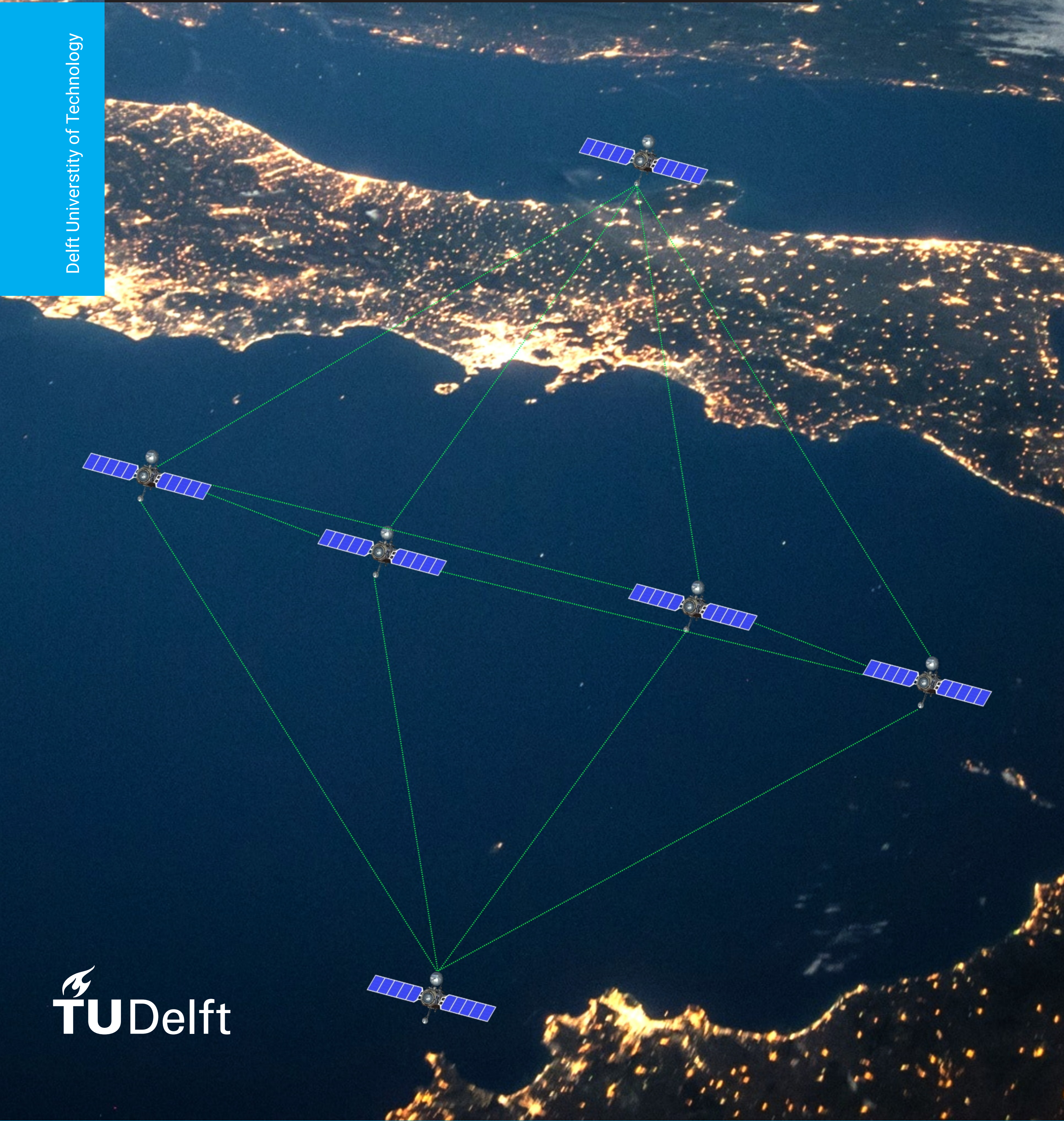
Distributed FDI using Recurrent Neural Networks for Thruster Faults in Formation Flying Satellites

Master Thesis

Martin Henkel

Delft University of Technology



TU Delft

Distributed FDI using Recurrent Neural Networks for Thruster Faults in Formation Flying Satellites

Master Thesis

by

Martin Henkel

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday, December 17, 2020 at 1:30 PM.

Student number:	4345509	
Project duration:	September 9, 2019 – December 17, 2020	
Thesis committee:	Dr. J. Guo,	TU Delft, supervisor
	Prof. E.K.A. Gill,	TU Delft
	Ir. C. de Wagter,	TU Delft

TU Delft

Preface

This document contains the Master thesis, the capstone of the MSc program in Aerospace Engineering at the Delft University of Technology. It contains the development as well as analysis of a neural network based fault detection and isolation system that is applied to a case study of formation flying satellites experiencing thruster faults.

I would like to thank both Dr. Robert Fonod and Dr. Jian Guo for their guidance and mentoring over the course of this thesis. Their input was always helpful, critical and thought provoking. I would also like to thank Professor Gill and Christophe de Wagter for agreeing to be on my thesis committee.
Furthermore, I thank my family as without them this thesis, and my entire education at the TU Delft, would not have been possible. Finally, I thank my fianceé from the bottom of my heart for supporting me during this period.

Martin Henkel
Siegen, December 2020

Summary

The use of distributed space systems, among them formation flying satellites, is increasing rapidly and these formation flying missions often require the use of precise orbital control in order to fulfil their mission goal. With this trend, a need is created for fast and robust fault detection and isolation (FDI), which can detect faults in the formation keeping control before the fault can lead to mission failure, either through a break-up of the formation or a collision. As the formation is made up of a spatially separated spacecraft, the information flow in between the satellites should be kept to a minimum in order to reduce cost and requirements on inter-satellite communication links.

In this thesis, the implementation of a distributed FDI methods based on multiple recurrent neural networks which utilize the already available relative positioning and velocity data is shown and these methods are tested on a representative formation flying mission. Specifically, so called Long Short-Term Memory (LSTM) networks were trained on data gathered from a formation simulation implemented in MATLAB. They are compared to a more traditional centralized Kalman filter based approach, which uses the entire formation state, in terms of their detection capability, isolation capability and their robustness.
Two separate neural network approaches were trained and tested; the first is a single network that was trained on the data of each satellite, while the second is the combination of six networks that each were only trained on the data coming from a single individual satellite. The former is called the 'naive' network, while the latter is called the 'combined individual' network.
Two types of thruster faults were investigated in particular. The first represents a blockage in a thruster valve, leading to a reduced output thrust, called the 'closed' fault for short. The second type of fault is a valve that cannot shut correctly, thereby always enabling propellant to leave the thruster, causing a constant force on the satellite, called the 'open' fault.

While the naive networks performed poorly both in almost all cases in detection and isolation, the combined individual network showed comparable performance to the Kalman filter in terms of isolating open thruster faults. The combined network isolation accuracy, precision and recall exceeded 99.9% for open faults. However, in regards to the closed fault type the performance was significantly worse with only a 22% detection recall and 50% isolation accuracy, compared to the 96% and 99.3% for the Kalman Filter. Furthermore, all networks were less robust and less capable of detecting low-intensity faults.
On the other hand, the Kalman filter was only capable of isolating thruster pairs, rather than individual thrusters. The networks also have the advantage of being less dependent on inter-satellite communication links. If the communication between satellites breaks down, the Kalman filter cannot cope with the lack of information, while the neural networks can still function as a de-centralized system. They also require less computational resources to make a decision, especially compared to a Kalman filter with time-varying innovation gain.

The performance on the closed faults can be explained by two factors. The first is the relative impact of closed faults compared to open. Due to their very nature, the closed faults are only having an impact on the system performance when the thruster is supposed to fire. This is not the case for the open faults which continuously impact the performance. A smaller impact will lead to a worse FDI performance and this was seen for all implemented methods.
The second is due to the training data pre-processing for closed faults. Due to the intermittency of the fault, the networks were sometimes trained on data that was indistinguishable from faultless data, but that was labeled faulty. This can cause confusion for the network and reduce the performance.
Overall, the neural network performed poorly in comparison the the centralized Kalman filter. A pre-processing procedure for the neural network training data that takes into account the individual faults and their impact on the network input is recommended and is expected to improve the performance to acceptable levels.

Contents

List of Figures

List of Tables

List of Symbols

A few notes regarding the notation presented in this thesis:

- **Bold** symbols represent vectors or matrices, lowercase symbols for vectors and uppercase symbols for matrices.
- A superscript T ($\boldsymbol{A}^T$) represent the transpose of a matrix.
- A dot ($\dot{x}$) over a symbol represents the derivative with respect to time of the quantity.
- The subscript 0 denotes the initial conditions of a quantity.
- For the Kalman filter, the superscripts $-$ and $+$ are used to indicate the pre- (*a priori*) and post-measurement (*a posteriori*) estimates, respectively.

Table 1: List of symbols

Symbol	Meaning	Unit
a	Semi-major axis	[m]
a_1	Activation of first neuron	[-]
A	Surface area	[m^2]
$\boldsymbol{a}^l$	Neural network activation vector of layer l	[-]
$\boldsymbol{A}$	System matrix	[-]
b_1	Bias of first hidden neuron	[-]
$\boldsymbol{b}^l$	Neural network bias vector of layer l	[-]
$\boldsymbol{B}$	Input matrix	[-]
$\boldsymbol{B}_{mag}$	Magnetic field vector	[T]
c	Speed of light	[m s^{-1}]
C	Generic cost function	[-]
C_{ce}	Cross-entropy cost function	[-]
C_{quad}	Quadratic cost function	[-]
C_D	Drag Coefficient	[-]
C_R	Reflectivity	[-]
$\boldsymbol{C}$	Measurement/Output matrix	[-]
$\boldsymbol{d}$	Disturbance vector	[-]
$\boldsymbol{D}$	Feed-forward matrix	[-]
e	Orbital eccentricity	[-]
$\boldsymbol{E_x}$	State disturbance matrix	[-]
$\boldsymbol{E_y}$	Measurement disturbance matrix	[-]
f_{thrust}	Thruster force magnitude	[N]
$\boldsymbol{f}$	Fault vector	[-]
$\boldsymbol{F}_{aero}$	Aerodynamic disturbance force	[N]
$\boldsymbol{F}_{grav}$	Gravitational disturbance force from third bodies	[N]
$\boldsymbol{F}_{mag}$	Magnetic (Lorentz) disturbance force	[N]
$\boldsymbol{F}_{rad}$	Radiation pressure disturbance force	[N]
$\boldsymbol{F_x}$	State fault matrix	[-]
$\boldsymbol{F_y}$	Measurement fault matrix	[-]
g	Decision Function	[-]
h_k	Kalman Filter Detection Threshold	[-]
h_n	Neural Network Detection Threshold	[-]
$\boldsymbol{H_{yd}}$	Disturbance transfer function	[-]
$\boldsymbol{H_{yf}}$	Fault transfer function	[-]
$\boldsymbol{H_{yu}}$	Input transfer function	[-]
$\boldsymbol{H_{yx}}$	State transfer function	[-]
i	Orbital inclination	[rad]
$\boldsymbol{I}$	Unit matrix	[-]
J	Quadratic performance index in Linear-Quadratic-Regulator	[-]
J_2	Second-order Zonal Coefficient	[-]
k	Discrete Time	[s]
$\boldsymbol{K}$	Kalman filter gain matrix	[-]
$\boldsymbol{K}_{LQR}$	Linear-Quadratic regulator rain matrix	[-]
M	Mean anomaly	[rad]

Table 2: List of symbols (continued)

Symbol	Meaning	Unit
n	Mean orbital motion	[]
p	Semi-latus rectum	[m]
$p_\theta(z)$	Probability of sampling observation z from distribution θ	[-]
$\boldsymbol{p}$	Position vector	[m]
$\boldsymbol{P}$	Solution matrix to algebraic Ricatti equation arising from LQR control	[-]
$\boldsymbol{P_k}$	Estimation error covariance matrix	[-]
q_{sat}	Electric charge accumulated by spacecraft	[C]
$\boldsymbol{Q}$	State weight matrix	[-]
$\boldsymbol{Q_x}$	Process noise covariance matrix	[-]
$\boldsymbol{Q_y}$	Measurement noise covariance matrix	[-]
$\boldsymbol{r}$	Position vector	[m]
R_E	Earth radius	[m]
$\boldsymbol{R}$	Input weight matrix	[-]
$\boldsymbol{R}_B^A$	Arbitrary rotation matrix	[-]
s	Complex frequency	[rad s^{-1}]
$s(z)$	Log-likelihood ratio in CUSUM method	[-]
S	Cumulative Sum in CUSUM method	[-]
$\boldsymbol{S}$	Final state weight matrix	[-]
$\boldsymbol{S}(\boldsymbol{\omega})$	Skew-symmetric matrix of vector $\boldsymbol{\omega}$	[-]
t	Continuous time	[s]
t_{fault}	Fault inception time	[s]
T	Orbital period	[s]
$\boldsymbol{T}$	Observer input matrix	[-]
$\boldsymbol{T}_B^A$	Frame transformation matrix from reference frame A to B [-]	
$\boldsymbol{T}_{config}$	Thruster configuration matrix	[-]
$\boldsymbol{T}_{ECEF}^{ECI}$	Frame transformation matrix from the ECI frame to ECEF frame	[-]
$\boldsymbol{T}_z$	Rotation matrix around z axis	[-]
$\boldsymbol{u}$	Input vector	[-]
U	Electric potential difference	[V]
$\boldsymbol{v}$	Process noise vector	[-]
$\boldsymbol{v}$	Velocity vector	[m s^{-1}]
$\boldsymbol{V}_ij$	Discrete Transfer function between signals i and j	[-]
w_1	Weight of first neuron connection	[-]
$\boldsymbol{w}$	Measurement Noise Vector	[-]
$\boldsymbol{W}^l$	Neural Network Weight matrix of layer l	[-]
x	Relative distance in radial direction	[m]
$\boldsymbol{x}$	State vector	[-]
$\boldsymbol{x}_{t1}$	State vector at time t_1	[-]
$\hat{\boldsymbol{x}}_A$	Unit vector in direction of the first axis of reference frame A	[-]
y	Relative distance in along-track direction	[m]
$\boldsymbol{y}$	Measurement/Output vector	[-]
$\hat{\boldsymbol{y}}_A$	Unit vector in direction of the second axis of reference frame A	[-]
z	Relative distance in cross-track direction	[m]
$\boldsymbol{z}$	General observer state vector	[-]
$\hat{\boldsymbol{z}}_A$	Unit vector in direction of the third axis of reference frame A	[-]

Table 3: List of Symbols (continued) - Greek letters

Symbol	Meaning	Unit
α	CUSUM distribution parameter	[-]
β	F-Measure Weighting Factor	[-]
γ	Angle used to construct quaternion	[rad]
δ	Network sensitivity	[-]
ΔV	Velocity Increment	[$\mathrm{m\,s^{-1}}$]
ϵ_0	Permittivity of free space	[$\mathrm{F\,m^{-1}}$]
μ	Mean of a distribution	[-]
μ_g	Standard gravitational parameter	[$\mathrm{m^3\,s^{-2}}$]
π	Circle constant	[-]
σ	Standard deviation of a distribution	[-]
θ	True anomaly	[rad]
ϕ	Parameter in Relative Motion Equation	[-]
ω	Argument of periapsis	[rad]
ω_{Earth}	Rotational velocity of the Earth	[$\mathrm{rad\,s^{-1}}$]
$\boldsymbol{\omega}$	Rotational velocity vector	[$\mathrm{rad\,s^{-1}}$]
Ω	Right Ascension of the Ascending Node (RAAN)	[rad]

Table 4: List of abbreviations

Abbreviations	Meaning
AAReST	Autonomous Assembly of a Reconfigurable Space Telescope
ADCS	Attitude Determination and Control System
ARE	Algebraic Ricatti Equation
ANN	Artificial Neural Network
CPOD	Cubesat Proximity Operations Demonstration
CSV	Comma-Seperated Values
CUSUM	Cumulative Sum
DARE	Discrete Algebraic Ricatti Equation
ECI	Earth-Centered Inertial
ECEF	Earth-Centered, Earth-Fixed
EKF	Extended Kalman Filter
FASTRAC	Formation Autonomy Spacecraft with Thrust, Relnav, Attitude and Crosslink
FDI	Fault Detection and Isolation
FF	Formation Flying
GLR	Generalized Likelihood Ratio
GNC	Guidance, Navigation and Control
GRACE	Gravity Recovery and Climate Experiment
GRAIL	Gravity Recovery and Interior Laboratory
HCW	Hill-Clohessy-Wiltshire
LOS	Line Of Sight
LQR	Linear-Quadratic Regulator
LSTM	Long Short-Term Memory
LTI	Linear Time-Invariant
MMS	Magnetospheric Multiscale Mission
MIMO	Multiple Input, Multiple Output
PROBA	Project for On-Board Autonomy
PRISMA	Prototype Research Instruments and Space Mission technology Advancement
RAAN	Right Ascension of the Ascending Node
RMS	Root-Mean-Square
RNN	Recurrent Neural Network
STM	State Transition Matrix
SXSY	Satellite No. X, Thruster No. Y
UIUC	University of Illinois at Urbana-Champaign
UKF	Unscented Kalman Filter
VRB	Virtual Rigid Body

1 Introduction

In this chapter the main topics of the thesis are introduced. First, the relevance of the work is explained in Section 1.1, followed by the the goals of the thesis and the research questions which this thesis aims to answer in Section 1.2. Finally, an overview of the thesis structure is given in Section 1.3.

1.1. Relevance

Planning, launching and operating a space mission is a difficult endeavor. Challenges arise from the environment via the radiation both from the Sun and other sources, the difficult thermal conditions as well as the physical distance to Earth. Due to these difficulties, machines and circuitry experience increased wear and require specific designs in order to cope with these conditions, and this is not to mention the difficulty that is presented for human space travel. This, in combination with the large launch and equipment costs, leads to space missions which need to be robust and reliable to avoid a mission failure.

In order to ensure the safety and reliability of space missions, spacecraft contain additional redundant systems and components to be able to cope with faults or failures in any part of the system. A fault in this case is defined as the deviation from the design parameters of any part to the detriment of the entire system [7]. For example a single bit-flip in a digital circuit, a mechanical component getting stuck or a sensor no longer updating its measurements. It is distinguished from a failure, which describes the "inability of a system or component to accomplish its function" [8, p.7] and is irrecoverable even with a system in place that can handle faults.
In addition to physical redundancies, there are also analytic or software redundancies which change the behavior of a system experiencing a fault to maintain the safe system operation.
However, all of these redundancies require the capability of detecting when a fault has occurred in the system in order to recover and return to the original safe design conditions. The consequences of these faults can range from benign, as in the case of a bit flip in an image file, slightly altering the color of one pixel, to catastrophic, for example in the explosion of the Ariane 5 rocket due to a software bug [9].

A complete fault-tolerant system generally can detect, isolate, identify and recover from faults. First a few definitions are given:

Fault Detection:	the binary decision ('yes' or 'no') on whether or not a fault has occurred anywhere in the system.
Fault Isolation:	determining the location of a fault in the system.
Fault Identification:	determining what kind of fault has occurred and how severe it is
Fault Recovery:	taking appropriate action to eliminate or mitigate the fault.

These systems are used in almost all fields of engineering, ranging from industrial [10], chemical [11], robotics [12], maritime [13] and of course aerospace engineering [14]. Their importance in ensuring the proper performance of system cannot be overstated.
Furthermore, in the field of aerospace engineering in particular there has been a trend towards distributed

systems. This includes both satellite formations which generally consist of few satellites in relative proximity and satellite constellations which generally are made up of many satellites spread over a large distance. According to Scharf et al, formation flying missions are characterized by their dynamics being "coupled by a common control law" [15]. This is in opposition to satellite constellations where the satellites keep their orbit independent of other satellites.
The amount of distributed space systems has steadily increased over the past years (see Figure 1.1[1]) with relatively small formations of satellite finding use in e.g. the Magnetospheric Multiscale Science (MMS) mission [16], the GRACE and GRAIL gravimetry missions [17, 18] or the planned PROBA-3 mission [19].

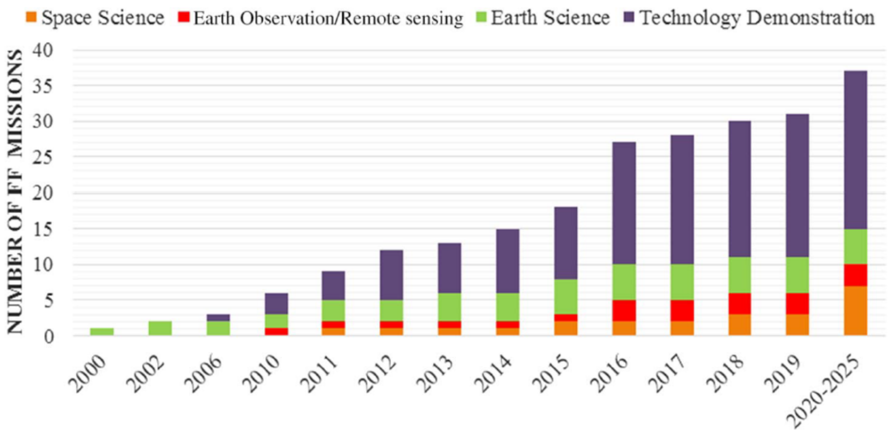


Figure 1.1: Cumulative formation flying mission launches classified based on their primary objective [1].

These distributed systems have distinct advantages: the spatial separation of the spacecraft can enable unique measurements. For example, the gravity recovery and climate experiment (GRACE) mission used two satellites in the same orbit in order to map Earth's gravity field [17]. A similar mission (GRAIL) was performed to determine the gravitational field of the Moon with unprecedented accuracy [18]. The PROBA-3 mission will use two satellites in order to perform coronagraph measurements of the sun, one satellite blocking the main body so that the other can image the outer layers of the Sun [20].
However, this approach also comes with downsides. The spacial separation introduces new risks, most prominently the risk of collision, which is especially important for close formations and would result in complete mission failure. To avoid this risk, the spacecraft need a well-functioning orbit control system, which in turn requires well functioning control thrusters. In order to ensure the functionality of the orbit control thruster, an FDI system is necessary.
Traditional FDI methods have access to all the information that is desired, however in a distributed system the exchange of information is associated with additional energy costs and time delays, as well as requirements on having a stable inter-satellite communication link. In this thesis, the following definitions for the terms *centralized, de-centralized* and *distributed* in the context of an information exchange are used, and will be illustrated using the example of a fleet of ships:

Centralized In a centralized system, all information is gathered at a single system which makes decisions (control or diagnosis) about all components of the system. For example, a fleet of ships all send the information to a central location (e.g. a central mothership or another location), which in turn makes a decision concerning every ship in the fleet.

De-Centralized In a de-centralized system, a system is broken up into several sub-system which have access to a subset of the information and independently make decisions using the available information. In such an approach, each ship in the fleet would make decisions based on the available information on the ship.

Distributed In a distributed system, several sub-systems only have access to a subset of the available information but communicate in order to reach a common goal. In this approach, each ship in the fleet would communicate some or all of its information or decisions which the other ships would incorporate in their own decision making.

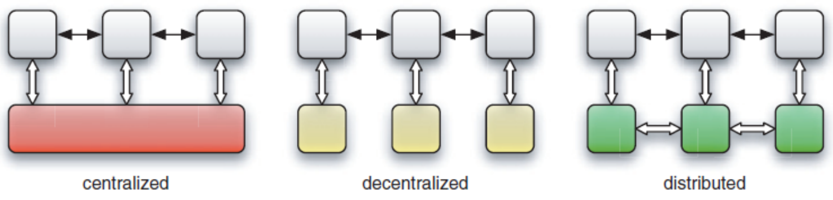


Figure 1.2: Graphical illustration of the difference between centralized, de-centralized and distributed systems.

These distinctions are graphically illustrated in Figure 1.2.
In a satellite formation, the communication between satellites is more complicated. This in turn makes a centralized system less appealing and makes it more advantageous to have a de-centralized or distributed system which can detect and isolate faults. The use of distributed FDI systems has been a topic in research for the past years for formation flying satellites [21–24] as well as other distributed systems, e.g. mobile robots or industrial systems [25–27].

The use of so called *artificial neural networks* has gotten significantly more attention in research and in general media in recent years, This is in part due to the impressive achievements made by deep learning systems such as Google's Deepmind [28, 29]. The advancement in computer hardware has allowed for handling larger amounts of data, which in turn generally improve the performance of a neural network [30, chp. 1].
The use of neural networks can also be seen in FDI systems, especially those for which faults are difficult to handle otherwise [31–33]. In addition to the capability of neural network to handle faults which are difficult to model, the neural networks also have benefits specific to spacecraft applications. Due to the low computational requirements the neural network can be evaluated very quickly after it has been trained.
The fact that neural networks tend to work best for large amounts of data may make it seem counter-intuitive to use them in an environment where sharing data is cost prohibitive, such as in a satellite formation. However, the satellites do already have the relative position and velocity data locally available as it is needed for the formation control system. This data can be used to train the network and in operation be used for the evaluation of the trained network. Sharing the resulting output then enables distributed FDI.

1.2. Thesis Goals and Research Questions

The topic of this thesis lies at the intersection of all the previously mentioned topics. In the previously conducted literature study it was found that the use of neural networks in distributed FDI is sparse, especially for formation flying satellites. This could be due to the limitations inherent to neural networks, the relative straightforwardness of their applications, or a lack of performance.
However, a neural network based system has several advantages. They are fast to compute once they have been trained. This is especially relevant for space applications which have limited computational resources. Furthermore, they can tackle problems which are difficult to solve otherwise.
This research aims to close the gap in knowledge regarding the use of neural network based distributed FDI compared to a more traditional centralized approach, in order to determine the efficacy of neural networks for this application. There are two main goals this thesis is trying to address: (1) to propose a solution to the problem of formation flying FDI for thruster faults by designing a new distributed FDI method based on neural networks and (2) to evaluate its performance by comparing it to a centralized model-based method in multiple fault detection and isolation performance parameters, when applied to a realistic formation flying scenario.
This leads to the main research question:

How does a distributed, neural-network based FDI architecture for thruster faults compare to a centralized model-based architecture in terms of detection capability, isolation capability and robustness in a close formation flying mission?

By considering the various aspects of this main question, it can be broken up into several sub-questions:

- *How does the detection and isolation performance (accuracy, precision and recall) compare between the centralized and distributed approaches?*
  The main point of comparison should be the performance of the methods. The three mentioned per-

formance parameter are often used to determine the quality of a classification system. The *accuracy* measures the overall percentage of correct classifications. The *precision* quantifies how often the system is correct on a positive decision (in this case, the presence of a fault). On the other hand, the *recall* determines how often a positive occurrence (the presence of a fault) is detected as such.

- *How quickly and consistently do the methods respond to the faults?*
  Two more important measures, unrelated to the classification performance itself, are the time it takes for the system to come to a decision and if the system sticks with its decision.
- *How robust are the FDI methods to scenarios they were not designed for?*
  In practice, the system will encounter situations for which they were not designed. Testing the robustness of the developed approaches to such scenarios is crucial in determining their value.
- *What is the influence of model uncertainties and disturbances on the detection and isolation?*
  One of the issues of neural networks is their sensitivity to noise. Even small changes can radically change the output of a network [34]. As such, it is important to determine how the effect of model uncertainties and various disturbance forces affect the systems.
- *What is the effect of distributing the FDI system on the computation time, bandwidth usage and time until fault detection?*
  As one of the systems will be distributed, it will be interesting to see the effect this has on the parameters, which are affected by the distribution. This includes the computation time, bandwidth usage and (due to communication delays) time until fault detection.

Answering these questions should give insight into the quality of neural-network based systems for FDI.
The scenario that will be used to achieve this aim and answer the research questions was identified during the literature study. A so-called Virtual-Rigid-Body (VRB) formation, similar to the formation used in the MMS mission, will be simulated in MATLAB. VRB formations require active control and therefore have a stronger need for an FDI system for the orbital control thrusters of the satellites in the formation.
Two types of thruster faults will be considered: a fault causing a reduction in the thrust (imagine a valve that is jammed, reducing the mass flow of the thruster) and one causing a constant thrust from the affected thruster (imagine a valve that suddenly fails and causes constant mass flow). For this scenario, the performance of the implemented FDI approaches will be analyzed and compared.

1.3. Thesis Outline

The thesis is structured as follows: First, the relevant theoretical background is explained in Chapter 2. Here, the necessary background is provided on satellite formation flying, neural networks as well as FDI basics in order to understand the remainder of the thesis.
After the necessary background has been explored, the set-up of the simulation required to meet the research aims and enable the data generation that the networks will be trained on is explored in Chapter 3. This chapter introduces the specific simulated mission, how the mission parameters were determined, the simulation structure and its verification as well as the set-up for generating the necessary training data.
This is followed by the specific fault detection and isolation methodology, detailed in Chapter 4. This chapter contains the set-up, structure, platform and training of the neural network as well as the model-based FDI comparison method.
After this, the two approaches are analyzed, compared and these results presented in Chapter 5. This analysis includes the detection and isolation performance as well as an investigation into the robustness of both methods.
Finally, conclusions and recommendation are presented in Chapter 6.

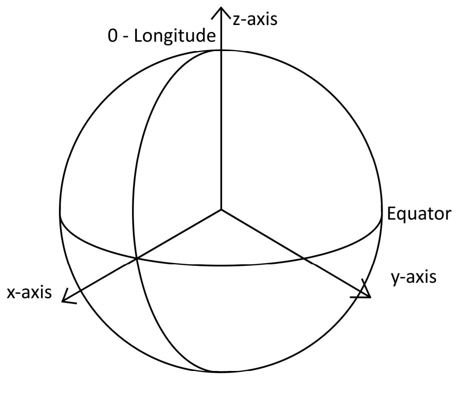
2 Background Information

In this chapter the relevant theoretical background for the thesis is explained. First, the relevant reference frames in this thesis are explained in Section 2.1. Then, a short survey of formation flying missions in general is presented in Section 2.2, followed by the relative dynamics of formations in Section 2.3. The topic of particular formation control is explored in Section 2.4. Furthermore, the basics of neural networks and their training are shown in Section 2.5. Finally, the Kalman filter is explained in Section 2.6.

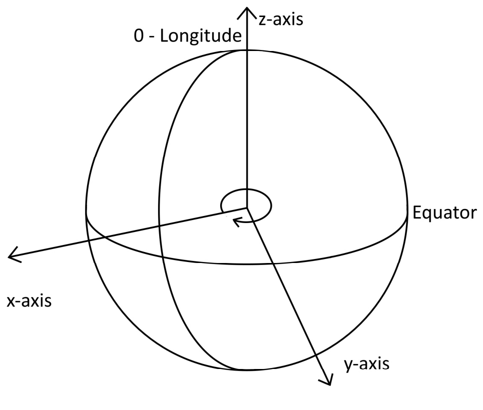
2.1. Reference Frames

Four main frames of reference are used in the thesis, and they are listed below

- Earth-Centered Earth-Fixed (ECEF) frame (shown in Figure 2.1a)
- Earth-Centered Inertial (ECI) frame (shown in Figure 2.1b)
- Euler-Hill frame (shown in Figure 2.2, left)
- Satellite Body-Fixed frame (shown in Figure 2.2, right)



(a) Illustration of the Earth-Centered Earth-Fixed (ECEF) reference frame.



(b) Illustration of the Earth-Centered Inertial (ECI) reference frame.

The Earth-Centered frames have their origin at the center of the Earth. The ECEF rotates with the Earth and is constructed by taking the x-axis radiating out from the center of the earth along the 0-longitude line at the equator, the z axis out of the north pole and letting the y axis complete the right-handed frame.
The Earth-Centered-Inertial frame shares the origin of the ECEF frame, but it does not rotate along with the Earth. Instead its axes stay fixed relative to the distant stars. The specific ECI frame in this thesis is chosen to

be the one where the axes of the ECEF and ECI coincide at the beginning of the simulation.
As such the conversion from the ECI to the ECEF frame is

$$\boldsymbol{T}_{ECEF}^{ECI} = \boldsymbol{T}_z\left(\omega_{Earth} t\right) \tag{2.1}$$

Where ω_{Earth} is the rotational speed of the earth and t is the time that has passed since the start of the simulation.

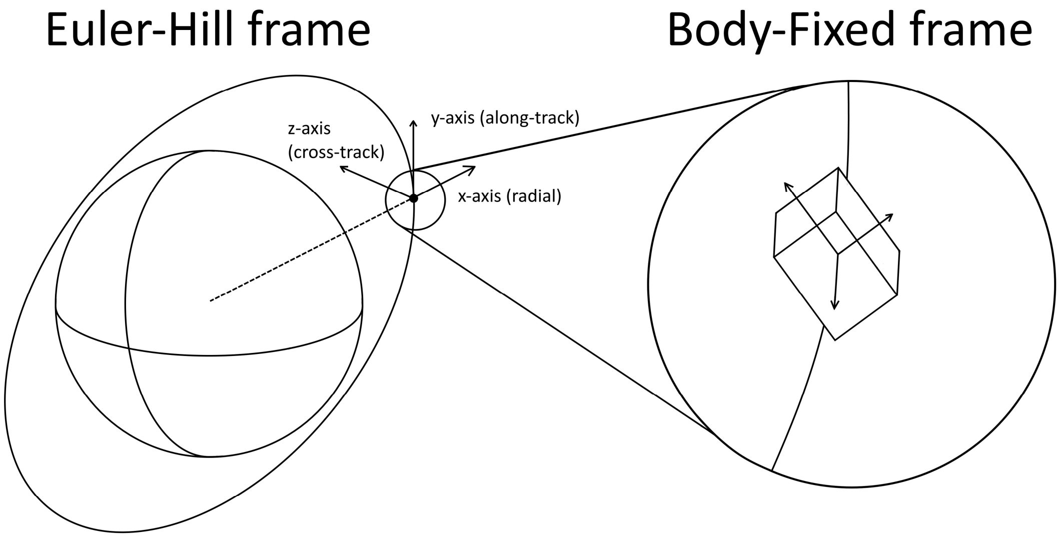


Figure 2.2: Illustration of the Euler-Hill reference frame (left) and the satellite body reference frame (right).

The Euler-Hill frame is a non-inertial reference frame fixed on a point in orbit. Its axes are defined as follows: the x-axis lies on the line connecting the orbiting point and the center of the Earth, pointing radially outward. The z-axis points in the direction of the angular momentum, i.e. it is perpendicular to the orbital plane. The y-axis completes the right handed frame. These x,y and z directions are also called the 'radial', 'along-track' and 'cross-track' direction, respectively. The direction cosine rotation matrix from the ECI frame to the Hill frame is constructed from these definitions. This can be achieved by normalizing the unit vector belonging to the frame and computing the matrix shown in Equation (2.2).

$$\boldsymbol{T}_{Hill}^{ECI} = \begin{bmatrix} \hat{\boldsymbol{x}}_{Hill}\cdot\hat{\boldsymbol{x}}_{ECI} & \hat{\boldsymbol{x}}_{Hill}\cdot\hat{\boldsymbol{y}}_{ECI} & \hat{\boldsymbol{x}}_{Hill}\cdot\hat{\boldsymbol{z}}_{ECI} \\ \hat{\boldsymbol{y}}_{Hill}\cdot\hat{\boldsymbol{x}}_{ECI} & \hat{\boldsymbol{y}}_{Hill}\cdot\hat{\boldsymbol{y}}_{ECI} & \hat{\boldsymbol{y}}_{Hill}\cdot\hat{\boldsymbol{z}}_{ECI} \\ \hat{\boldsymbol{z}}_{Hill}\cdot\hat{\boldsymbol{x}}_{ECI} & \hat{\boldsymbol{z}}_{Hill}\cdot\hat{\boldsymbol{y}}_{ECI} & \hat{\boldsymbol{z}}_{Hill}\cdot\hat{\boldsymbol{z}}_{ECI} \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{x}}_{Hill}^T \\ \hat{\boldsymbol{y}}_{Hill}^T \\ \hat{\boldsymbol{z}}_{Hill}^T \end{bmatrix} \tag{2.2}$$

It should be noted that due to the short time scales involved in this thesis, the precession and nutation of the Earth's rotational axis are not taken into account.

Due the rotation of the frame, the conversion of velocity from the ECI to the Hill frame the derivative of the rotation matrix needs to be taken into account, as shown in Equations (2.3) to (2.5).

$$\boldsymbol{r}_{Hill} = \boldsymbol{T}_{Hill}^{ECI}\left(\boldsymbol{p}_2 - \boldsymbol{p}_1\right)_{ECI} \tag{2.3}$$

$$\rightarrow \dot{\boldsymbol{r}}_{Hill} = \boldsymbol{T}_{Hill}^{ECI}\left(\dot{\boldsymbol{p}}_2 - \dot{\boldsymbol{p}}_1\right)_{ECI} + \dot{\boldsymbol{T}}_{Hill}^{ECI}\left(\boldsymbol{p}_2 - \boldsymbol{p}_1\right)_{ECI} \tag{2.4}$$

$$\boldsymbol{v}_{Hill} = \boldsymbol{T}_{Hill}^{ECI}\left(\boldsymbol{v}_2 - \boldsymbol{v}_1\right)_{ECI} + \dot{\boldsymbol{T}}_{Hill}^{ECI}\left(\boldsymbol{p}_2 - \boldsymbol{p}_1\right)_{ECI} \tag{2.5}$$

For any rotation matrix its time derivative can be expressed by Equation (2.6) [35, 36].

$$\dot{\boldsymbol{R}}_B^A = -\boldsymbol{S}(\boldsymbol{\omega}_B)\boldsymbol{R}_B^A \tag{2.6}$$

Where $\boldsymbol{R}_B^A$ is the rotation matrix from frame A to frame B and $\boldsymbol{S}$ is a skew-symmetric matrix corresponding to the angular velocity vector $\boldsymbol{\omega}$, expressed in frame B. Hence the conversion between the two frames is given by Equations (2.7) and (2.8).

$$\boldsymbol{r}_{Hill} = \boldsymbol{T}_{Hill}^{ECI} \left(\boldsymbol{p}_2 - \boldsymbol{p}_1\right)_{ECI} \tag{2.7}$$

$$\boldsymbol{v}_{Hill} = \boldsymbol{T}_{Hill}^{ECI} \left(\boldsymbol{v}_2 - \boldsymbol{v}_1\right)_{ECI} - \boldsymbol{S}\left(n\boldsymbol{z}\right) \boldsymbol{T}_{Hill}^{ECI} (\Omega) \left(\boldsymbol{p}_2 - \boldsymbol{p}_1\right)_{ECI} \tag{2.8}$$

2.2. Formation Flying Missions

According to Scharf et al, formation flying missions are distinguished from the other common type of distributed space system, a constellation, by the fact that the dynamics of the spacecraft are "coupled through a common control law" [15]. Formations of satellites can furthermore be divided into so called 'passive' and 'active' formations. The former requires no active control to keep the formation close, as the orbits themselves are designed to keep the formation from drifting apart. 'Active' formations on the other hand follow non-Keplerian orbits and as such require constant orbit corrections in order to keep the formation. An example of an active formation using the so-called Virtual-Rigid-Body concept is the Magnetospheric Multiscale Science Mission (MMS) [16]. In this mission, the satellites used active control to stay in fixed position relative to each other for a portion of each orbit [37].
An overview of formation flying missions that were surveyed in the literature study can be found in Table 2.1. As can be seen, most formation flying missions that have already been launched consist of only a small number of satellites (less than 5). However, ambitious concepts such as the Stellar Imager (SI) missione exist which proposes the use of 30 satellites. Table 2.1 shows that, while many formation missions only involve a small amount of satellites, the number of satellites is expected to increase as the technology for precise formation flying matures.

2.3. Formation Relative Dynamics Model

The first attempts made at describing the relative dynamics of bodies in an orbital environment were made by G.W. Hill in 1878 [54], specifically in the lunar environment. During the 1960s the same equations for orbital rendezvous were derived independently by W.H. Clohessy and R.S. Wiltshire [55]. The so called Hill-Clohessy-Wiltshire (HCW) Equations are very useful tools for describing the relative motion in close formations, and are given in Equations (2.9) to (2.11).

$$\ddot{x} - 2n\dot{y} - 3n^2 x = u_x \tag{2.9}$$

$$\ddot{y} + 2n\dot{x} = u_y \tag{2.10}$$

$$\ddot{z} + n^2 z = u_z \tag{2.11}$$

Where x, y, z are the relative distances in radial, along track and cross track direction respectively, n is the mean orbital motion of the reference satellite, and u_x,u_y and u_z are additional accelerations in the directions indicated by the subscript. These directions are based on the Hill coordinate frame, which is defined as follows: the radial direction is in the direction of the vector from the center of the orbited body (in this case the Earth) to the satellite, the cross-track direction is in the direction of the angular momentum vector of the satellite and the along-track direction is chosen to complete the right-handed frame. For a circular reference orbit, the along-track direction is parallel to the velocity vector of the reference satellite.
The assumption used to derive these equations are as follows: The object in a circular reference orbit ($e = 0$) and no disturbances such as drag, solar radiation pressure or gravitational disturbances are present. The

[1]https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Proba_Missions/About_Proba-3 accessed 16.07.2020
[2]https://www.nasa.gov/directorates/spacetech/small_spacecraft/cpod_project.html accessed 16.07.2020

Table 2.1: Overview of surveyed formation flying missions.

Mission Name	**Description**	**No. of Satellites**	**Launch Year**	**Relative Distance [km]**
PROBA-3	A technology demonstration mission to validate high accuracy formation flying and to gather coronagraph measurements [19, 20, 38]	2	2022[1]	0.025-0.25
TanDEM-X	A scientific mission that generated a digital elevation map of the Earth using synthetic aperture radar techniques [39, 40]	2	2010	20
PRISMA	Technology demonstration mission to act as "an in-orbit test bed for Guidance, Navigation and Control (GNC) algorithms for advanced closed-loop formation flying and rendezvous" [41]	2	2010	0.15-2.7
CanX-4/ CanX-5	A technology demonstration mission to showcase high accuracy formation flying with nano-satellites [42–44]	2	2014	0.05-2
MMS	Investigating the tail of Earth's magnetic field using four satellites to approximate its gradient [16]	4	2015	10-400
GRACE	Studied Earth's gravitational field using the relative separation of two satellites in order to generate spherical harmonics model [45]	2	2002	250
GRAIL	Studied the Moon's gravitational field using the relative separation of two satellites in order to generate spherical harmonics model (see GRACE) [18]	2	2011	200 (40)
FASTRAC	Student-run technology demonstration mission to show formation flying capabilities of nano-satellites [46, 47, 47]	2	2010	n/a
AAReST	Technology demonstration mission that aims to demonstrate the in-orbit assembly and reconfiguration of a space telescope from micro-satellites [48]	3	n/a	n/a (docking)
CPOD	Mission to demonstrate docking and close-proximity operations with CubeSats [49]	2	2021 [2]	n/a (docking)
XEUS	Scientific mission to study the early universe by use of an x-ray telescope comprised of two spatially separated spacecraft [50]	2	n/a	0.05
Exo-S	Scientific mission to gaher direct images of exo-planets [51]	2	n/a	250
Stellar Imager (SI)	Scientific mission concept to capture the surface of distant stars [52]	30	n/a	0.5
UIUC-JPL Missions	Two separate mission concepts that aim to show advanced formation flying using a group of four or six nano-satellites [53]	4/6	n/a	0.05

HCW equations have analytical solutions, shown in Equations (2.12) to (2.14) [6, p.130].

$$x(t) = \left(\dot{x}_0/n\right)\sin(nt) - \left(3x_0 + 2\dot{y}_0/n\right)\cos(nt) + 4x_0 + 2\dot{y}_0/n \tag{2.12}$$

$$y(t) = \left(2\dot{x}_0/n\right)\cos(nt) + \left(6x_0 + 4\dot{y}_0/n\right)\sin(nt) - \left(6nx_0 + 3\dot{y}_0\right)t - 2\dot{x}_0/n + y_0 \tag{2.13}$$

$$z(t) = \left(\dot{z}_0/n\right)\sin(nt) + z_0\cos(nt) \tag{2.14}$$

Where the subscript "0" indicates the initial conditions.
In order to use these linearized dynamics in the controller it will be useful to express them in the state space form, consisting of the state vector $\boldsymbol{x} = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T$, and the input vector $\boldsymbol{u} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^T$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 3n^2 & 0 & 0 & 0 & 2n & 0 \\ 0 & 0 & 0 & -2n & 0 & 0 \\ 0 & 0 & -n^2 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{u} \tag{2.15}$$

These equations assume not only the absence of disturbance forces but also the homogeneity of the gravitational field. Various adaptions exist which take into account elliptical reference orbits or the various irregularities in the Earth's shape and size. The most prominent of these irregularities is the bulging around the equator, with the resulting acceleration known as the J2 effect [56]. A variation of the relative dynamics is shown in Equation (2.16) which takes into account the average effect of this additional material around the equator over one orbital period [57].

$$\boldsymbol{A}_{J2} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ (5\phi^2 - 2)n^2 & 0 & 0 & 0 & 2n\phi & 0 \\ 0 & 0 & 0 & -2n\phi & 0 & 0 \\ 0 & 0 & -(3\phi^2 - 2)n^2 & 0 & 0 & 0 \end{bmatrix} \tag{2.16}$$

$$\phi = \sqrt{1 + (1 + 3\cos(2i))\frac{3J_2R_E^2}{8r^2}} \tag{2.17}$$

Where i is the orbital inclination r is the radius of the circular orbit, R_E is the radius of the Earth and J_2 is the second zonal coefficient with a value of approximately 1082.645e−6 [58].

2.4. Formation Control

The control approach used in this thesis is based on the state space model representation of a system. Specifically, that of a discrete system which is shown in Equations (2.18) and (2.19)

$$\boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) \tag{2.18}$$

$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x}(k) + \boldsymbol{D}\boldsymbol{u}(k) \tag{2.19}$$

Where $\boldsymbol{x}$ is the state vector of the system, $\boldsymbol{u}$ is the input vector, $\boldsymbol{y}$ is the output or measurement vector, $\boldsymbol{A}$ is the state transition matrix, $\boldsymbol{B}$ is the input matrix, $\boldsymbol{C}$ is the measurement matrix, $\boldsymbol{D}$ is the feed-through matrix and k is the discrete time step.
The general approaches to formation control are shown in Section 2.4.1 and the use of the linear-quadratic regulator as a means to derive a control gain is explained in Section 2.4.2. The formation-flying specific control concept called "virtual-center-control" in shown Section 2.4.3.

2.4.1. Control Architectures

The architectures used to control the formation can be divided into five categories: Multiple-Input Multiple-Output (MIMO), Leader/Follower, Virtual Structure, Cyclic and Behavioral [59], although it should be mentioned that there is significant overlap between the different categories. Similar architecture distinctions were made by G.P. Liu and S. Zhang [60].

- The MIMO assumes the entire formation to be a single system with multiple inputs and outputs. This requires extensive information exchange between the different spacecraft as every single spacecraft will require information about the entire formation to make control decisions. However, the high amount of available information generally enables higher accuracy. Additionally, the general stability of such systems can easily be guaranteed.
- In the Leader/Follower (L/F) (also called target/chaser) architecture, the leader spacecraft are used as a reference for the follower spacecraft. When each follower spacecraft can only have one leader, this is also referred to as single-leader L/F. This approach is currently the most common, as it highly suited for missions with two or three satellites [59]. The leader(s) can also be virtual, i.e. non-existent, in which case this approach becomes similar to the virtual structure approach.
- The Virtual Structure (VS) or Virtual-Rigid-Body (VRB) architecture treats the spacecraft as vertices embedded in a larger, virtual rigid body. The motion of this body is specified according to the needs of the mission, and the motion of the vertices is then used to generate reference-trajectories. A distinction between the control and the guidance has to be made here [59]: the VRB guidance has the the same property, which gets used to determine the trajectories of each satellite. If the satellite merely follows the trajectory, it is not strictly speaking a control architecture. However, if the VRB is kept via relative navigation, the states do depend on the other members and as such it can be considered a distributed control architecture.
- A Cyclic architecture is similar to the L/F in the sense that some spacecraft directly depend on the states of others. Unlike L/F however, the cyclic architecture allows for loops to form in the directed graph of the control dependencies. This avoids the single point of failure of the L/F architecture, which fails if the leader satellites breaks down.
- In a behavioral architecture, there are multiple goals, each of which has its own controller and whose outputs get combined to result in a specific behaviour. These multiple goals are dependent on the specific constraints of the system, but can include goals like collision avoidance, formation keeping or moving to a specific goal location. This is the most abstract of the control architectures, as the individual goals themselves can be achieved through various control strategies.

The control dependencies of the different architectures can be visualized using directed graphs, with the direction of the connections indicating a dependency. Examples for the L/F and the cyclic architecture are shown in Figure 2.3.

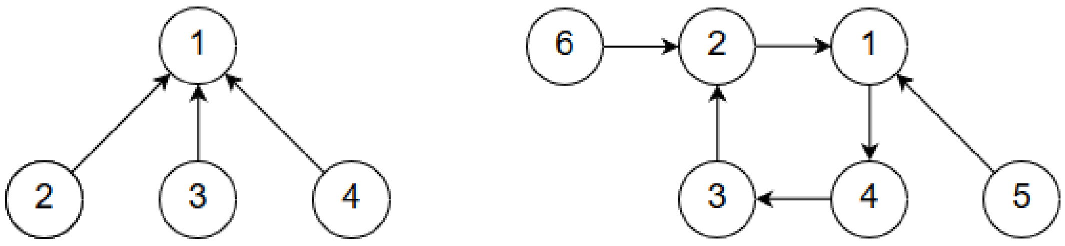


Figure 2.3: Examples of directed graphs of the Leader/Follower architecture (left) and the Cyclic architecture (right).

2.4.2. Linear-Quadratic Regulator

A Linear-Quadratic Regulator (LQR) is a control approach from the field of optimal control. This approach aims to minimize a quadratic cost function for a linear system as seen in Equations (2.18) and (2.19). The cost function is a weighted quadratic sum of the control error and the control effort. It is shown in Equation (2.20) [61, p.306].

$$J(\boldsymbol{u}) = \frac{1}{2}\boldsymbol{x}_{t_1}^T \boldsymbol{S}_{t_1} \boldsymbol{x}_{t_1} + \frac{1}{2}\int_{t_0}^{t_1} \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{u}^T \boldsymbol{R} \boldsymbol{u} \, \mathrm{d}t \tag{2.20}$$

Where J is the quadratic performance index, $\boldsymbol{x}$ and $\boldsymbol{u}$ are the state and input vectors, respectively, $\boldsymbol{S}, \boldsymbol{Q}$ and $\boldsymbol{R}$ are the weight matrices, and t_0 and t_1 are the start and end times, respectively.
The solution to minimizing the performance index J is given by the control input $\boldsymbol{u}$ shown in Equation (2.21) [61, p.309].

$$\boldsymbol{u} = -\boldsymbol{K}_{LQR}\boldsymbol{x} = -\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P}\boldsymbol{x} \tag{2.21}$$

Where $\boldsymbol{K}_{LQR}$ is the LQR gain and $\boldsymbol{P}$ is the solution to the Algebraic Ricatti equation (ARE) shown in Equation (2.22) [61, p.309].

$$-\dot{\boldsymbol{P}} = \boldsymbol{P}\boldsymbol{A} + \boldsymbol{A}^T\boldsymbol{P} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} + \boldsymbol{Q} \tag{2.22}$$

Since the system at hand is not continuous, the discrete LQR controller is used in the simulation. The discretized LQR problem is very similar to the continuous case. In the discrete case, the discrete performance J (given by Equation (2.23)) can be minimized by using the control input shown in Equation (2.24)[62, p.49]

$$J_k = \sum_{i=0}^{k} \boldsymbol{x}^T(i)\boldsymbol{Q}_k\boldsymbol{x}(i) + \boldsymbol{u}^T(i)\boldsymbol{R}_k\boldsymbol{u}(i) \tag{2.23}$$

$$\boldsymbol{u}(k) = -\left(\boldsymbol{B}^T\boldsymbol{S}\boldsymbol{B} + \boldsymbol{R}\right)^{-1}\boldsymbol{B}^T\boldsymbol{S}\boldsymbol{A}\,\boldsymbol{x}(k) \tag{2.24}$$

Where the k is the discrete time and as a subscript indicates the discrete versions of the parameters shown in Equation (2.20). The matrix $\boldsymbol{P}$ in that equation is the solution to the discrete algebraic Ricatti equation (DARE) shown in Equation (2.25)[63, p.69].

$$\boldsymbol{A}^T\boldsymbol{P}\boldsymbol{A} - \boldsymbol{P} - \boldsymbol{A}^T\boldsymbol{P}\boldsymbol{B}\left(\boldsymbol{B}^T\boldsymbol{P}\boldsymbol{B} + \boldsymbol{R}\right)^{-1}\boldsymbol{B}^T\boldsymbol{P}\boldsymbol{A}_k + \boldsymbol{Q} = \boldsymbol{0} \tag{2.25}$$

The steady state linear gain is then given by Equation (2.26) [63, p.69]

$$\boldsymbol{K}_{LQR} = \left(\boldsymbol{B}^T\boldsymbol{P}\boldsymbol{B} + \boldsymbol{R}\right)^{-1}\boldsymbol{B}^T\boldsymbol{P}\boldsymbol{A} \tag{2.26}$$

2.4.3. Virtual Center Formation Control

The virtual center method is a formation flying control architecture for virtual structure formations [2]. Instead of controlling the formation relative to a leader-spacecraft or a propagated orbit, the control error is calculated based on the estimation of the central location of the formation. For a given geometry, this center can be estimated by the relative positioning measurements taken on each satellite. This is graphically illustrated in Figure 2.4 [2].

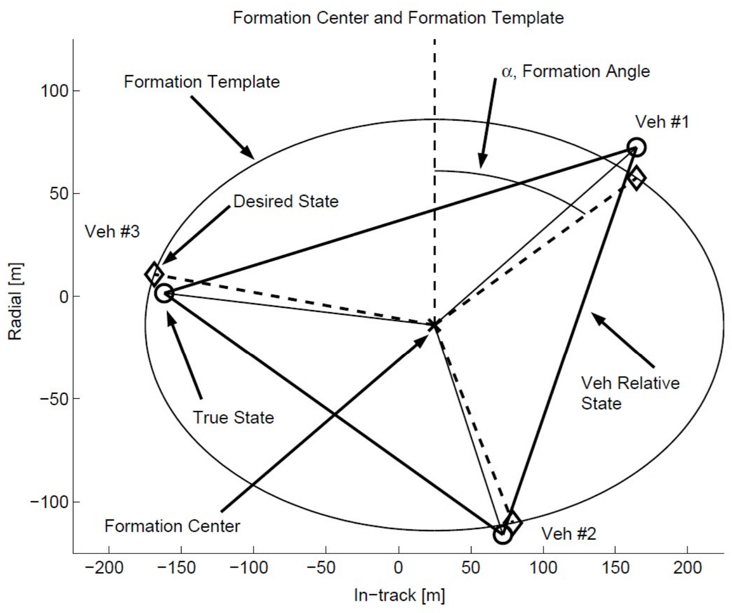


Figure 2.4: Illustration of the control error calculation in the Virtual Center approach [2].

For the octahedral case, this is especially simple as the center of the formation in nominal conditions is the mean value of the four edges of any vertex. Therefore, the mean of the relative distance measurement should give the current relation to the center position.

This method has an advantage over an approach that is based on propagating a reference point, or using a leader-follower type architecture where the position of one spacecraft in the formation is taken as the reference. This advantage is the inclusion of the actual mean motion of the spacecraft which includes model errors and disturbances, as this allows for correcting them without having to explicitly model the disturbances [2]. Another advantage specific to virtual-structure formation types is that they require the measurement of the relative positions in order to achieve their scientific objective. This means that these measurements can then also be used to control the formation, obviating the need for further measurements or communication.

Certain drawbacks exist as well. For example, this approach assumes that the center is stationary over the planned control action. However, this is not the case as each member of the formation acts via this control scheme and as such the relative measurements change.

2.5. Neural Network Basics

Artificial neural networks (ANN) are a powerful tool for generating models of complex systems. Inspired by the structure of biological nervous systems, these networks consist of nodes, called artificial neurons. These neurons have a certain activation value based on their connection to other neurons. In this section, the basics of ANNs are explained to the extent necessary to understand the thesis topic, starting with the motivation for and ideas behind neural networks Section 2.5.1. Then, the general structure of neural networks and two prominent adaptions is shown in Section 2.5.2. This is followed by the activations functions used in the network in Section 2.5.3. Finally, the back-propagation algorithm for finding the parameters of a neural network ("training" the network) is explained in Section 2.5.4.
A specific type of recurrent neural network called the "Long short-term memory" (LSTM) network is particularly useful in the application of time-series input data and is used for the fault detection and isolation of the thruster faults. As such, it is explained in more detail in Section 2.5.5.

2.5.1. Motivation behind Neural Networks

Neural network were first developed in the 1940's, in an effort to emulate the way biological neural pathways work [64, chp.2]. Neural networks are a general computational framework, capable of computing any continuous function [65, chp.4]. They contain many parameters that will be explained in Section 2.5.2, but these parameters fully define how the network computes an output for a give input.

Determining these parameters can be done through many procedures. Crucially however, they do not need to be picked manually. If a dataset exist that contains a relationship between a desired input and output, this dataset can be used to determine the sets of parameters that will produce a function for the desired relationship. For example, imagine a set of images and a set of labels for whether these images feature a bird. It would be very difficult to write an algorithm by hand that can detect whether or not a bird is visible in an image. However, such image recognition problems are easy to approach with neural networks.
Various algorithms exist that can determine the desired relationship, *any* relationship (as long as it can be represented as a continuous function), as long as a sufficiently useful dataset exists.

The process of determining the set of parameters that will yield the desired function is called "training" the network, and the network is said to "learn" to perform a task. However, in the end a neural network is simply a complex computational framework and the supposed "learning" process is very different from the way humans learn.

The property, that neural networks can approximate any function and that the necessary parameters can be picked by an algorithm, is a benefit and a drawback at the same time. It is advantageous, because the function to be approximated can be very complicated. They can be so complicated in fact, that no other approach for achieving the problem is known (like in the example of image recognition of birds).
Furthermore, the networks are computationally relatively simple, once they have been trained. This makes them attractive in situations where the computational resources are limited.
However, the drawback is the fact that the neural networks, once trained, are black-boxes. It is almost impossible to understand the interactions in such networks, and as such their behavior is difficult to fix, if the networks do not show sufficient performance.

Nonetheless, the achievements of neural networks in recent years have been impressive and their use should be explored further.

2.5.2. Network Elements & Structure

The most basic structure for a neural network is the so-called feed-forward structure. In this structure, the neurons are organized in layers with one sided connections which go from the input layer through the so called "hidden layers" to the output layer (see Figure 2.6) [30, 65]. Each neuron has a connection with every neuron in the past layer which influences its activation value, and as such these layers are also called "fully connected".
The elements making up an ANN and their functionality will be illustrated based on a very simple example with only a single neuron in each layer, shown in Figure 2.5.

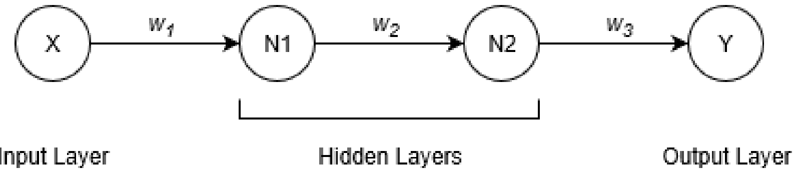


Figure 2.5: A simplified example structure of a neural network.

The input to the network (in this case a single real number) is multiplied by the weight w_1 and serves as the input to the first neuron N_1. This neuron has a certain *bias*, b_1, which indicates how easily it can be activated. The activation z of this neuron can then be found by computing the so called activation function with the input coming from the previous layer, shown in Equation (2.27).

$$a_1 = f(w_1 \cdot x + b_1) \tag{2.27}$$

Where a is the output of the neuron, f is the activation function, w is the connection weight, x is the value of the input layer and b is the neuron activation bias.
The activation of the first neuron N_1 is then passed through the next connection, gets multiplied by the weight w_2 and added to the bias, forming the input for the next neuron. This forms the basis for neural networks.
In more complex networks, the neurons receive weighted inputs from multiple neurons of the previous layer. In that case, the equation for the activation of each neuron can easily by summarized in a single vector equation by using a matrix of weights $\boldsymbol{W}$, a vector of neuron activations $\boldsymbol{a}$, and a vector of biases $\boldsymbol{b}$.

$$\boldsymbol{a}^l = f\left(\boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l\right) \tag{2.28}$$

In this equation, the function f is assumed to act on each element of the vector $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{a} + \boldsymbol{b}$ separately and the superscript l indicates the lth layer of neurons. Here, a word on notation is in order. The weight element w^l_{jk} is associated with the connection from node k in the $l-1$th layer to the node j in the lth layer.

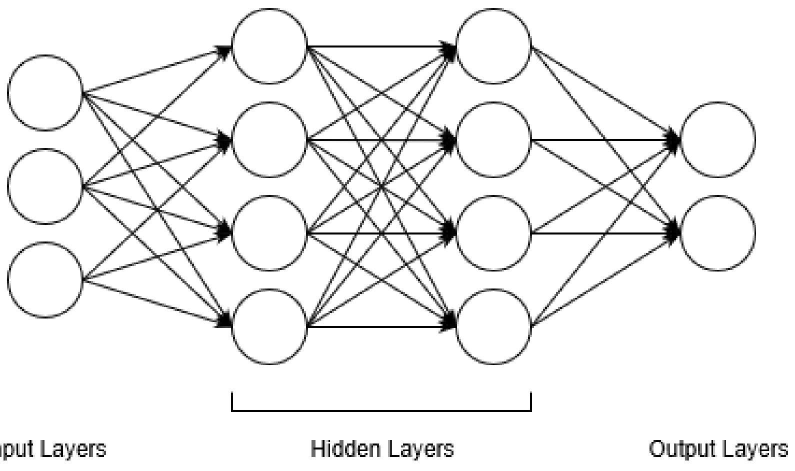
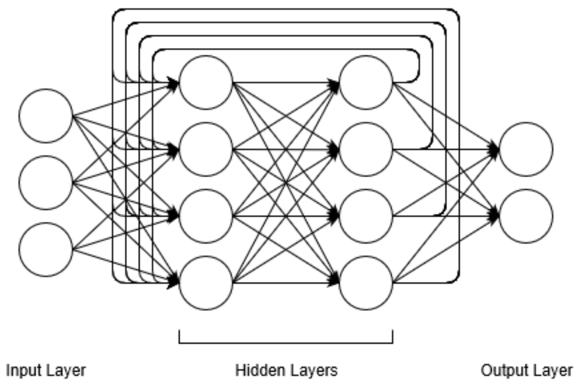


Figure 2.6: Exemplary feed-forward network structure with two hidden layers.

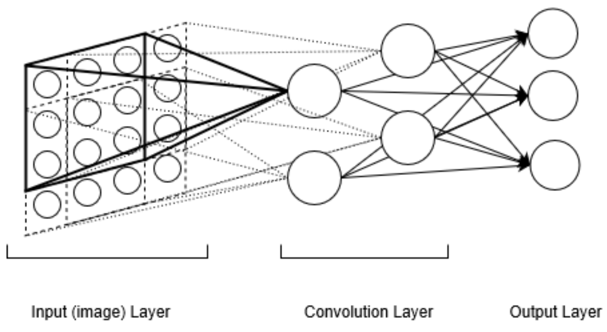
Table 2.2: Overview of the most common adaptions to the dense feed forward structure.

Network Type	Principal	Advantages
Fully Connected	Connect all neurons to every other neuron to create a dense structure	Most general structure suitable for many applications
Convolutional	Connect only some neurons to the next layer with equal weight to detect spatial patterns	Specifically suited for grid-organized data such as images
Recurrent	Feed the network state back with a time delay to create time dependencies	Very useful for networks with time series as input or output

Adaptions of this static feed-forward structure exist for various purposes, a few of which are presented in Table 2.2. The above described network structure is unchanging, which can guarantee stability under certain conditions, but also means that the system is completely independent of time. For the purpose of modeling dynamic systems, it can be useful to enable feedback in the network, i.e. letting states in later layers affect the previous ones, with a time delay, illustrated in Figure 2.7a.



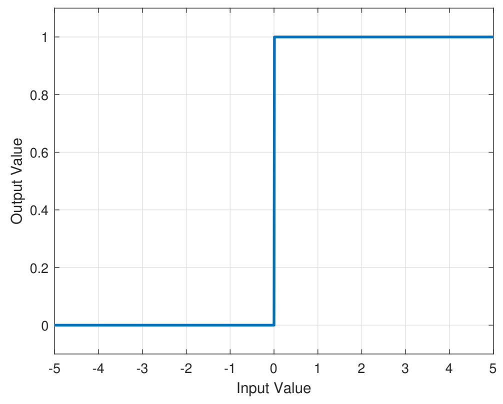
(a) Recurrent network example.



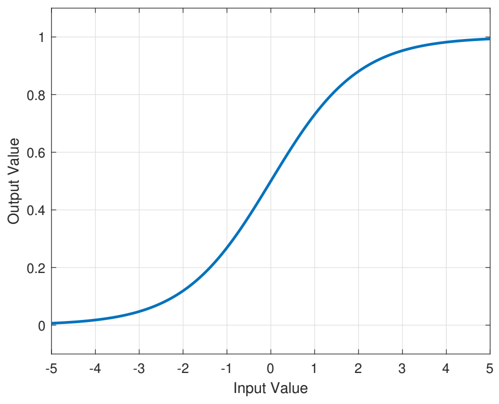
(b) Convolution network example.

Figure 2.7: Adaptations of the feed-forward structure.

Another variation which is often used in the analysis of images (e.g. for computer vision) is the convolutional neural network. In this approach, the input is a two-dimensional grid, which gets subdivided into smaller grids of a fixed size n x n, called the kernel size. Each of the sub-grids is then linked to a single output neuron in the next layer, with the only connections being from the sub-grids. Additionally, all of the connections to the next layer in that manner have the same weight, i.e. each kernel has a single weight rather than n x n ones. This obviously reduces the number of parameters that need to be determined and therefore speeds up training.
Another type of layer are the so called pooling layers, which are similar to convolutional layers in the sense that they are only dependent on a sub-grid of neurons from the previous layer and have no connections to neurons outside that sub-grid. However, their activation is not the result of an activation function like the sigmoid or step function, but one which "condenses" the information of the previous layer into fewer neurons. A simple example of this is an averaging pool in which the activation of a neuron in the pooling layer is the average activation of its kernel from the previous layer. This type is often combined with the convolutional layers [66, 67].
Of course, the designer is not limited to a single type of network. These kinds of layers can also be chained together, e.g. two convolutional layers followed by two regular feed-forward layers. An example of a network with one convolutional layer and one fully connected layer can be seen in Figure 2.7b. In this example. the input layer contains 4 x 4 neurons for illustration purposes, with each sub-grid of 3 x 3 being connected to a neuron in the next layer. In real images, the number of inputs are generally much larger while the the size of the sub-grid is comparatively smaller.
These kinds of structures can be motivated by the specific nature of the problem to be solved. For example in



(a) Step function.



(b) Sigmoid function.

Figure 2.8: Activation functions.

image processing it can be argued that the first convolutional layer detects small scale features, the next layer can determine larger scale features and the remaining layers determine the desired result from the detected features. However, whether a network actually performs in the manner that is envisioned before the training process is usually not discernible later on, due to the very large number of connections (which scale with $m \cdot n$, where m and n are the number of neurons in the connected layers).
These meta-parameters of the network, (number of layers, number of neurons per layer, layer structure, type of activation function) depend on heuristics related to the problem at hand as well as practical factors which can influence the time it takes to train the network.

2.5.3. Activation Functions

There are many available options for the activation function f An intuitive activation function is the step function which outputs either 0 if the input is negative or 1 if the input is positive (seen in Equation (2.29)).

$$\text{step}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \tag{2.29}$$

This is very similar to the way actual neurons behave in the brain, which are either resting or firing with no in-between states.
However, this activation function is problematic. Ideally, small changes in the input should correspond to small changes in the output (i.e. the output is continuous over the input), as this enables differentiating the network output with respect to the input (which will become important in training). However this is not the case with the step function. It can easily be seen that causing a tiny change to the input in a more complex network could cause a series of neurons to flip in difficult to predict ways which can completely alter the outcome. This can be remedied by choosing a smoothed out version of the step function, called the sigmoid function, given in Equation (2.30).

$$sig(z) = \frac{1}{1+e^{-z}} \tag{2.30}$$

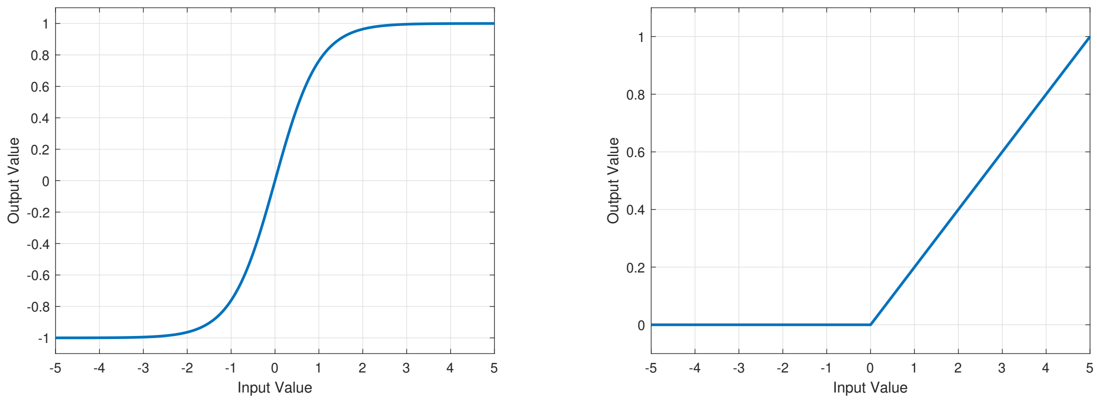
This function is close to 0 for very large negative values of the input z and close to 1 for very large positive input values. Both the step function and the sigmoid function can be seen in Figure 2.8. The sigmoid function solves the problem of a discontinuous output, but more practical activation function exist and will be mentioned later on.

In addition to the activation functions mentioned above, two other functions are commonly used. The hyperbolic tangent function and the rectifier function, shown in Figure 2.9 and Equations (2.31) and (2.32).
The hyperbolic tangent is very similar to the sigmoid when comparing their shape and algebraic form (in fact $(1+\tanh(x))/2 = sig(2x)$), so it might not be obvious why one would be more useful then another. However, the key difference is the fact that the sigmoid function is strictly positive. No input to the function can cause

the output to be 0. But in a complex network there might be many neurons which should not activate, depending on the given input. This leads to very large valued weights in the training of the network and can make the network unstable. Choosing the *tanh* function is hence preferred for practical reasons [68].

$$\mathrm{rect}(z) = \begin{cases} 0 & \text{if } z < 0 \\ \alpha \cdot z & \text{if } z > 0 \end{cases} \tag{2.31}$$

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.32}$$



(a) Hyperbolic tangent function. (b) Rectifier function (with $\alpha = 1/5$).

Figure 2.9: Activation functions.

Furthermore, there is another activation function in the case of classification problems with more than two categories. In those cases, it might be desirable to express the output as a probability distribution. This gives a direct interpretation to the network output as a certain "confidence" in its result.
One commonly used function for this purpose is the softmax function, shown in Equation (2.33) [65, p.70].

$$\mathrm{rect}(\boldsymbol{z}_i = \frac{e^{z_i}}{\sum_j^n e^{z_j}} \tag{2.33}$$

Where $\boldsymbol{z}$ is now a vector of inputs, i and j are indices ranging from 1 to n, where n is the amount of elements in the input vector.
As can be easily checked, the sum of the components of this output function is 1.

2.5.4. Backpropagation Training Algorithm

Determining the network parameters, i.e. the weights and biases of the network, is in essence an optimization problem with many free variables: Given the training data, find the sets of weights and biases that minimize a certain cost function expressing the quality of the network output.

$$\min_{\boldsymbol{W},\boldsymbol{b}} C(\boldsymbol{W}, \boldsymbol{b})$$

Where C is the chosen cost function, $\boldsymbol{W}$ is the set of weights and $\boldsymbol{b}$ is the set of biases. The choice of cost function is another parameter that needs to be determined based on the problem at hand. Examples of cost functions are the quadratic cost shown in Equation (2.34) or the cross-entropy cost function shown in Equation (2.35). Depending on the problem at hand, other cost functions may be chosen.

$$C_{\text{quad}}(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{n} \sum_n \left(\boldsymbol{y} - \boldsymbol{a}\right)^2 \tag{2.34}$$

$$C_{\text{ce}}(\boldsymbol{W}, \boldsymbol{b}) = -\frac{1}{n} \sum_n y \ln a + \left(1 - y\right) \ln(1 - a) \tag{2.35}$$

Where $\boldsymbol{a}$ is the vector of outputs, n is the number of training samples used to determine the cost and $\boldsymbol{y}$ is the desired outcome.
A very common approach which lends itself to neural networks specifically are gradient-based methods to find the minimum, e.g. gradient descent or conjugate gradient methods. In these approaches, the gradient of the cost function is determined w.r.t the weights and biases and the parameters are adjusted in the direction of that gradient.
Due to the nature of the neural network, a simple way to determine the gradients of the weights and biases is through a technique called "backpropagation". The partial derivatives making up the gradient (sometimes also called *sensitivities*) are first calculated for the last layer. Following this, a recursive relationship is established which allows for calculating the partial derivatives going back one layer at a time.

This is summarized in Equations (2.36) to (2.40), which will not be derived here, but can be easily determined through the application of the chain-rule [65, chp. 2].

$$\delta_j^l := \frac{\partial C}{\partial z_j^l} \tag{2.36}$$

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{2.37}$$

$$\delta_j^l = \sum_k w_{kj}^{l+1}\delta_j^{l+1}\sigma'(z_j^L) \tag{2.38}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.39}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{2.40}$$

$$\tag{2.41}$$

Equation (2.36) defines the sensitivity δ_j^l as the partial derivative of the cost function with respect to the input z of the jth node in the lth layer. Equation (2.37) gives the sensitivities of the last layer L, as the partial derivative of the cost function with respect to the output of the neuron j mutipied with the derivative of the activation function. Equation (2.38) then gives the relationship to determine the sensitivity of jth node in the lth layer as a function of the sensitivities of all nodes in the layer $l+1$.

Finally, Equations (2.39) and (2.40) relate the sensitivity to the desired quantities: These are the partial derivatives of the cost function with respect to the bias and weights. This back-propagation algorithm can be summarized in the following steps.

(0. Initialize the network parameters $\boldsymbol{W}, \boldsymbol{b}$)

1. Select a training sample $\boldsymbol{x}$
2. Go forward through the network by calculating the activations a_j^l of all layers up to the last using Equation (2.28)
3. Calculate the sensitivities of the final layer using Equation (2.37)
4. Move backwards through the network layer by layer by calculating the sensitivities using Equation (2.38)
5. Determine the elements of the gradient vector with Equations (2.39) and (2.40)

These steps give the gradient for a single training example. In practice, this process is performed for a batch of samples (batch-learning) and the resulting gradients are averaged for a step of the optimization algorithm, which adjusts the network parameters.
This continues until all batches in the dataset have been used to compute a step in the optimization algorithm. Going over the dataset in its entirety is called an *epoch*. In most cases, a single epoch is not enough

to have reached a minimum in the cost function, so the process starts over in the next epoch. The decision to stop the training is usually not evaluated based on the value of the cost function over the training set, as the networks can be prone to over-fitting. Instead, a separate validation dataset is used. This dataset is only used to evaluate the performance of the network at a certain point during the optimization process, usually once per epoch. As the training algorithm is not using the validation to compute the gradient, it can be used to judge whether the network is beginning to specialize on the training data. When the performance on the training set continues to improve, but the performance on the validation set stagnates or decreases, the training is stopped. This avoids over-fitting the network parameters to the training dataset,

Various adaptions of this simple training method exist. For example, for recurrent neural networks which feature a dependence on time, the computational graph is "unrolled" up to a certain point in time. From there, the computation of the gradient can commence. This adapted approach is called backpropagation through time (BPTT).

2.5.5. Long Short-Term Memory Networks

A specific type of recurrent neural network called the "Long short-term memory" (LSTM) network is particularly useful in the application of time-series input data and is used in the development of the neural network model in this thesis[69].
It differs from the usual network structure by its basic unit not being composed of a neuron with a single activation value and an output. Instead, the basic LSTM unit is made up of a central neuron which holds the value of the cell, an 'input' gate which regulates whether the value in the central neuron can be altered by input, an 'output' gate which in turn regulates to what extent the value in the central neuron gets passed on to the next layers and a 'forget' gate which regulates how quickly the value in the central neuron decays. This is illustrated in Figure 2.10[3].

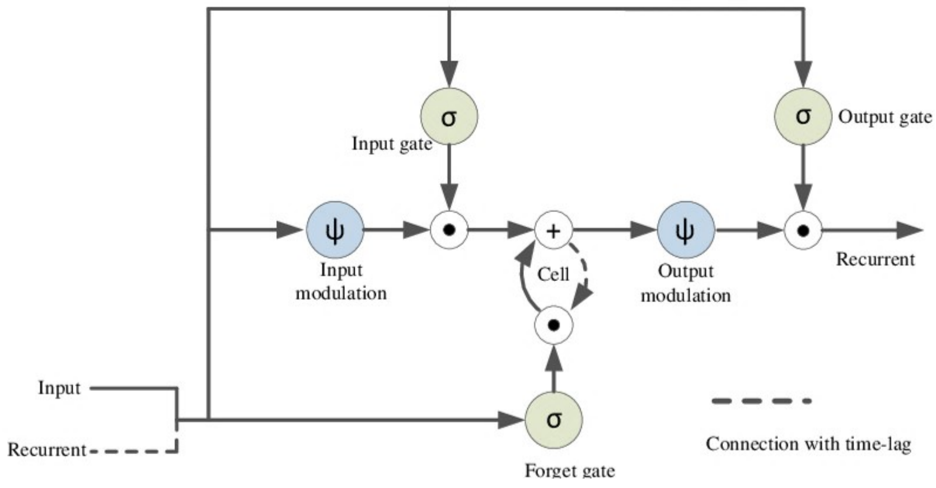


Figure 2.10: Illustration of the basic Long Short-Term Memory (LSTM) network unit [3].

The figure shows that a single LSTM cell has four locations where the collected input either from previous layers or recurrent layers is used. The first is the input modulation, which is similar to a regular neural network cell. This is followed by a multiplication with the value of the input gate. The resulting values is *added* to the cell value. This is a crucial difference to other recurrent networks as it makes the network resistant to the vanishing gradient problem which occurs in other recurrent neural networks [70]. The current value of the cell is then multiplied with the output of the forget gate. This combined cell value is then passed through an activation function, most commonly the hyperbolic tangent function. Finally, the cell output is multiplied with the value of the output gate.
As a network with LSTM units effectively has four cells which require a weighted sum of the input and each have their own biases, the total amount of parameters is four times greater than for a regular dense network with the same amount of units.

2.6. Kalman Filter

The Kalman filter is an observer for a stochastic system that is based on the system model given by Equations (2.18) and (2.19). The filter aims to give an estimate of the system state with minimal error covariance in the least-squares sense. This is achieved by combining the model-prediction with the state measurement in an optimal way.
The discrete Kalman filter works in a predictor-corrector fashion. That means, the next state is estimated based on the current state using the system model, resulting in the so called *a priori* estimate. Using the next measurement the *a priori* estimate $\hat{\mathbf{x}}^-(k+1)$ is improved upon, resulting in the *a posteriori* estimate $\hat{\mathbf{x}}^+(k+1)$. This is done by combining the *a priori* estimate with a weighted combination of the measurement and measurement estimate, called the innovation, in such a way as to minimize the variance P_k of the error in the new estimate in the least squares sense [71, p.22]. The algorithm for the discrete Kalman filter works according to the following steps [72]:

1. Given the previous estimate for the mean and the error covariance, compute the *a priori* estimate for the next state
$$\hat{\boldsymbol{x}}^-(k+1) = \boldsymbol{A}\hat{\boldsymbol{x}}^+(k) + \boldsymbol{B}\boldsymbol{u}(k) \tag{2.42}$$

2. Compute the *a priori* estimate for the next error covariance by Equation (2.43)
$$\boldsymbol{P}^-(k+1) = \boldsymbol{A}\boldsymbol{P}^+(k)\boldsymbol{A}^T + \boldsymbol{Q}_x \tag{2.43}$$

3. Compute the Kalman filter gain by Equation (2.44)
$$\boldsymbol{K}(k+1) = \boldsymbol{P}^-(k+1)\boldsymbol{C}^T(\boldsymbol{C}\boldsymbol{P}^-(k+1)\boldsymbol{C}^T + \boldsymbol{Q}_y)^{-1} \tag{2.44}$$

4. Compute the *a posteriori* state estimate
$$\hat{\boldsymbol{x}}^+(k+1) = \hat{\boldsymbol{x}}^-(k+1) + \boldsymbol{K}_k(k+1)\left(\boldsymbol{y}(k+1) + \boldsymbol{C}\hat{\boldsymbol{x}}^-(k+1)\right) \tag{2.45}$$

5. Update the error covariance matrix by Equation (2.46).
$$\boldsymbol{P}^+(k+1) = (\boldsymbol{I} - \boldsymbol{K}(k+1)\boldsymbol{C})\boldsymbol{P}^-(k+1)(\boldsymbol{I} - \boldsymbol{K}(k+1)\boldsymbol{C})^T + \boldsymbol{K}(k+1)\boldsymbol{Q}_y\boldsymbol{K}(k+1)^T \tag{2.46}$$

The blending of new and old information is called the *innovation* and is given again in Equation (2.47).

$$\boldsymbol{r}(k+1) = \boldsymbol{y}(k+1) - \hat{\boldsymbol{y}}(k+1) = \boldsymbol{y}(k+1) + \boldsymbol{C}\hat{\boldsymbol{x}}^-(k+1) \tag{2.47}$$

A sub-optimal version of the filter which does not require recomputing the gain can be used, since the recursive update equation usually converges to a constant. The steady state Kalman filter gain is shown in Equation (2.48) where the constant error covariance matrix is given by the solution of the discrete algebraic Ricatti equation shown in Equation (2.49).

$$\boldsymbol{K} = \left(\boldsymbol{A}\boldsymbol{P}\boldsymbol{C}^T\right)\left(\boldsymbol{C}\boldsymbol{P}\boldsymbol{C}^T + \boldsymbol{R}\right)^{-1} \tag{2.48}$$

$$\boldsymbol{P} = \boldsymbol{A}\boldsymbol{P}\boldsymbol{A}^T + \boldsymbol{Q} - \left(\boldsymbol{A}\boldsymbol{P}\boldsymbol{C}^T\right)\left(\boldsymbol{C}\boldsymbol{P}\boldsymbol{C}^T + \boldsymbol{R}\right)^{-1}\left(\boldsymbol{C}\boldsymbol{P}\boldsymbol{A}^T\right) \tag{2.49}$$

3
Simulation & Data Generation

Because there is little real data from formation flying missions which fit the parameters of the thesis, a representative simulation of the satellite formation and the relative dynamics was created. The most important sections of the code can be found in Appendix B.2, but in this chapter, the simulation is explained. First, the mission setup is detailed in Section 3.1, followed by an overview of the separate parts of the simulation in Section 3.2. Then, the implemented forces are explained in Section 3.3 and the Guidance, Navigation and Control (GNC) system of the formation is shown in Section 3.4. Finally, the verification of the simulation is shown in Section 3.5.

3.1. Simulated Mission Scenario

In this section the mission scenario that is used for the analysis is detailed. Section 3.1.1 gives information on the type of the formation, followed by a description of the process used to determine some of the mission parameters in Section 3.1.2.

3.1.1. Mission Overview

In order to fulfill the aims of the thesis, a close formation is necessary. In addition, a formation with active use of thrusters is preferable as thruster faults are to be investigated. Therefore, the chosen formation type is a virtual structure (i.e. a formation in which an exact shape is maintained by the spacecraft), specifically that of a regular polyhedron with satellites at the vertices. The number of satellites was chosen to be 6, as the resulting shape (octahedron, shown in Figure 3.1) is the first regular polyhedron (based on amount of vertices) where not every vertex has a connection to every other vertex, thus allowing for investigating the case of missing connections and their effect on the FDI performance.

Of course not only the formation type has to be decided but also the satellites making up that formation have to be characterized. Of particular importance are the properties pertaining to the behavior in the orbit. The spacecraft parameters can be seen in Table 3.1. The spacecraft is assumed to be a micro-satellite with a wet mass of 100 kg and is modelled as a completely rigid, solid cube with a side length of 1 meter. The thruster are assumed to be similar to the orbital control thrusters of the MMS mission, which were mono-propellant thrusters capable of a 1 lbf thrust, approximately 4.5 N [73]. While additional data beyond the thrust could not be found for this particular thruster, a brochure by arianegroup lists a similar 5N monopropellant thruster with a specific impulse range of 206 to 226 s and a minimum impulse bit from 0.03 Ns to 0.1 Ns [74]. For this simulation, the numbers were rounded to 4N for the thruster, 0.1 Ns for the minimum impulse bit and 200 s for the specific impulse.

In addition to the parameters mentioned above, the configuration of the control thrusters needs to be decided. In order to keep the thrust allocation problem simple, a configuration of 6 thruster per satellite was chosen, with a pair of thruster per principal axis of the satellite. This means a thruster pointing away from any face of the cube-like satellite. It is assumed that the thrust vector points through the center of mass of the satellite. The chosen configuration is illustrated in Figure 3.2.

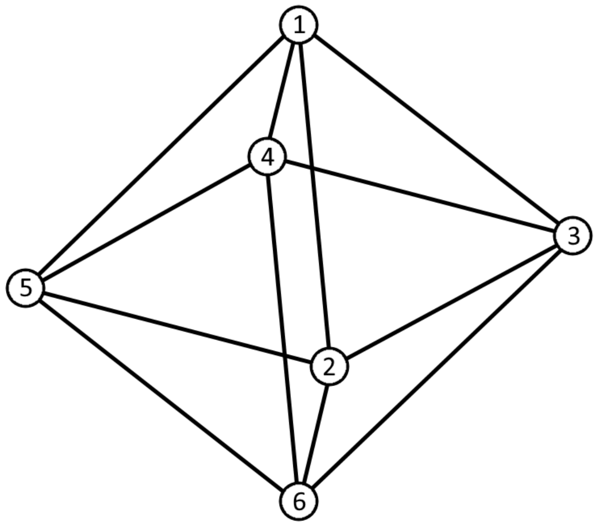


Figure 3.1: Illustration of the satellites' numbering and relative positioning.

Table 3.1: Spacecraft parameters for MATLAB simulation.

Spacecraft Parameter	Value	Unit
Mass	100	[kg]
Dimensions	1 x 1 x 1	[m]
Moment of Inertia	16.6	[kgm^2]
Thrust	4	[N]
Minimum Impulse Bit	0.1	[Ns]
Specific Impulse	200	[s]

3.1.2. Mission Parameters

In addition to the type of formation, a particular orbit has to be selected. There have not been many uses of the virtual structure formation type in flown missions, but the MMS mission used a highly eccentric orbit in order to study the reconnection of the magnetic field [75]. Such an orbit would complicate the guidance and control of the implemented mission. As this is not the topic of the thesis, a simplified approach is taken. A circular, low-Earth orbit was chosen.

The exact mission parameters such as orbital height, the orbital inclination, the size of the formation (i.e. the distance between satellites) and the orientation of the virtual octahedron were determined through a simple optimization routine in order to find a sensible mission parameters.

The cost function in this routine is based on the required ΔV that needs to be expended in order to maintain the formation over one orbital period. Specifically, the cost function is a weighted sum of the average ΔV over all satellites in the formation and the standard deviation of the ΔV. This can be seen in Equation (3.1).

$$C(\boldsymbol{\Delta V}) = \alpha \cdot \frac{\bar{\boldsymbol{\Delta V}}}{n} + (1-\alpha) \cdot \frac{std(\boldsymbol{\Delta V})}{\bar{\boldsymbol{\Delta V_i}}} \tag{3.1}$$

Where α is the weighting factor, $\bar{\boldsymbol{\Delta V}}$ is the average velocity increment for the formation, $std(\boldsymbol{\Delta V})$ is the standard deviation of the velocity increments of the formation, and n is a normalization factor. In this case n was chosen to be 0.9 after some trial and error, as it yielded good results.

This optimization was done in order to both reduce the overall effort necessary to maintain the formation as well as keep the effort balanced across the formation. This is important for the lifetime of the mission as the spacecraft are assumed to be identical. As such the formation will not be able to be maintained, as soon as the first member runs out of propellant.

Furthermore, a balanced ΔV across the formation corresponds to an equal amount of thrust expended over one orbit, which has implications for the fault detectability and potential biases for the training data.

The optimization considered 7 variables comprised of the four components of the orientation quaternion, the orbital altitude, the inclination and the formation size. As the quaternion component are naturally restricted

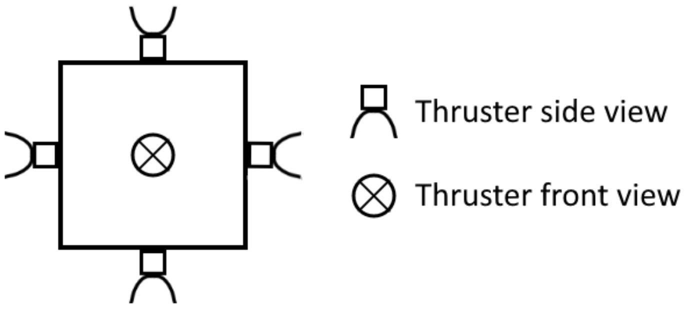


Figure 3.2: Illustration of the chosen thruster configuration.

to range of [-1,1], the other components need to be normalized to a similar range for proper performance of the optimization algorithm. The variables are constrained within these bounds as to not get results above LEO or formations whose satellites are too far apart. There is another constraint as the quaternion is restricted to a 4d unit-vector, if it is meant to represent an orientation. The `fmincon()` function from the MATLAB Optimization Toolbox was used for the optimization process. The bounds and the resulting parameters can be seen in Table 3.2.

Table 3.2: Mission parameters determined by `fmincon()` optimization script.

Mission Parameter	Bounds	Final Value	Unit
Inter-satellite distance	[100,1000]	142.39	[m]
Inclination	[0,90]	62.811	[°]
Altitude	[350,1000]	687.48	[km]
Formation orientation (Quaternion)	[-1,1]	[0.9328,-0.0854,0.0314,0.3488]	[-]

3.2. Simulation Overview

The satellite formation is implemented in MATLAB, making use of some of its object-oriented programming features. The simulation is started in a main file, creating objects of two custom classes, formation and spacecraft. The methods of the classes as well as separate functions are used to propagate the formation through time. A graphical overview of the connections between various parts of the simulation can be found in Figure 3.3. The classes involved in the simulation are elaborated on in Section 3.2.1, while the functions are detailed in Section 3.2.2. An overview of the dependencies of the simulation can be seen in Figure 3.3.

3.2.1. Classes

Two classes are used in the simulation of the formation. The first is the 'spacecraft' class with relevant parameters such as its position, velocity, thruster configuration, etc. (detailed below) and the second is the 'formation' class, which contains multiple objects of the spacecraft class and information such as the orbital parameters of the formation reference orbit.

Spacecraft The spacecraft class contains the following methods

- **Constructor** `spacecraft()`
  The constructor of the spacecraft class initializes the object with all relevant parameters. These include the starting position and velocity, its attitude and spin, an array of additional parameters as well as an array of parameters characterizing the formation.
  The spacecraft parameters given in the array `spacecraftParameters` consists of 17 values, including the mass, dimensions as well as thruster parameters. The `formationParameters` array contains the size of the formation (i.e. the distance between the satellites), the orientation of the formation structure with respect to the ECI frame, and the orbital parameters of the reference orbit the formation is

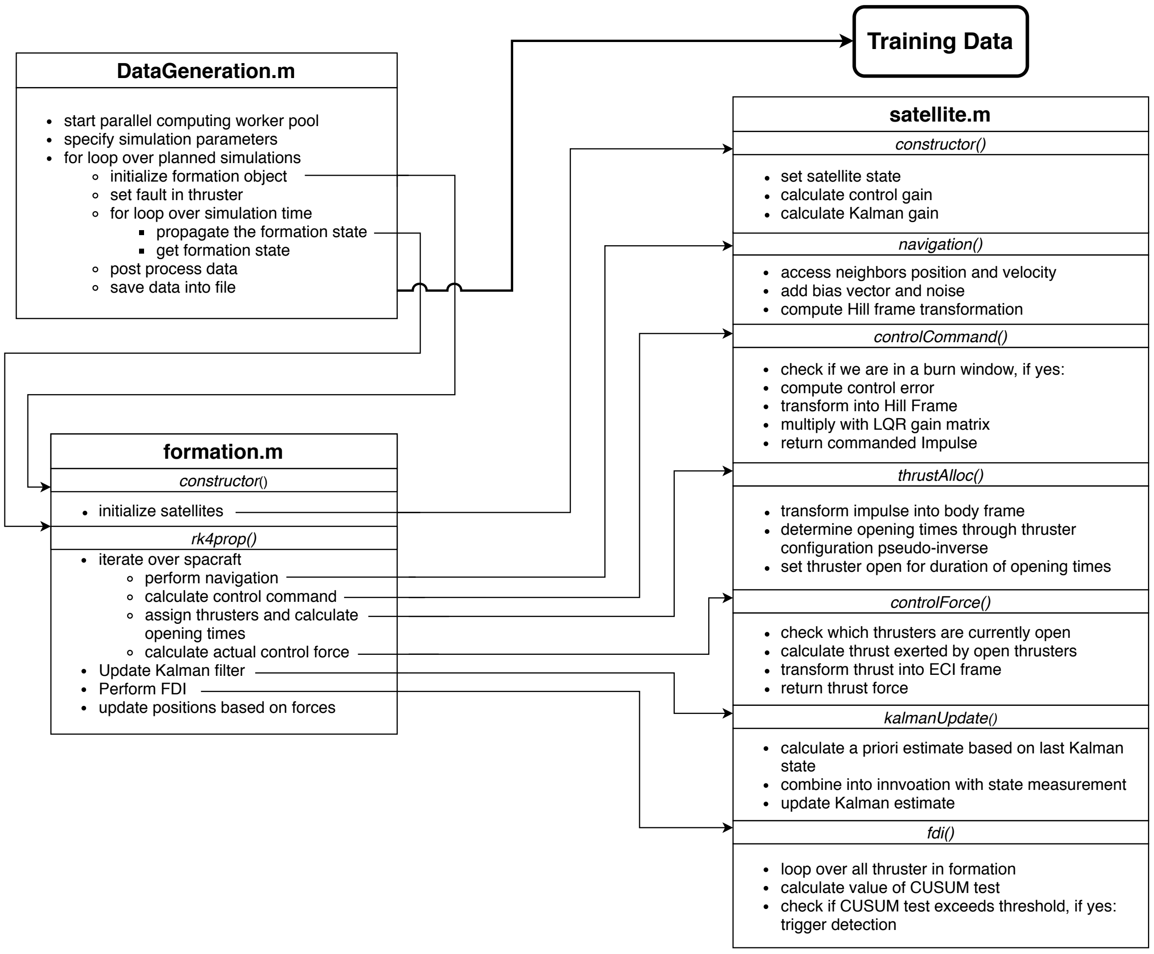

Training Data
DataGeneration.m
start parallel computing worker pool
specify simulation parameters
for loop over planned simulations
initialize formation object
set fault in thruster
for loop over simulation time
propagate the formation state
get formation state
post process data
save data into file
formation.m
constructor()
initialize satellites
rk4prop()
iterate over spacraft
perform navigation
calculate control command
assign thrusters and calculate opening times
calculate actual control force
Update Kalman filter
Perform FDI
update positions based on forces
satellite.m
constructor()
set satellite state
calculate control gain
calculate Kalman gain
navigation()
access neighbors position and velocity
add bias vector and noise
compute Hill frame transformation
controlCommand()
check if we are in a burn window, if yes:
compute control error
transform into Hill Frame
multiply with LQR gain matrix
return commanded Impulse
thrustAlloc()
transform impulse into body frame
determine opening times through thruster configuration pseudo-inverse
set thruster open for duration of opening times
controlForce()
check which thrusters are currently open
calculate thrust exerted by open thrusters
transform thrust into ECI frame
return thrust force
kalmanUpdate()
calculate a priori estimate based on last Kalman state
combine into innvoation with state measurement
update Kalman estimate
fdi()
loop over all thruster in formation
calculate value of CUSUM test
check if CUSUM test exceeds threshold, if yes: trigger detection

Figure 3.3: Overview of the numerical simulation.

intended to follow.
The constructor uses this data to assign it to the corresponding object parameters. In addition to this simple assignment, the constructor also computes various matrices relating to the LQR controller and the Kalman filter FDI method.

- `updatePos()` and `updateAtt()`
  The update functions take a deviation in either the position and velocity (`updatePos()`) or the attitude as a quaternion and rotation vector (`updateAtt()`) and apply the given change to the satellite. This function is used during the Runge-Kutta 4 propagation function of the satellite formation.

- `getState()` and `setState()`
  `getState()` returns the satellite position and velocity vector. while `setState()` takes a position-velocity pair as input and sets them as the satellite's position and velocity.

- `navigation()`, `selfNav()`,`relNav()` and `relNavDistAngle()`
  The navigation functions simulate the measurement procedure that would take place in a real satellite, in a very simplified manner. The functions `selfNav()` and `relNav()` give the absolute position and velocity of the satellite itself, and the relative position and velocity of its four neigbors in the ECI frame, respectively. A random-direction bias as well as Gaussian noise in every component is added to both the absolute and relative navigation, depending on the parameters set in the object.
  The `relNavDistAngle()` function works similarly to `relNav()`, however implementing a different navigation approach. Instead of the noise being added to the component of the relative position, a noise component is added to the range and the relative angles instead, emulating a range-based navigation method.
  The `navigation()` function calls the aforementioned functions and sets the current estimate of the relevant satellite parameters to the returned values. Furthermore, `navigation()` also calculates the transformation matrix to the current Hill frame and its derivative, as they are used in various other methods.

- `kalmanUpdate()`
  The Kalman filters functions implements the regular Kalman Filter discussed in Section 2.6.

- `fdi()` The fault detection function uses the innovation of the Kalman filter for fault detection purposes. The exact algorithm is described in Section 4.2.

- `guidance()`
  The guidance function propagates the reference orbit using Keplerian Dynamics and uses the solution of Lambert's Problem [] to determine the necessary instantaneous ΔV commands to reach the next position in the correct time. This function is not used in the propagation itself, but was used during the determination of the mission parameters, discussed in Section 3.1.2.

- `errVirtualCenter()` and `errTrackedCenter()`
  The `err[...]()` functions calculate the control error based on two separate approaches, used for testing. `errVirtualCenter()` uses a control approach based on [2], while `errTrackedCenter()` uses an approach where the center of the formation is propagated on-board of each satellite.

- `controlCommand()`,`thrustAlloc()` and `controlForce()`
  The three control functions `controlCommand()`, `thrustAlloc()` and `controlForce()` each perform a separate step of the control system. The function `controlCommand()` takes the current relative position to the formation center calculated from `errVirtualCenter()` to determine the current control acceleration. Then, `thrustAlloc()` determines based on the current control acceleration which thrusters need to open and the corresponding opening times. Finally, `controlForce()` determines which thrusters are currently open and calculates the force that is acting on the satellite. This force is fed back into the propagation of the state.

Formation The formation class contains a number of satellite objects making up that formation, as well as general information such as information about the reference orbit. The following functions are used in the simulation

- **Constructor** `formation()`
  The constructor of the `formation` class takes the following input: the number of spacecraft, the type of the formation, the position and velocity of the virtual center (which define the reference orbit), the formation size and orientation, the array of spacecraft parameters that are passed on and information on the navigation method. In addition, it also takes the mean of the residual vector for the Kalman filter based FDI method.
- `rk4Prop()`
  This is the main function of the formation class and is used to propagate the state of the entire formation by one time step Δt using the Runge-Kutta 4 Integration scheme with the `dynamics()` function.
- `getStates()`,`setStates()` and `getRelStates()`
  The functions `getStates()` and `getRelStates()` return the true position and velocity in the ECI frame, and the measured relative position and velocity from each satellite, respectively. The `setStates()` on the other hand takes a set of positions and velocities and assigns them to the satellites in the formation.
- `getAbsoluteMeasurement()`
  This function returns the measured positions and velocities of the satellites in the formation, in the ECI frame.
- `getControlCommands()`
  This function returns the current commanded control acceleration in the Hill frame of each spacecraft in the formation.
- `setFault()`
  This function sets a particular fault (open or closed) in the given satellite and thruster.

3.2.2. Functions
Multiple stand-alone functions have been used, which do not require direct access to object properties or are simple functions used for improved code-readability. The following list explains the important stand-alone functions used in the simulation.

- `dynamics()`
  The `dynamics()` function calculates the derivative of the position and velocity. It in turn calls the various functions which calculate the forces acting on the body in order to determine the acceleration.
- `rotX()`,`rotY()`,`rotZ()`,`rotZdot()`
  These functions are used to determine the rotation matrices around the X,Y and Z axis, as well as the time derivative of the Z axis, as it is used in the determination of the Hill Frame.

3.3. Forces Model
In this section the models for the various forces involved in the simulation are detailed. First, the gravity model is shown in Section 3.3.1, followed by the disturbance forces in Section 3.3.2.

3.3.1. Gravity
The EGM 2008 Model is used in the calculation of the gravitational acceleration. An appropriate function implementing this model exists in MATLAB already and is called `gravitysphericalharmonic()`. It takes the position of an object in the ECEF frame and returns the acceleration in the ECEF frame based on the EGM 2008 model. This function was slightly modified to be able to change the degree of the model by a value in the `Constants` file.

3.3.2. Disturbances
Four disturbances are simulated [76, chp. 20]:

1. Aerodynamic drag in the upper atmosphere
2. Gravitational influence of the Sun and the Moon
3. Lorentz force due to the movement through Earth's magnetic field

4. Radiation pressure from Sun- and Earth-emmitted radiation

Aerodynamic Drag The remaining atmosphere at the orbital altitude will produce some drag from the collisions with air molecules. It is modelled similarly to the aerodynamic force in Aeronautics, shown in Equation (3.2) [76, p.534].

$$\mathbf{F}_{aero} = -C_D\frac{1}{2}\rho A|\mathbf{v}_{rel}|\mathbf{v}_{rel} \tag{3.2}$$

Where F_{aero} is the aerodynamics force, C_D is the drag coefficient, ρ is the atmospheric density, A is the cross-sectional area and $\mathbf{v}_{rel}$ is the velocity vector relative to the air. In this case it is assumed that the air rotates with the Earth. Furthermore it is assumed that the exposed area is constant, regardless of the orientation. A worst case estimate is taken in order not to underestimate the effect.
As such, a drag coefficient of 3 is assumed and the atmospheric density is assumed to be constant and equal to 1e−12 kg m^{-3} [76, p.538].

Gravitational Influence of Sun and Moon The most influential gravitational bodies for an orbit around the Earth are the Sun and Moon. Their disturbance acceleration is modelled by Equation (3.3) [76, p.540].

$$\mathbf{F}_{grav} = m\cdot\mu_g\left(\frac{\mathbf{r}_j - \mathbf{r_i}}{|\mathbf{r}_{ij}|^3} - \frac{\mathbf{r_j}}{|\mathbf{r}_j|^3}\right) \tag{3.3}$$

Where $\mathbf{F}_{grav}$ is the perturbing acceleration of a third body, μ_g is the standard gravitational parameter of said body, $\mathbf{r}_i$ $\mathbf{r}_j$, and $\mathbf{r}_{ij}$ are the position vector of the orbiting object, the perturbing body and the distance between the orbiting body and the perturbing body, respectively.
In order to know the gravitational influence of the Sun and the Moon, their positions have to be known for a given time. The scripts `createMoonPositions()` and `createSunPositions` create arrays of these positions using the `planetEphemeris()` function (a part of the Aerospace Toolbox for MATLAB), which are then linearly interpolated during the simulation in order to find the approximate location and determine the influence according to Equation (3.3).

Lorentz Force Due to the presence of ions in the atmosphere, a spacecraft can pick up an electric charge. This in combination with the movement through Earth's magnetic field can create a force on the satellite, shown in Equation (3.4)[76, p.534].

$$\mathbf{F}_{mag} = q_{sat}\mathbf{v}\times\mathbf{B}_{mag} \tag{3.4}$$

Where $\mathbf{F}_{mag}$ is the force due to the magnetic field, q_{sat} is the charge of the spacecraft, $\mathbf{v}$ is the velocity of the charged particles (the spacecraft in this case) and $\mathbf{B}_{mag}$ is the magnetic field vector.
The charge accumulated by a sphere is given by Equation (3.5)[76, pp.548-549].

$$q = U\cdot 4\pi\varepsilon_0 R_s \tag{3.5}$$

Where U is the electric potential difference between the satellite and the surrounding plasma, ε_0 is the permittivity of free space with a value of 8.854e−12 F m^{-1}, and R_s is the radius of the sphere. In estimating the charge accumulated by the spacecraft a few assumptions are made: It is assumed that a conservative estimate is achieved by a sphere with a diameter of the diagonal length of the spacecraft and the electric potential difference is 100 V. With these assumptions, the accumulated charge is equal to 1.9271e−8 C.
The model of the magnetic field that is used in the simulation is the "International Geomagnetic Reference Field" (IGRF) model, which already has an implementation in MATLAB's Aerospace Toolbox [77].

Radiation Pressure The radiation coming from the sun and that reflected from the Earth create a force on the satellite. This disturbance force is modelled by Equation (3.6) [76, p.534].

$$\mathbf{F}_{rad} = C_R\frac{WA}{c}\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} \tag{3.6}$$

Where $\mathbf{F}_{rad}$ is the disturbance force due to the radiation pressure, C_R is the satellite's reflectivity, W is the energy flux of the incoming radiation, A is the exposed area, c is the speed of light and $\mathbf{r}_{ij}$ is the position vector from the radiating body i and the receiving body j. This also requires knowing the position of the Sun. The energy flux of the Sun's radiation in LEO is approximately equal to 1360 W m^{-2}. The earth itself radiates approximately 240 $\mathrm{W\,m^{-2}}$ over a cloud covered area and it is assumed that 32% is reflected, yielding approximately 428 $\mathrm{W\,m^{-2}}$ [76, p.542]. The reflectivity of the satellite is assumed to be 0.9 for the entire range of wavelengths that the satellite receives. [1]

3.4. Simulation Navigation and Control

The formation navigation and control are described in this section.

Navigation The navigation of the formation is heavily simplified. Two navigation methods are used, absolute and relative navigation. Each method only adds a constant but random direction bias vector of a certain size as well as normally distributed noise to each component of the position and velocity information. The sizes of the bias vectors as well as the standard deviations of the Gaussian noise for both absolute and relative positions and velocities are shown in Table 3.3.

Table 3.3: Navigation noise and bias sizes for absolute and relative measurements [6, chp 6-7] [1].

	Absolute position	Relative position	Relative velocity
Bias size	1 m	0.05 m	0.0005 $\mathrm{m\,s^{-1}}$
Standard deviation size	2 m	0.02 m	0.0005 $\mathrm{m\,s^{-1}}$

Control A linear-quadratic regulator using the Hill-Clohessy-Wiltshire dynamics was selected, the theoretical background of which is described in Section 2.4. The actual implementation in the simulation is achieved in the functions `errVirtualCenter()`, `controlCommand()`, `thrustAlloc()` and `controlForce()`.
While the control error is calculated simply by the deviation from the nominal position in the Hill frame, the command is then achieved by multiplying said error with the control gain, derived from the LQR control, shown in Section 2.4.2.
The discrete system describing the relative dynamics was achieved by discretizing the continues HCW equations, which were shown in Section 2.3. The discretization time-step was chosen to be 60 s. This was deemed a good balance between firing the thrusters too often (which might require constant attitude adjustments) and not firing them often enough (which might degrade the performance). Furthermore, as was seen from the analytical solutions to the HCW equations, the uncontrolled system has a frequency of the inverse orbital period. This means that the discretized controller should not run into problems due to information loss from under-sampling with a time step of60 s. The discretization method used is the impulse matching method, which preserves the impulse response of the system. Considering that the controller activates the thruster in short bursts, rather than extended periods of time, this method should result in the best control performance.

The thruster allocation problem (i.e. which thruster to fire and for how long) is solved by utilizing the pseudo-inverse of the thruster configuration matrix, shown in Equations (3.7) and (3.8).

$$\mathbf{T}_{config} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \tag{3.7}$$

$$\mathbf{T}_{config}^{-1} = \begin{bmatrix} 0.5 & 0 & 0 \\ -0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & -0.5 & 0 \\ 0 & 0 & 0.5 \\ 0 & 0 & -0.5 \end{bmatrix} \tag{3.8}$$

As the delivered impulse over a certain time period is given by Equation (3.9), the pseudo-inverse can be used

[1] `https://laserbeamproducts.wordpress.com/2014/06/19/reflectivity-of-aluminium-uv-visible-and-infrared/`, last accessed 25.09.20

to deduce the opening times as shown in Equation (3.10).

$$\mathbf{I}_{control} = f_{thruster}\mathbf{T}_{config} \cdot \mathbf{t}_{open} \tag{3.9}$$

$$\mathbf{t}_{open} = \frac{1}{f_{thruster}}\mathbf{T}^{-1}_{config} \cdot \mathbf{I}_{control} \tag{3.10}$$

Where $\mathbf{I}_{control}$ is the delivered impulse over a certain time period, $f_{thruster}$ is the net force delivered by the thruster and $\mathbf{t}_{open}$ is the vector of the opening times for the thruster.
However, using Equation (3.10) can result in negative opening times due to the negative signs in the pseudo-inverse. This can be remedied by simply adding the absolute value of the thruster with the negative opening time to the one opposite it, which results in the same effective impulse.
Finally, the actual force at a given time is then calculated by `controlForce()` based on the opening times.

3.5. Verification

In this section the verification efforts for the simulation are shown. The largest unit tests are described here as well as the system tests applied to the entire simulation. The orbit propagation verification is shown in Section 3.5.1, the verification of the reference frames is shown in Section 3.5.2, the control system verification is shown in Section 3.5.3, the Kalman filter verification is shown in Section 3.5.4 and finally a few system tests are presented in Section 3.5.5.

3.5.1. Orbit Propagation

The Earth Gravitational Model (EGM) 2008 is used in the calculation of the gravitational acceleration [78]. As the related function is available from mathworks, the model itself will not be verified here, but the orbit propagation will be verified using a test case. For a satellite in a Kepler orbit the orbital period can be calculated by Equation (3.11)[79, p.81].

$$T = 2\pi\sqrt{\frac{a^3}{\mu_g}} \tag{3.11}$$

Where T is the orbital period, a is the semi-major axis (in this case the radius of the circular orbit) and μ_g is the standard gravitational parameter of the body that is orbited.
For a satellite with a height of 400 km, in orbit around the Earth, the values on the right hand side of the equation, taken from the EGM model, are [78]

$$r = 6378.137 + 1200 = 7578.137 \text{ km}$$

$$\mu_g = 0.39860 \cdot 10^6 \ \frac{\text{km}^3}{\text{s}^2}$$

Which leads to an orbital period of $T = 6565.3013$ s. When a satellite is placed at a radius of 7578.137 km and given the appropriate velocity to achieve a circular orbit (shown in Equation (3.12)[79, p.81]), the time until the satellite returns to its original location can be determined from the numerical simulation.

$$v_{circ} = \sqrt{\frac{\mu_g}{r}} \tag{3.12}$$

Where v_{circ} is the velocity of an object in a circular orbit, which for the figures specified above amounts to 7252.499 $\text{m}\,\text{s}^{-1}$. This velocity is imparted horizontally (i.e in the x-y plane) in the ECI frame on the satellite, which should result in an equatorial orbit. Since a time-step of 1 second is used in the simulation, by considering the position output only, the period could only be determined up to 1 s. However, for the purposes of the verification, the time step is reduced to 0.001 seconds, which enables checking the model consistency of the error up to 0.0005 seconds. The minimum distance from the initial position of the satellite occurs at 6565.301 s, which is off by 0.0002 s, within the predicted error range of less than 0.0005 s. A three-dimensional plot of this orbit is shown in Figure 3.4a.
In order to further verify the orbit propagation, a non-circular, non-equatorial orbit will be propagated for a given eccentricity e and inclination i. If the initial position of the satellite is solely in the x-direction (i.e $r = [r_0, 0, 0]$ and the velocity $\boldsymbol{v}$ of the satellite is directed in the y and z directions as such

$$\boldsymbol{v} = \begin{bmatrix} 0 \\ \|\boldsymbol{v}\| \cdot \cos(i) \\ \|\boldsymbol{v}\| \cdot \sin(i) \end{bmatrix}$$

The orbital plane should be inclined by the angle i, i.e. the inclination. To determine the magnitude of the initial velocity $\|\boldsymbol{v}\|$, the vis-viva equation can be used, shown in Equation (3.13) [79, p.89].

$$\frac{1}{2}\|\boldsymbol{v}\|^2 - \frac{\mu_g}{\|\boldsymbol{r}\|} = -\frac{\mu_g}{2a} \tag{3.13}$$

Where $\boldsymbol{r}$ is the position vector of the satellite. The definition of the semi-major axis and the eccentricity can be used to relate the semi-major axis and the distance at periapsis as follows [79, chp. 2]

$$a = \frac{r_p + r_a}{2} \tag{3.14}$$

$$e = \frac{r_p - r_a}{r_p + r_a} \tag{3.15}$$

$$\rightarrow r_a = r_p \frac{1+e}{1-e} \tag{3.16}$$

$$\rightarrow a = r_p \frac{1}{1-e} \tag{3.17}$$

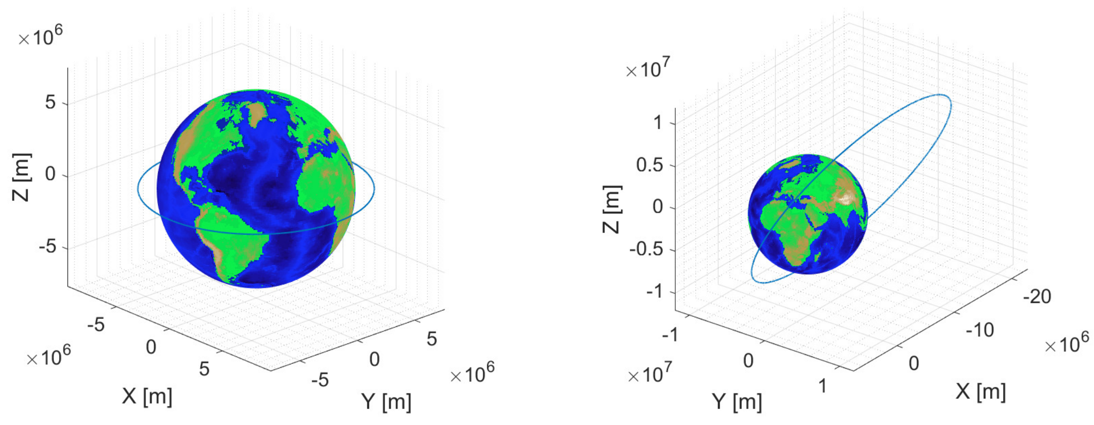
For an eccentricity of 0.5 it follows that $a = 2r_p$ and hence the magnitude of the initial velocity (i.e. the velocity at periapsis) should equal

$$\|\boldsymbol{v_p}\| = \sqrt{-\frac{\mu_g}{a} + \frac{2\mu_g}{r_p}} = \sqrt{-\frac{\mu_g}{2r_p} + \frac{2\mu_g}{r_p}} = \sqrt{\frac{3\mu_g}{2r_p}}$$

Which in this case with a height of 1200 km results in a velocity at periapsis $\|\boldsymbol{v_p}\| = 8882.461\,\mathrm{m\,s^{-1}}$. In addition to the orbital period, which for this elliptical orbit case using Equation (3.11) amounts to 18569.476 s, the maximum distance, i.e. the distance at apoapsis can be checked for accuracy. For an eccentricity of 0.5 and using

$$e = \frac{r_p - r_a}{r_p + r_a} \rightarrow r_a = r_p \frac{1+e}{1-e}$$

The distance at apoapsis should be three times the size of the distance at periapsis, i.e. the initial distance of 7578.137 km, and therefore be 22734.411 km.
The numerical simulation distance at apoapsis shows a difference of only −3.0696e−6 m and an orbital period of 18569.476 s, verifying the orbit propagation.



(a) Circular test orbit with $h_p = 1200, e = 0$, $i = 0°$.

(b) Elliptical test orbit with $h_p = 1200$, $e = 0.5, i = 60°$.

Figure 3.4: Test orbits to verify the simulated orbit propagation, Earth sphere created with [4].

It should be noted that due to the need for analytical solutions, only the first degree of the EGM 2008 model was used for the orbit propagation. It is assumed that the gravitational acceleration calculation from the remaining degrees of the model is correct.

3.5.2. Reference Frame Conversion

There are four main frames which are used in the simulation, the Earth-Centered-Inertial (ECI) frame, the Earth-Centered-Earth-Fixed (ECEF) frame, the Hill frame and the Satellite-Body frame. As the ECI frame is used for performing the main orbital propagation, all positions are initially expressed in the ECI frame. For certain calculations however, they need to converted to some of the other frames.

ECI to ECEF In the calculation of the gravitational acceleration it is important to determine a satellite's position relative to the fixed geometry of the Earth, as it determines the acceleration that the satellite experiences. As the ECI frame and the ECEF frame share a common z-axis, this conversion is simply a time based rotation around the z-axis. This angle of the rotation is dependent on the rotation of the Earth. In this simulation it is assumed that this rotation is constant. To test this rotation, a single point in space was chosen that stays fixed in the ECI frame, 1000 km above the 0-longitude line at the equator at an initial time. This point should rotate westward and return to its initial location after one full Earth rotation, or exactly 23 hours, 56 minutes and 4.2 seconds [80]. A plot of the movement due to the rotation of the Earth can be seen in Figure 3.5, where the conversion from ECI to ECEF has been calculated from $t = 0$ to $t = 18$ hours to see the rotation.

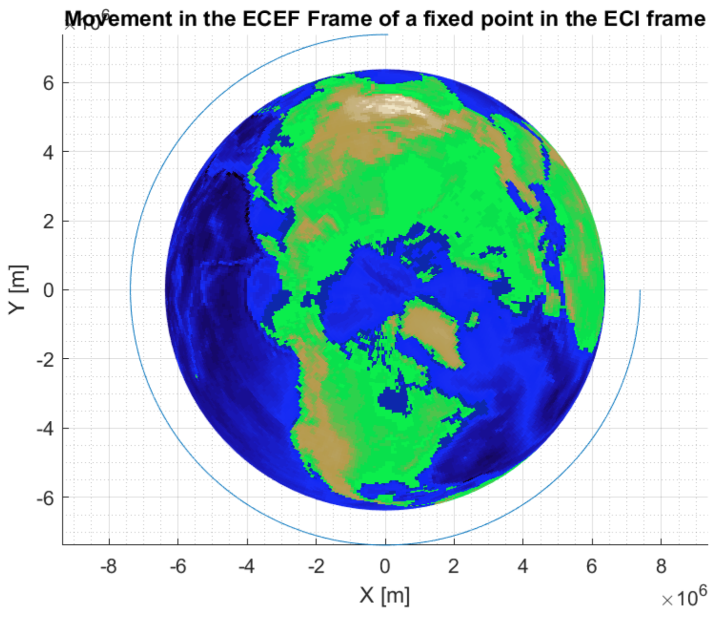


Figure 3.5: Movement in the ECEF frame of a fixed point in the ECI frame.

As can be seen, the point moves westward in the ECEF frame, as expected. The time until it returns to its original location is 23 hours, 56 minutes and 4.2 seconds, verifying the transformation.

ECI to Hill frame The Hill frame is used in the Clohessy-Wiltshire equations, and therefore in the control subsystem. As the Hill-frame is non-inertial, there are additional terms that need to be taken into account. The general formulation of this transformation was already seen in Section 2.1 and is repeated here for reference.

$$\boldsymbol{r}_{Hill} = \boldsymbol{T}_{Hill}^{ECI} \left(\boldsymbol{p}_2 - \boldsymbol{p}_1\right)_{ECI}$$

$$\boldsymbol{v}_{Hill} = \boldsymbol{T}_{Hill}^{ECI} (\boldsymbol{v}_2 - \boldsymbol{v}_1)_{ECI} - \boldsymbol{S}(n\boldsymbol{z})\, \boldsymbol{T}_{Hill}^{ECI}(\Omega) \left(\boldsymbol{p}_2 - \boldsymbol{p}_1\right)_{ECI}$$

The following test case is constructed: consider a chief satellite at a position of $[0, r_0, 0]$ and a deputy satellite at $[0, r_0 + 1, 0]$ (one meter further out), both in circular orbits with an inclination of $i = 60°$ and with an initial height h_0 =400 km.
First, the two rotation matrices that are involved are checked. From the definition of the Hill frame it is clear that at that point, the rotation matrix between the Hill and ECI frame is a simple 90° rotation around the z axis,

followed by a rotation around the x axis by the inclination, i.e. 60°. More generally, the frame transformation can also be constructed from Equation (3.18).

$$\boldsymbol{T}_{Hill}^{ECI} = \boldsymbol{T}_z(\theta)\,\boldsymbol{T}_x(i)\,\boldsymbol{T}_z(\Omega) \tag{3.18}$$

As the Hill frame is a rotating reference frame, also the time derivative of the rotation matrix needs to be checked. Taking the time derivative of Equation (3.18) yields

$$\dot{\boldsymbol{T}}_{ECI}^{Hill} = \dot{\boldsymbol{T}}_z(\theta)\,\boldsymbol{T}_x(i)\,\boldsymbol{T}_z(\Omega) + \boldsymbol{T}_z(\theta)\,\dot{\boldsymbol{T}}_x(i)\,\boldsymbol{T}_z(\Omega) + \boldsymbol{T}_z(\theta)\,\boldsymbol{T}_x(i)\,\dot{\boldsymbol{T}}_z(\Omega) \tag{3.19}$$

In a Kepler orbit neither inclination nor the RAAN are functions of time and such the rotation matrices based on them are both zero matrices. As such only the first term remains, containing the rotation matrix

$$\dot{\boldsymbol{T}}_z(\theta) = \frac{\mathrm{d}}{\mathrm{dt}}\left(\begin{bmatrix}\cos(\theta) & \sin(\theta) & 0\\ \sin(\theta) & \cos(\theta) & 0\\ 0 & 0 & 1\end{bmatrix}\right) = \begin{bmatrix}-\sin(\theta)\dot{\theta} & \cos(\theta)\dot{\theta} & 0\\ -\cos(\theta)\dot{\theta} & -\sin(\theta)\dot{\theta} & 0\\ 0 & 0 & 0\end{bmatrix} = \dot{\theta}\begin{bmatrix}-\sin(\theta) & \cos(\theta) & 0\\ -\cos(\theta) & -\sin(\theta) & 0\\ 0 & 0 & 0\end{bmatrix} = \dot{\theta}\frac{\mathrm{d}}{\mathrm{d}\theta}(\boldsymbol{T}_z(\theta))$$

Inserting this expression back into Equation (3.19) yields Equation (3.20), the analytical equation for the derivative of the rotation matrix.

$$\dot{\boldsymbol{T}}_{ECI}^{Hill} = \dot{\theta}\frac{\mathrm{d}}{\mathrm{d}\theta}(\boldsymbol{T}_z(\theta))\;\boldsymbol{T}_x(i)\,\boldsymbol{T}_z(\Omega) \tag{3.20}$$

As explained in Section 2.1, the time derivative of any rotation matrix where $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$ holds can also be expressed by [35, 36]

$$\dot{\boldsymbol{R}}_B^A = -\boldsymbol{S}(\boldsymbol{\omega})\boldsymbol{R}_B^A$$

Where $\boldsymbol{R}_B^A$ $\boldsymbol{S}$ is a skew-symmetric matrix constructed from the angular velocity vector $\boldsymbol{\omega}$ expressed in frame B. This enables checking the rotation matrix derivative.
As the chief satellite is at a circular orbits with radius $r_0 = R_e + h_0 = 6778.1\,\mathrm{km}$, the mean motion is $n = \frac{2\pi}{T} = 0.00124\,\mathrm{rad\,s^{-1}}$ [79]. Furthermore, $\theta = 0$, $i = 60°$ and $\Omega = \frac{\pi}{2}$.
Using Equations (3.18) and (3.20) as comparisons for the methods presented in Section 2.1, the differences in the two approaches could be calculated. The components of the matrix only differed by less than 1e−16, verifying the matrices.
Second, the computation of the relative position and velocity is verified. The position of the deputy satellite relative to the chief satellite is simply 1 meter in the radial direction, i.e. x direction of the Hill frame, while the relative velocity will be made up from two terms, one from the relative velocity in the inertial frame and one from the rotation of the Hill frame. The first is simply the difference in the ECI frame rotated by 90°, which in this case equals

$$v_{rel} = v_2 - v_1 = \sqrt{\frac{\mu_g}{r_0+1}} - \sqrt{\frac{\mu_g}{r_0}} = 0.0062\mathrm{ms}^{-1}$$

which will be in the along-track direction, i.e. y component of the Hill frame only. The term due to the rotation of the frame is the relative position multiplied by the time derivative of the rotation matrix. As the rotation of the Hill Frame is always around the local z-axis, this component will only be in the x-y plane of the Hill frame. In addition, as the relative position is only in the positive radial direction, the resulting velocity due to the rotation will only be in the negative along-track direction.
As such the relative velocity in the Hill frame is given by

$$\boldsymbol{v}_{rel} = \left[0, \sqrt{\frac{\mu_g}{r_0+1}} - \sqrt{\frac{\mu_g}{r_0}} - n, 0\right] = [0, -0.0019, 0]\,\mathrm{ms}^{-1}$$

Computing the relative position and velocity using the matrices and comparing the difference to the state as well as in the matrices, the differences are less than 1e−12, again verifying the simulation.

ECI to Satellite frame The attitude of the satellites is represented with a quaternion, and conversions between the two frames are performed using the `quatrotate()` function from MATLAB's aerospace toolbox. Two checks are performed to verify this frame transformation. The first is using a reference quaternion while

the second is converting a test vector back and forth using the inverse quaternion.
The reference quaternion is constructed using a set rotation axis $\boldsymbol{\omega}$ and angle γ by Equation (3.21).

$$\boldsymbol{q} = \begin{bmatrix} \cos\left(\frac{\gamma}{2}\right) \\ \omega_1 \sin\left(\frac{\gamma}{2}\right) \\ \omega_2 \sin\left(\frac{\gamma}{2}\right) \\ \omega_3 \sin\left(\frac{\gamma}{2}\right) \end{bmatrix} \tag{3.21}$$

Using $\boldsymbol{\omega} = [0,1,0]$ and $\gamma = \frac{\pi}{2}$, the resulting quaternion represents a 90° rotation around the y axis of the frame to construct the new frame. Four test vectors were rotated using this quaternion and the output can be seen in Table 3.4. As can be seen, the output corresponds to the desired 90° rotation around the y-axis. The sec-

Table 3.4: Frame conversion tests using the quaternion function.

Vector in frame A (input)	Vector in frame B (output)
[1,0,0]	[0,0,1]
[0,1,0]	[0,1,0]
[0,0,1]	[-1,0,0]
[3,4,5]	[-5,4,3]

ond tests involves the quaternion inversion function `quatinv()`. The vector $\boldsymbol{p} = [3,4,5]$ is rotated around a random quaternion and then rotated back by the quaternion inverse. The euclidean norm of the difference between the original and the twice rotated vector is 2.03e−15, verifying that the quaternion inverse can be used as the reverse transformation.

3.5.3. LQR Control

First it is checked whether the linear gain matrix $\boldsymbol{K}$ is correctly computed from MATLABs discrete algebraic Ricatti equation (DARE) solver. According to the documentation the `idare()` function with inputs A, B, Q, R, S and E, it produces the computes the unique stabilizing solution X, state-feedback gain K, and the closed-loop eigenvalues L of the discrete algebraic Ricatti equation given by Equation (3.22). Note that the input matrices here are given the subscript M in order to differentiate them from similarly named matrices from Equation (2.25).

$$\boldsymbol{A}_M^T \boldsymbol{X} \boldsymbol{A}_M - \boldsymbol{E}_M^T \boldsymbol{X} \boldsymbol{E}_M - \left(\boldsymbol{A}_M^T \boldsymbol{X} \boldsymbol{B}_M + \boldsymbol{S}_M\right)\left(\boldsymbol{B}_M^T \boldsymbol{X} \boldsymbol{B}_M + \boldsymbol{R}_M\right)^{-1}\left(\boldsymbol{A}_M^T \boldsymbol{X} \boldsymbol{B}_M + \boldsymbol{S}_M\right)^T + \boldsymbol{Q}_M = \boldsymbol{0} \tag{3.22}$$

The DARE resulting from the LQR problem which was shown in Equation (2.25) is repeated here for reference [63, p.69].

$$\boldsymbol{A}^T \boldsymbol{S} \boldsymbol{A} - \boldsymbol{S} - \left(\boldsymbol{A}^T \boldsymbol{S} \boldsymbol{B}\right)\left(\boldsymbol{B}^T \boldsymbol{S} \boldsymbol{B} + \boldsymbol{R}^{-1}\right)\left(\boldsymbol{B}^T \boldsymbol{S} \boldsymbol{A}_k\right) + \boldsymbol{Q} = \boldsymbol{0}$$

It can be seen that the there are a few additional terms in the MATLAB input ($\boldsymbol{E}_M$ and $\boldsymbol{S}_M$) which are not present in the discrete Ricatti equation resulting from the LQR problem. However, it can be easily seen that if $\boldsymbol{E}_M = \boldsymbol{I}$ is the identity matrix of appropriate size and $\boldsymbol{S}_M = \boldsymbol{0}$ is the zero matrix of appropriate size, the two equations match.
A simplified example from [63] is reproduced here to verify the correctness of the LQR controller. Consider the system given by Equation (3.23)

$$x_{k+1} = a x_k + b u_k \tag{3.23}$$

With the cost function given by Equation (3.24).

$$J = \frac{1}{2} \sum_{k=0}^{\infty} \left(q x_k^2 + r u_k^2\right) \tag{3.24}$$

According to [63, p.75], the corresponding steady state gain is given by

$$k = \frac{abs}{b^2 s + r} \tag{3.25}$$

Where s is the positive solution to Equation (2.22), which can be solved as shown below to result in Equation (3.28).

$$s = a^2 s - \frac{a^2 b^2 s^2}{b^2 s + r} + q \tag{3.26}$$

$$s \cdot \left(b^2 s + r\right) = a^2 s \cdot \left(b^2 s + r\right) - a^2 b^2 s^2 + q \cdot \left(b^2 s + r\right) \tag{3.27}$$

$$b^2 s^2 + rs = a^2 b^2 s^2 + a^2 sr - a^2 b^2 s^2 + qb^2 s + qr$$

$$0 = a^2 b^2 s^2 + a^2 sr - a^2 b^2 s^2 + qb^2 s + qr - b^2 s^2 - rs$$

$$0 = \left(-b^2\right) s^2 + \left(a^2 r + qb^2 - r\right) s + qr$$

$$\rightarrow s_{1,2} = \frac{\left(a^2 r + b^2 q - r\right) \pm \sqrt{\left(a^2 r + b^2 q - r\right)^2 - 4\left(-b^2\right)\left(qr\right)}}{2b^2} \tag{3.28}$$

For the example values $a = 0.1, b = 5, q = 5$ and $r = 0.1$, the positive solution to Equation (3.26) is $s = 5.1002$ and the corresponding value of the gain $k = 0.0981$. Using MATLAB to solve the problem using the `idare()` function, the difference in the value of s is 4.44e−16, which is small enough to be acceptable.

3.5.4. Kalman Filter Verification

The Kalman filter will be verified on a simple one dimensional example, similar to the verification of the LQR. Consider the one dimensional system described in Equations (3.29) and (3.30)

$$x_{k+1} = ax_k + bu_k + v_k \tag{3.29}$$

$$y_k = cx_k + du_k + w_k \tag{3.30}$$

Where v_k and w_k are normally distributed noise with variances q and r, respectively.
As the computation of the Kalman steady state gain matrix is also handled by the `idare()` function of MATLAB, a similar approach to the verification of the LQR controller is taken. For reference, the Ricatti equation for the steady state error covariance is repeated here:

$$\boldsymbol{P} = \boldsymbol{APA}^T + \boldsymbol{Q} - \left(\boldsymbol{APC}^T\right)\left(\boldsymbol{CPC}^T + \boldsymbol{R}\right)^{-1}\left(\boldsymbol{CPA}^T\right)$$

By comparing this to Equation (3.22) it can be seen that the inputs to the function have to be $\boldsymbol{A}^T, \boldsymbol{C}^T, \boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{0}$ and $\boldsymbol{I}$.
The steady state Kalman filter gain that is shown in Section 4.2, in the one dimensional case amounts to solving the same quadratic equation as shown in Equation (3.26).

$$k = \frac{cp}{c^2 p + r} \tag{3.31}$$

$$p = a^2 p - \frac{a^2 c^2 p}{c^2 p + r} + q \tag{3.32}$$

$$p_{1,2} = \frac{\left(a^2 r + c^2 q - r\right) \pm \sqrt{\left(a^2 r + qc^2 - r\right)^2 - 4\left(-c^2\right)\left(qr\right)}}{2c^2} \tag{3.33}$$

Taking $a = 0.6703, c = 1, q = 0.2$ and $r = 3$, the resulting steady state error covariance is $p = 0.3356$ Again, the difference between this and the solution from MATLAB's `idare` function is less than `1e−16`. The system state, measurement and resulting Kalman output for the simplified test case described above can be seen in Figure 3.6. It can be seen that the Kalman estimate follows the state quite closely, only being influenced strongly by large excitations in the noisy measurement.

For the ideal Kalman filter, the residual normally distributed, with zero mean and variance $\boldsymbol{CPC}^T + R$. The distribution of the simulated residual can be seen in Figure 3.7, for n = 10,000 discrete time steps. The resulting residual follows a normal distribution very well as can be seen from the figure. The mean of this distribution is 4.36e−4 with a standard deviation of 1.8238. This matches closely with the expected mean of 0 and standard deviation of $\sqrt{\boldsymbol{CPC}^T + \boldsymbol{R}} = \sqrt{c^2 p + r} = 1.8264$.

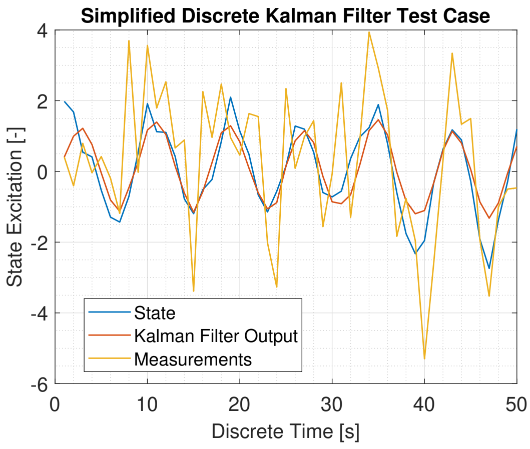


Figure 3.6: One dimensional test system response, measurement and Kalman filter output.

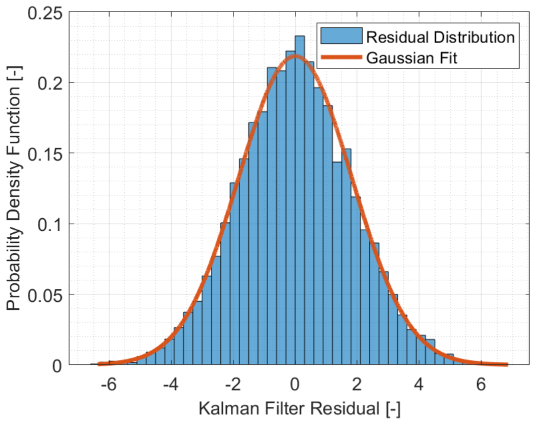


Figure 3.7: Distribution of Kalman residual in the simplified test case with Gaussian fit.

In addition to the one-dimensional example seen above, the residual vector in the for the relative positions of satellite 1 in the full simulation can be seen below.

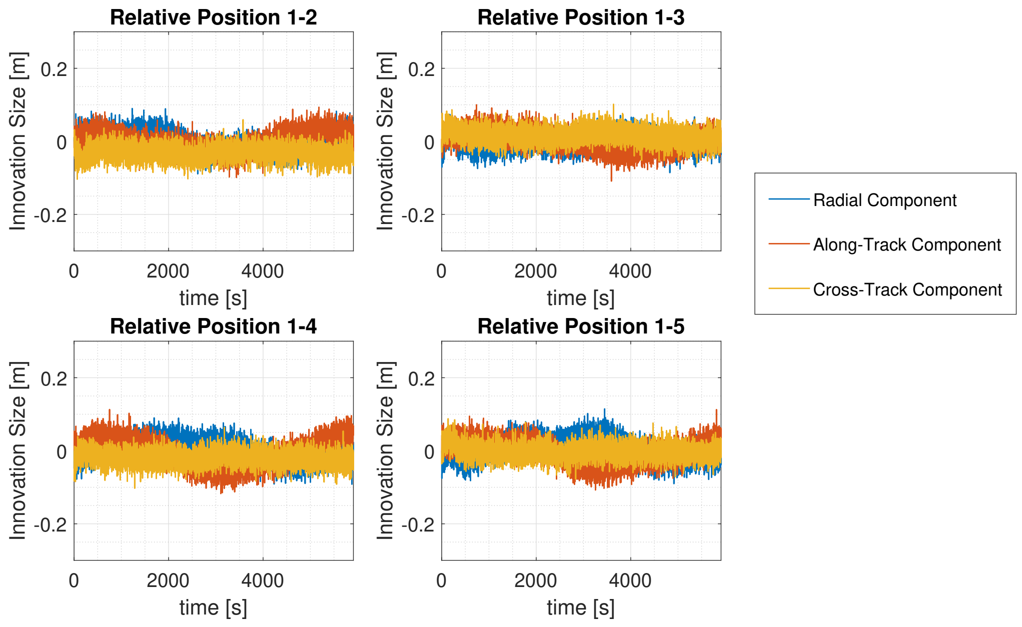


Figure 3.8: Residual vector for position components of formation state vector.

It can be seen that there is still time dependent variation in the residual signal. The ideal Kalman filter has a mean of 0, however the Clohessy Wiltshire equations are a linearization of the full non-linear relative motion model. One would expect this deviation from the zero mean to decrease with a smaller formation size, as it makes the assumptions of the linearization more close to valid. The residual for a closer (14.2 m distance, rather than 142 m) formation can be seen in Figure 3.9. In addition, the noise was reduced by a factor of 10 in order to see the variation of the mean more closely.

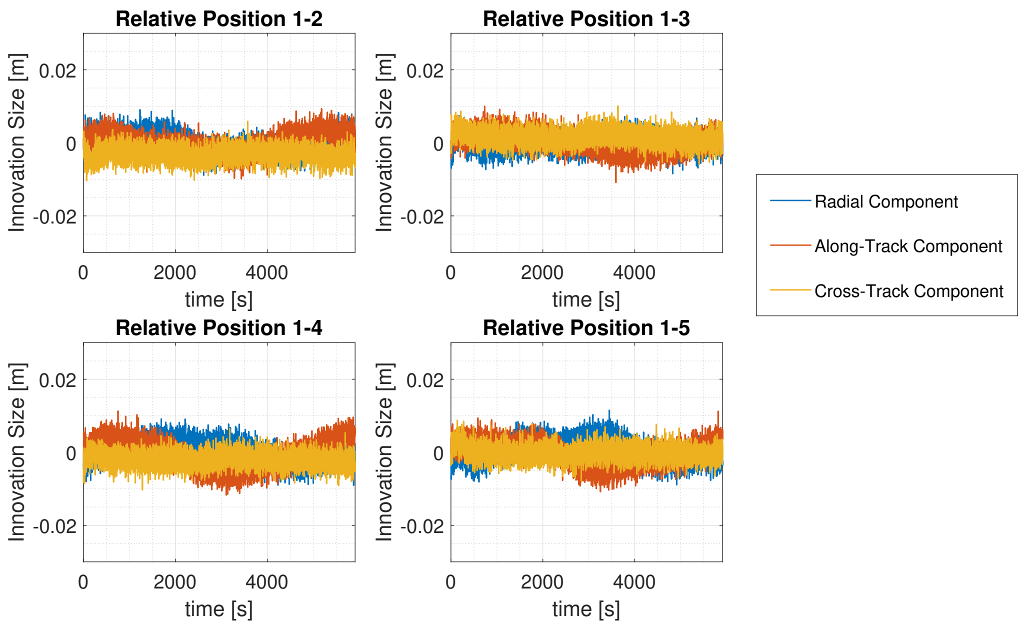


Figure 3.9: Residual of the Kalman filter for a closer formation with less measurement noise.

As expected, the mean is much closer to zero, indicating that the deviation seen in Figure 3.8 is due to the linearization. With this, the Kalman filter is verified.

3.5.5. System Tests

Three overall system tests will be performed to see if the simulation works as expected For the mission parameters of a nominal distance of 143 m formation size, the relative distances are shown in Figure 3.10.

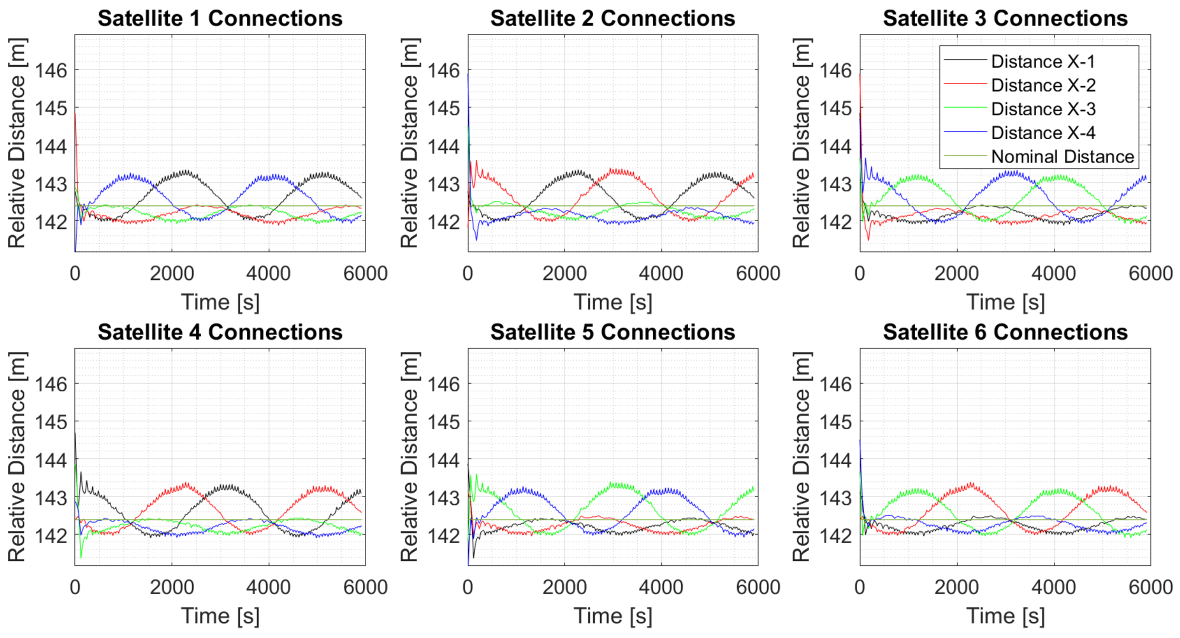


Figure 3.10: Relative distances over time for the default formation size.

If the satellites are placed five times further apart and the corresponding formation size that the satellites aim to keep is increased, the control should hold the formation at the greater distance as well. The only difference should be the error in keeping the formation, which should increase as the spacecraft will drift further apart from their natural orbits for an increased distance. The resulting distance plots can be seen in Figure 3.11

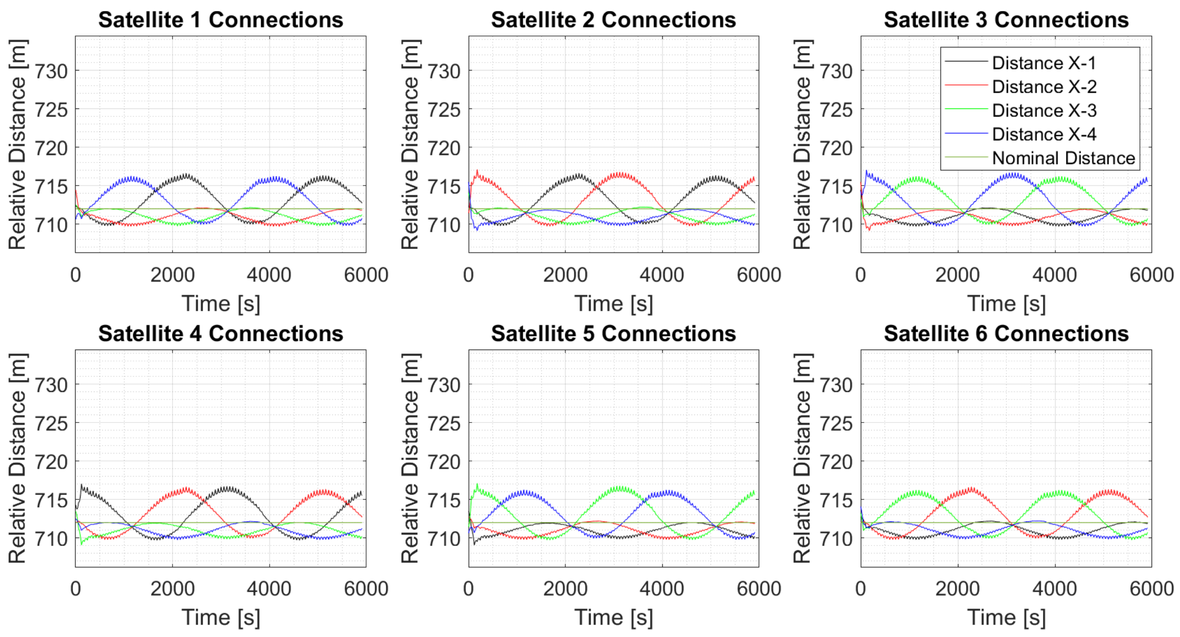


Figure 3.11: Relative distances over time for a five-fold increased formation size.

As can be seen, the formation still holds, albeit with the expected increase in the error. Whereas in the default formation size the error varies approximately sinusoidally with an amplitude of around 1 m, it can be seen that in the increased formation size the error amplitude is up to 6m.

The second system test is a tenfold decrease in the thrust value. A decreased thrust should increase the burn-times of each thruster. The corresponding relative distances can be seen in Figure 3.12. Furthermore it was found that the mean thruster duration increased from 0.1143 s to 1.1378 s, approximately a factor of ten increase. This is not surprising considering that the control system still aims to deliver the same change in momentum each thrust period. With a tenfold reduced thrust value, the opening duration has do increase by a factor of ten.

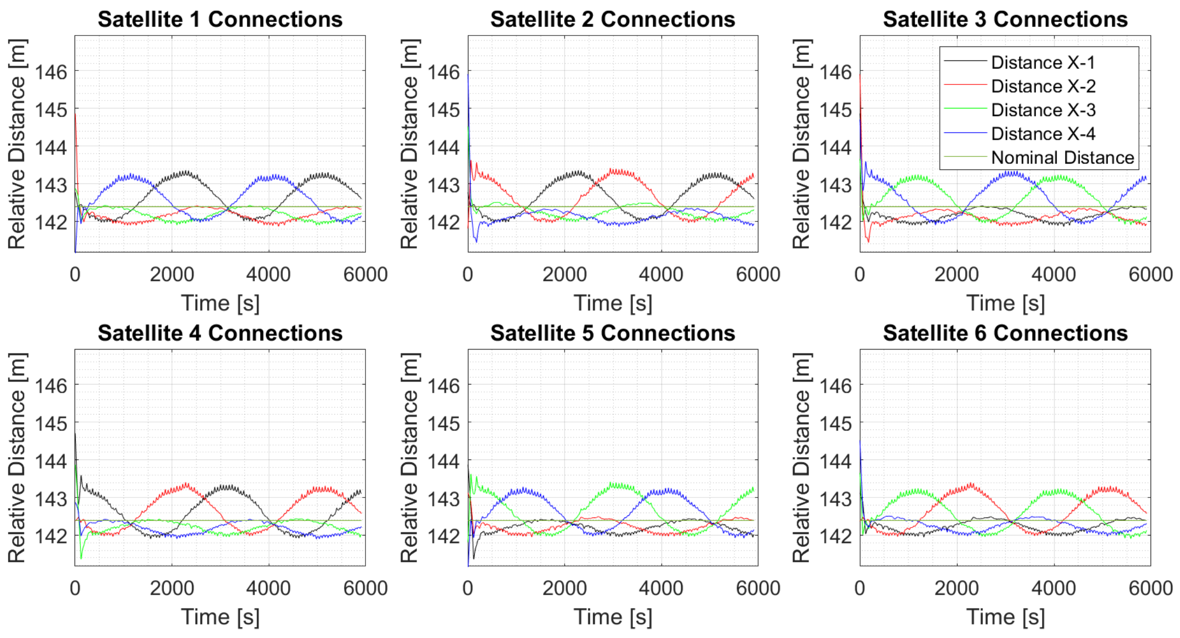


Figure 3.12: Relative distances over time for a tenfold decreased thrust capability.

Finally, the last system test is reducing the time in between corrective bursts from 60 s to 10 s. It would be expected that the error in the relative distances goes to zero as the control system has more opportunities to counteract the build-up of errors. The resulting relative distances can be seen in Figure 3.13.

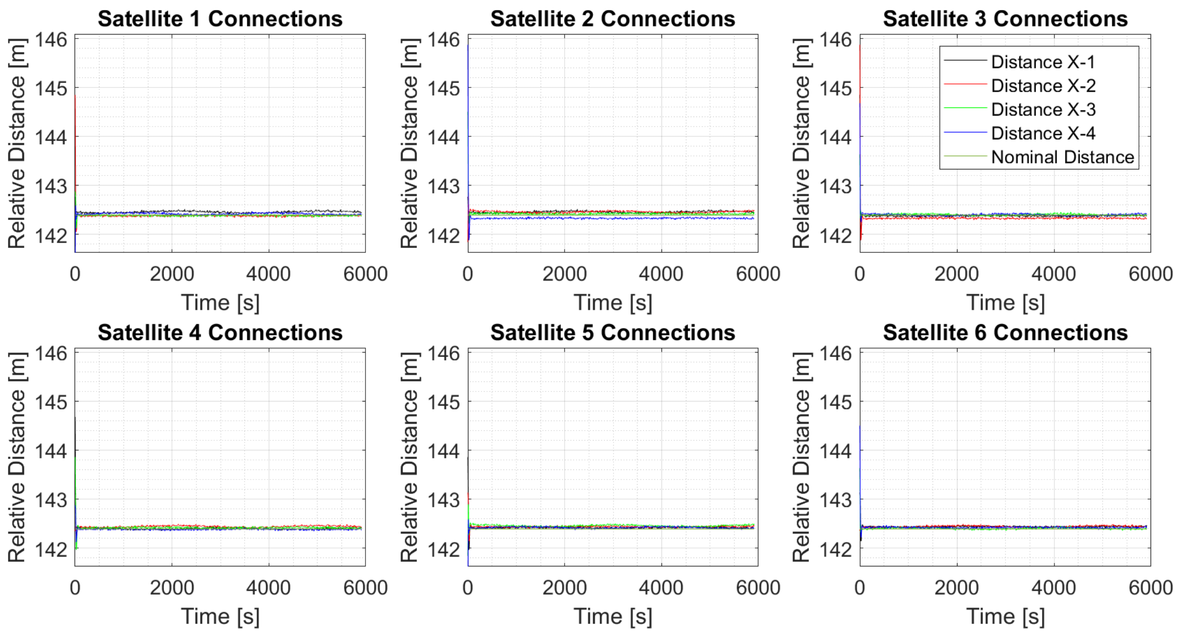


Figure 3.13: Relative distances over time for a decrease in the time between thrusts to 10 s.

As can be seen, the deviation in the relative distances from the nominal distance of 142.39 m is severely reduced. The previous amplitude of the error was around 1 m, which has now reduced to less than 10 cm. The positioning error still shows sinusoidal behavior, however it is now even more clear that the mean of that oscillation is offset from the nominal distance by a certain bias. These biases in the control error are the result

of the measurement biases and disappears if the size of the biases is reduced to zero.
With this, the simulation is considered verified. It has given correct results for all unit tests and intuitive behavior when the behavior of the entire simulation is tested against various conditions.

3.6. Fault Modeling

Two types of faults will be considered in this thesis both relating to the operation of orbital control thrusters. The first is a fault in some of the valves in the cold-gas thruster which prevent the valves from opening fully, thus reducing the effective thrust that is delivered and the second is a valve stuck in an open position allowing propellant to escape and thus creating an unintended force on the satellite. The first is called a *closed fault* and the second an *open fault* for brevity.
The exact implementation of either fault in the simulation is discussed in Section 3.6.2, while the impact of the faults on the relative positioning is explored in Section 3.6.2.

3.6.1. Implementation

Both faults are categorized by a fault parameter ϕ, which specifies their intensity. This parameter can range from 0 to 1, 0 representing no fault and 1 representing the most intense the particular fault can be. For a closed fault, a fault parameter of 1 represents a complete closure of the valve, the thruster cannot produce any more thrust. For an open fault, a fault parameter of 1 means a complete opening of the thruster, the thruster fires at full thrust for the entire time.
The implementation in the code is achieved by the function `setFault()` of the formation class. It sets the type and location of the fault, as well as the value of the fault intensity in the correct satellite. Then, in the satellite function `controlForce()`, the calculation of the actual force resulting from the thruster openings is adjusted based on the fault type. For a closed fault, the thrust output is reduced according to the fault intensity, while for the open fault an additional term is added.

3.6.2. Fault Impact

The faults will have an impact on the relative motion in the formation, as the formation keeping is dependent on the control forces.

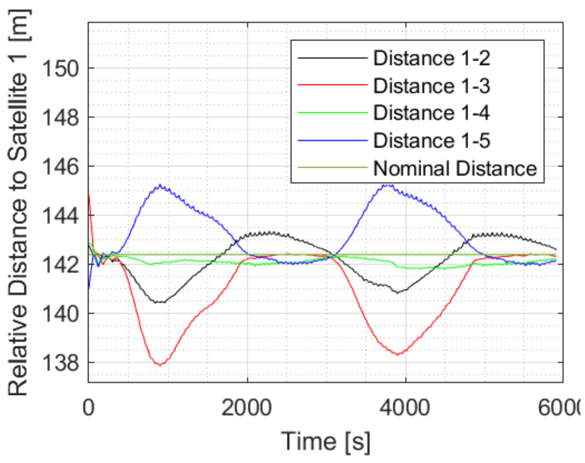


Figure 3.14: Relative distances of satellite 1 to its neighbors under the influence of a closed thruster fault in S1T2.

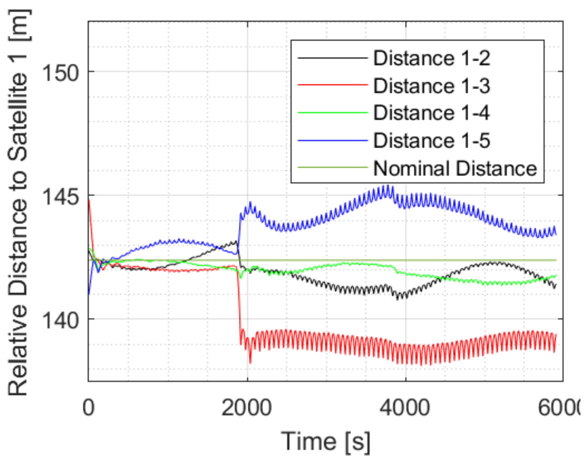


Figure 3.15: Relative distances of satellite 1 to its neighbors under the influence of an open thruster fault in S1T1.

The impact of the closed and open fault can be seen in Figure 3.14 and Figure 3.15, respectively. When compared to the behavior seen in Figure 3.10, it can be seen that the response of the formation is quite different. While the closed thruster type fault results in a temporary deviation from the typical sinusoidal pattern, the open fault results in a constant deviation with a higher frequency oscillatory behavior around this new nominal distance.

These effects are not surprising as the closed faults are only apparent when the faulty thruster is scheduled to fire. When this is not the case, the formation recovers and returns to its usual performance. The open fault on the other hand causes a constant disruption which requires much more control action to maintain the formation. This can also be seen in the required ΔV over one orbital period for the two fault cases. For a fault intensity of 1 (i.e. a complete failure of the thruster), the closed fault increases the average ΔV from $1.00\,\mathrm{m\,s^{-1}}$ to $1.08\,\mathrm{m\,s^{-1}}$, while an open fault of intensity 0.1 already increases ΔV to $4.96\,\mathrm{m\,s^{-1}}$. The satellite experiencing the open fault shows the largest increase, as the open thruster continuously emits propellant, thereby increasing the imparted momentum on the spacecraft. However, the other spacecraft in the formation also experience an increase up to a ΔV of $3.14\,\mathrm{m\,s^{-1}}$.
As such it can already be seen that the open faults are much taxing on the formation than a closed fault with the same fault intensity.

3.7. Data Generation

In this section, the process of generating the training data is detailed. First, the setup for the data generation in MATLAB will be described in Section 3.7.1, followed by a discussion on the influence of the disturbance forces on the relative positioning in Section 3.7.2. Them the format of the saved data will be explained in Section 3.7.3. Finally, a small issue regarding the dataset with regards to compatibility with the Kalman filter method is mentioned in Section 3.7.4.

3.7.1. Setup

The data for the training will be generated by the simulation described in this chapter. In order to create a varied dataset, certain options are randomized for each simulation run. These include the time of fault inception, the fault intensity and the initial position deviation from the nominal positions, in addition to the already random noise and bias that are added to the measurements.
As the response to a particular fault will differ based on the position in the orbit, which varies with time, the inception time of the fault should be randomized. As can be seen in Figure 3.10, there is a periodic variation in the relative distances which repeats twice per orbit. In order to still be able to see the effects of a fault in the remaining simulation time, the fault should be injected well before the end of the first orbit.
In addition to the fault time, the initial positions are randomized as well. This is achieved by not placing the satellites at the exact distance of the octahedron, but also adding a normally distributed variation to each component of the position, with a standard deviation of 1 m. This has the consequence that the formation is not immediately in the proper relative positioning and needs some time to properly establish the formation and reach the nominal relative positioning. It was seen that this takes approximately 300 s. The network should not be given the data during this time as it might be confusing to see large deviations from the nominal values without a fault present.
The fault time is therefore randomly distributed in the interval $[300\text{s}, \frac{T}{2} + 300\text{s}]$. This should ensure that no bias with respect to the fault time is present in the final dataset.
Furthermore, in the final dataset each class should be represented equally. For the detection networks this means an equal amount of faultless and faulty samples, while for the isolation network this means an equal amount of open and closed fault types across all thrusters in the formation. In order to have a sufficiently representative sample, each fault type is simulated one thousand times for all thrusters in the formation, amounting to 72000 simulations. As these simulations contain both a certain amount of faulty and faultless data, the average fault time needs to be taken into account when determining the amount of purely faultless simulations that will be necessary to balance the classes out. The expected value of a uniform distribution over the interval $[a, b]$ is $\bar{x} = \frac{a+b}{2}$. As the fault time is uniformly distributed over the interval $[300\text{s}, \frac{T}{2} + 300\text{s}]$, the expected average value is $\bar{t}_{fault} = \frac{T}{4} + 300 = 1777.78$. Therefore in any simulation run there are on average 30.08% faultless time steps. As 72000 runs are performed, approximately 21654 runs worth of data contain no faults. In order to balance this with the 50346 simulation worth containing faulty time steps, roughly an additional 28700 faultless simulation runs are needed to balance out the faultless and faulty data. This results in approximately 100,000 simulations, with a total size of around 700 GB.
The fault intensity is randomly generated on the interval [0.1,1]. The lower bound of 0.1 was selected as to not generate data that is too similar to the faultless case, as this can cause confusion in the network training process.

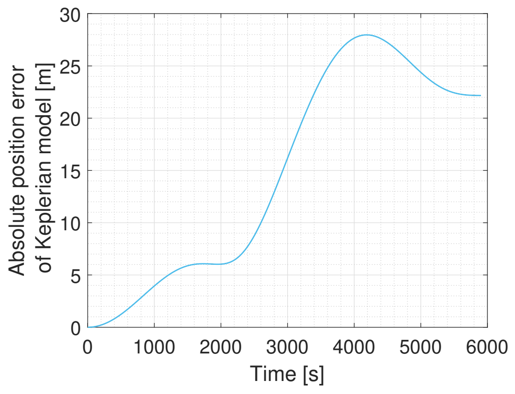
3.7.2. Influence of Disturbance Forces and Discretization Time Step on Relative Positions

For proper performance of the neural networks, they need to be trained on a large amount of data. As there is only a limited amount of time available, it is crucial that the simulation runs are short. Therefore, the code needed to be optimized to decrease the run time. In particular, the components of the simulation which take the longest time need to be reduced, this being the the gravity model and the disturbance forces. For the gravity model, this is due to the need to compute the Lagrange polynomials involved in the spherical harmonics. Similarly for the disturbances, the Earth's magnetic field model needs to be accessed (which also involve spherical harmonics), slowing down the simulation.
However, it may not be necessary to use the complete model to generate the data, as the networks will be fed noisy measurement data. In order to determine what degree of accuracy is necessary for the generation of the data, the impact of computing the disturbance forces as well as the degree of the spherical harmonic gravity model were determined.
Table 3.5 shows the error between various simplified models and the highest fidelity model that the model allows (120 degrees of the spherical harmonic model and all disturbances). In addition, the absolute error over one orbital period for all the considered models is shown in Figure 3.16.

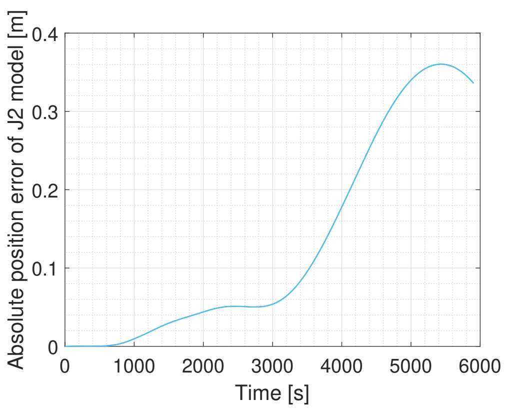
Table 3.5: Comparing the performance of various models in terms of maximum absolute and relative position error as well as computation time.

Model Description	Absolute position error	Relative position error	Time per run	Shown in
Basic Keplerian Orbit (n = 1, no disturbances)	28 m	0.38 m	37.0 s	Figures 3.16a and 3.17a
Gravity model including J2 Effect (n = 2, no disturbances)	0.38 m	5.2e−3 m	49.8 s	Figures 3.16b and 3.17b
J2 Effect and active disturbance forces (n = 2, with disturbances)	0.38 m	5.2e−3 m	60.6 s	Figures 3.16c and 3.17c
Up to 10th spherical harmonic and active disturbance forces (n = 10, with disturbances)	8.4e−3 m	5.5e−4 m	161.7 s	Figures 3.16d and 3.17d

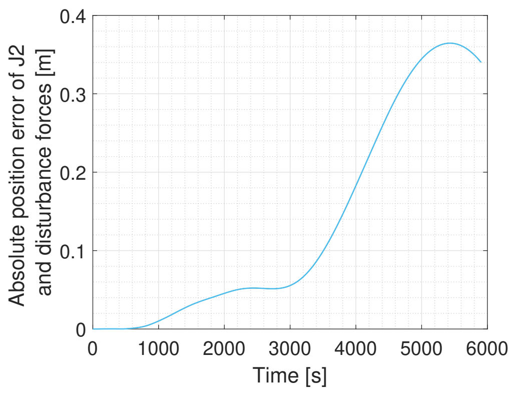
What can be seen is that for the Kepler orbit (see Figure 3.16a), the absolute position error is very large with a maximum deviation of approximately 28 m from the best model. However, by simply accounting for the J2 effect by increasing the degrees of the gravity model, this error drops to 0.36 m, shown in Figure 3.16b. Increasing the degree of the gravity model to 10 gives another significant decrease in the error, decreasing it to 8.4e−3 m, shown in Figure 3.16d. The positioning error when the disturbance forces are included can be seen in Figure 3.16c. By comparing Figure 3.16b and Figure 3.16c, it can clearly be seen that including the disturbances makes little difference in the accuracy of the model, as the error barely changes.



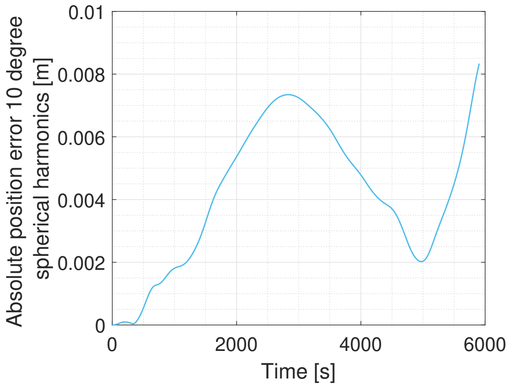
(a) Absolute error for no disturbances and n = 1 (i.e. Keplerian orbit).



(b) Absolute error for no disturbances and n = 2.



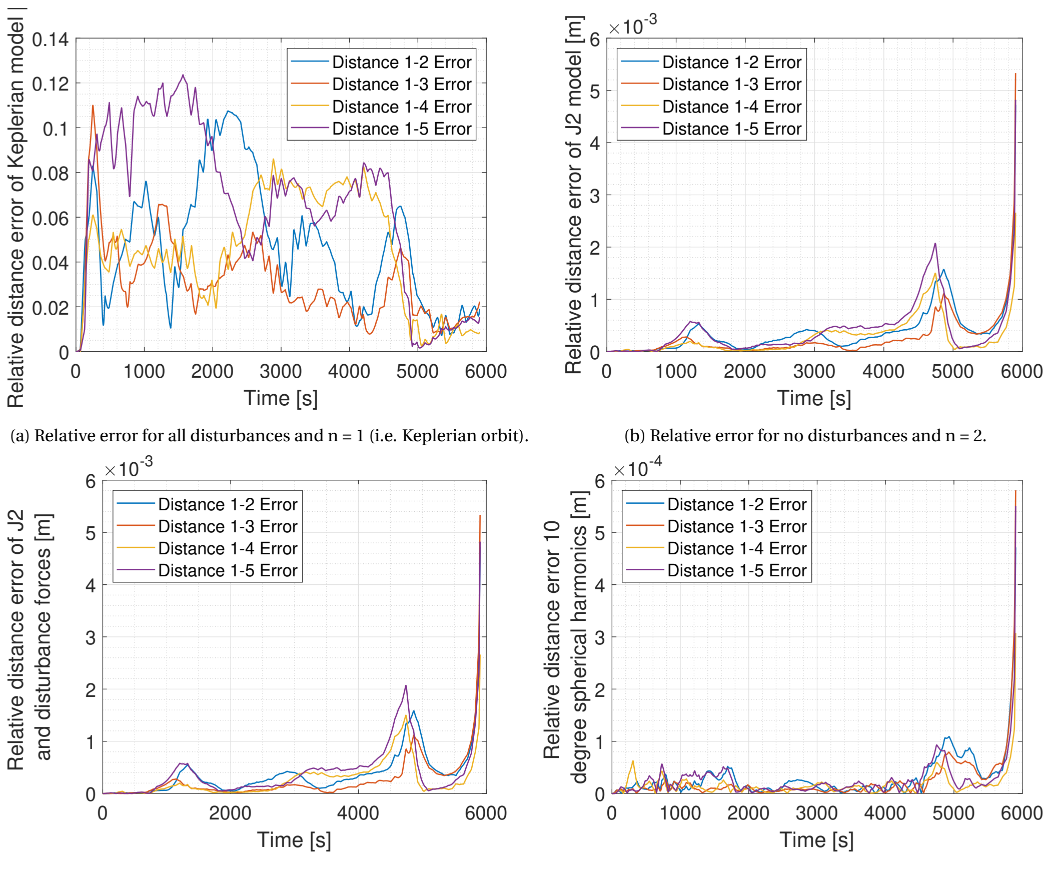
(c) Absolute error for all disturbances and n = 2.



(d) Absolute error for all disturbances and n = 10.

Figure 3.16: Absolute position error of satellite 1 for various simulation settings and degrees n of the spherical harmonic gravity model (EGM 2008).

While the absolute accuracy positioning is important for the simulation as a whole, the accuracy of the relative motion is a more relevant parameter for the neural networks. The neural networks receive the relative positions and velocities as input to determine whether or not a fault has occurred, and where a fault has occurred. The reason for this is the fact that they can be measured more accurately than the absolute quantities. As such, a better evaluation criteria is the error in the relative distances. These are shown in Figure 3.17. As can be seen, the relative distance errors are much smaller in magnitude compared to the absolute position errors, as already for the Kepler model (shown in Figure 3.17a) the maximum relative distance error is only 0.12 m. By including the J2 effect, shown in Figure 3.17b, the maximum error drops to 5.2e−3 m. It is also notable that for most of the orbit the error stays below 2e−3 m.
By inspecting Figure 3.16c it can be seen that, again, the inclusion of the disturbances makes no visible difference in the error. Another order of magnitude decrease in the maximum error can be achieved by increasing the degree of the gravity model, visible in Figure 3.17d, where the maximum error reaches 5.8e−4 m.



(a) Relative error for all disturbances and n = 1 (i.e. Keplerian orbit).

(b) Relative error for no disturbances and n = 2.

(c) Relative error for all disturbances and n = 2.

(d) Relative error for all disturbances and n = 10.

Figure 3.17: Relative position error of the distances of satellite 1 to its neighbors for various simulation settings and degrees n of the spherical harmonic gravity model (EGM 2008).

Considering that the noise level for the positions will be in the centimeter to decimeter range, having a simulation which produces relative distances with an error below the millimeter scale is not worth it given the time increase that comes with the higher accuracy models.
As such, for the generation of the data, the **J2 model with no disturbances** will be used.

The time step used for the numerical simulation is an important variable that mainly controls how long the simulation takes to run and how accurate the numerical simulation is. A time step of 1 second is used for various reasons; The first is that for orbit propagation this should result in a reasonable amount of accuracy. At the same time, this allows for relatively fast computation time for each simulation run. Considering the amount of runs that need to be performed, this is crucial.
The error that builds up over time due to the time step is shown in Figure 3.18. The orbit was propagated using a time step of 0.001 s to serve as a comparison to the much faster case where the time step is at 1 s. As can be seen, the error does build up over time but does not exceed 9e−6 This error is considered acceptable for the purposes of training the network, considering the much larger noise in the measurements.

3.7.3. Data Format

The MATLAB script saves each simulation in a CSV (comma seperated value) text file which is formatted in the following manner. The first line gives information about the fault, specifically the fault time, the faulty satellite number, the number of the faulty thruster on said satellite,the fault type (closed [0] or open [1]), and the fault intensity parameter.
What follows is the relative positioning data in blocks of 24 lines. Each line gives the relative position errors, i.e. the difference between the measured relative position and the intended relative position and velocities in the ECI frame. The first four lines of each block correspond to the relative positions of the neighbors of satellite 1, i.e. satellites 2, 3, 4 and 5, to satellite 1. The next four correspond to the neighbors of satellite 2 and so on.

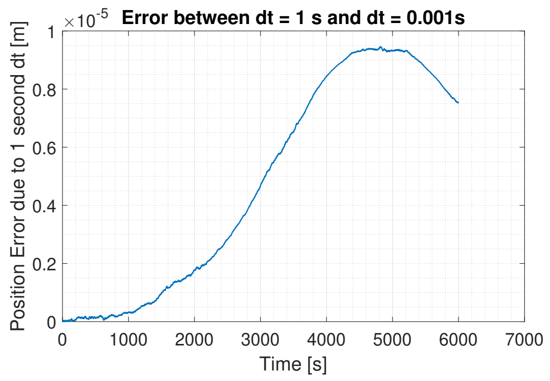


Figure 3.18: Numerical error in the position due to discretization time step.

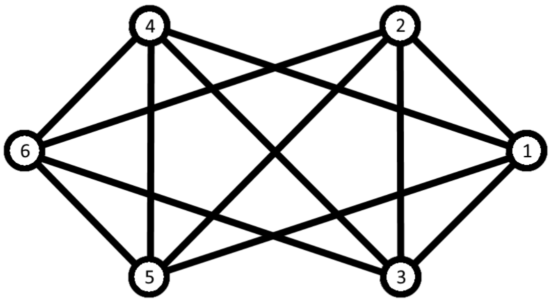


Figure 3.19: Connection graph of the Formation.

The ordering of these connections is summarized in Table 3.6. They are also graphically illustrated in Figure 3.19. For example, the first neighbor of satellite 4 is satellite 3 and as such its data is the first row for satellite 4, making it the 13th row for a particular block. After the last connection of satellite 6, the next block corresponding to the next time step starts.
As one file consists of one simulation run, amounting to 5902 time steps, the final files consist of 147551 lines of text, with an approximate file size of 7.4 MB.

Table 3.6: Ordering of satellite connections.

Satellite	Neighbor 1	Neighbor 2	Neighbor 3	Neighbor 4	Not connected
1	2	3	4	5	6
2	1	5	6	3	4
3	1	2	6	4	5
4	3	6	5	1	2
5	4	6	2	1	3
6	5	4	3	2	1

The connection graph in Figure 3.19 shows a certain projection of the connections of an octahedron onto a two-dimensional plane. This particular figure shows well, which satellites do not have a connection. The outermost two satellites (1 and 6), as well as the pairs which are beside each other in the center (2 and 4, 3 and 5).

3.7.4. Issue with Network Dataset

Due to the considerable size of the dataset that was created for the purpose of training and evaluating the neural network, only the relevant input (being the deviation of relative position and velocities from the default) and relevant data for classification purposes (fault type, fault intensity, fault time, etc) was saved to files. As the Kalman filter requires the control input as additional information, as well as requiring absolute posi-

tion and velocity information in order to compute the Hill-Frame, it cannot be evaluated on the exact same dataset as the neural networks.
However, the filter will be tested on a representative dataset with equal distribution of faultless, fault 1 and fault 2 cases as well as equal fault intensity distribution. This enables a base of comparison.

3.8. Simulation & Data Generation Summary

The simulation that was designed to test the FDI approaches has been introduced in this chapter. One of its main goals is to generate data that can be used to train the neural network approaches. In this thesis, the necessary data is the relative positioning and velocity of the satellites in the formation. To achieve this aim, the simulated mission, faults and the specifics of the data generation were shown.

The simulated mission scenario is a virtual-rigid-body type formation, in low Earth orbit with six satellites in an octahedral shape. The satellites maintain a constant distance to each other, and the orientation of the formation stays fixed. The formation is maintained through shorts corrective bursts every minute, which are determined through a control-system based on a linear-quadratic regulator making use of the Hill-Clohesssy-Wilthire equations.
This mission is simulated in MATLAB, making use of its object-oriented features. Two custom classes were written, one for the formation as a whole and one to represent the individual spacecraft.
The simulation uses a Runge-Kutta 4 integration scheme, where each time-step the relevant forces are calculated and used to determine the next state. The forces involved include the gravity of the Earth, simulated by making use of the EGM 2008 model, as well as the most prominent disturbance forces. These are: The aerodynamic drag, the Lorentz force, the radiation pressure and the effect of the Sun's and Moon's gravity.

To investigate the efficacy of the fault detection and isolation system, two types of thruster faults are simulated in this thesis. The first results in a blockage of propellant, leading to a reduction of the available thrust. The second results in a propellant leak, leading to a constant thrust force. Both are integrated into the simulation and can occur with varying intensity, characterized by a parameter ranging from 0 to 1. A value of 0 represents no fault, while a value of 1 represents the most intense the fault can be. For the closed faults this means a complete blockage, meaning no thrust can be produced. For an open fault on the other hand, this means a full opening of the thruster, resulting in maximum thrust at all times.

These faults are simulate to determine the response of the formation. The necessary relative position and velocity data is then generated by repeatedly simulating the formation, with random starting conditions, under the effects of one of the two faults. To generate a sufficiently large dataset, 1000 simulations were performed for each of the possible locations of a thruster fault. Together with the faultless simulations, this resulted in approximately 100,000 simulations, amounting to approximately 700 GB of data. For this data generation, it was found that the higher degrees of the EGM 2008 gravity model and the disturbance forces provide very little improved accuracy while rapidly increasing the amount of time spent on each simulation run. Due to the amount of simulation runs, the disturbances are not used for the data generation, and the gravity model is limited to the second degree of the spherical harmonics, J2.

4 Fault Detection and Isolation Methodology

In this chapter, the methods used for detecting and isolating the considered thruster faults are described. First, the distributed approach via artificial neural networks (ANNs) is shown in Section 4.1, followed by the the centralized, model-based Kalman Filter Section 4.2.

4.1. Neural Network Approaches

The considered approach for FDI using ANN is elaborated on in this section. First, the selected architecture of the neural network based method is presented and motivated in Section 4.1.1, with the selection of the hyper-parameters in Section 4.1.2. This is followed by a description of the training environment and organization in Section 4.1.3. After this, the required data processing of the simulation data is given in Section 4.1.4. The conversion from a continuous output to a discrete detection is explained in Section 4.1.5. Then the training and pre-procesing is verified in Section 4.1.6. Finally, the process for the network evaluation for the analysis in the next chapter is elaborated on in Section 4.1.7.

4.1.1. Neural Network Architecture

The structure of the neural networks, or their *architecture*, is comprised of various elements. This includes the type of neurons that are used in the network, the connections between the neurons, and the way the input and output to the network is handled. All of these points will be addressed in this section, starting with the type of network.

Motivation for choosing RNNs Of all the various types of neural networks described in Section 2.5, a suitable type has to be selected. In general, there are many types of possible networks, with varying numbers of forward or backward connections and layer structures. However, only the standard choices will be considered as the expertise to decide on a more specific structure is not available. These standard networks are the densely connected feed-forward networks, convolutional neural networks and recurrent neural networks. Convolutional neural networks (CNN) are mainly used in applications involving the analysis of grid-organized data, e.g. images or pictures, as the connections involved mimic the two dimensional *convolution* operation. According to Jiuxiang Gu et al, most of the areas of application in recent years involve image processing to some extent, for example image classification, object detection, or object tracking, among others [81]. In these visual-based applications, the CNNs show very good performance [82, 83]. Only a few attempts are made to use CNNs on time dependent data and in those cases the CNN is used in conjunction with other methods, such as using a character level CNN as input to a series of LSTM layers for natural language processing [84].
Densely connected neural networks on the other hand are potentially good at many tasks, due to the generality of their setup. As dense neural networks are not set up to take advantage of any structure in the input data, they can potentially be used for any task. However, they have the disadvantage of having to make a decision based only on a snapshot of data in time. Especially for a dynamic system which is constantly changing its state, this is undesirable.
Recurrent Neural Networks (RNN) are especially suited towards modeling dynamic systems and other time-

series prediction problems [30, 85]. They are furthermore used in natural language processing as well as control systems [86, 87]. Compared to the other methods presented, RNNs have the unique advantage of having 'memory', i.e. a state that can persist over time. This can enable better predictions for data that is time-dependent. They are also used in the field of FDI [32, 88]. As the scenario investigated in this thesis involves the analysis of a dynamic (i.e. time-varying) system, the use of RNNs is intuitively advantageous. The ability to learn time-dependencies and using this in the diagnosis of the system makes it more beneficial compared to the use of only CNNs or simple dense networks.

Of the various RNNs the simplest method to use and implement is the LSTM network, which was presented in Section 2.5.5. According to Hochreiter and Schmidhuber, the LSTM type network can learn connections over up to a thousand time-steps [69]. This also makes it useful for the problem at hand, as depending on the fault type and severity, the effect of a fault may only be visible in the measurements after some amount of time.
For this thesis, the specific type of RNN chosen is the Long Short-Term Memomry units (LSTM) introduced in **??**. Many more types of RNNs exist, e.g. the Gated Recurrent Unit (GRU) networks or the Bidirectional Recurrent Neural Networks (BRNN) [89], and some have better performance than an LSTM network for certain applications. However, a network with LSTM units can achieve performance close to the optimal architecture when a constant bias is added to the 'forget' gate [90]. This particular procedure is applied to the LSTM networks trained in this thesis, in order to enhance the performance of the networks.

After deciding on the general type of neural network, the layer amount and structure have to be decided. It was decided to use three hidden layers in total, made up of two layers of LSTM units and one densely connected layer, illustrated in Figure 4.1.

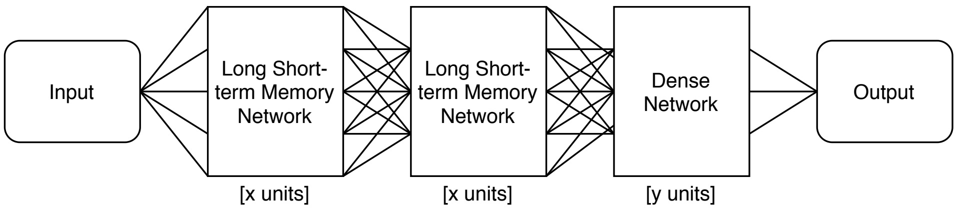

Input
Long Short-term Memory Network
[x units]
Long Short-term Memory Network
[x units]
Dense Network
[y units]
Output

Figure 4.1: General architecture of the networks considered in this thesis.

As can be seen in Figure 4.1, the processed input to the neural networks is fed into a layer of LSTM units, followed by another LSTM layer, followed by a third layer of densely connected neurons (with a different amount of units, generally) which then connect to the output layer. Only three hidden layers are used as increasing the amount of layers drastically increases the training time. The reason for including a dense layer in between the LSTM layers and the output is twofold. First, it increases the depth of the network which can result in better performance for complex problems [91]. Second, it is reasonable to assume that the output of the LSTM layers requires additional processing before it can lead to a decision on detection or isolation of a fault.
In principal, even more diverse and complex architectures could be tested, but are not investigated in this work due to time-constraints and a lack of experience with the structuring of neural networks. Ideas for adaptations of this structure are given in the recommendations in Chapter 6.

Application to Detection and Isolation As the problem at hand, fault detection and isolation, is comprised of two distinct activities, the question arises whether the same network should be used for both detection and isolation at once, or whether two networks should be trained, one for detection and one for isolation. The latter approach has several advantages. First, detection and isolation differ vastly in their complexity. Having a dedicated, smaller network to first detect a fault and only then run the fault isolation network could improve computational time, as the detection network would not necessarily need to be as large or deep.
Second, it is advantageous to be able to distinguish the nominal case (no fault) from the group of faulty scenarios independently. This split allows for training the isolation network only on data which is in fact faulty, which makes the problem clearer and should reduce the amount of false positives [92].
Third, it makes the system more modular. The detection network could be easily adjusted to account for more faults, if their effect on the input signal is different enough from the nominal case.

Therefore, the problem of fault detection and isolation with ANNs is approached by training one dedicated network for detection and one network for isolation.

Between the two approaches, the activation functions of the final layers differ. The reason for this is the different goals the networks have to fulfill. The output of the detection network should represent the likelihood of a fault according to the neural network. As such, the output is a single value between 0 and 1. A good activation function which can map the weighted input that the final neuron receives to this interval is the simple sigmoid function which was presented in Section 2.5.
As the output of the isolation networks should be a vector of probabilities, the sum of which is 1 (as the sum of the probability of all events must equal 1), a set of sigmoid neuron is no longer appropriate. Instead, the *softmax* function is used, which results in the desired outcome.
One downside of the *softmax* activation function is the tendency to amplify the largest component in a vector and diminish the rest. This is disadvantageous in the case where multiple faults need to be detected. An illustration of this effect can be seen in Figure 4.2. This figure shows the relative size of the input (in this case the vector [5, 0.5, 0.5, 0.5, 0.5, 2]) and the output when passed through the *softmax* function. As can be seen, the relative sizes of of the elements are exaggerated. The largest element becomes even larger and the rest becomes smaller.

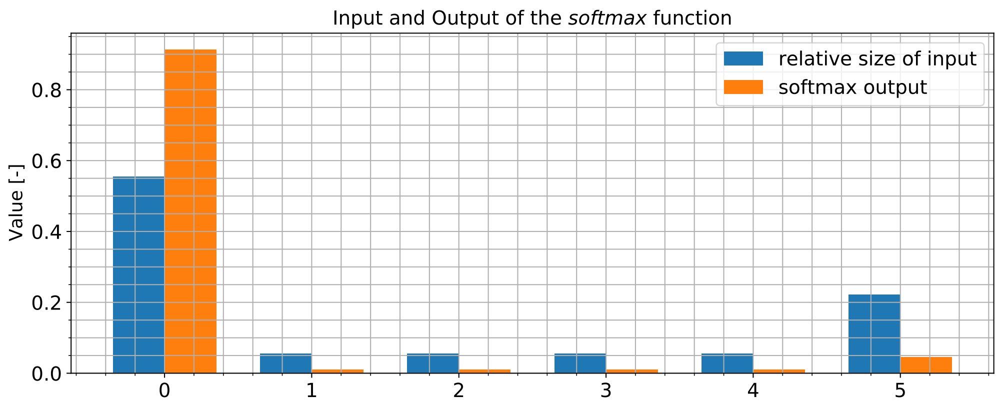


Figure 4.2: Illustration of the *softmax* activation function showing the impact on the relative size of the input.

Handling the Input Data The data that is available has to be processed before it is usable as input to a neural network. Furthermore, the question of what exact subset of the relative position and velocity data of the entire formation should be given to the network has to be answered. Three different methods for handling the relative position and velocity data are distinguished.
The first is to simply give the network all the relative position and velocity data that could be available on any given satellite. That is to say, a single network for all the satellites in the formation is trained, on a mixed dataset comprised of the data of all the satellites. This method will be called the 'naive' approach in this thesis, as no particular care is given to how the network might react to differing sets of data. While it is not expected that this will yield very good results, the capabilities of deep learning methods for learning complicated relationships between the input and output might still result in a network capable of isolation or detection.
The second approach is to separate the data from all the satellites and train a different network for each of the satellites in the formation. This comes with the disadvantage of having to train additional networks. However, it is also expected that the training time per network will be shorter, as the relationship between input and output is clearer, compared to the naive approach. This method will be called the 'individual' approach.
The final approach is to only train a single network for all the satellites, as in the naive approach, but transform the data in some way in order to achieve a more consistent relationship between the input and the FDI decision. This approach was ultimately discarded due to time constraints, as no easy transformation (e.g. rotation or swapping data indices) could be found that resulted in similar data patterns.

Therefore, multiple networks each are trained and evaluated with varying parameters. The distinction between the naive and individual approach is graphically illustrated in Figures 4.3 and 4.4.

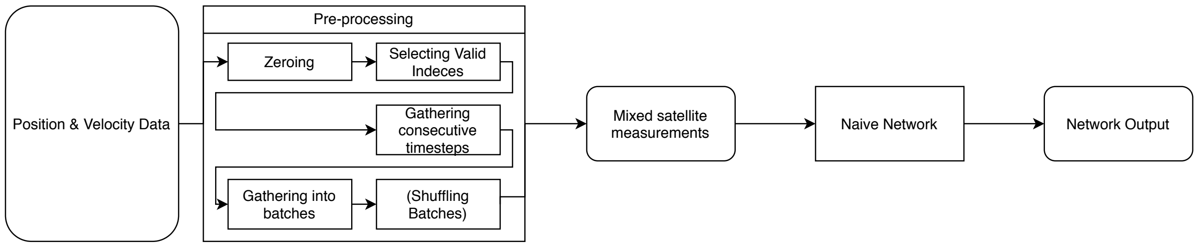


Figure 4.3: Illustration of overall architecture of naive networks.

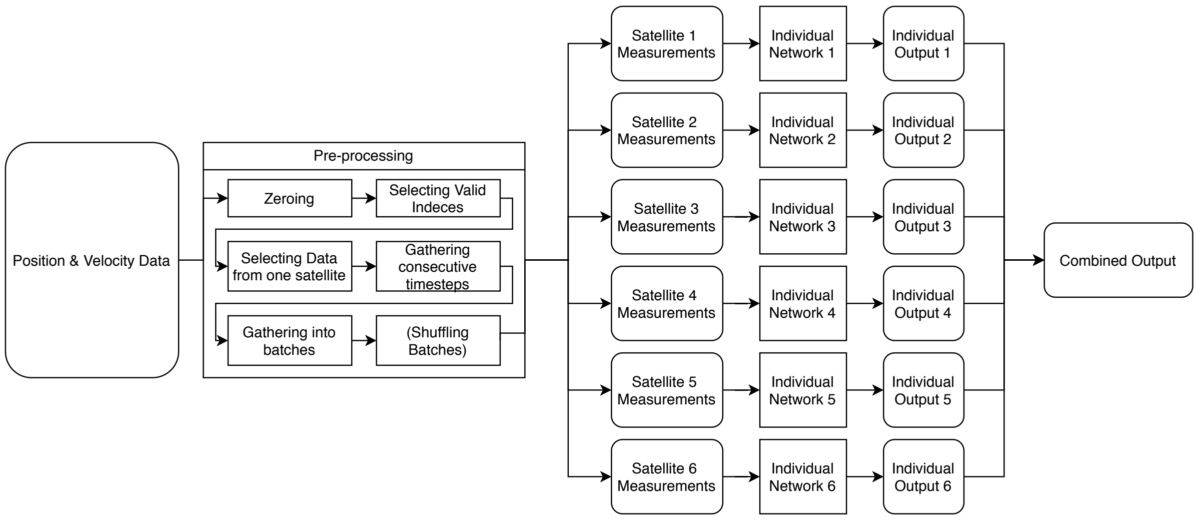


Figure 4.4: Illustration of overall architecture of individual networks.

One thing all LSTM networks have in common is the need to gather consecutive time steps to pass them to to the network. The reason for this is the training algorithm for recurrent neural networks, the Backpropagation through time (BPTT) algorithm. Similar to the regular backpropagation algorithm, the gradients are computed backwards starting from the finely layer. However, as the networks receive input not only from previous layers, but also from a previous time instant of themselves, the computation of the gradient not only goes backwards through the layers, but also backwards in time. Of course, this process cannot go on indefinitely. Therefore, a limit is set and the data of consecutive time steps is bundled together for easier computation. This process is further explained in Section 4.1.4. This results in an input to the network that is three-dimensional and has the shape (batch size, time steps, size of vector).

Combining the Network Output In order to reach a common diagnosis across the formation, the isolation network outputs (representing the probability of a particular fault) are shared across the formation. These fault probabilities are then combined using a simple weighted sum, resulting in a single consistent fault isolation across the formation. In order to reduce the communication requirements, these outputs are only shared after a fault is detected by any of the detection networks, which are running independently on the satellites. This process is illustrated inFigure 4.5. Of course, it is possible that a fault is not detected quickly enough, resulting in a break-down of the communication link before the local isolation results can be shared. In that case, the satellites have to rely on their local isolation network output to make a decision.

The relevant weights used to combine the local network output are different for each of the satellites and are based on *a priori* knowledge of each individual networks performance. In this particular case, the normalized confusion matrix is chosen as the method of choice. The confusion matrix is a natural description of a networks performance as it categorizes probabilities that a network prediction is accurate. With a small adjustment this can be used to combine the network output: The correct diagnoses of the network lie on the diagonal of the confusion matrix. This is where the output of the network matches with the correct fault location. All off-diagonal terms are incorrect isolations. In order to distinguish between correct and incorrect

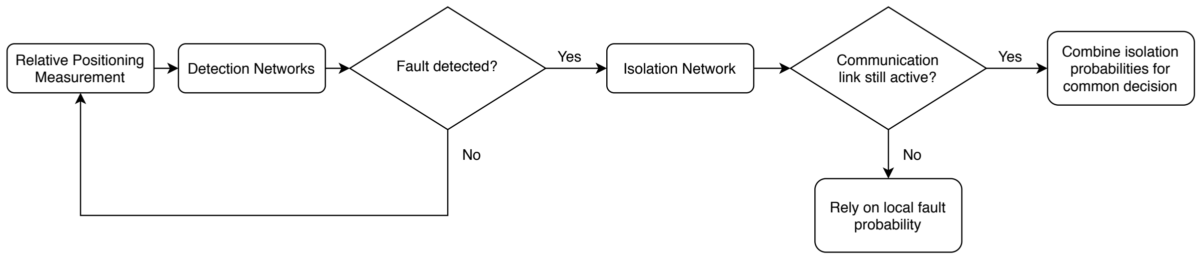

Relative Positioning Measurement
Detection Networks
Fault detected?
Yes
No
Isolation Network
Communication link still active?
Yes
No
Combine isolation probabilities for common decision
Rely on local fault probability

Figure 4.5: Illustration of the entire neural network based FDI system for the formation.

isolation, the off-diagonal terms are multiplied by -1. In the weighted sum, this means that a particular diagnosis of a network that has often been wrong in isolating that particular fault will be discouraged. This is shown in Equation (4.2).

$$\boldsymbol{p}_{formation} = \sum_{i=1}^{n} \boldsymbol{C}_i^* \cdot \boldsymbol{p}_i \tag{4.1}$$

$$C^*[l,m] = \begin{cases} 1 \cdot C[l,m] & \text{if } l = m \\ -1 \cdot C[l,m] & \text{if } l \neq m \end{cases} \tag{4.2}$$

Where $\boldsymbol{p}_{formation}$ is the probability vector for the entire formation, n is the number of satellites in the formation, $\boldsymbol{C}_i$ is the confusion matrix for the ith satellite, $\boldsymbol{C}^*$ is the adjusted confusion matrix where all off-diagonal terms have been multiplied by -1, and $\boldsymbol{p}_i$ is the probability vector for the occurrence of faults, i.e. the network output of the ith satellite.
Combining the output of the networks in this way results in a unified isolation across the formation and compensates for shortcomings in the individual networks. For example, if satellite 1 always misdiagnoses faults in satellite 5, its output should not be taken into account for a general diagnosis. This is achieved by this method.

Summary In total, four different kinds of neural networks are trained in this thesis. All of them have the same basic structure of three hidden layers, two of them made up of LSTM units and one regular densely connected layer.
Two distinctions are made. The first is a matter of which output is desired of the network. Two of the network types will be used for fault *detection* purposes only, while the other two are dedicated to fault *isolation*. The detection networks use a simple sigmoid as the final layer activation, while the isolation networks use the *softmax* function.
The second distinction is made according to what input the networks are trained on. In practice, all the networks will receive the relative measurement data from the satellite they run on, but the *naive* networks will be trained on input data from all satellites. On the other hand, the *individual* networks are trained on only the data belonging to a single satellite, resulting in 6 different networks.

4.1.2. Selecting Neural Network FDI Parameters

Various so called hyper-parameters affect the performance of a network. These parameters include the number of neuron-layers, the amount of neurons per layer, the type of neurons in a layer, etc. There is no general method for choosing the hyper-parameters of the network, and while certain approaches for hyper-parameter optimization exist [93], these are not feasible with the computational resources that were available. As such, the hyper parameters were selected by trial and error. In general, the two networks for detection and isolation differ only in the amount of units per layer, rather than the amount or type of layers. The reasons for the increased amount of units per layer in the isolation networks is the hope of better performance considering the increased complexity of the problem, compared to the detection case.
The amount of units per layer is summarized in Table 4.1. For most of the networks tested, the amount of neurons per network is a power of two. This is due to a supposed increase in performance, as the processors can make use of some internal optimizations, as they most often handle powers of two. However, as can be seen for the isolation networks, the amount of neurons in the dense layer is not a power of two and this did not severely affect the training speed or performance.

Table 4.1: Amount of units and layers per network approach.

Networks	Naive Detection	Naive Isolation	Individual Detection	Individual Isolation
Layers & Units	[128, 128],[64]	[256, 256],[100]	[128, 128],[64]	[256, 356],[100]

Furthermore, the amount of time-steps that are gathered into one input for the LSTM networks was chosen to be 50, with a sampling period of 1 s. This is a trade-off, as more data can mean better performance for the network, but also means longer training times as well as a need to store more measurements in the on-board computer (OBC). In the end, the amount was chosen to be slightly less than the time between thruster firings, which is 60 s.

4.1.3. Training Environment

Due to the amount of computations involved in training a neural network of considerable size, the local computing resources are not sufficient to complete the training in a reasonable amount of time. However, the online platform "Google Colab" is suitable for performing the training, for various reason [94]. First, the platform is integrated with Google's "Drive" cloud data storage system, making it easy to access the training data as soon as it is on the "Drive". Secondly, Colab offers access to hardware accelerators such as GPU and the specifically for networks designed TPUs (tensor-processung unit). The interface of Colab allows running Jupyter notebooks on a virtual machine on the Google servers, which can run tensorflow, Google's library for training and evaluating neural networks.
The training of the neural networks is performed with the Keras API of the Tensorflow package, due to its ease of use. This allows to easily set up and customize the chosen network architectures.

The training of the network requires a data pipeline in order to efficiently load the training data into the virtual machine running the training program. Due to the integration of Colab with the "Drive" cloud service, all the training data was uploaded to the Google servers and then loaded from there onto the virtual machine. A special tensorflow class, called `Dataset`, can be used to programmatically load, process and integrate the data into the training process. The construction of the dataset was the same for all networks which were trained. First, a list of all training files was generated. This was used as an input to an initial `Dataset` object, only containing the file paths. This list of files then was decoded from its binary format into usable data. Then, a user-made pre-processing function was used to slice the data correctly (the pre-preocessing procedure is further explained in Section 4.1.4). The result is the input to the network. The dataset containing these inputs could then be batched, shuffled and pre-loaded using functions from the tensorflow library.

Furthermore, the use of distributed training was attempted but ultimately discarded. The reason for this was mainly the inability of the distributed program to handle the data pipeline from Google drive, as the services were not intended to load data from Drive. Switching to a different service (Google cloud storage (GCS) buckets) would have fixed this problem, however it is a paid service intended for use by businesses, which made it infeasible for use in this project at the time.

One particular training approach which has proven to be very useful in certain areas of application is "Reinforcement Learning", the driving force behind impressive endeavours, including Google's Alphastar project [29]. While the results speak for themselves, it is important to understand the limitations of this approach. Reinforcement learning is only applicable in situations where the best output of the network at a given time is either unknowable or too difficult to compute. An example is determining the best move in the board-game "Go" [28]. There is no feasible way to define a cost function which only takes into account the current state of the game due to the sheer amount of possible moves. However, the network output can be judged on whether the resulting move causes the network to eventually lose the match. Another example is teaching new skills to a robot which are difficult to model analytically, where the result however can easily be evaluated, such as flipping a pancake [95].
Transferring this analogy to the field of FDI, the use of Reinforcement Learning for detection and isolation is not beneficial, as the correct output of the network (the state of the system) is known when creating the training dataset. However, if the step after isolation, 'reconfiguration', is taken into account, the network does have an impact on the system, which is not easily assessable. In such a case, reinforcement learning techniques could be investigated for FDIR.

Instead, a standard stochastic gradient-descent algorithm was used for the training, the so called Adaptive

Moment Estimation (Adam)-optimizer [96]. A learning rate of 0.001 was chosen, as it produced good results during the training. The cost function that was selected for the training was the categorical cross-entropy function, a generalized version of the cost function described in Section 2.5. The reason for choosing this function is the improved performance in training compared to the simple quadratic cost function [65, p.65]. Furthermore, it is particularly suited for classification tasks (especially when there are multiple classes) and is the primary choice when using an output layer with the softmax activation function, as minimising it corresponds to the maximum-likelihood estimation between two distributions. [30, p.193].
The cut-off point was selected to be 3 epochs of no improvement on the validation cost function. It was seen often during the training that due to the randomness of the stochastic gradient descent, the performance on the validation set decreased for one or two epochs, only to increase again above its previous best performance. In principle it is also possible that the performance on the validation set decreases for longer, only to increase again afterwards. However, due to the long training times, 3 epochs was chosen as it presents a good trade-off between the amount of time wasted (as no there is no performance gain for the last 3 epochs) and the potential gain in performance if the network starts to perform better again.

There are four different types of network that need to be trained: naive detection, individual detection, naive isolation and individual isolation. The training of these networks is split over four "Colab Notebooks", i.e. the Jupyter Notebooks that are run on the Google servers. There are four different pre-processing functions, as each type of network requires a different dataset. An overview of the entire workflow involving the Google Colab platform can be seen in Figure 4.6.

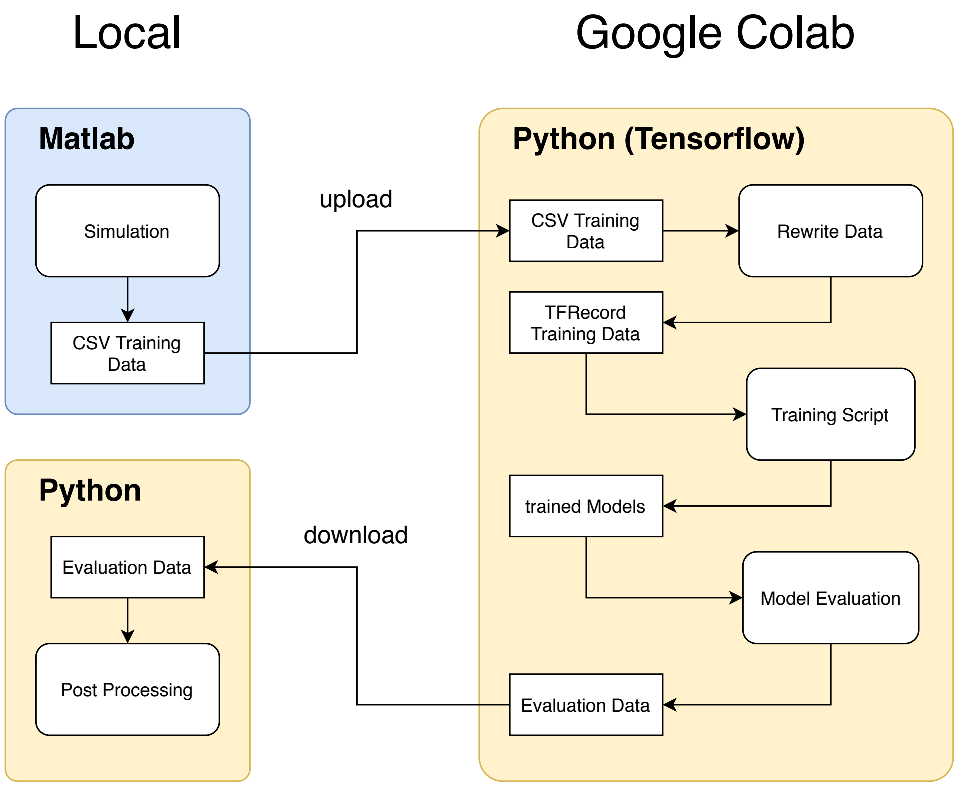


Figure 4.6: Interaction within and Interface of Google Colab.

In the figure, it can be seen that the data from the simulation is uploaded and stored in the Google Drive. From there it is re-written to the appropriate data format that will ease handling and speed up the training. In the program this TFRecord data is loaded and decoded into usable data, as well as pre-processed, according to the procedure described in Section 4.1.4.
A separate script was used for evaluating the trained models. The neural network output was then downloaded again for further analysis in Python.

4.1.4. Processing the Simulation Output

The output of the simulation data needs to be pre-processed before it can be used as the input for the neural networks. The process of pre-processing the simulation output is illustrated in Figure 4.7. As can be seen in the figure, the data is first organized into individual slices: successive relative position measurements in time. The relative measurements which are vectors of size 4(amount of neighbors) x 6(position and velocity) = 24 are then gathered into a two-dimensional array of size 24 x n where n is the amount of time steps.

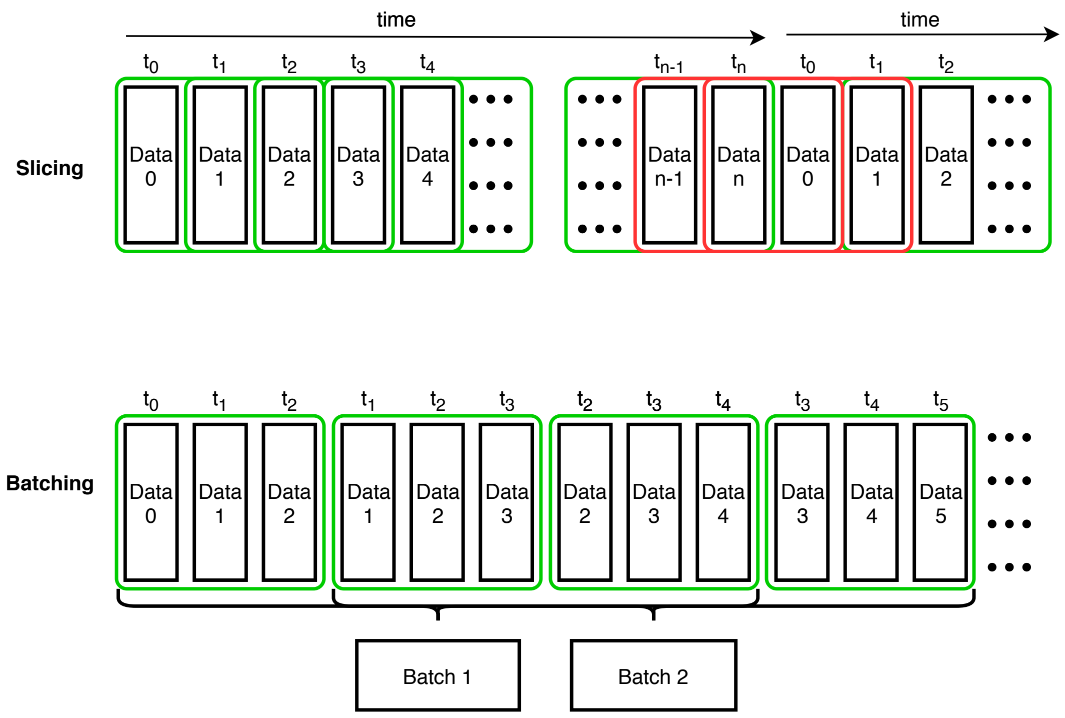


Figure 4.7: Illustration of the data pre-processing: slicing and batching; red slices are invalid and discarded.

Not all slices in the data generated this way are valid. Some would contain data from two different satellite perspectives or different simulation runs. In order to avoid such slices, the following condition is evaluated on the index to be sliced: The time at the start of the index i, i.e. $t[i]$, must be less than the time at the index at the end of the slice, i.e. $t[i+n]$. In the case of the individual network, an additional condition needs to be met and that is that the data comes from a single satellite source.
Additionally, the pre-processing function assigns a label to each slice, representing the correct detection or isolation output. In the case of the detection, where the output is only a single number, a faultless slice receives a label of '0', while a slice containing data from a fault receives a '1'. The isolation case is slightly different as the output is a vector rather than a single value. As such, the corresponding label is a vector of 35 zeroes with a single '1' in the location of the fault (1-6 corresponding to satellite 1, 7-12 to satellite 2, etc.). This can easily be done by using the `tf.one_hot()` function, which creates a `tensor` of such vectors from a `tensor` of integers. The slices are then gathered into batches for training purposes. A batch size of 4096 was selected after some trial and error as a good amount.

During the training it was found that going over the entire dataset was too time-consuming. Even with the use of hardware accelerators such as the available GPUs or TPUs, the time for a single epoch over the dataset exceeded 10 hours, which made it completely infeasible. Attempts were made to speed up training by pre-loading the data onto the local drive of the virtual machine, but this did not increase the training speed.
The second attempt involved rewriting the training data from CSV text files to the custom tensorflow binary data format called "TFRecord". This format not only reduces the disk space required (by approximately a factor of three), but also increased training speed for three reasons. First, the decreased file-size reduces the amount of data that needs to be downloaded to the virtual machine working memory. Secondly, the TFRecord format gives tensorflow the opportunity to exclusively use Tensorflow based functions, which utilize C++ and C code for optimization as opposed to needing to use NumPy functions for pre-processing the data.

Finally, pre-fetching data in the training process works best with files that are on the order of hundreds of MB large, as this reduces the amount of times a request for downloading the file has to be issued, reducing the overhead associated with downloading the files. The simulation runs are gathered into groups of 100 files for the detection and 200 files for the isolation, which makes the resulting TFRecord files large enough to see a difference in this respect.

While this sped up the training, it still amounted to very large training times. The networks were therefore only trained on a subset of the generated data. As the relation between the faulty input data and the fault diagnosis should be clearest for the most intense faults the network should learn this relationship the quickest with high intensity fault data. However, the networks should also learn to detect or isolate the least intense faults. As such, they need to be represented in the dataset as well. Therefore, the dataset was structured around the most and least intense faults. Instead of an even distribution, 40% of the training set were taken from the intensity range [0.8,1], another 40% was taken from the range [0.1,0.3], the lower end of the intensity range. The remaining 20% were spread over the interval [0.3,0.8]. The reduced training dataset was gathered by randomly selecting data from the full set, to match the above fault intensity distribution. It should be noted that this does not affect the validation data, which still contains a uniform distribution of the fault intensity.

Furthermore, a subset of the data has to be reserved for validation purposes, i.e. to judge when the training of the networks transitions from improving to overfitting. A 90/10 split was chosen, i.e. the networks will be trained on 90% of the data, while 10% of the data is reserved for validation purposes.

4.1.5. Converting Continuous Output to Discrete Decision

Another aspect that needs to be addressed is the process of turning the output of the networks (a single continuous number in [0,1] in the detection case or a vector of such numbers in the isolation case) into a decision on the classification. This output can be interpreted as the networks confidence in a particular class, seen as a probability. In the detection case this is the probability of the occurrence of any fault, whereas the isolation is a vector of probabilities, whose component sum is 1.
During the network training, the value used to calculate the accuracy and therefore the value of the loss function in the detection case is 0.5 or 50 %, the center of the range of possible values [0,1]. However, this might not be the optimal threshold as the performance could potentially be increased with a higher or lower threshold. The question then becomes how to judge the performance of the classifier. Two common performance characteristics in the field of machine learning are the *precision* (the proportion of correct positive predictions out of all positive predictions) and the *recall* (the proportion of correct positive predictions out of all occurrences of the positive case). This distinction is also graphically illustrated in Figure 4.8[5]. The former gives a measure of how often a positive occurrence is classified, while the latter gives a measure of how often a positive prediction of the network is actually correct.

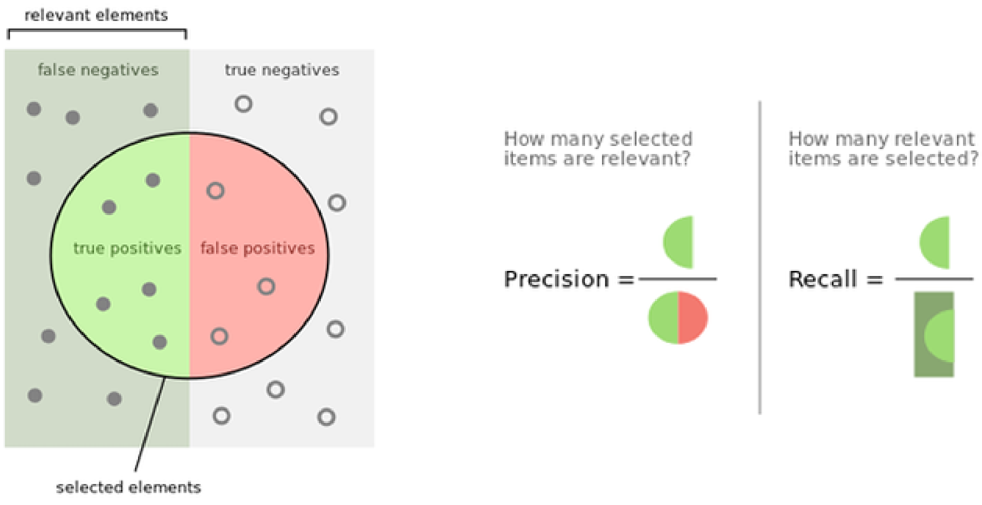


Figure 4.8: Graphical illustration of the definitions of recall and precision [5].

A commonly used performance measure is the F_β-measure, which is the weighted harmonic mean of the precision and recall, shown in Equation (4.3).

$$F_\beta = (1+\beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision}} \tag{4.3}$$

This measure can be used to judge the overall performance of a classifier and will be used to determine the threshold.
In the case of fault detection, a lower threshold will mean more faults get detected (higher recall), however the amount of false positives will also go up (lower precision). As the predictive value and therefore the precision of an fault detection system is important, the F measure should be weighted in favor of precision. In this case, a weighting factor $\beta = \frac{1}{2}$ was chosen, i.e. precision is weighed twice as important as recall. This $F_{\frac{1}{2}}$ measure will be optimized over the range of possible thresholds.
In the isolation case, this approach does not work as there are many more available classes. For isolation, simply the maximum element of the output vector (i.e. the option with the highest confidence) is chosen.

4.1.6. Training Verification

In this section the `Dataset` structure and data pipeline is verified. As it is assumed that the tensorflow `fit()` works correctly when providing the right input, i.e. a network and a properly labelled dataset, the training algorithm itself is considered verified when the data is constructed correctly. This will be shown in this section.
The first part that is verified is the generation of the TFRecord files from the CSV data output of the simulation. Secondly, the pre-processing of the TFRecord files into `dataset` objects such that they can be used for the detection and isolation networks is checked, which require different labels and structure. Finally, the selection of only the subset of data belonging to a single satellite viewpoint is verified.
The relevant code sections for all procedures described here can be found in Appendix B.1.

Writing and Decoding the TFRecord Files The TFRecord functions are tested by comparing the data from the original CSV file against the decoded content of the written TFRecord file. A separate Jupyter notebook was used to perform this operation. By comparing the numerical values it was found that there were small differences, on the order of 1e-9, which are considered acceptable for training purposes as they are far below the noise floor.

Network Model Generation The neural network generation function can be easily checked by using the function `tf.keras.utils.plot_model()`, that generates a figure of the network model. Such a graphic for the individual detection network can be seen in Figure 4.9.

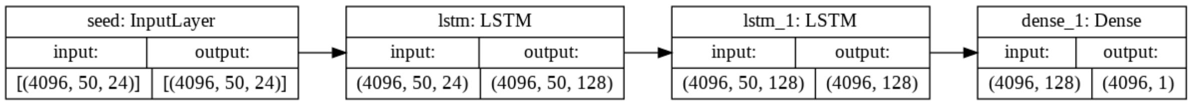


Figure 4.9: Illustration of individual detection network model.

As can be seen, the input of shape (batch size, time steps, data size) is correctly passed through two successive layers of LSTM units, followed by a dense layer which condenses the output into the shape (batch size, 1).

Pre-processing the Detection and Isolation files The pre-processing output was checked manually to see if the correct shape was achieved. A shape of (time steps,data size) is expected, in this case (50.24) with either a label of size 1 or a vector of size (36,1), which is exactly the shape that is provided by the pre-processing function.
Another test to verify the pre-processing is performed by comparing the output of the pre-processing function against manual slices of the original data. As the original data contain slices which are not valid (as they go from one simulation run into the next), there should be a specific amount of discrepancies in between the pre-processing output and the manual slices, i.e. there should be more slices in the unprocessed data. For a single simulation run, a total number of $5 \cdot 50 = 250$ discrepancies are expected. In the verification procedure, these discrepancies were counted for the file of a single simulation run and 250 were found at the correct indices (close to multiples of the simulation run-time).
Furthermore, the fault labels were also compared, and no discrepancies were found for the correct slices. This verifies the data pre-processing.

Selecting data only from a single satellite The selection of only data from a single satellite viewpoint is achieved by comparing the decoded data against the output of the *individual* pre-processing function. If only the data of the selected satellite remains after processing, this functionality is verified.
After comparing the output, similar to the procedure for verifying the pre-processing in general, it was found that the only output that remained was from the selected satellite. The only difference to the above procedure is that with only a single satellite, only 50 incorrect slices are expected in the comparison, which is exactly what was found, regardless of the satellite selected.
This verifies this part of the pre-processing.

4.1.7. Network Evaluation

The network evaluation is performed on the complete validation dataset that was set out at the generation of the simulation data. Due to the size of this dataset, the evaluation was performed in two steps. First, the data is given to the network and the output of the network is saved, and then the data is analyzed.
Due to the need to be able analyze the performance of the network with respect to the various parameters of the dataset (fault location, fault type, fault time, fault intensity), each simulation file was separately written into a TFRecord file. During the evaluation, a `dataset` object was constructed from a single file and the network tested on the resulting dataset. The output of the network for each time-step was then written to a CSV file, together with identifying information of the file for debugging purposes, as well as the fault parameters and the correct output.
These CSV files (one for each network) could then be downloaded and loaded into a `pandas` dataset object in Python. This then allowed for offline analysis.

4.2. Kalman Filter approach

One standard method for performing FDI is through the use of an observer or a filter, for a deterministic or stochastic system, respectively [97]. The Kalman filter specifically is a fairly simple method that can still yield good results for linear systems. As the non-linearities in the relative dynamics of satellites depend on their distance and the formation considered here is very close, a standard Kalman filter was chosen due to its ease of implementation. First, the combined system model that is used to capture the dynamics of the formation is shown in Section 4.2.1. The detection and isolation procedure for the Kalman filter is presented in Section 4.2.2. Finally, the selection of two tunable parameters of the Kalman filter is presented in Section 4.2.3.

4.2.1. Combined Formation Model

The general structure of the centralized Kalman filter is shown in Figure 4.10. The relative position and velocity measurements that are taken by each satellite in the formation are transformed into the Hill-Frame and then sent to the diagnosing satellite, in this case satellite 1. The gathered data is then used in the Kalman filter to generate the residual vector, which can then be analyzed statistically in order to detect changes in the distribution of the residual vector. As the residual is distributed in a bell-curve, changes in the residual distribution would indicate a fault. This decision can then be broadcast back to the formation for reconfiguration purposes, but that is beyond the scope of this thesis.

In order to have a centralized Kalman filter, a state space model of the entire formation is needed. It is assumed that the satellites are close enough, such that the differences in the state matrix $\boldsymbol{A}$ for each of the satellite connections are negligible. As each of the relative distances have no interconnection, the use of a block-diagonal state space matrix, as shown in Equation (4.4), is possible.

$$\boldsymbol{A}_{formation} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \cdots \\ \boldsymbol{0}_{6x6} & \boldsymbol{A} & \boldsymbol{0}_{6x6} & \cdots \\ \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{A} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{4.4}$$

There are only 12 unique edges in an octahedron and therefore only 12 unique connections in the formation, and each connection is characterized by a relative position and velocity vector. Therefore, the state vector of the entire formation will have size 12 x (3 + 3) = 72. More specifically, the states are arranged as seen in Equation (4.5).

$$\boldsymbol{x}_{formation} = \begin{bmatrix} \boldsymbol{x}_{12}^T & \boldsymbol{x}_{13}^T & \boldsymbol{x}_{14}^T & \boldsymbol{x}_{15}^T & \boldsymbol{x}_{23}^T & \boldsymbol{x}_{34}^T & \boldsymbol{x}_{45}^T & \boldsymbol{x}_{52}^T & \boldsymbol{x}_{65}^T & \boldsymbol{x}_{64}^T & \boldsymbol{x}_{63}^T & \boldsymbol{x}_{62}^T \end{bmatrix}^T \tag{4.5}$$

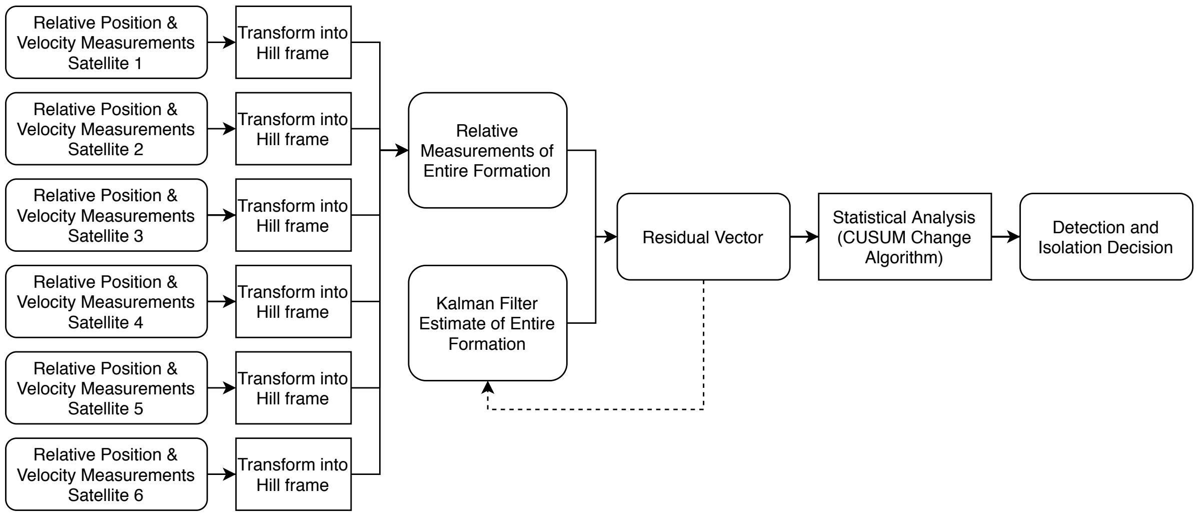

Relative Position & Velocity Measurements Satellite 1
Relative Position & Velocity Measurements Satellite 2
Relative Position & Velocity Measurements Satellite 3
Relative Position & Velocity Measurements Satellite 4
Relative Position & Velocity Measurements Satellite 5
Relative Position & Velocity Measurements Satellite 6
Transform into Hill frame
Relative Measurements of Entire Formation
Kalman Filter Estimate of Entire Formation
Residual Vector
Statistical Analysis (CUSUM Change Algorithm)
Detection and Isolation Decision

Figure 4.10: Flow-chart of centralized Kalman filter FDI scheme.

Where $\boldsymbol{x}_{formation}$ is the state of the combined formation model and $\boldsymbol{x}_{ij}$ is the state vector (i.e. relative position and velocity in the Hill frame) of the connection from satellite i to satellite j. This ordering is illustrated in Figure 4.11.

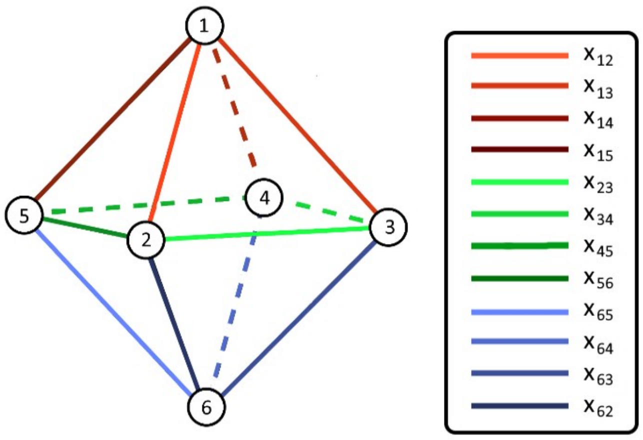


Figure 4.11: Illustration of formation state ordering.

The specific ordering is arbitrary as it does not change the underlying dynamics. This particular ordering of the relative distances flows from satellite 1 to satellite 6, first covering all the connections of satellite 1 (shown in shades of red), then all the connections not concerning satellite 1 or 6 (shown in shades of green) and then all connections of satellite 6 (shown in shades of blue).

Similarly, the formation control input $\boldsymbol{u}_{formation}$ is the vertically stacked control input of every satellite in the formation as seen in Equation (4.6).

$$\boldsymbol{u}_{formation} = \begin{bmatrix} \boldsymbol{u}_1^T & \boldsymbol{u}_2^T & \boldsymbol{u}_3^T & \boldsymbol{u}_4^T & \boldsymbol{u}_5^T & \boldsymbol{u}_6^T \end{bmatrix} \tag{4.6}$$

The corresponding input matrix can be seen in Equation (4.7), resulting in a 72x18 matrix.

$$\boldsymbol{B}_{formation} = \begin{bmatrix} -\boldsymbol{B} & \boldsymbol{B} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} \\ -\boldsymbol{B} & \boldsymbol{0}_{6x6} & \boldsymbol{B} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} \\ -\boldsymbol{B} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{B} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} \\ -\boldsymbol{B} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{B} & \boldsymbol{0}_{6x6} \\ \boldsymbol{0}_{6x6} & -\boldsymbol{B} & \boldsymbol{B} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} \\ \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & -\boldsymbol{B} & \boldsymbol{B} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} \\ \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & -\boldsymbol{B} & \boldsymbol{B} & \boldsymbol{0}_{6x6} \\ \boldsymbol{0}_{6x6} & \boldsymbol{B} & \boldsymbol{0} & \boldsymbol{0}_{6x6} & -\boldsymbol{B} & \boldsymbol{0}_{6x6} \\ \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{B} & -\boldsymbol{B} \\ \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{B} & \boldsymbol{0}_{6x6} & -\boldsymbol{B} \\ \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{B} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & -\boldsymbol{B} \\ \boldsymbol{0}_{6x6} & \boldsymbol{B} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & \boldsymbol{0}_{6x6} & -\boldsymbol{B} \end{bmatrix} \tag{4.7}$$

As each satellite involved in a connection has an influence via the control forces on the relative state, the $\boldsymbol{B}$ matrix appears twice for each connection. For example, the connection $\boldsymbol{x}_{12}$ involves both the control input from satellite 1 and 2, and such both these position have a copy of the B matrix. The negative signs can be deduced from the Clohessy-Wiltshire equations.
While the formation state is captured accurately by the 12 unique connections, there are 4x6 = 24 measurements available, i.e. each connection is measured twice. The connections are arranged in the ordering shown in Equation (4.10).

$$\boldsymbol{y}_{formation} = \begin{bmatrix} \boldsymbol{y}_1^T & \boldsymbol{y}_2^T \end{bmatrix}^T \tag{4.8}$$

$$\boldsymbol{y}_1 = \begin{bmatrix} \boldsymbol{x}_{12}^T & \boldsymbol{x}_{13}^T & \boldsymbol{x}_{14}^T & \boldsymbol{x}_{15}^T & \boldsymbol{x}_{23}^T & \boldsymbol{x}_{34}^T & \boldsymbol{x}_{45}^T & \boldsymbol{x}_{52}^T & \boldsymbol{x}_{65}^T & \boldsymbol{x}_{64}^T & \boldsymbol{x}_{63}^T & \boldsymbol{x}_{62}^T \end{bmatrix}^T \tag{4.9}$$

$$\boldsymbol{y}_2 = \begin{bmatrix} \boldsymbol{x}_{21}^T & \boldsymbol{x}_{31}^T & \boldsymbol{x}_{41}^T & \boldsymbol{x}_{51}^T & \boldsymbol{x}_{32}^T & \boldsymbol{x}_{43}^T & \boldsymbol{x}_{54}^T & \boldsymbol{x}_{25}^T & \boldsymbol{x}_{56}^T & \boldsymbol{x}_{46}^T & \boldsymbol{x}_{36}^T & \boldsymbol{x}_{26}^T \end{bmatrix}^T \tag{4.10}$$

$$\boldsymbol{y}_2 = \begin{bmatrix} -\boldsymbol{x}_{12}^T & -\boldsymbol{x}_{13}^T & -\boldsymbol{x}_{14}^T & -\boldsymbol{x}_{15}^T & -\boldsymbol{x}_{23}^T & -\boldsymbol{x}_{34}^T & -\boldsymbol{x}_{45}^T & -\boldsymbol{x}_{52}^T & -\boldsymbol{x}_{65}^T & -\boldsymbol{x}_{64}^T & -\boldsymbol{x}_{63}^T & -\boldsymbol{x}_{62}^T \end{bmatrix}^T$$

As $\boldsymbol{x}_{ij} = -\boldsymbol{x}_{ji}$. This results in the measurement matrix shown in Equation (4.11).

$$\boldsymbol{C}_{formation} = \begin{bmatrix} \boldsymbol{I}_{72x72} \\ -\boldsymbol{I}_{72x72} \end{bmatrix} \tag{4.11}$$

4.2.2. Detection and Isolation

The standard Kalman filter can be used as a method to generate fault residuals by using the innovation vector as described in the equations in Section 2.6, which are repeated here for ease of reading.

$$\hat{\boldsymbol{x}}^-(k) = \boldsymbol{A}\hat{\boldsymbol{x}}^+(k-1) + \boldsymbol{B}\boldsymbol{u}(k)$$

$$\boldsymbol{P}^-(k) = \boldsymbol{A}\boldsymbol{P}^+(k-1)\boldsymbol{A}^T + \boldsymbol{Q}$$

$$\hat{\boldsymbol{x}}^+(k) = \hat{\boldsymbol{x}}^-(k) + \boldsymbol{K}\left[\boldsymbol{y}(k) - \boldsymbol{C}\hat{\boldsymbol{x}}^-(k) - \boldsymbol{D}\boldsymbol{u}(k)\right]$$

$$\boldsymbol{r}(k) = \boldsymbol{y}(k) - \boldsymbol{C}\hat{\boldsymbol{x}}^-(k) - \boldsymbol{D}\boldsymbol{u}(k)$$

$$\boldsymbol{P}^+(k) = [\boldsymbol{I} - \boldsymbol{K}\boldsymbol{C}]\boldsymbol{P}^-(k)[\boldsymbol{I} - \boldsymbol{K}\boldsymbol{C}]^T + \boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^T$$

The resulting innovation vector $\boldsymbol{r}$ can be used as a fault residual since in the ideal case the innovation is a randomly distributed vector with mean $\boldsymbol{0}$ and covariance $\boldsymbol{Q} = \boldsymbol{C}\boldsymbol{P}\boldsymbol{C}^T + \boldsymbol{R}$. A change in the mean of this distribution would indicate a fault.
In order to detect if a change in the mean of the the residual vector has occurred, a log-likelihood ratio test

is implemented. This test aims to detect if a vector quantity is sampled from one particular distribution or another. In the scalar case, this test is given by Equation (4.12).

$$s(z) = \ln \frac{p_{\theta 1}(z)}{p_{\theta 0}(z)} \tag{4.12}$$

Where s is the so called log-likelihood ratio, z is a particular observation from a certain distribution and p_θ is the probability of that particular observation occurring under the distribution θ. If more information about the distribution is known, this expression can be simplified. For example. if the two distributions θ_0 and θ_1 are both normal distributions with means μ_0 and μ_1, respectively, the log-likelihood ratio can be simplified to equation Equation (4.16)

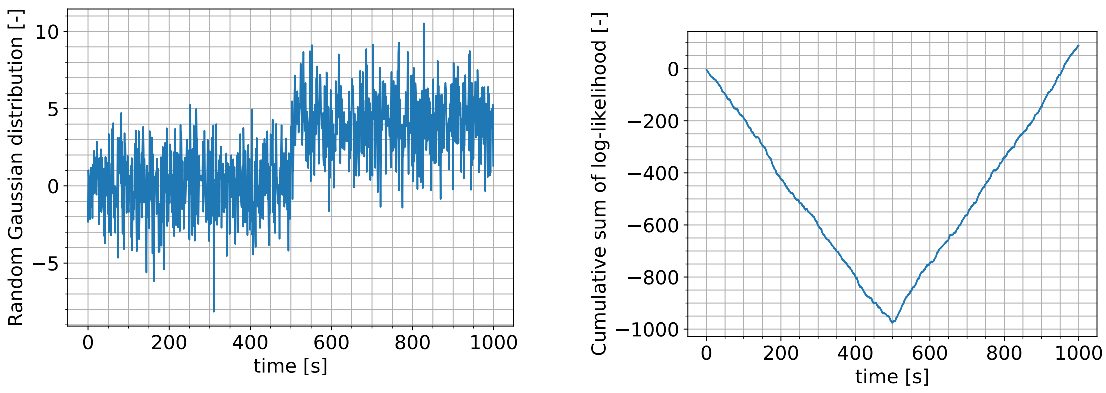
$$p_\mu(z) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right) \tag{4.13}$$

$$s(z) = \ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(z-\mu_1)^2}{2\sigma^2}\right)}{\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(z-\mu_0)^2}{2\sigma^2}\right)}\right) \tag{4.14}$$

$$s(z) = \ln\left(\exp\left(-\frac{(z-\mu_1)^2}{2\sigma^2} + \frac{(z-\mu_0)^2}{2\sigma^2}\right)\right) \tag{4.15}$$

$$s(z) = \frac{(z-\mu_0)^2 - (z-\mu_1)^2}{2\sigma^2} \tag{4.16}$$

One algorithm that uses this ratio to detect a change is the so called cumulative sum (CUSUM) algorithm [8, chp.7].



(a) Observations drawn from normal distribution; a change in the mean occurs at $t = 500$.

(b) Cumulative sum of the log-likelihood.

Figure 4.12: Illustration of the CUSUM method: observations drawn from normal distributions and the cumulative sum of the log-likelihood test.

An illustration of this method is seen in Figure 4.12 (reproduced from [8, chp.7]). The figure on the left shows a series of observations, first drawn from a normal distribution with mean $\mu_0 = 0$ and standard deviation $\sigma = 2$, but after 500 s the distribution shifts to a new mean of $\mu_1 = 4$. The right figure shows the cumulative sum of the log-likelihood computed from Equation (4.16). As can be seen, the cumulative sum shows as distinctive change after the distribution changes. The slope of the cumulative sum changes from negative to positive.

A decision function $g(k)$ can then be established and recursively computed by Equation (4.17).

$$g(k) = \max\left(0, g(k-1) + s(z(k))\right) \tag{4.17}$$

If this decision function exceeds a chosen threshold h_k, a detection is triggered.

In the vector case, where the probability of a certain observation vector $\boldsymbol{z}$ is drawn from a normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{Q}$ is given by.

$$p_{\boldsymbol{\mu}}(\boldsymbol{z}) = \frac{1}{\sqrt{(2\pi)^n \det \boldsymbol{Q}}} \exp\left(-\frac{1}{2}(\boldsymbol{z}-\boldsymbol{\mu})^T \boldsymbol{Q}^{-1}(\boldsymbol{z}-\boldsymbol{\mu})\right) \tag{4.18}$$

Again, assuming only a change of the mean in the two distributions p_0 and p_1, and inserting Equation (4.18) into the expression for the log-likelihood shown in Equation (4.12), Equation (4.19) is the result after a small amount of algebra [8, p.297].

$$s(\boldsymbol{z}) = \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0\right)^T \boldsymbol{Q}^{-1} \left(\boldsymbol{z} - \frac{1}{2}\left(\boldsymbol{\mu}_0 + \boldsymbol{\mu}_1\right)\right) \tag{4.19}$$

Determining the Change in Mean In the case of a deterministic fault inserted into a linear system, the residual will experience a change in mean, but not in the the covariance [8, p.327]. As such, for a known change in the mean, Equation (4.19) can be used to detect a fault. In order to determine the change of the mean for particular faults, the transfer function between a fault signal and the residual needs to be determined.
For a general system, the transfer function between the input of a discrete state space system and the output is given in Equation (4.23).

$$zX(z) = \boldsymbol{A}X(z) + \boldsymbol{B}U(z) \tag{4.20}$$

$$Y(z) = \boldsymbol{C}X(z) + \boldsymbol{D}U(z) \tag{4.21}$$

$$\rightarrow Y(z) = \left[\boldsymbol{C}((z\boldsymbol{I}-\boldsymbol{A})^{-1} + \boldsymbol{B}) + \boldsymbol{D}\right] U(z) \tag{4.22}$$

$$\rightarrow \boldsymbol{V}_{yu}(z) = \boldsymbol{C}((z\boldsymbol{I}-\boldsymbol{A})^{-1} + \boldsymbol{B}) + \boldsymbol{D} \tag{4.23}$$

Combining the Kalman filter equations into a recursive a priori time-update form [63, p.74], using the residual as the measurement equation, the following system is acquired.

$$\hat{\boldsymbol{x}}^-(k+1) = [\boldsymbol{A} - \boldsymbol{AKC}]\,\hat{\boldsymbol{x}}^-(k) + [\boldsymbol{B} - \boldsymbol{AKD}]\,\boldsymbol{u}(k) + \boldsymbol{AKy}(k) \tag{4.24}$$

$$\boldsymbol{r} = -\boldsymbol{Cx}^-(k) - \boldsymbol{Du}(k) + \boldsymbol{y}(k) \tag{4.25}$$

As the faults considered have the same impact on the residual as the control input (they act on the system like a control force from a thruster would), the transfer function between the control input $\boldsymbol{u}$ and the residual is the same as the fault signal $\boldsymbol{f}$ and the residual.
Comparing Equation (4.23) and Equations (4.24) and (4.25), the transfer function between the input and the residual can be deduced to be Equation (4.26).
The steady state change in the mean can then be determined from Equation (4.27), for a steady state value of the fault signal $\bar{\boldsymbol{f}}$.

$$\boldsymbol{V}_{ru}(z) = -\boldsymbol{C}(z\boldsymbol{I} - \boldsymbol{A} + \boldsymbol{AKC})^{-1}(\boldsymbol{B} - \boldsymbol{AKD})) - \boldsymbol{D} \tag{4.26}$$

$$\boldsymbol{\mu}_f = \boldsymbol{V}_{rf}(1)\cdot\bar{\boldsymbol{f}} = \left[-\boldsymbol{C}\cdot(\boldsymbol{I} - \boldsymbol{A} + \boldsymbol{KC})^{-1}\cdot\left(\boldsymbol{F}_f - \boldsymbol{AKE}_f\right) - \boldsymbol{E}_f\right]\bar{\boldsymbol{f}} \tag{4.27}$$

This value can be calculated for all thrusters by changing the value of $\bar{\boldsymbol{f}}$ in Equation (4.27) to match the faulty thruster. Then, the value of the log-likelihood ratio can be computed by iterating over all thrusters using Equation (4.19), resulting in a statistical test-vector $\boldsymbol{s}$, the aggregate of all individual tests. When any of them exceed the detection threshold h_k, a fault detection is triggered and the component of this vector with the largest value is isolated. The output of this vector in a simple fault case in thruster 1 of satellite 1 is shown in Figure 4.13.

As can be seen, the vector shows a response in the appropriate component.
There are two drawbacks to this method. First, as the system that the Kalman filter is applied to here is not

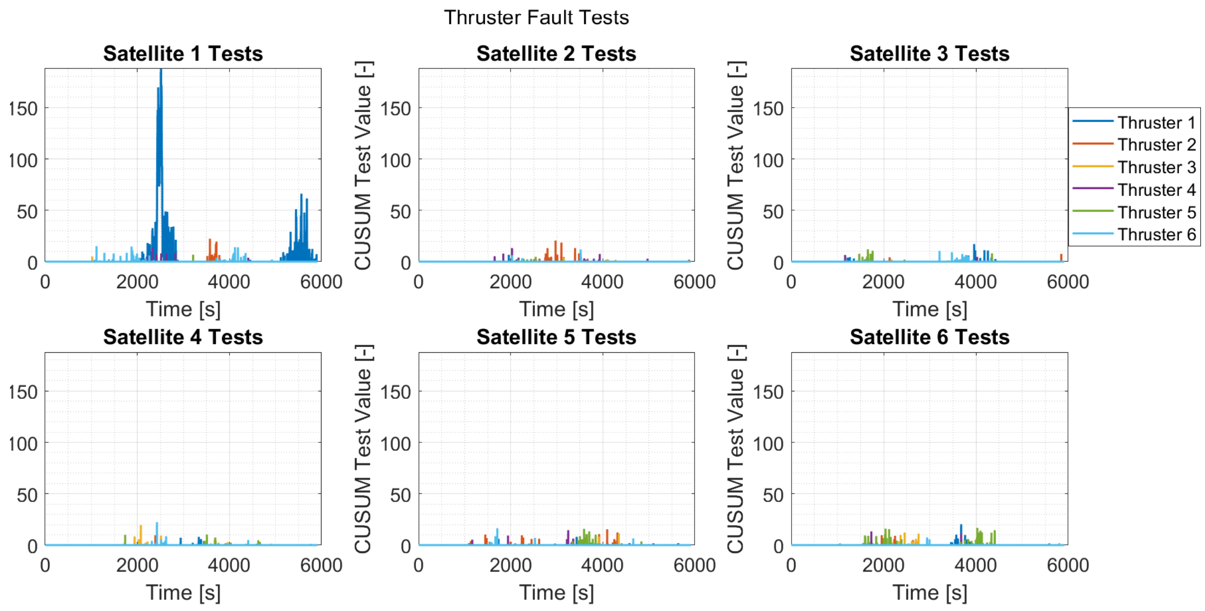


Figure 4.13: Test vector components in a scenario with a thruster fault in satellite 1 thruster 1.

linear in nature, but only an approximation of a non-linear system, there are both transients and deviations from the 0 mean present in the residual vector. In order to account for this, the mean in the faultless case ($\boldsymbol{\mu}_0$) was determined via the simulation by averaging the residual over one orbital period (after settling down). This value is used instead of a zero mean in Equation (4.19).
And second, as the deviation from the mean is essentially determined by a force, it will be impossible to distinguish between the absence of a force in a positive direction and the presence of a force in the negative direction, or vice versa. This has implications for the fault isolation, as an open fault is a presence of an unwanted force and a closed fault is the reduction of an intended force.

4.2.3. Selecting Kalman Filter FDI Parameters

A few parameters of the simple Kalman filter can be adjusted to affect the performance of the filter. Two parameters were tuned to reach the final performance, the first of which is the detection threshold h_k. The second is the process noise value of $\bar{f}$ in the computation of the change in the mean $\boldsymbol{\mu}_f$ with Equation (4.27). Both have to be selected together as the former is the cut-off point for the value of the statistical test s, while the second influences how easily s changes from 0. These values were selected through a trial and error process, testing the Kalman filer on simulation runs without faults and observing the peaks in the values of the statistical test vector. In the end, a value of $h_k = 20$ and $\bar{f} = 5\text{e}{-4}$ were selected. This ensured that no detection of the Kalman filter is a false positive, as the values were chosen such that the statistical tests remain below the threshold for faultless simulations.

4.3. Methodology Summary

In this chapter, the two approaches that are to be compared have been presented. They consist of an approach utilizing neural networks, which only have access to a limited amount of information, and a Kalman filter, which is centralized and hence has access to all the information in the formation.

The neural network based approach makes use of so called Recurrent Neural Network (RNN), which feature connections with a time-delay, allowing the network to have a time-dependency. Specifically, Long Short-Term Memory (LSTM) units were selected, due to their near optimal performance compared to other RNN architectures and their ease of implementation. The networks were chosen to have three hidden layers, the first two of which use LSTM units, while the last hidden layer is a regular, fully-connected layer.
The problem of fault detection is much simpler than the problem of fault isolation. Due to this, it was decided to train two separate networks, one type for detection and one for isolation. This split allows for training the isolation network only on the truly faulty data, which should result in a performance increase.

In addition to the splitting the networks along the lines of detection and isolation, the question has to be answered whether there should be a single network running on all spacecraft, or each spacecraft should make use of its own neural network. Both approaches will be evaluated in this thesis.
The two approaches differ in the training. For a network to be able to run on any satellite, it needs to be trained on the data from all of the spacecraft. This makes sure that the network will be able to handle data from satellites with different viewpoints. This will be called the "naive" network as no further considerations to the data handling are given.
The second approach involves training six separate networks, each trained on the data of one satellite only. This approach will be called "individual", as the trained networks will differ for each spacecraft.

Regardless of the approach, during operation the satellites may come to different decisions regarding the location of a fault. To remedy this, the following procedure is proposed: Once any detection network triggers a detection, the isolation network is started and the decision of the isolation network is broadcast. The other satellites then start their own isolation networks and share the results. These results are combined using a weighted sum, resulting in a combined, unified fault isolation. This method does rely on quick and reliable detection, as it requires an active communication link between the satellites, which may break up for certain faults. However, if the link does break down between some satellites, the other can still use the available measurements to come to a common diagnosis among themselves.

These neural networks need to be trained on the generated simulation data. The training platform used is Google Colab, making use of the tensorflow library. Specifically, the adaptive moment estimation algorithm, a variation of stochastic gradient descent, is used to find the network parameters. During training it was found that the dataset generated resulted in very large training times. Despite efforts to speed up training, this made using the entire dataset infeasible. Instead, the networks are only trained on a subset of the generated data.

The networks will be compared in their performance to a centralized method. This comparison method, the Kalman filter, has access to the entire formation state. The state space model it uses to generate a residual is the combined model of all individual connections. The innovation resulting from this filter is analyzed statistically using the cumulative sum of the log-likelihood ratio. The faults considered cause a deviation in the mean of the residual, thus allowing for detection. The fault isolation is achieved by determining the direction of change in the high-dimensional residual space and using a vector based version of the cumulative sum of the log-likelihood test in order to determine the fault location.

5 Evaluation and Analysis

The approaches described in Chapter 4 based on the data generated by the simulation described in Chapter 3 will be analyzed and compared in this chapter. First, the impact of the size of the training dataset on the performance on the ANNs will be evaluated in Section 5.1. This is followed by the analysis of the categorization accuracy of both the ANNs as well as the Centralized Model-based approach in Section 5.2. Then, the robustness of both approaches to various scenarios is analyzed and the results compared in Section 5.4. Finally, the results are summarized in Section 5.6.

5.1. Dataset Size

For neural networks the size of the dataset has a considerable impact on the final output accuracy. Generally, the larger and more diverse the dataset, the better the resulting performance of the network will be. In this section the impact of the size of the dataset is evaluated. Due to the considerable training duration, only 4 training session with varying dataset sizes were performed. The results can be seen in Table 5.1. This data only reflects the performance of the individual isolation network described in Section 4.1, specifically the individual network for satellite 1.

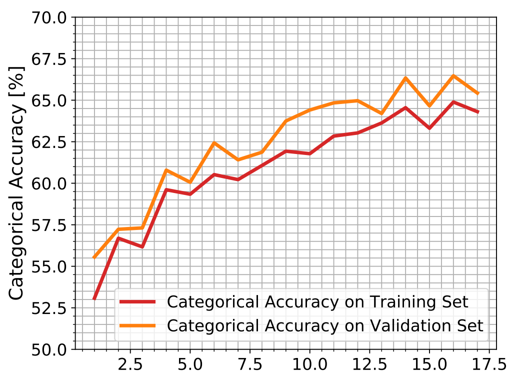
Table 5.1: Network categorical accuracy for varying sizes of the training dataset

Qualifier	0.5% Dataset	1% Dataset	2% Dataset	4% Dataset
Maximum categorical accuracy on training set	59.14%	61.20%	64.89%	66.02%
Categorical accuracy on validation set at minimum loss	59.86%	62.16%	66.34%	66.33%
Minimum loss for validation set	1.4072	1.2305	1.1587	1.1354
Number of epochs trained	11	16	17	17

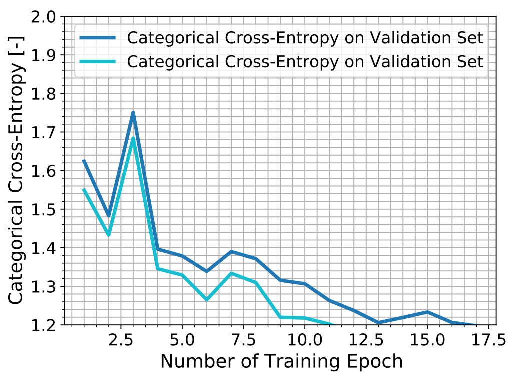
As can be seen from the table, the performance of the network does increase with a larger dataset, as expected. However, there are diminishing returns. As the dataset size is doubled, the performance in terms of accuracy on the training and validation set seem to increase at roughly 2%, while the loss drops at smaller and smaller increments, only showing a decrease of 0.0233 or 2.01% for the last doubling. The simplest explanation for this is that the networks 'learn' better from a varied dataset and at some point the subset accurately reflects the distribution of the full dataset.
Furthermore, it can be seen that the amount of epochs that the network was trained before the stopping condition was met is 16 to 17, except for the smallest dataset. Given the smaller dataset size, this means that the network reach their highest potential sooner the less data they are trained with, as 16 epochs of a dataset that is half as big are only amount to only half the computation steps in the optimization process. This is even more obvious for the smallest dataset tested, which reached the stopping criteria after only 11 epochs. While this number is influenced by the randomness of the stochastic gradient descent, the equivalent number of

epochs should be twice that of the next larger dataset to reach the same amount of optimization steps. If the equivalent amount of training steps is compared (i.e. at half the epochs of the smaller dataset), it can be seen that the network with the larger dataset actually show a larger value of the cost function, i.e. worse performance. That means that per training step, which is equivalent to per unit of time, the smaller sets learn more quickly. However, their performance limit is considerably lower than for the larger datasets. In that respect, the hypothesis that the networks reach better performance when trained on a larger dataset is true, however they take considerably longer to reach this limit and in fact learn slower in a per optimization step comparison.
For practical reasons, 2% of the full set were used for training the networks for the evaluation that follows. This allows for having multiple uninterrupted epochs of training to see the progression of the network performance as well as being able to evaluate a stopping criteria for the networks. Due to internet connectivity issues or an automatic disconnect from the virtual machine on the Google servers, there are frequent interruptions in the training process. The larger dataset, while giving a better performance has the drawback of very long training epochs, which make it impossible to reliably train over multiple epochs, and therefore to see if the network has reached the 'memorization' (i.e. overfitting) stage of training.
Similar tests with respect to the impact of the dataset size were performed for the naive isolation network, which showed little to no change in its performance depending on the amount of data it was trained on. The reason for this will be clear when the performance of the naive isolation network is evaluated in Section 5.2.2.

In order to better understand the training process, the performance over the training period is plotted in Figure 5.1, showing both the categorical accuracy as well as the value of the loss function on the training and validation set.



(a) Plot of satellite 1 network categorical accuracy over the course of training.



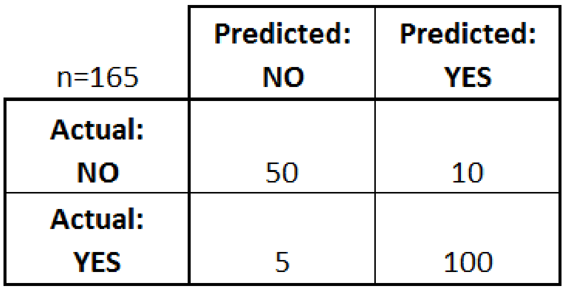
(b) Plot of satellite 1 network categorical cross-entropy over the course of training.

Figure 5.1: Plots of the performance of the individual isolation network for satellite over the training period.

The categorical accuracy of the network on both the training and validation dataset can be seen on the left in Figure 5.1a, while the value of the loss function (categorical cross-entropy) is shown on the right in Figure 5.1b. As expected, the value of the accuracy increases over the training period while the loss function decreases. The minimum of the loss on the validation set is reached at epoch 14, after which it starts to increase while the loss on the training set continues to decrease. As discussed in Section 4.1.3, the point after which there is no further improvement on the validation set for three epochs is chosen as the cut-off point.
Occasionally, it can be seen that the loss or the accuracy become worse after completing an epoch. This is a feature of the stochastic gradient descent, as the cost function value shown in Figure 5.1 is determined over the entire dataset (training or validation) that is used, but for each optimization step the gradient is only approximated using the data in one sample batch.
An observation that could be made during training that is not reflected in Figure 5.1 is the progress during the first epoch. The starting accuracy of the network is between 2% and 3%, as would be expected of a random set of weights, which will produce essentially random output. Therefore, the chance of a correct isolation is $\frac{1}{36}$ or approximately 2.78%. The biggest progress is made during the first epoch, where the accuracy rises to approximately 53% for the selected dataset, corresponding to the start of the graph shown in Figure 5.1.

5.2. Performance

The quality of either approach is judged by comparing a few relevant performance parameters, including the detection rates. A compact way to visualize a classifier performance is the *confusion matrix*. It is a representation of the amount of correctly and incorrectly classified data. The position on the vertical axis represents the true state, in the detection case with only two options: fault or no fault. The horizontal axis on the other hand shows the output of the classifier. Therefore, the diagonal of this representation (i.e. when true state and classifier are the same) shows the correct classification, while the off-diagonal elements show the incorrect classifier decisions. This is illustrated in Figure 5.2[1].



n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Figure 5.2: Example confusion matrix for explanatory purposes[1].

In the example confusion matrix, there are 50 instances where the classifier and the data agreed on the label 'NO', while there are 10 instances where the classifier incorrectly labeled a sample as 'YES'. Similarly, there are are 100 instances where the classifier correctly identified a sample as 'YES', and only 5 instances when it predicted the label 'NO' when it should have been 'YES'.
In the detection case, the incorrect decisions are the false positive (detection of a fault in the actually faultless case) and the false negative rate (missed detection of a fault in the faulty case), in the top right and bottom left of the confusion matrix, respectively.
All confusion matrices that are shown in this chapter are normalized to the true occurrences of a certain class (e.g. fault in a particular thruster), i.e. divided by the sum of the entries in each row (row normalized). This means, that the values represent the fraction of the classifier decisions that match the true state out of all occurrences of that state, rather than the the fraction of times the true state matches a particular classifier output out of all occurrences of that output. In the binary case, the former is also called the *recall* an the latter is called the *precision*, explained in Section 2.5.
For example, when considering the matrix in Figure 5.2, the normalized entries would read 0.833 and 0.167 for the top row, and 0.048 and 0.952 for the bottom row.

As mentioned before in Section 4.2, the Kalman filter approach is not evaluated on the exact same dataset due to a lack of absolute position information in the saved dataset. However, the Kalman filter was tested on a representative dataset, with the same distribution of faultless and faulty scenarios and the same (uniform) distribution of fault intensity as for the network data.
It should be noted that the Kalman filter has additional information compared to the neural networks in both the control input as well as implicit information about the position and velocity of the formation via the Hill-Frame. This means that from a purely systematic view, the Kalman filter has a higher potential for accuracy, as it has a higher amount of available information.
First, the approaches are compared on detecting faults in Section 5.2.1 and then on isolating a fault to a specific thruster in Section 5.2.2.

5.2.1. Detection

Before the detection network can be evaluated, a detection threshold has to be determined.

Selecting a Detection Threshold As discussed in Section 4.1.5, the weighted F measure will be used to select a threshold above which the network is counted as having detected a fault. The performance of the networks was evaluated with various thresholds, the results of which can be seen in Figure 5.3 and Figure 5.4, for the

[1]`https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/` last accessed 25.09.20

naive network and the individual network, respectively. These figures show various quality measures, including the precision, recall and F measure.

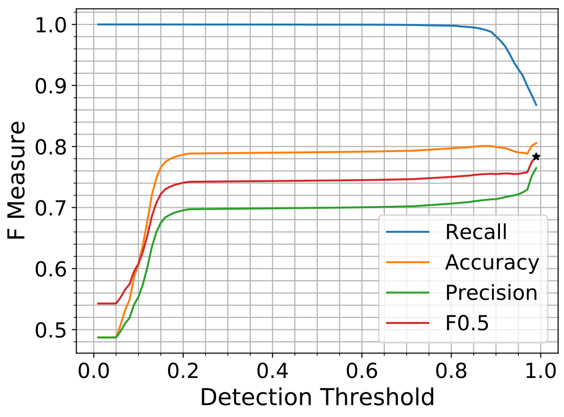


Figure 5.3: Classifier quality measures for naive detection network, black dot marks the maximum of the F measure.

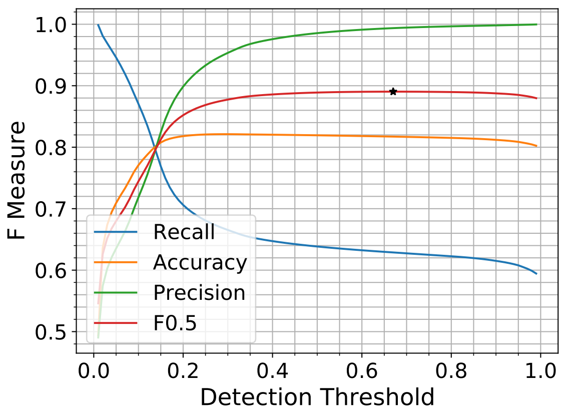


Figure 5.4: Classifier quality measures for the individual detection network, black dot marks the maximum of the F measure.

As expected, the higher the threshold, the more precise the network evaluation is and the lower the recall. It can be seen that the optimum F measure for the naive network is reached at the highest checked threshold of 0.99, whereas the individual network reaches the maximum at a threshold of 0.67. This is indicative of the naive network having a generally higher output value compared to the individual network, as for a good performance its threshold needs to be much higher. Considering the almost flat region seen in Figure 5.3, it appears as though the network almost never occupies states in the range of 0.2 to 0.7, as the change in the threshold there makes little difference in its performance. This is not the case for the individual network, which shows much smoother progression in all the performance measures. However it can be seen that the accuracy and the F measure flatten towards the higher threshold values.
Having chosen a threshold value for both networks, the detection performance can be evaluated. The confusion matrices in the detection case for the various classifiers can be seen in Tables 5.2 to 5.4. As mentioned before, these matrices are row normalized, i.e. the number of incidences are divided by the total amount of occurrences of the no fault and fault case. This means the entries show the proportion of true states matching the prediction out of all occurrences of that true state.

Table 5.2: Naive network detection rates in percent, row normalized.

	Predicted No Fault	Predicted Fault
True No Fault	74.63	25.37
True Fault	13.19	86.81

Table 5.3: Individual network detection rates in percent, row normalized.

	Predicted No Fault	Predicted Fault
True No Fault	99.62	0.38
True Fault	37.12	62.88

The detection confusion matrices for the naive and individual detection network specifically can be seen in Table 5.2 and Table 5.3, respectively. The individual network performs much more reliably than the naive detection network, as seen by the much lower percentage of false positives (25.37% vs 0.38%). However, it also detects fewer faults with a 62.88% true positive rate compared o the 86.81%. While the recall of the naive network is higher, this could simply be due to a much higher activation in general, leading to more positive decisions. Considering that the threshold for the naive network was selected at 0.99, there seems to be little distinction between the output in a faulty situation versus a faultless one.

Conversely, the confusion matrix for the centralized Kalman filter be seen in Table 5.4. As can be seen from the table, the Kalman filter works very well for detecting the considered faults. In fact, all faults were detected, the only difference is how long it took for the filter to detect them after the fault inception. Furthermore,

Table 5.4: Kalman filter detection rates in percent, row normalized.

	Predicted No Fault	Predicted Fault
True No Fault	100	0.0
True Fault	1.67	98.32

there is no instance of the filter miscategorizing the faultless state, shown by the entries in the upper row of Table 5.4. The false positive rate is 0, meaning no fault is supposedly detected when in fact the system is in fact faultltess. The false negative rate of 1.67% is due to the delay in detection time that will be further addressed in Section 5.3.1.

Comparing the three approaches, it can be immediately seen that the networks perform worse across the board. True positive and true negative rates are both considerably lower in the networks. The true positive rate is at 86.81% and 62.88% for the naive and individual network, compared to the 99.49% for the Kalman filter. While there are no false positives in the Kalman filter approach, the naive network approach has a considerable amount of false positive detections with 25.37%. The individual network performs much better in this regard with a true negative rate of 99.62%, and therefore only 0.38% false positives. However, the false negative rate is much higher compared to the Kalman case with 13.19% for the naive network and 37.12%, indicating that the time until detection is larger, or that certain faults are completely missed.

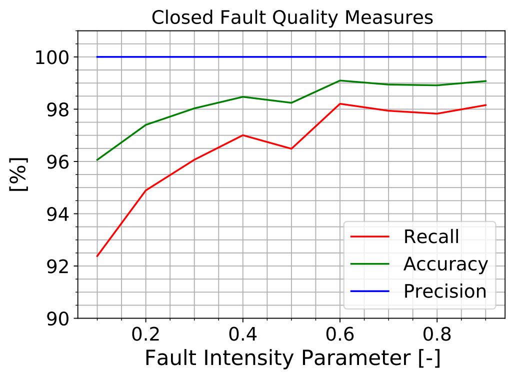
While the average performance of the classifiers is interesting to consider, it is also worthwhile to inspect the performance on the two fault types separately. The performance of all three fault detectors split according to the fault type is summarized in Table 5.5.

Table 5.5: Average accuracy (Acc.), precision (Prec.) and recall (Rec.) for each detection network split by fault type, in percent.

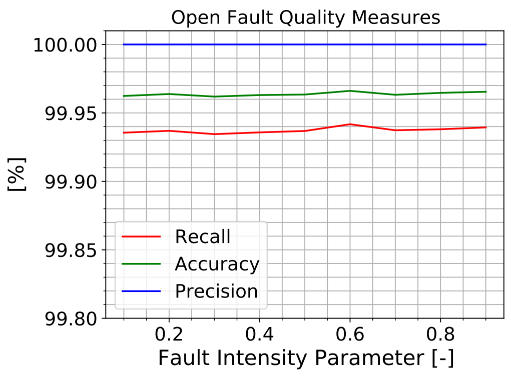
Dataset	Naive Network			Individual Network 1			Kalman		
	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.
Closed Faults	61.85	70.20	72.58	48.51	99.29	21.57	98.22	100	96.48
Open Faults	82.73	81.18	99.51	99.74	99.89	99.76	99.96	100	99.94
Complete Dataset	80.56	76.47	86.81	81.72	99.37	62.88	99.09	100	98.32

As can be seen for all detectors, the performance is worse on the closed faults compared to the open type faults. The only exception in this regards is the precision of the Kalman filter, which remains at 100%.
The reasons for this difference in performance in the detectors is in the relative impact that closed and open fault have on the relative positioning. This is inherent to the nature of these respective faults. The closed faults are only visible when a thruster is scheduled to fire, as only then its function is impaired. In this setup, the time in between thrust windows is 60 s. Compared to the constant effect of an open fault, which causes the thruster to continuously fire, this is a much smaller time frame in which the fault can have an effect on the relative positioning. In addition, with the considered thruster on the satellite, the average opening time of a thruster is only 0.115 s, leading to an even smaller impact of that fault type.
Furthermore, it can be seen that the difference between the open and closed faults is much bigger for the neural network based detectors compared to the Kalman filter. This difference could be due to fact that the Kalman method has information about the thruster commands which are supposed to be executed, and the effect they should have on the relative positioning. A deviation from this predicted effect will raise the statistical tests and therefore cause a detection. The second reason for the Kalman filter much higher performance is the availability of the entire set of formation data, as opposed to only a subset of the data corresponding to the neighbors of only a single satellite.

Another aspect to consider is the performance variation by fault intensity. In order to get a representative average, the data was split into buckets of width 0.1, e.g. all faults in the half open interval [0.1, 0.2) were gathered. The accuracy, precision and recall averaged over a 0.1 fault interval can be seen in Figures 5.5 to 5.7. As can be seen in Figure 5.5b, the quality measures for the open fault cases show almost no change over the fault intensity. The same cannot be said for the closed faults, which show decreases in the accuracy and recall from approximately 99% and 98%, to 96% and 92.5%, respectively. The precision of the Kalman filter stays

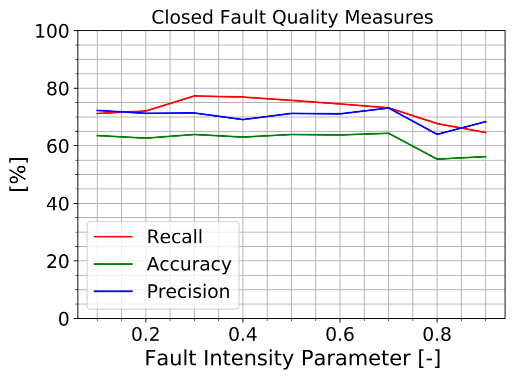


(a) Kalman filter detection quality measures for closed faults.

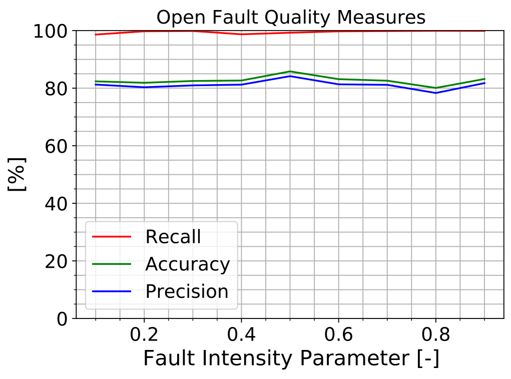


(b) Kalman filter detection quality measures for open faults.

Figure 5.5: Kalman filter detection quality measures of there fault intensity interval [0.1, 1].



(a) Naive network detection quality measures for closed faults.



(b) Naive network detection quality measures for open faults.

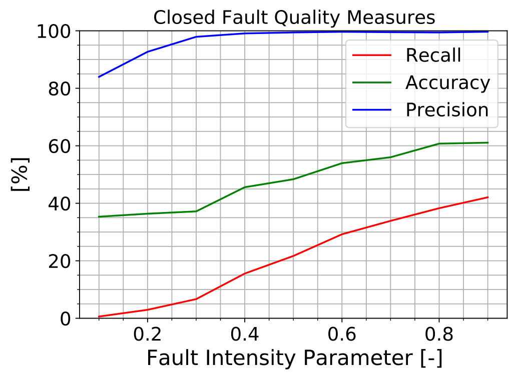
Figure 5.6: Naive network quality measures over the fault intensity interval [0.1, 1].

at 100%, meaning that no false positives are present. The general drop in recall and accuracy indicates an increase in the detection time for the closed faults with lower fault intensities.

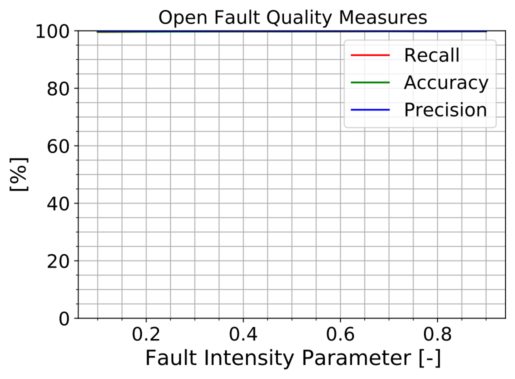
The naive network shows little change over the considered fault intensities for both closed and open fault cases. This is surprising as the impact of (especially the closed) faults can vary considerably based on the fault intensity and therefore one would expect a drop in detection quality towards the lower end of fault intensities. This indicates that the naive network does not react properly to the data.

The individual detection network of satellite 1 shows similar trends to the Kalman filter. For the open faults, shown in Figure 5.7b, the performance stays close to optimal, with accuracy, precision and recall staying above 99.5%. However, unlike the Kalman filter, there does appear to be a downwards trend towards the lower fault intensities.
The performance on the closed faults show an even clearer trend, as the recall drops from around 40% at a fault intensity of 1.0 to less than 5% at a fault intensity of 0.1. It seems the individual network is approach its lower limit of the detectability for the closed faults. The precision stays above 95% up until the fault intensity drops to 0.2.

Finally, it may be possible that the detection differs based on the location of the fault. Especially for the individual network this may be the case, as it was only trained on a subset of the data corresponding to the neighbors of satellite 1. In addition, the naive network, if it were to actually run on the satellites would only have access to the relative distance measurements of that satellite's neighbors. Therefore, the question of whether the accuracy, precision or recall of the detectors varies by fault location should be investigated. These quality measures averaged over the thrusters of each satellite are illustrated as bar plots in Figures 5.8 and 5.9. As can be seen from the figures, there is only little difference (1-2 percent points) in accuracy, precision and recall



(a) Individual network of satellite 1 detection quality measures for closed faults.



(b) Individual network of satellite 1 detection quality measures for open faults.

Figure 5.7: Individual network of satellite 1 detection quality measures over the fault intensity interval [0.1, 1]

based on the location of the fault. This is interesting considering that both the naive and individual networks only have access to a subset of the full formation information, but (at least in terms of fault detection) can make detections for a fault in any part of the formation with almost equal performance. For the Kalman filter, there is similarly no difference in detection performance based on the fault location, as expected.

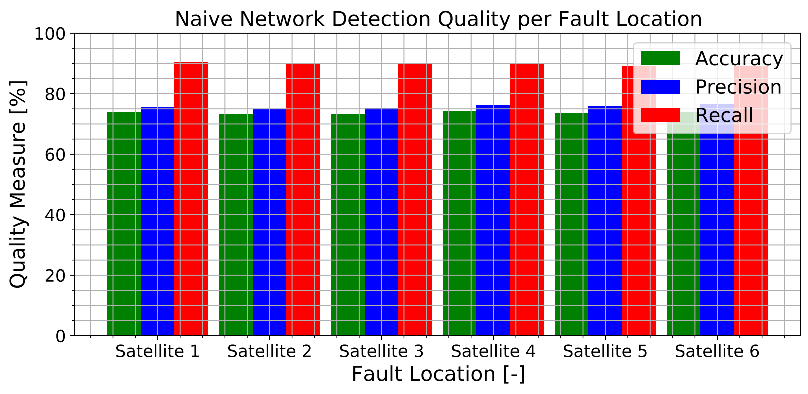


Figure 5.8: Bar plot of the quality measures per fault location for the naive network.

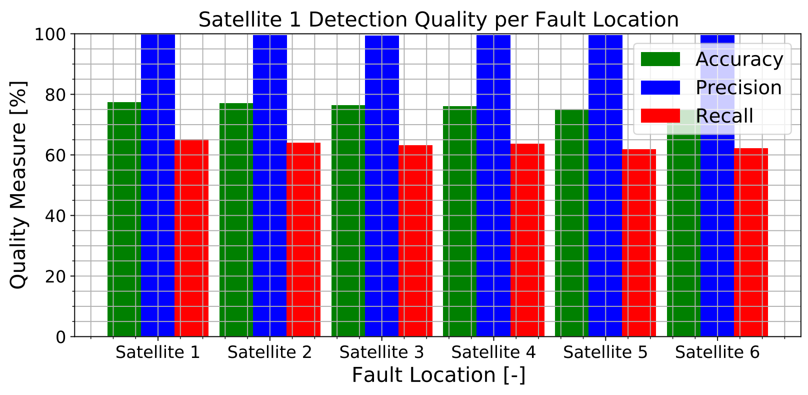


Figure 5.9: Bar plot of the quality measures per fault location for the individual network of satellite 1.

5.2.2. Isolation

The problem of isolating the fault is, in this case, the determination of which thruster experiences either the closed or open valve fault. This is a substantially harder problem to solve than the issue of fault detection. Not only does isolation necessitate prior detection, but the networks also needs to determine based on the relative motion alone where a fault has occurred. As mentioned before, the confusion matrices will be the main tool to investigate the isolation performance of the methods.

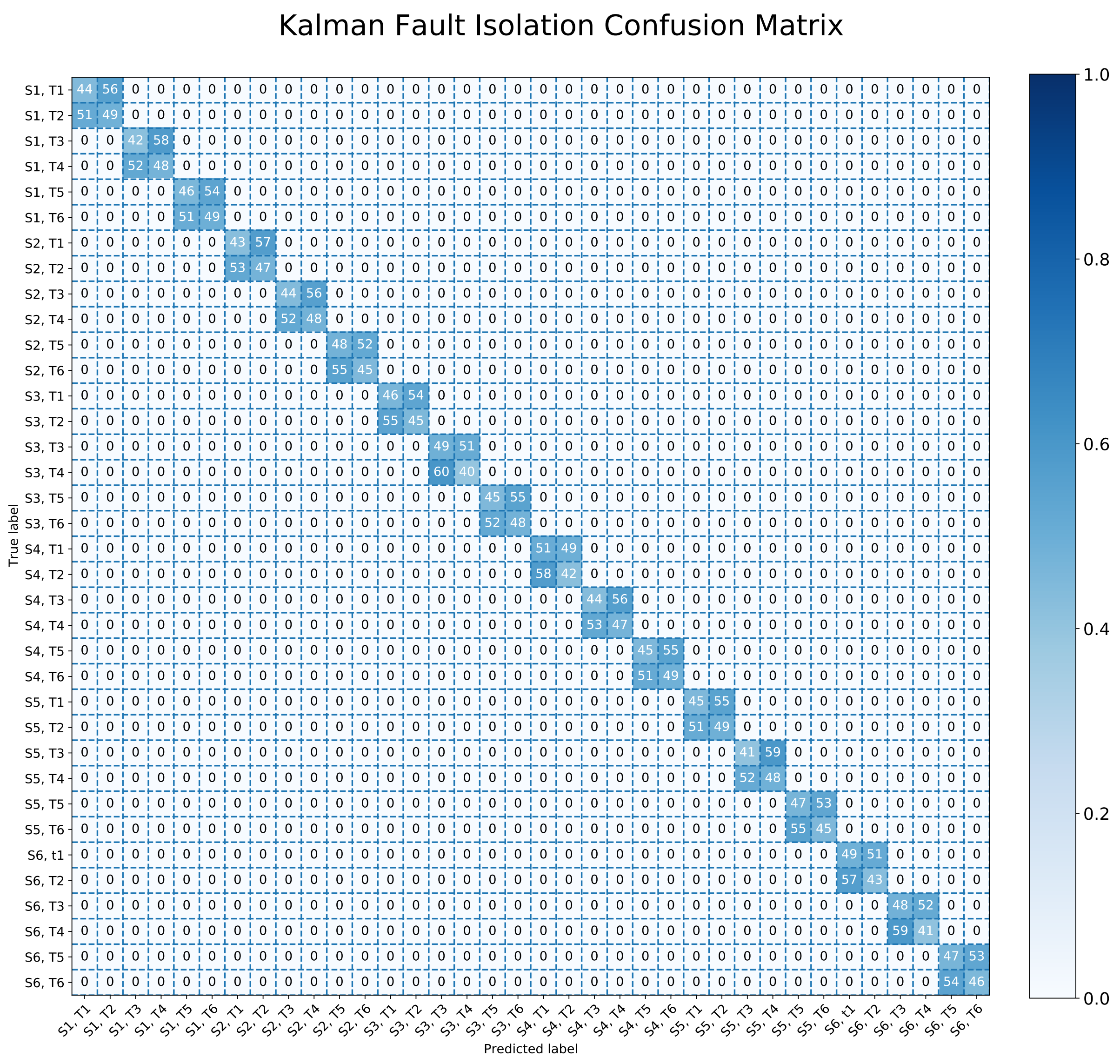
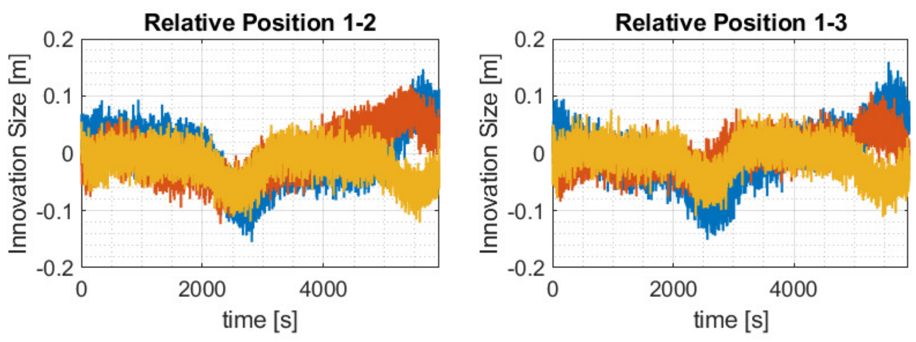


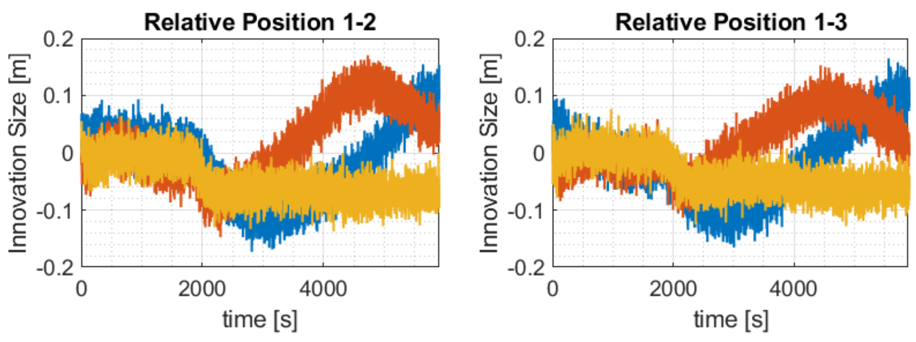
Figure 5.10: Kalman filter isolation confusion matrix, numbers in percent.

The full isolation confusion matrix for the Kalman filter can be seen in Figure 5.10. One interesting observation is that the the filter only appears to misisolate the thrusters which are mounted in opposite directions, e.g. thrusters 1 and 2, 3 and 4, or 5 and 6 of any given satellite. No other misisolations take place.

Due to the fact that the isolation is based on the assumption of an external force in a certain direction which influences the residual mean, the reduction or absence of a force in one direction cannot be distinguished from a present force in the opposite direction from the residual alone. This is illustrated in Figures 5.11a and 5.11b, which shows the residual in the case of a closed fault in thruster 1 of satellite 1, and the residual in the case of an similarly intense open fault in the opposing thruster 2 of satellite 1.



(a) Kalman innovation for a closed fault in thruster 1-1.



(b) Kalman innovation for an open fault in thruster 1-2.

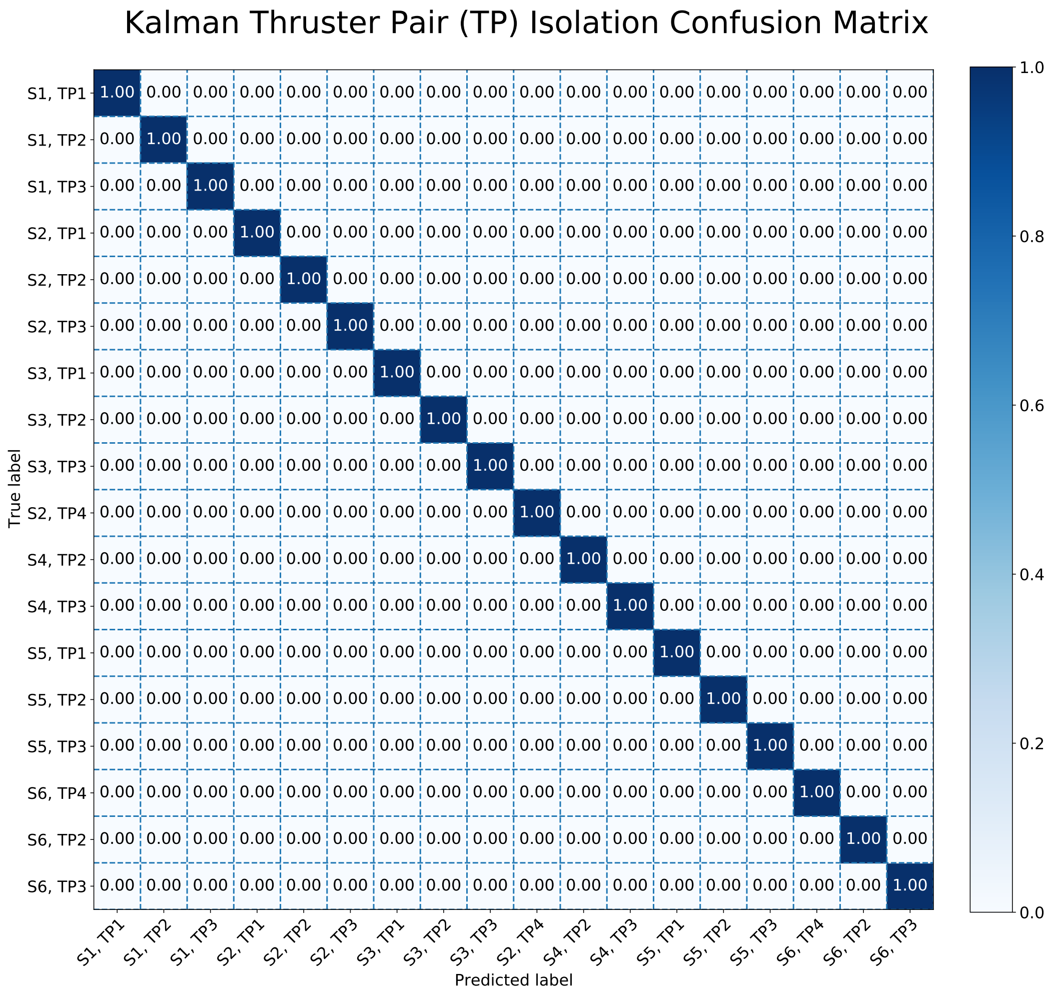
Figure 5.11: Kalman filter innovation for the two considered fault types to showcase their similarity.

As can be seen when comparing the two figures, both faults show a deviation in the residual vector in the exact same direction. The only difference is that the open fault causes the fault residual to constantly deviate from the 0 mean, due to the fact that the open fault causes the thruster to continuously fire. This is not the case for the closed fault, whose impact can only be seen when the thruster is supposed to be firing. Since thruster 1 is completely inactive for a period of time, the fault signal returns to the 0 mean and only starts deviating again when the thruster is scheduled to fire again.

The performance of the Kalman filter when only considering the problem of isolating the fault in a thruster pair can be seen in the confusion matrix in Figure 5.12. As can be seen, the performance in that aspect is perfect, with no thruster pair being misisolated. This is very good performance and shows that when the filter flags a faulty thruster, one can at least be certain that one thruster in that pair is faulty.

As for the performance of the networks, the isolation matrix of the naive network is shown in Figure 5.13, the complete confusion matrix for the individual network of satellite 1 is shown in Figure 5.16 and the confusion matrix for the combined networks is shown in Figure 5.19. The remainder of the individual networks complete isolation matrices are omitted here for space reasons, but can be found in Appendix A.

It can be seen that the naive isolation network has not learned to isolate faults at all. Instead, it only gives one of two possible responses to any input: either satellite 2 thruster 6 is flagged or satellite 4 thruster 5. This could be due to the dataset being too inconsistent or "confusing" for the network to learn any particular features which could allow for isolation of a fault. The naive network is supposed to be able to isolate any particular thruster when given the relative positioning data of any satellite in the formation, but it is trained on the relative positioning data of the entire formation. This leads to inconsistent labeling of data. At one point the network receives relative positioning data from the perspective of satellite 1, which will have a completely different fault signature than the relative positioning data from the perspective of any other satellite, yet the true label that is used to compute the loss function is the same for both vastly different sets of input. This lack of clear correlation between the input and fault label of said input is most likely the cause for the failure to train the naive network.
This explains the behavior of the naive neural networks during the training period, where the naive network,



True label \ Predicted label	S1, TP1	S1, TP2	S1, TP3	S2, TP1	S2, TP2	S2, TP3	S3, TP1	S3, TP2	S3, TP3	S2, TP4	S4, TP2	S4, TP3	S5, TP1	S5, TP2	S5, TP3	S6, TP4	S6, TP2	S6, TP3
S1, TP1	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S1, TP2	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S1, TP3	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S2, TP1	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S2, TP2	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S2, TP3	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S3, TP1	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S3, TP2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S3, TP3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S2, TP4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S4, TP2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S4, TP3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
S5, TP1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
S5, TP2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
S5, TP3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
S6, TP4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
S6, TP2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
S6, TP3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Figure 5.12: Kalman filter thruster pair (TP) isolation confusion matrix.

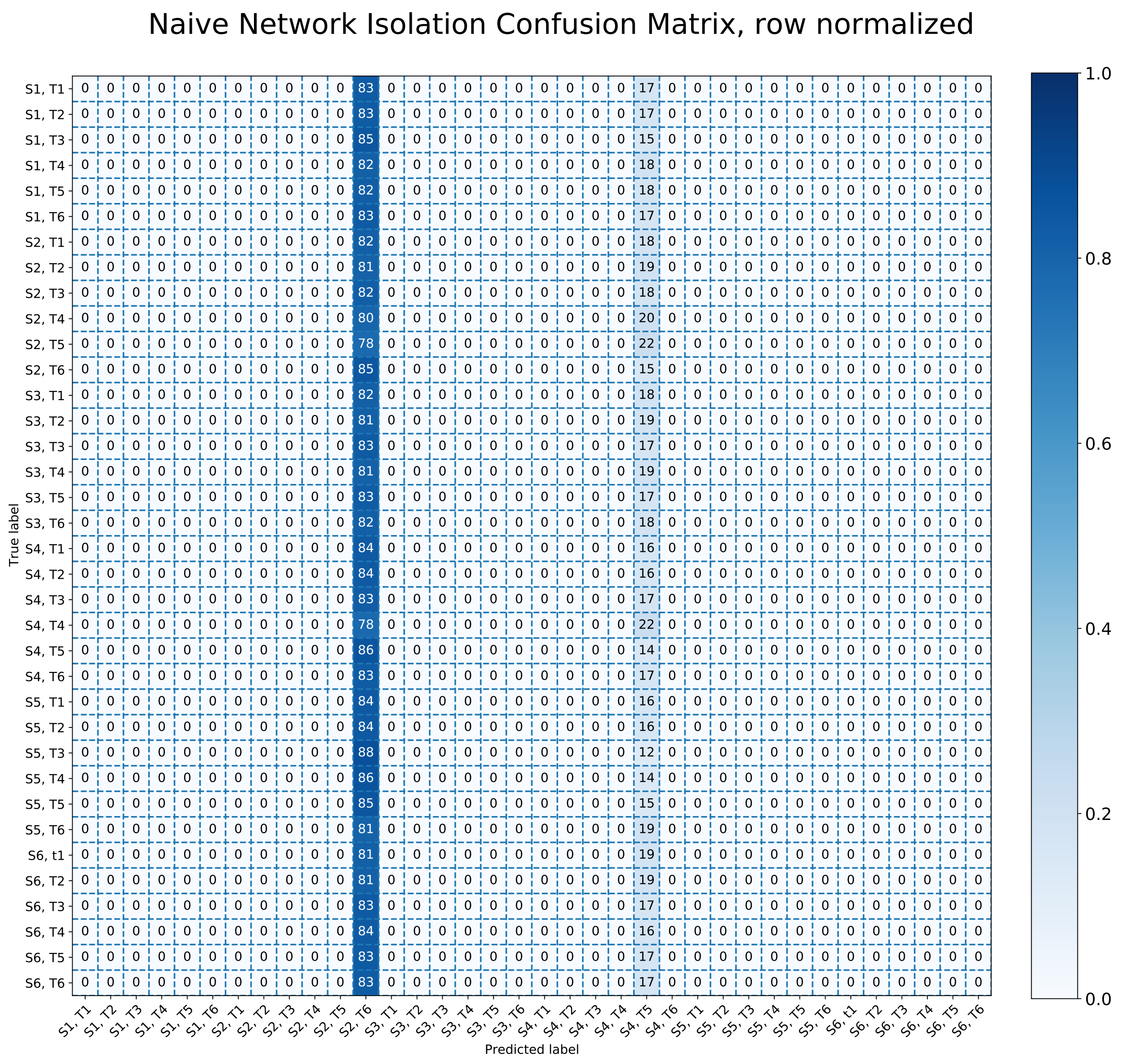


Figure 5.13: Naive network isolation confusion matrix for thruster isolation.

despite changes to the size of the dataset or the values of its hyper-parameters did not show any significant change to its performance.
As for why these two outputs seem to be the one that the network selects, there could be a few reasons. The first is that the training set might be biased slightly towards one of these two fault scenarios. While the overall dataset was constructed so that every fault occurs equally often in every thruster, the networks were not trained on the complete dataset, but a reduced one, containing only 2% of the data. As described in Section 4.1, the way this reduced dataset was produced was through selecting the simulation runs with the highest and lowest fault intensity. The individual runs were seleted randomly, therefore the selected reduced dataset contained some variation in the amount of certain fault types and locations. In addition, there is a smaller effect which may have exacerbated this issue. The time of the fault occurrence was also determined randomly during the simulation. Hence, there is an inherent imbalance in the distribution of the fault data.
This imbalance in the distribution of the fault labels together with the lack of a clear correlation between input and output could lead the network to only output a single value as that could represent a local minimum in the cost function.
Investigating the true distribution of faults for this reduced dataset resulted in Figure 5.14 which shows the distribution of fault labels for this training dataset.

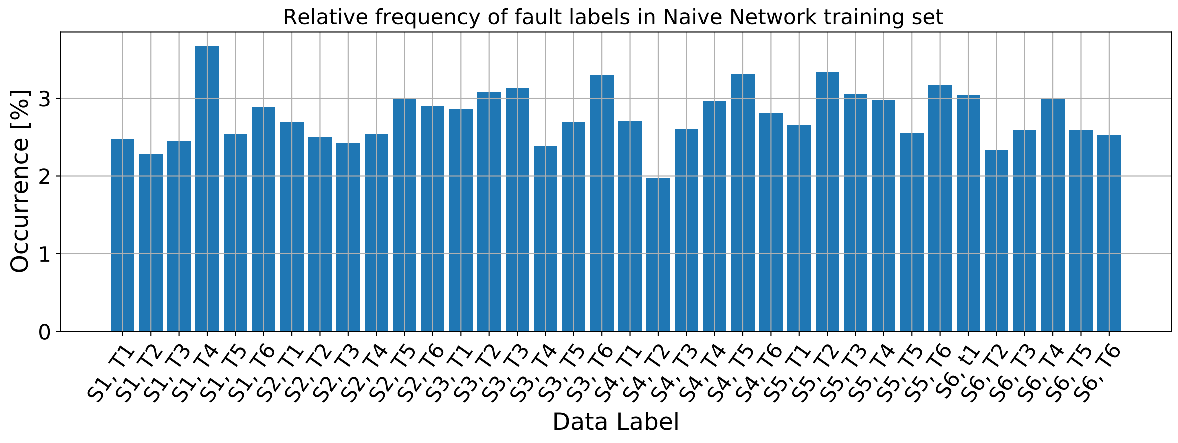


Figure 5.14: Distribution of the frequency of the fault labels in the naive training set, numbers in percent.

It can be seen that there is a variation in the distribution of the fault labels, ranging from 3.6% as the most to 1.9% as the least common. However, the three labels which occur the most (satellite 1 thruster 4, satellite 3 thruster 6 and satellite 5 thruster 2) are not the ones which the network favors. Furthermore, two other of naive networks with a different set of hyper-parameters were tested against the validation set. The first is a network with double the amount of LSTM units per layer. It showed a similar problem, only that it always selected satellite 2 thruster 6. Similarly, the other tested network with three layers of hidden LSTM units always selected satellite 6, thruster 6.
In order to investigate this behavior further, the network weights saved at each training epoch were investigated. The same behavior of solely selecting a single fault location all the time was seen at every step of the training. The most commonly selected locations can be seen in Figure 5.15. Unfortunately, the weights for epochs 4-10 were lost and so that data is missing from the graph.

As can be seen from the figure, the network changes its preferred output almost every epoch. The outputs chosen most often are satellite 1 thruster 4 and satellite 2 thruster 6. As was seen in Figure 5.14, the fault location that is most common in the naive training set is in fact satellite 1 thruster 4, so it is not surprising that the network seems to favor this output as it will be more correct on average than the other options. However, this does not explain why the network is seemingly incapable of learning the patterns in the data and sticks to only selecting a single output. Due to the stochastic nature of computing the gradient for each step in the training, there is some randomness involved. The computation of the gradient relies on a small batch of the data rather then the full training set. This could explain why the network does not converge to a specific output and frequently changes its preferred output, as the most common fault label in a batch is likely to not be the same as the one that is most frequent in the entire dataset.

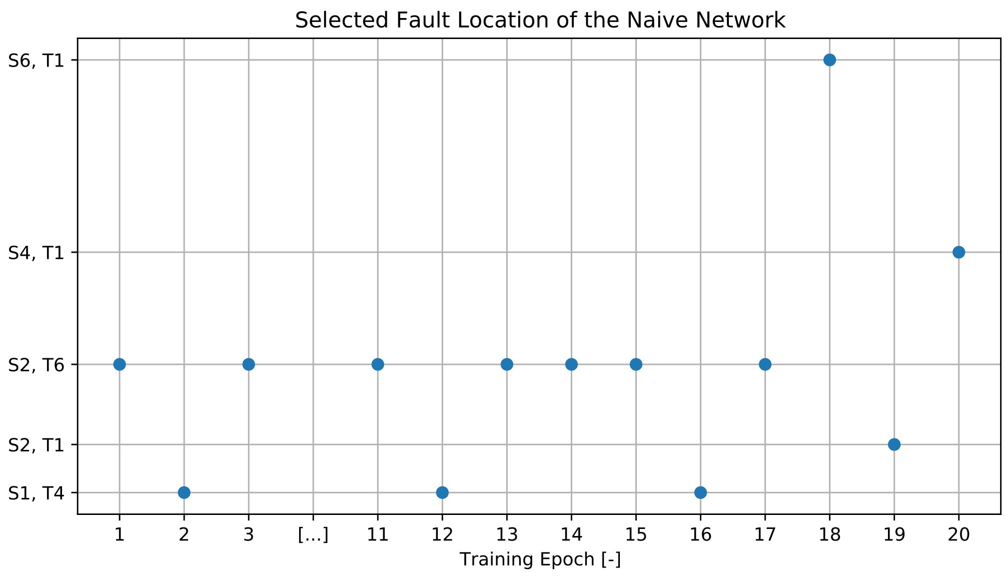


Figure 5.15: Most selected fault location for the naive network versus the training epoch.

Overall it is likely that the naive network's performances is in part due to the imbalance in the training set, in part due to the inconsistent relation between the input of the network and the desired output, and the type of network approach chosen.

The individual networks show much better isolation performance, as seen in Figure 5.16. While only the complete isolation matrix for satellite 1's individual network is shown here, the remainder can be seen in Appendix A. The majority of isolations is correct which can be seen from the high-valued entries in the diagonal of the matrix, which range from the low value 8 % for the isolations of satellite 6 thruster 3, up to 91 & of satellite 3 thruster 6. There is furthermore a pattern here that is not easy to spot but can be seen more clearly when the isolation is reduced to simply the satellite itself, rather than the thrusters of the entire formation. The confusion matrices for this case are shown in Figure 5.17.

The biggest area of misisolation lies in the satellite itself and the one it has no direct connection to, e.g. satellite 1 is poorest in isolating faults in itself and satellite 6 on the other end of the formation. This trend can be seen with every satellite of the formation. On the one hand, it is intuitive why the network's performance is poorest in isolating the satellite of which no direct information is given to the network. On the other hand, its seeming inability to determine faults in itself is surprising. The fault signature of a fault in any particular thruster of itself is distinct from that of a fault in its neighbors, and it is clearly visible in the relative motion, at least for the more intense faults.
However, the situation becomes more clear when the effect of a particular fault on the relative positioning is investigated. The relative distances of all 12 unique satellite connections after a thruster fault in the first thruster of satellite 1 are shown in Figure 5.18. It can be seen that the effect of this thruster fault is largest on the connections of satellite 1 (1-2, 1-3, 1-4, and 1-5), while the connections among those satellites (i.e. 2, 3, 4 and 5) are relatively undisturbed. But surprisingly, the effect of a thruster fault in satellite 1 can also be seen very well in the the relative connections of satellite 6. The precise reason for this propagation of the fault is complex as the control system itself takes into account the relative distances, but this explains why the network has difficulty in distinguishing between faults in itself and the opposing satellite in the formation. The fault signature is similar, only weaker for a fault in the opposing satellite.
If the confusion matrix shown in Figure 5.16 is looked at more closely, it can be seen that the misisolations affect the opposite thruster on opposing satellite in the formation. For example, a fault in satellite 1 thruster 1 is most misisolated as a fault in satellite 6 thruster 2. Conversely, a fault in satellite 6 thruster 1 is most often misisolated as a fault in satellite 1 thruster 2.

There is an additional observation to be made, as the network often misisolates faults as belonging to satellite

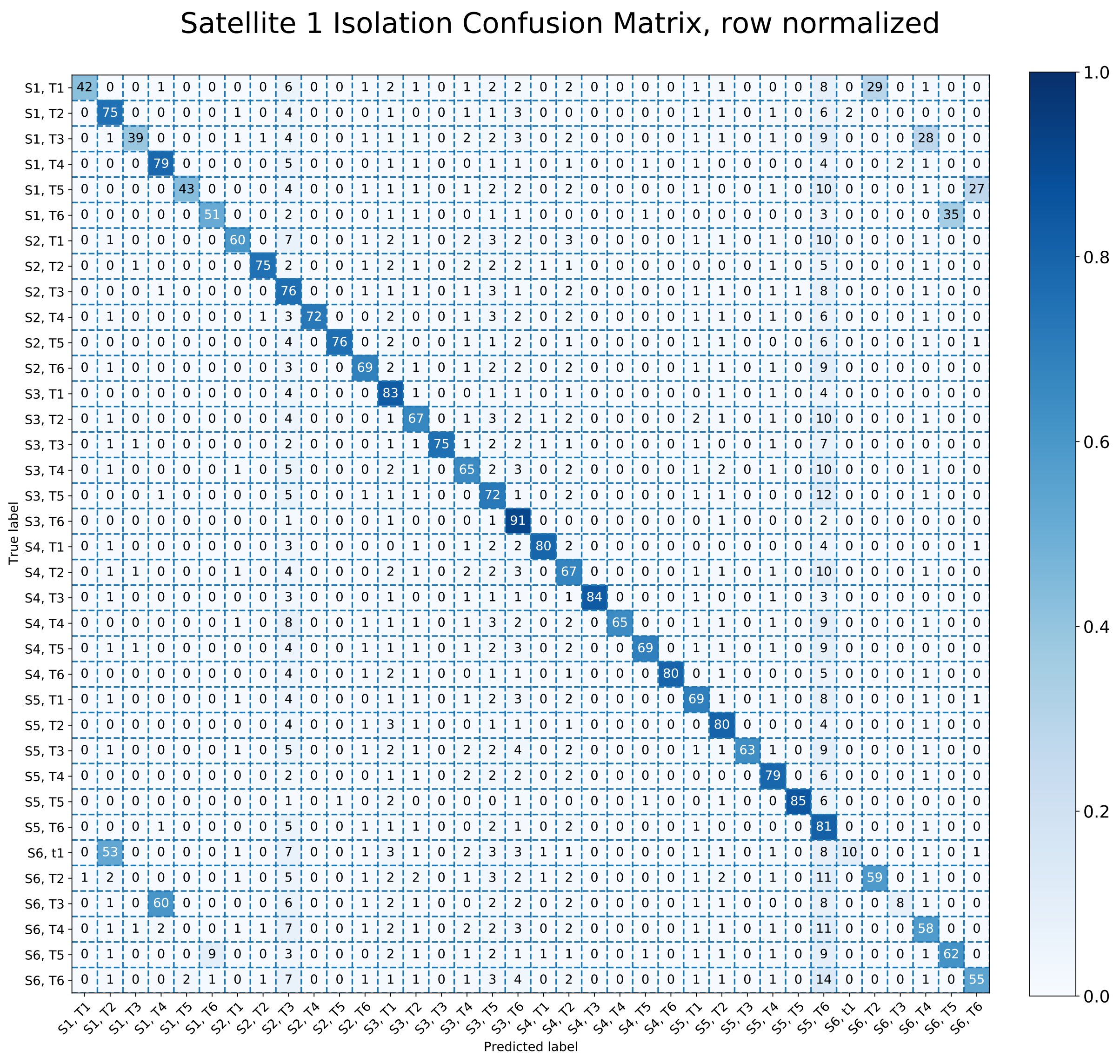
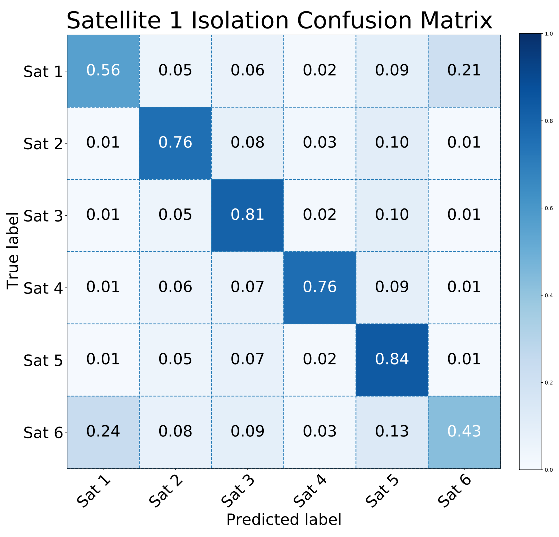
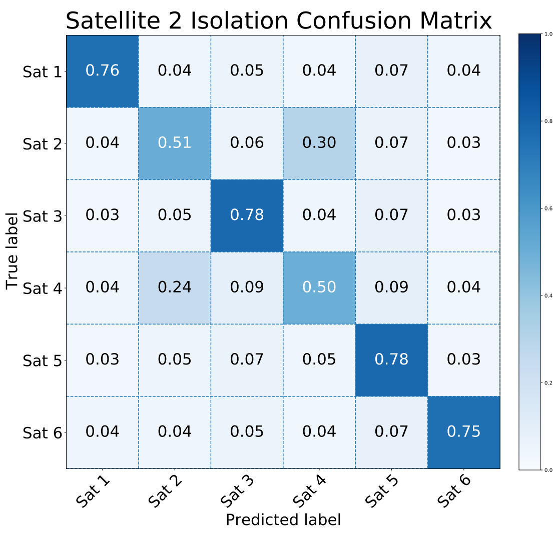


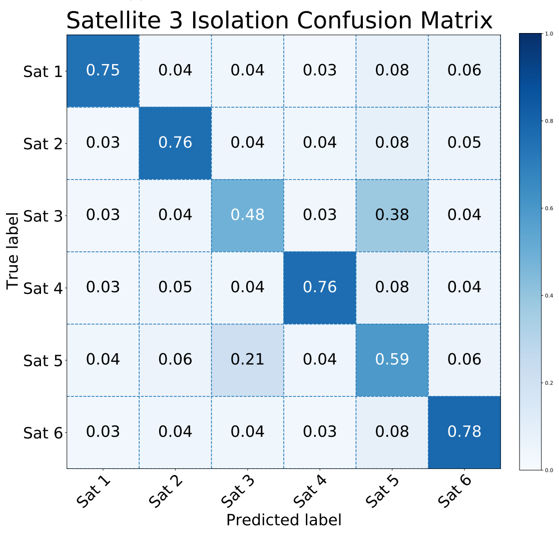
Figure 5.16: Satellite 1 isolation confusion matrix for thruster isolation, numbers in percent.



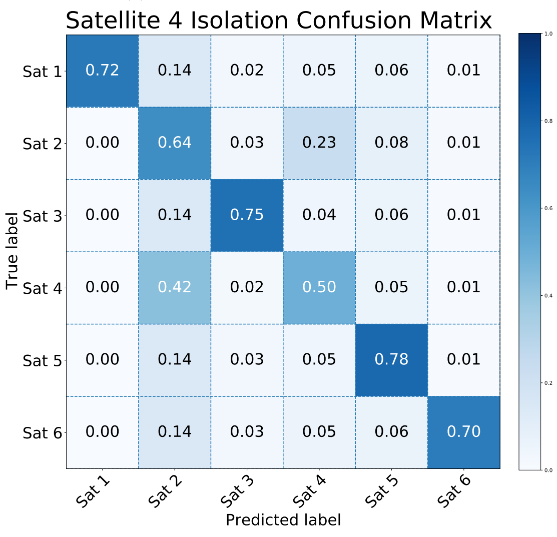
(a) Satellite 1 network confusion matrix.



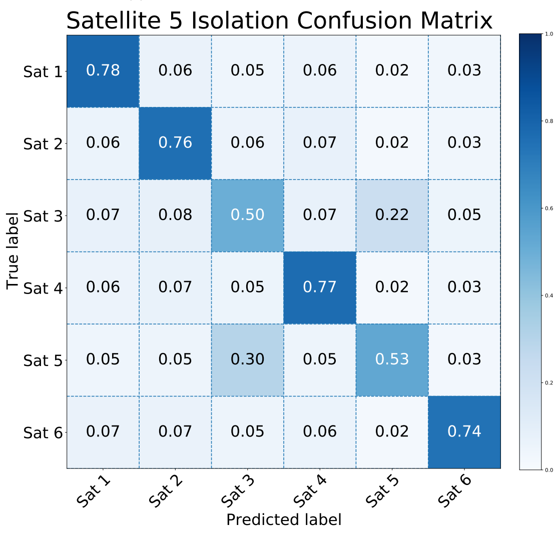
(b) Satellite 2 network confusion matrix.



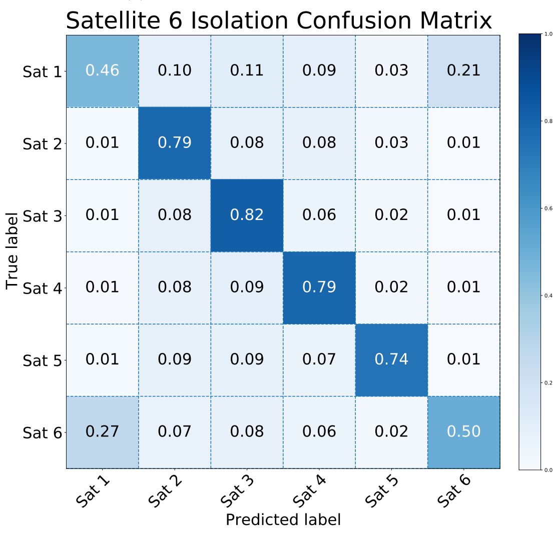
(c) Satellite 3 network confusion matrix.



(d) Satellite 4 network confusion matrix.



(e) Satellite 5 network confusion matrix.



(f) Satellite 6 network confusion matrix.

Figure 5.17: Satellite isolation confusion matrices for each individual network.

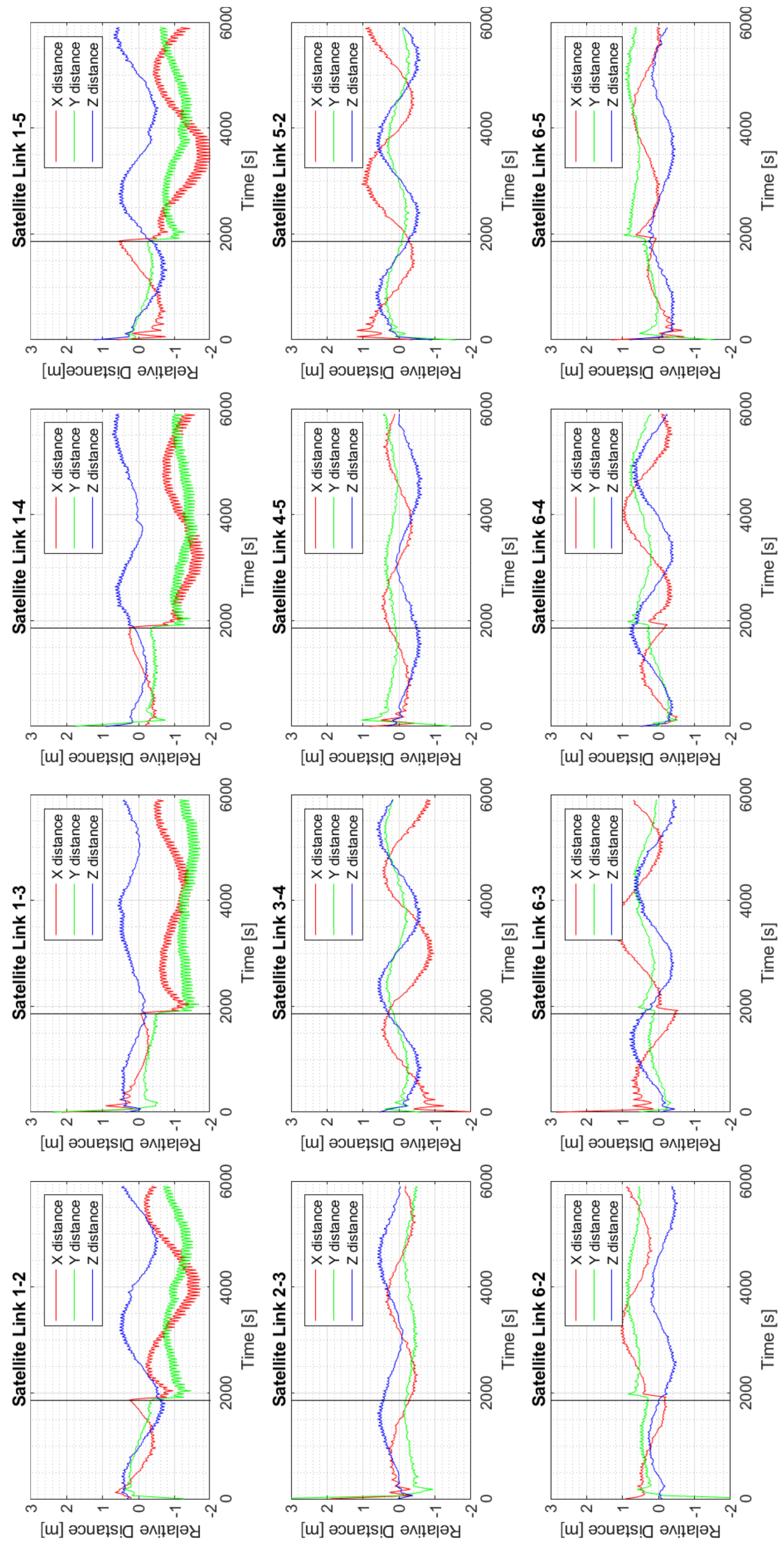


Figure 5.18: Relative positions of all satellite connections after the occurrence of an open fault in satellite 1, thruster 1 (S1T1).

5 thruster 6, seen by the light blue strip on the right hand side of the confusion matrix. This behavior can also be seen in the other individual networks. Considering the output behavior of the naive networks, it is likely that this is a remnant of the training phase. The networks, due to the initial random assignment of weights and biases, start selecting an output at random and over the training period the network starts to separate the faults.
One way to get around the issues of a single network is to combine the network predictions as described in Section 4.1.1, using the confusion matrices as a way to quantify the uncertainty of a prediction. The confusion matrix for this combined output can be seen in Figure 5.19.

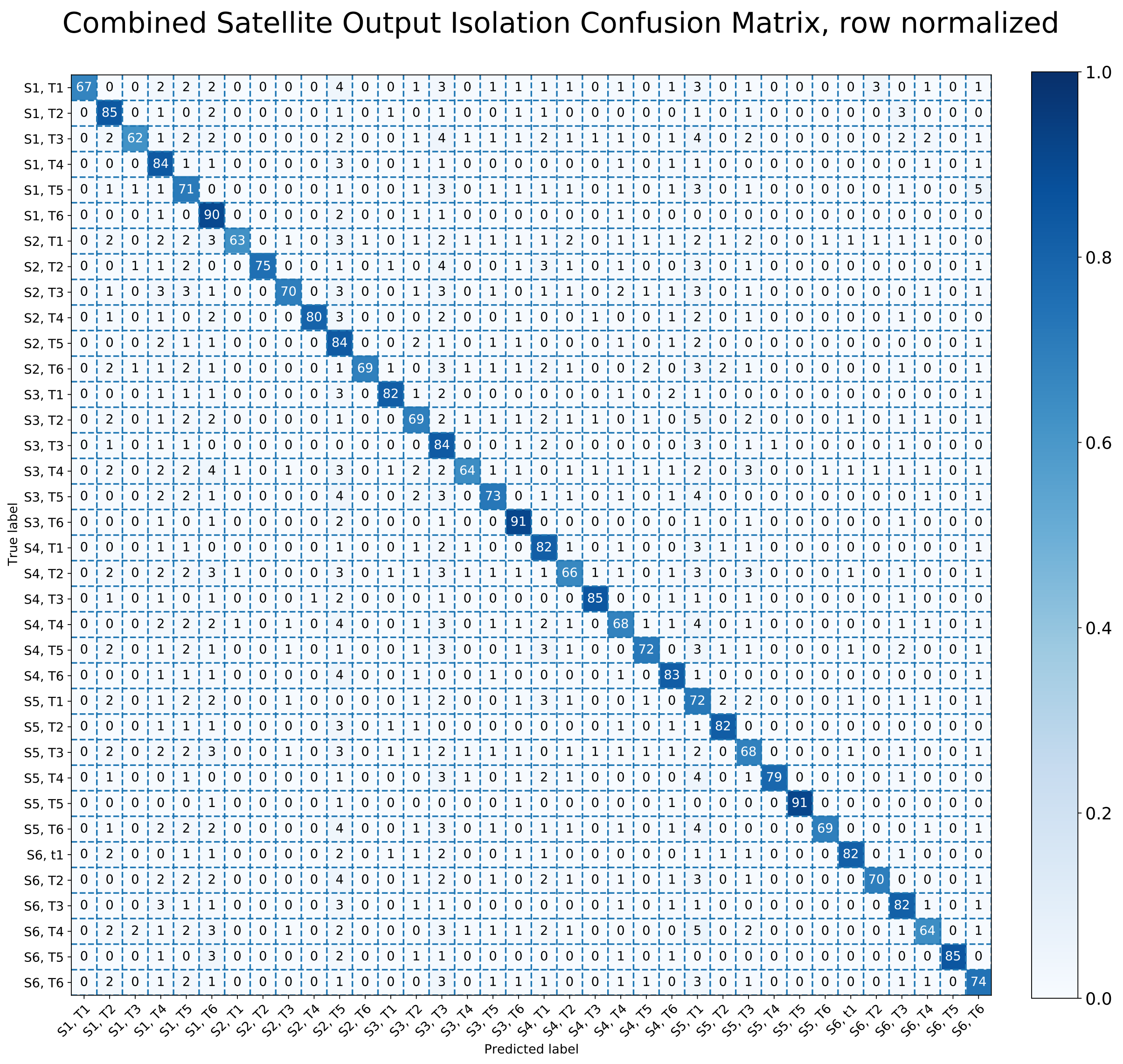


Figure 5.19: Combined network output confusion matrix for thruster isolation, numbers in percent.

Overall, the effect of combining the network output is a performances improvement. The misisolation seen in the individual networks disappear, although the results are still far from perfect. While the overall performance of the classifiers is interesting to consider, it is also worthwhile to inspect the performance on the fault types and fault intensities. The performance of the isolation approaches is summarized in Table 5.6. As the quality measures such as precision and recall are only defined for scenarios with merely 2 cases, the values in Table 5.6 are the mean value of each individual fault precision and recall. That is to say, for each fault location (e.g. satellite 2 thruster 3), a precision can be calculated as the ratio of correct isolations of said thruster divided by all occurrences of that particular network output. Similarly, the recall for each fault location can be calculated as the ratio of correct isolations of said thruster divided by all time steps which contained a fault in that thruster. The isolation accuracy is simply the amount of correct isolations divided by all isolations. These values are averaged over all fault locations to reach the figures in Table 5.6.

Table 5.6: Average isolation accuracy (Acc.), isolation precision (Prec.) and isolation recall (Rec.) for each isolation network split by fault type, in percent; Individual networks are averaged.

Dataset	Naive Network			Individual Networks			Combined Networks			Kalman		
	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.
Closed Faults	2.57	2.49	2.71	35.74	52.77	35.11	49.61	54.19	48.63	99.30	99.27	99.33
Open Faults	2.73	1.37	2.78	89.98	91.54	90.01	99.97	99.97	99.97	0.87	0.85	0.87
Complete Dataset	2.66	2.46	2.76	64.46	72.84	64.38	76.28	78.93	76.08	45.98	47.30	46.61

As expected, the naive network shows little to no change in its performance. The accuracy remains at approximately 2.5%, close to the value of $\frac{1}{36}$ one would expect when a dataset contains 36 different classes and the output is always the same regardless of input. The little variation there is between the open and closed faults can be traced back to different fault times, leading to different amount of time steps per simulation and therefore an imbalance in the dataset, despite there being an equal amount of simulations for either case.
The individual networks show a clear difference in the isolation of closed and open faults in all quality measures. the closed faults are much more difficult to isolate, with only an average isolation accuracy of 35.74% for the individual neural networks. The precision and recall are not much better, with the precision reaching a value of 52.77% and the recall 35.11%. For the open faults on the other hand, the networks show significantly better performance. The performance measures are close to 90% for accuracy, precision and recall, with the precision reaching up to 91.54%.
From Table 5.6, the quantitative increase due to the combination of the individual network output into a single output can be seen. For the closed faults, this mainly corresponds to an approximate increase of 15% for isolation accuracy and an almost 20% increase in isolation precision. For the open faults, the combination leads to an approximate increase of 10% across all quality measures, raising all of them to 99.97%. This shows that at least for the open faults, this isolation method is very reliable.
The table also shows the performance of the Kalman filter split across the fault types. As expected and explained in Section 4.2.2, almost all closed faults are correctly isolated, while almost all open faults are incorrectly isolated. It is not clear, why there are a few cases which do not follow this general trend.
The quality of the isolation networks also depends on the fault intensity. The isolation accuracy of the combined network for either fault can be seen in Figure 5.20.

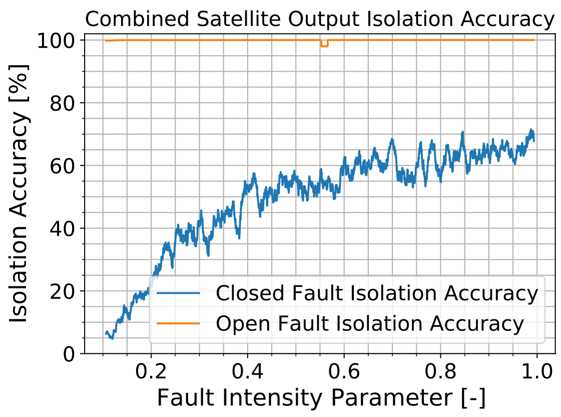


Figure 5.20: Combined satellite isolation accuracy split by fault type vs fault intensity.

It can be seen that the isolation accuracy for the open faults seems to not depend on the intensity. However, despite a very noisy signal, the closed fault isolation accuracy shows a clear downwards trend, dropping below

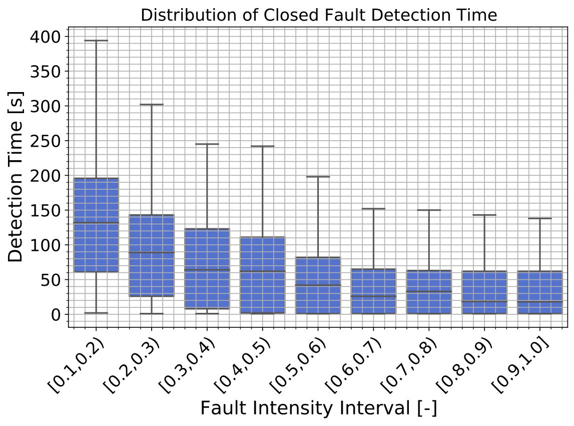
10% isolation accuracy at the lower fault intensity bound of 0.1.
The Kalman filter in fact does not show a decrease with the fault intensity for either closed or open faults, hence a figure is omitted.

5.3. Time Response

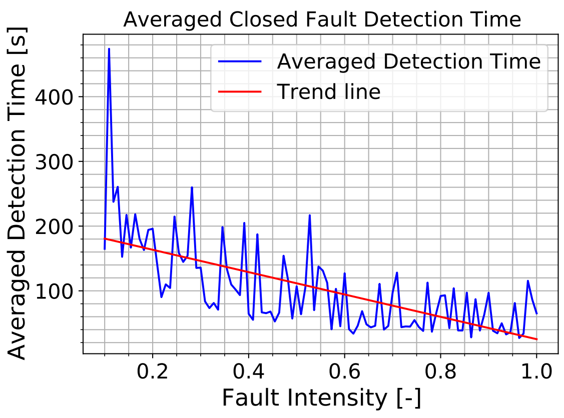
In addition to the overall measures of accuracy, it is also important to evaluate the behavior of the FDI methods over time. That includes how fast a fault is detected, see Section 5.3.1, as well as how consistent the detection and isolation signals are, see Section 5.3.2. Finally, a very brief consideration to the computation time of the two methods is given in Section 5.3.3.

5.3.1. Detection Time

The detection time is defined as the duration between the occurrence of a fault and its detection by the fault detection system. The way detections are derived for each of the methods is described in Sections 4.1.5 and 4.2.2 and the results when evaluated on the test sets are presented here.
Box plots of the detection time for the Kalman filter in relation to the fault intensity parameter can be seen in Figure 5.21a and Figure 5.22a, while line plots of the detection time averaged over all thrusters versus the fault intensity can be seen in Figure 5.21b and Figure 5.22b. Overall, the median detection time for the closed thruster faults is 61.0 s, while for open faults the median time is only 1.5 s. For the closed fault cases a downward trend of the detection time with increasing fault intensity can be seen. This is to be expected as the effect on the residual is directly dependent on the magnitude of the fault. Thus, the likelihood ratio test takes longer to reach the predefined threshold necessary for detection to occur when the intensity is lower.



(a) Box plot of Kalman FDI response time to closed thruster faults.

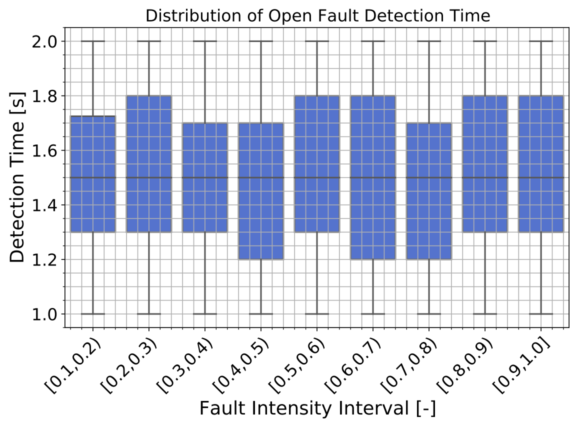


(b) Line plot of Kalman FDI response time to closed thruster faults.

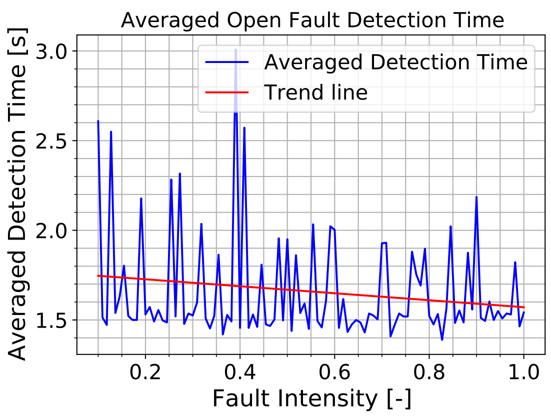
Figure 5.21: Kalman filter response times to closed thruster faults.

This is especially prevalent in the case for closed faults, as seen in Figure 5.21, where the detection time ranges from close to 1 second, i.e. immediately after the fault occurrence up to 400 s. On the other hand, almost all detections of the open faults occur within 1 or 2 time steps. While a very slight downward trend can also be seen for the open faults, considering the box plot in Figure 5.22a, it is possible that this is simply the result of random chance.

For the naive detection network, a false detection occurs somewhere before the fault actually begins for the vast majority of simulations, and therefore by the method to determine the detection time described in Section 4.1, the closed fault median detection time is -1653.0 s, while the open fault median detection time is -1172.0 s. Considering that almost 25% of the naive network output are false positives, this is not particularly surprising. This shows further that the naive network seems to not react to a change in the input in the way one would expect and simply seems to ’guess’, leading to a high recall, but very poor performance in the detection time and precision.
The detection times of the individual networks are shown in Figure 5.24. The overall median detection time for the closed Faults of the Individual Detection Network is 257.5 s, while it is only 11.0 s for the open faults.

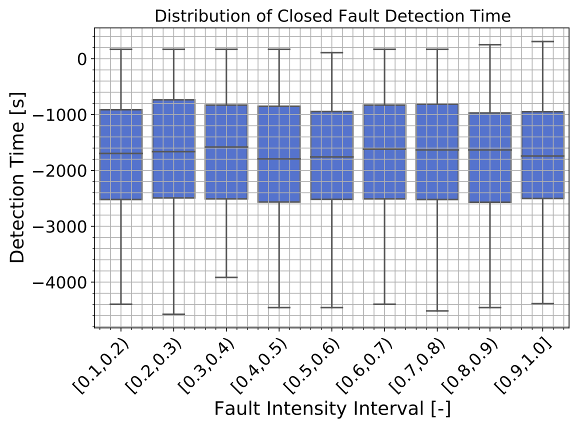


(a) Box plot of Kalman FDI response time to open thruster faults.

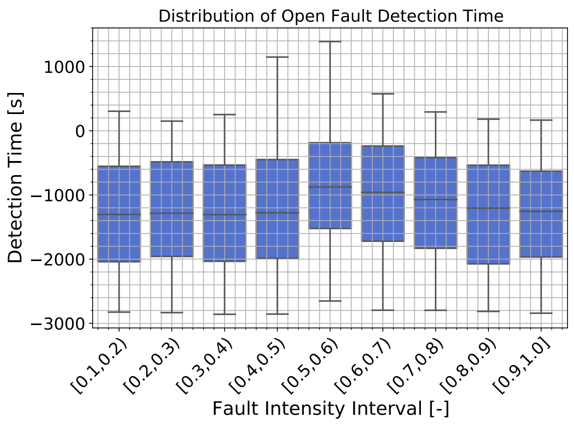


(b) Line plot of Kalman FDI response time to open thruster faults.

Figure 5.22: Kalman filter response time to open thruster faults

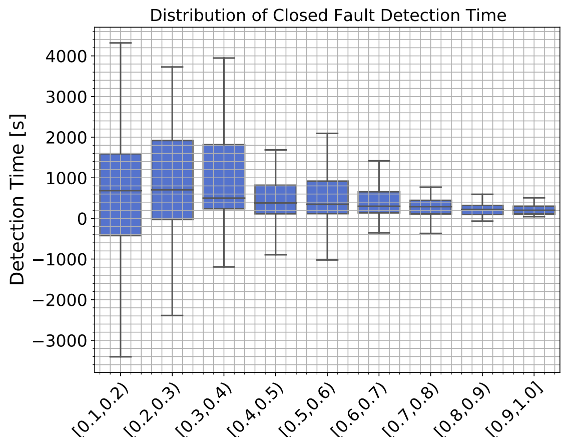


(a) Box plot of the naive network FDI response time to closed thruster faults.

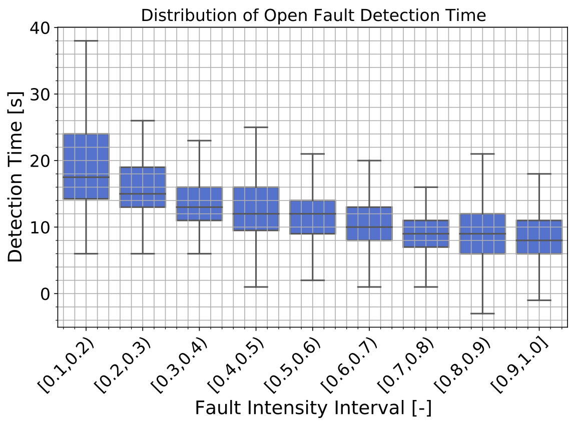


(b) Box plot of the naive network FDI response time to open thruster faults.

Figure 5.23: Naive network box plot of response times.



(a) Box plot of the individual network FDI response time to closed thruster faults



(b) Box plot of the individual network FDI response time to open thruster faults

Figure 5.24: Individual network box pot of response times.

The detection times with regards to the open faults show decreasing trend of the detection times with increasing fault intensity, with most detections occurring between 6 and 24 seconds. Almost all detections times are positive, meaning that there are few false positive detections. The closed fault behavior on the other hand is peculiar at first glance. While for the highest fault intensities (i.e. > 0.9) the detection times are generally above or at least close to 0 with a median of around 200 s and a relatively tight distribution, there is both an increase in the median detection time as well as a widening of the distribution for the lower fault intensities, with the lowest fault intensities having a difference between the upper and lower quartile bounds of approximately 2000 s. This indicates an increase of false positives with decreasing fault intensity. However, as these false positives are flagged before a fault has occurred, i.e. before the fault could have had any impact on the input data to the networks, there should be no reason as to why the network shows a difference based on the fault intensity. The fault intensity should only have an impact after the fault occurred, potentially in an increase in the detection time, as seen for the Kalman filter in Figure 5.21a.
However, there is an explanation of this behavior. As could be seen in Figure 5.7, the performance drops significantly towards the lower fault intensities. This is due to increasing detection times, to the point where a lot of faults are not detected at all. These undetected faults are not included in Figure 5.24 (as without a detection there can be no detection time) and therefore the amount of samples in each of the box-plots is not equal. In fact, the uppermost intensity range of 0.9 to 1.0 contains 182 cases where the network detects a fault. However, the lowest range from 0.1 to 2.0 only contains 30 supposed detections.
As Figure 5.25 shows. The number of false detections which result in a negative detection time does not increase as the fault intensity decreases, unlike what Figure 5.24a would imply. However, the relative amount of false detections goes up, to the point where below some fault intensities only detections due to random fluctuations are left. This results in the detection times being negative for a large percentage of the detections.

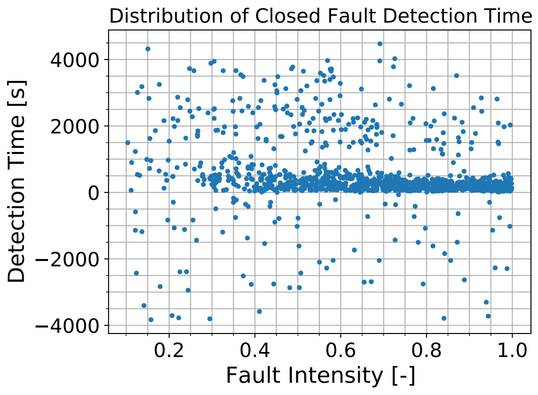


Figure 5.25: Scatter plot of detection times of individual detection network for closed faults vs fault intensity.

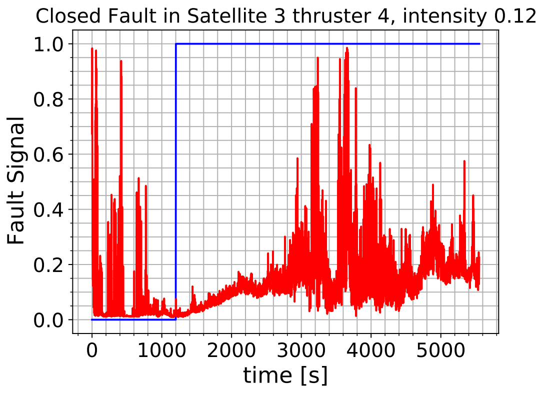


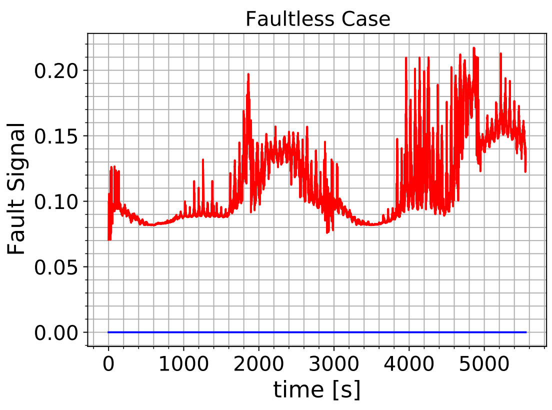
Figure 5.26: Time response of the individual detection to a closed fault with low fault intensity.

Figure 5.26 shows one example of such random false positive detections. As can be seen, there are multiple spikes which exceed the detection threshold. the first of these is at approximately 60 s, but there are more before and after the fault actually occurs. It is impossible to say why the network spikes at these particular times, but they are the cause of the false positive detections.

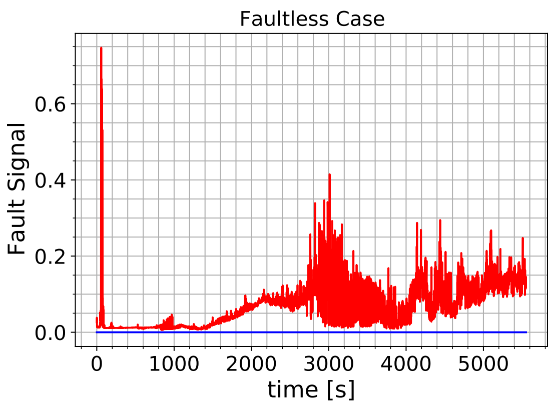
5.3.2. Signal Stability

Another important measure is how stable the detection or isolation signal is over time. Ideally, the signal perfectly reflects the true state. The stability of the detection network is mainly dependent on the way the continuous output of the network is interpreted into the discrete decision of "fault" or "no fault". The actual response of the two trained detection networks can be seen in

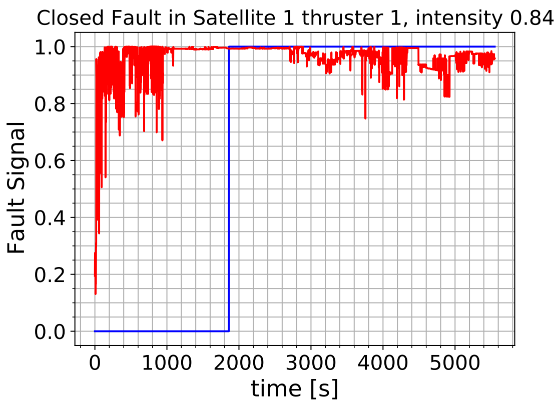
As can be seen in Figure 5.27, there are fluctuations in the signal, especially for the closed fault case. The main reasons this occurs for the close faults is due to the nature of the fault: the effect of a thruster that cannot open fully can only be detected when the thruster is supposed to fire. Due to the attitude of the satellites and their positions in the orbit, there are large periods of time (up to 15 minutes) where any particular thruster



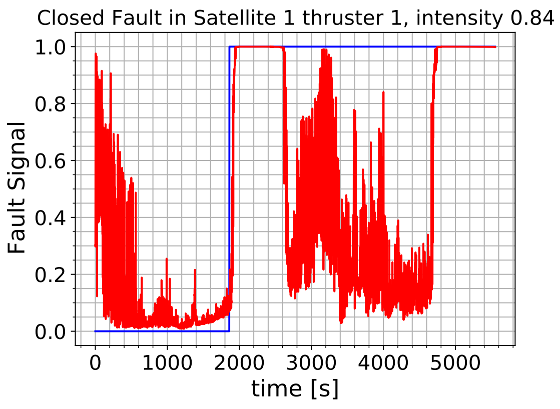
(a) Naive network output in the faultless case



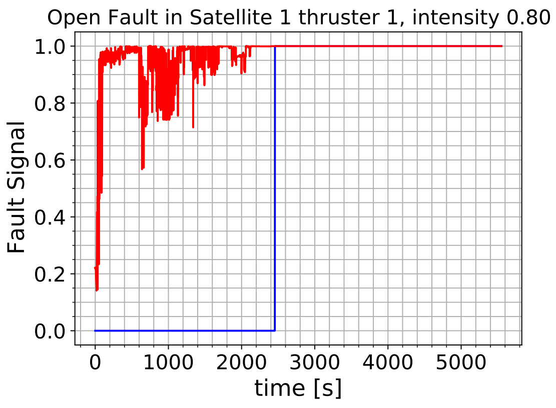
(b) Satellite 1 individual network output in the faultless case



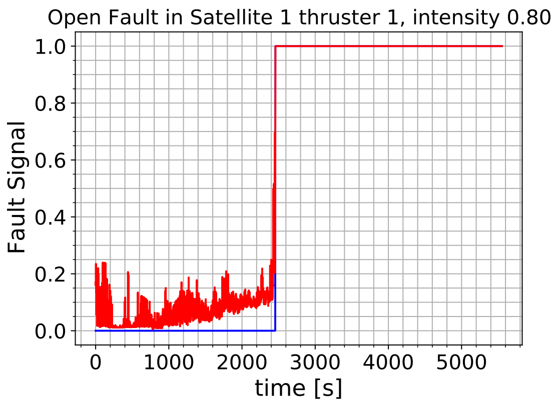
(c) Naive network output in the closed fault case



(d) Satellite 1 individual network output in the closed fault case



(e) Naive network output in the open fault case



(f) Satellite 1 individual network Output in the open fault case

Figure 5.27: Network output and fault status for the detection networks

does not have to fire. In those time frames the control system corrects for the initial deviation and the fault becomes effectively invisible. This can be seen when looking at the relative distance between satellite 1 and its neighbors in the case of a closed thruster fault in satellite 1, shown in Figure 5.28.

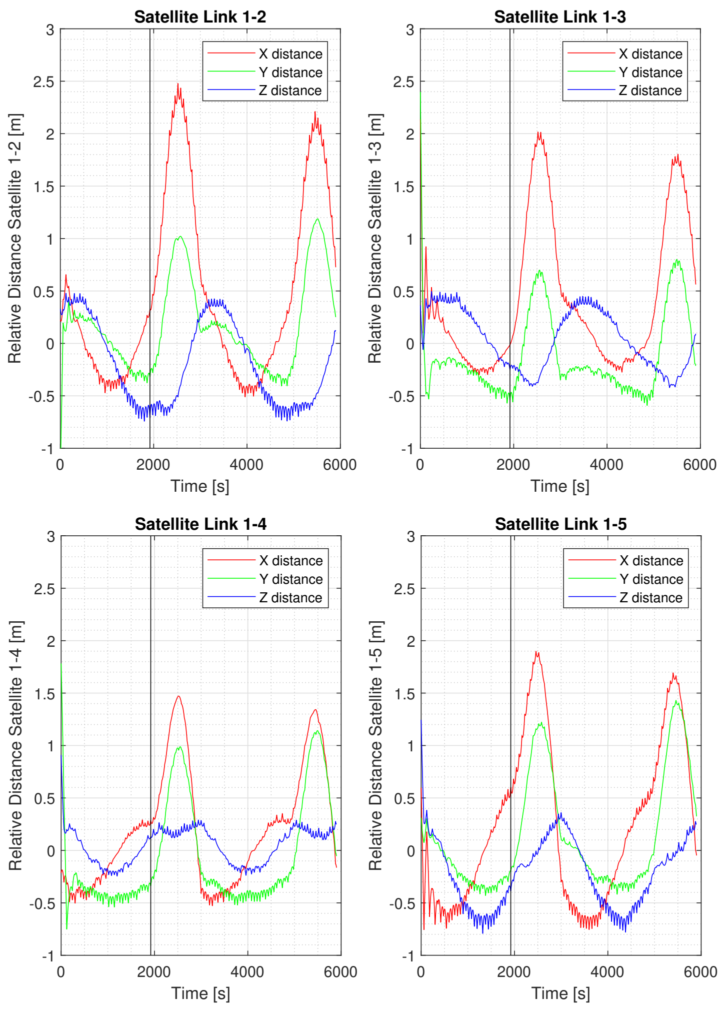


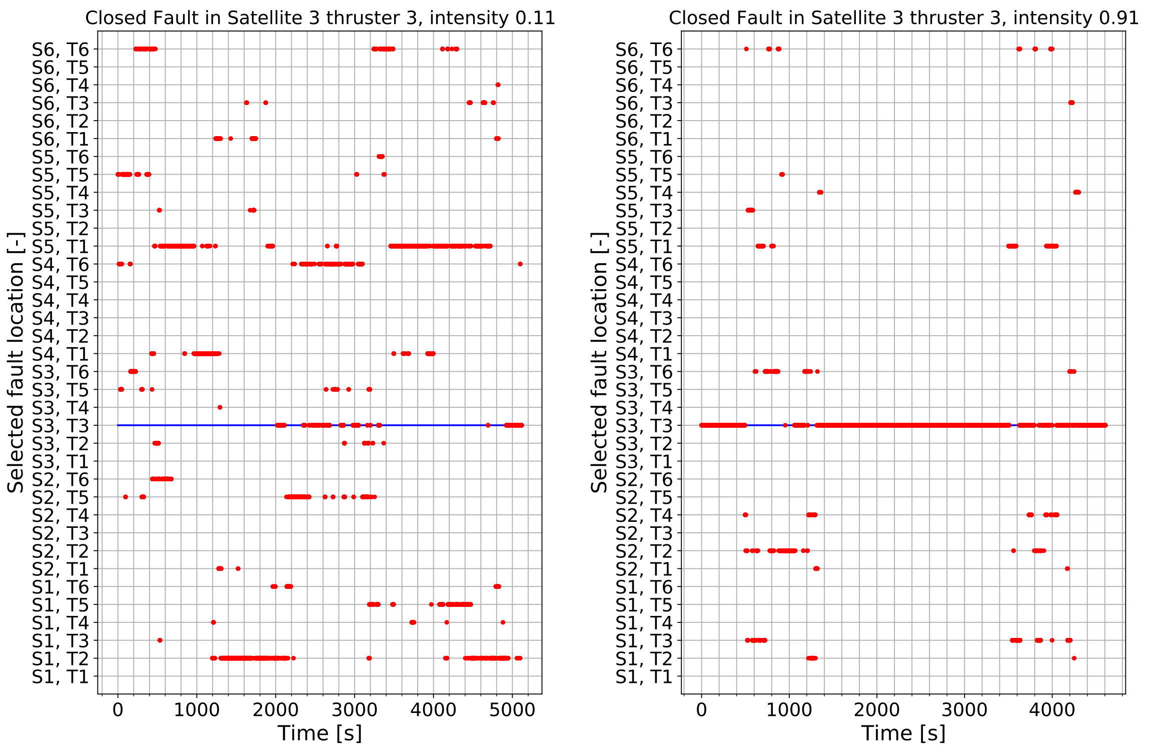
Figure 5.28: Closed fault impact on relative positioning of satellite 1's neighbors, zeroed around nominal distance. The vertical black line marks the occurrence of a fault.

The vertical black line marks the inception of the fault. Immediately it can be seen that the distances start to differ from the usual behavior. However, at approximately the 3000 seconds mark, the relative distance have returned to their usual states. The fault is still in the system, only the effect is no longer visible because the thruster is not scheduled to fire in that window. At around 4900 seconds, almost 3000 seconds (roughly half an orbital period) after the initial fault inception, the thruster is again supposed to fire and the effect of the fault becomes visible again.

However, while this explains the behavior of the individual network response, seen in Figure 5.27d, the naive network response is much more constant, even before the fault occurs in the first place. When comparing Figures 5.27a, 5.27c and 5.27e, it can be seen that this is only the case in the closed and open fault case. And while these three runs may be outliers, the same effect can be seen when inspecting other simulation runs and when considering the detection times of the networks, shown in Figure 5.23. It appears as though the network memorized the closed fault runs and instead of learning how to respond to this fault, instead it aimed for a very high output in the hopes of it being good enough. This is surprising as the same cannot be seen in the faultless cases and the open fault cases, where the network *does* respond appropriately.

In terms of the isolation capability the Kalman filter is very constant. There are no changes in the isolation of a thruster; as soon as one is isolated, that thruster is the choice of the network for the remainder of the simulation.

The isolation networks, however, do show changes over time. Only the time behavior for the combined neural network is presented here as the naive network always shows the same behavior. As could be seen from Figure 5.20, the isolation accuracy of the combined network shows almost no dependence on the fault intensity when it comes to isolating the open type faults. However, a clear dependence was visible for the closed faults. Figure 5.29 shows the response (i.e. the highest probability fault location) of the combined neural network output so a closed fault in thruster 3 of satellite 3.



(a) Combined network output for a low intensity closed fault in satellite 3 thruster 3

(b) Combined network output for a high intensity closed fault in satellite 3 thruster 3

Figure 5.29: Combined network most likely fault location over time in the case of a low and high intensity closed fault

As can be seen on the left, in Figure 5.29a, the network is changing its output very often and even for stretches of time where the signal is more constant, there are still occasional isolations outside the apparent line. For example, the apparently 'constant' stretch of identifications of satellite 1 thruster 2 from 1300 s to 2200 s is comprised of segments of varying lengths where the isolation is indeed "S1, T2" in a row, interrupted by outliers in other thrusters. These segments can be up to 200 s long, however.

The right side, Figure 5.29b shows the response of the combined networks to a more intense fault in the same thruster. It can be seen that the network is correct in its isolation for much longer periods of time. There appear to be two distinct regions where the network becomes less certain. The first starts at approximately 500 s after fault occurrence and lasts until 1300 s, after which the network continuously isolates the fault correctly until the next period of uncertainty. This second period occurs almost 3000 s or half an orbital period later, starting at 3500 s and ending at 4300 s. This periodic behavior is reminiscent of the fault detection networks, which also showed a period of no fault detection for the closed faults. It is interesting however, that despite the relatively short effect of a closed fault, shown in Figure 5.28, which lasts only for approximately 1200 s the isolation network is continuously isolating the correct thruster for a longer period of time.

However, these two examples only show the performance on a single run each. In order to quantify the signal stability, the segment length of continuous isolations for over the fault intensities was determined and can be seen in Figure 5.30. This was only performed for the closed faults. as the combined network has very high performance (>99.9% accuracy) on open faults which is independent of the fault intensity (> 0.1).

As can be seen from Figure 5.30, the lower the fault intensity, the more changes there are on average and therefore the less consistent the output is. This leads to the next question: if the signal changes more often, is the signal more correct the longer it stays on a single thruster? This can be answered by considering Figure 5.31, which show the isolation accuracy on a single simulation run plotted versus the longest continuous isolation output, normalized to the total simulation time.

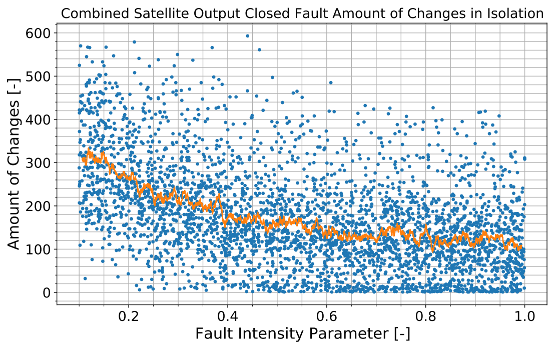


Figure 5.30: Scatter plot of amount of changes in the isolation signal versus fault intensity for the combined network output, orange line represents a moving average of 50 elements

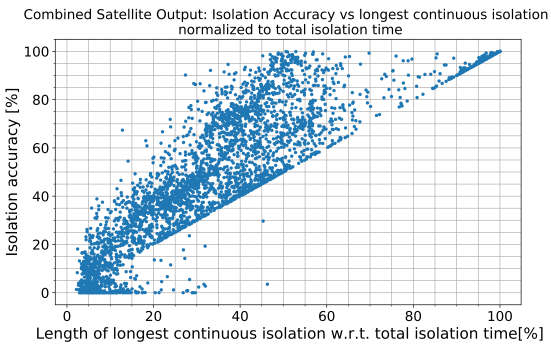


Figure 5.31: Scatter plot of isolation accuracy on a single simulation run versus the longest isolation segment for that simulation for the combined network output

Immediately it is clear that the longer the signal goes on, the more likely it is to be correct. Furthermore, the figure shows a clear lower bounding line for the accuracy. This line represents cases where the output stays on the correct thruster for the longest time compared to any other thruster output, but then never returns back to the correct output after switching. All points above that line show cases where the longest continuous isolation is correct, there were some switches to other values, and then another period where the isolation was correct.

There are two outliers above the 40% range, but in general this trend only starts to break down below the 30% line and then completely unreliable below 10% of the time. This can be used to get a measure of reliability of the isolation. By simply keeping track of the longest continuous isolation so far and the total amount of time, it is very likely that the isolation of the so far longest continuous stretch is correct if it is longer than 40% of the time so far. This will of course only become accurate for longer stretches of time.

5.3.3. Computation Time

The computation time for the two methods is also an important measure, especially in the space engineering context, where every bit of performance matters.

The times for the networks and the Kalman filter were compared on the local laptop PC, with an Intel i7-4910 MQ processor. The isolation networks take approximately 550 ms for a full batch of 4096 sets of input data, averaging to 0.13 ms for a single input evaluation. The Kalman filter, on the other hand, takes 8.96 ms for a single evaluation.

This shows that the networks are considerably faster than the Kalman filter. However, such a comparison should only be taken as an indication of the relative times (i.e. which one is faster), as there are multiple flaws with this comparison. First, the programs, while run on the same hardware, used different programs (Python vs MATLAB) with differing levels of code optimization. Also, the hardware used in an actual spacecraft might be optimized to handle different operations compared to a consumer processor, thus affecting the computation times.

A further investigation of these times is recommended before a definitive, quantitative answer on the relative computation time can be given. This could include an exhaustive bench-marking test or an in-depth analysis of the precise computations involved in each method.

5.4. Robustness

In addition to checking for the expected fault cases which were used to train the ANN approach as well as tuning the Kalman Filter, additional fault scenarios will be examined to judge the robustness of both approaches. The following scenarios are examined:

1. **Decreased fault intensity** (Section 5.4.1). In order to test the lower limits of detectability and isolability a dataset of 100 runs for both the closed and open fault type was generated to cover the range of fault

intensities of 1e−5 to 1e−1. Due to time restrictions, this was only performed for a single thruster, specifically satellite 1 thruster 1.

2. **Lower maximum-thrust capability** (Section 5.4.2). A lower thrust changes the thrusting behavior of the satellites in the formation and therefore might present a challenge to isolate for a network which is used to short bursts of thrust. In this scenario, the thrust was reduced by a factor of 100. Similar to the low fault intensity scenario, 100 runs each were conducted for both closed and open faults, but only for satellite 1 thruster 1.

3. **Increased noise level** (Section 5.4.3). Neural networks often have issues with noisy inputs and the Kalman filter is also dependent on the noise level. In order to see the effect of the noise on the FDI quality, the biases and standard deviations of the noise distribution were increased tenfold. Five runs were performed for each thruster on each satellite with fault intensities at [0.2, 0.4, 0.6, 0.8, 1.0], for a total of 360 runs.

4. **Range based navigation** (Section 5.4.4). As the current simulation of the navigation is very simplified, the noise follows a Gaussian distribution in each direction of the ECI coordinate system, with a constant bias vector. This is not necessarily accurate, as the noise in all directions is equal and uncorrelated. In this scenario, the navigation is based on range and angle measurements which are subject to biases and noise and from which the relative position is reconstructed. Five runs were performed for each thruster on each satellite with fault intensities at [0.2, 0.4, 0.6, 0.8, 1.0], resulting in 360 simulation runs.

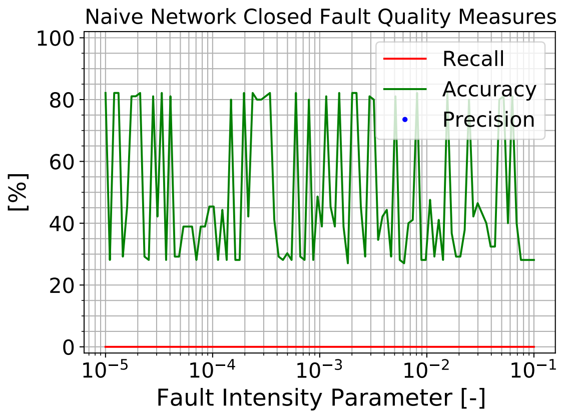
5. **Multiple simultaneous faults** (Section 5.4.5). Currently, both the neural network based methods and the Kalman filter are only designed to detect and isolate a single fault. In this scenario, an open fault is injected into thruster 1 of satellite 1 at a random time and 100 seconds later another open fault is injected into another thruster. The output of the networks and the Kalman filter will be analysed in more detail than before to see if the methods are easily extensible to a multi-fault scenario. Ten runs were performed for each thruster and satellite, except for the first thruster on satellite 1 as it contains a fault anyway in all scenarios. This results in 350 simulation runs. The fault intensity here is spaced evenly in the range [0.5, 1].

As it was found that the naive isolation network does not respond any differently to any of the scenarios, it will not be analyzed beyond a general overview in Table 5.9, found in Section 5.4.6.

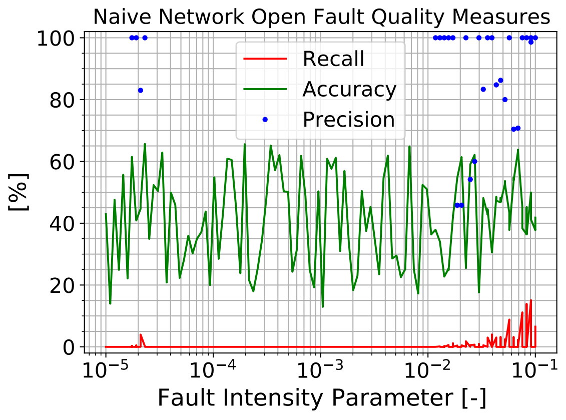
5.4.1. Decreased Fault Intensity

An interesting aspect to investigate is the lower bound of detectability and isolability for each considered fault type. In order to investigate this, another set of data was generated with fault intensities going below the threshold of 0.1 of the regular dataset. The fault intensities in this set are logarithmically spaced from a lower value of 10^{-5} to 10^{-1}, for a total of 100 simulation runs for either fault type. Due to the need for a relatively fine spacing, the dataset only considers a single faulty thruster for both closed and open faults, in this case the first thruster of satellite 1. The detection quality measures (accuracy, precision, recall) versus the fault intensity for the detectors can be seen in Figures 5.32 to 5.34.

The open network quality measures as shown in Figure 5.32 demonstrate this network's overall poor performance. The figure show those quality measures evaluated over the single simulation run with that particular fault intensity. For both the closed faults (shown on the left) and the open faults (shown on the right), the recall of this detector is close to 0, with an overall average of 0.02%. Therefore, in almost all cases the network does not trigger a detection and hence the accuracy is merely a representation of the proportion of faultless states in the dataset. The precision in the few cases where the fault actually gets detected is close to 100% , with a few outliers in the range from 45% to 85%, but there seems to be no pattern as to whether the fault is detected or not. The average precision for the naive network is 97.24%.
The quality measures for the individual network are shown in Figure 5.33. It can be seen that there is a noticeable change in the detection behavior over the fault intensities when considering the open thruster faults. It can be seen that there is a change in all the detection quality parameters over the range $4 \cdot 10^{-3}$ to $1.5 \cdot 10^{-3}$ where the recall drops from close to 100% to below 10% and then stays below or around 5% for the lower intensities. Over the same time frame the accuracy drops in a similar fashion to around 50%. The precision stays near 100% for slightly longer, until approximately $2 \cdot 10^{-3}$, but then also drops and is very inconsistent for the intensities lower than 10^{-3}, similar to the naive network shown above. This same erratic behavior can be seen for the detection of the closed faults. It seems that this low recall with high precision erratic behavior

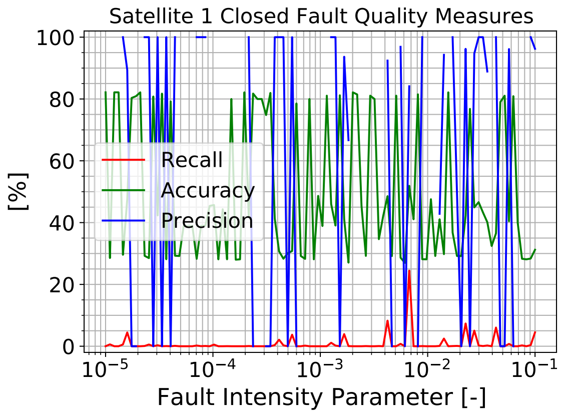


(a) Closed fault quality measures versus fault intensity

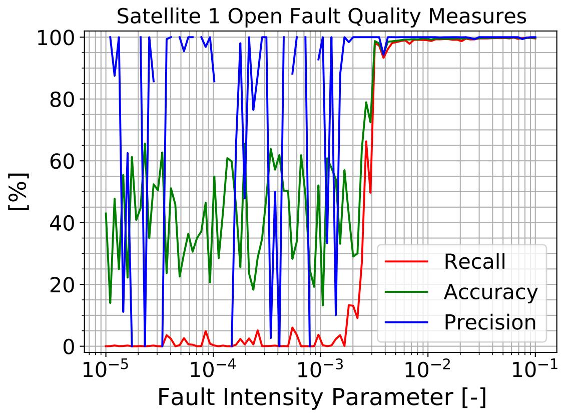


(b) Open fault quality measures versus fault intensity

Figure 5.32: Naive network quality measures (accuracy, precision, recall) versus fault intensity for the low fault intensity scenario

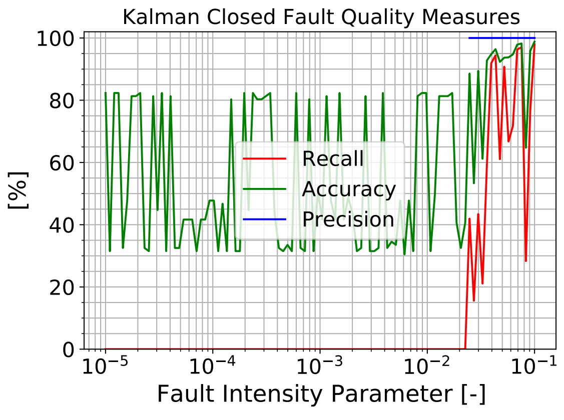


(a) Closed fault quality measures versus fault intensity

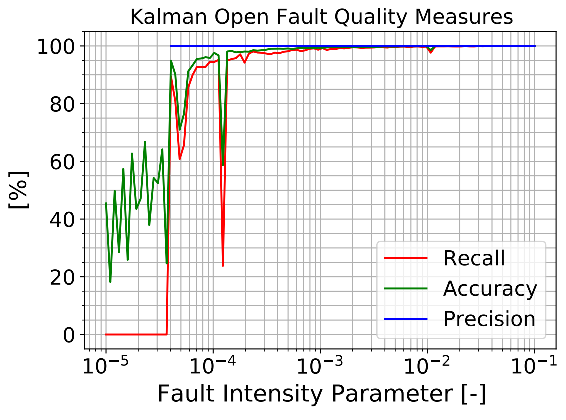


(b) OpenfFault quality measures versus fault intensity

Figure 5.33: Satellite 1 network quality measures (accuracy, precision, recall) versus fault intensity for the low fault intensity scenario



(a) Closed fault quality measures versus fault intensity



(b) Open fault quality measures versus fault intensity

Figure 5.34: Kalman filter quality measures (accuracy, precision, recall) versus fault intensity for the low fault intensity scenario

is present when the fault is not intense enough to be detected.
In general it can be said that below 10^{-3}, the network output only very rarely reaches above the detection threshold and therefore is unable to detect the open faults below that threshold reliably. Consequently, the accuracy is approximately 50% as the faulty and faultless cases are approximately equally well represented in the dataset.
The Kalman filter performs considerably better for the low fault intensities, as visible in Figure 5.34. For the closed faults the recall drops to 0 at a fault intensity of approximately 2.1e−2, considerably lower than for the neural networks. After this value, the Kalman filter does not produce any positive results and as such, the precision is not defined and therefore the line ends. For the open faults, this point is at a much smaller value. There is an initial drop in performance at 1.2e−4, but the point where the detection actually stop is at 3.9e−5. This is almost two orders of magnitude lower than for the better of two tested neural networks. In terms of its detection times, for the faults it does detect, they rose considerably. For closed faults, the median time rose to 369.0 s, while for open faults it rose to 21.1 s.
The networks on the other hand, the naive network in particular, showed much worse in performance in terms of detection times. The naive network detects no closed faults and of the open faults it detects, the median detection time is 452.5s. The individual network shows many false positives for the closed faults, resulting in a median detection time of -631.0s. Comparatively, the open faults median detection merely increased to 35.0 s.
In terms of isolation capability, the combined network output isolation accuracy compared to the fault intensity is shown in Figure 5.35, while the Kalman filter isolation accuracy can be seen in Figure 5.36.

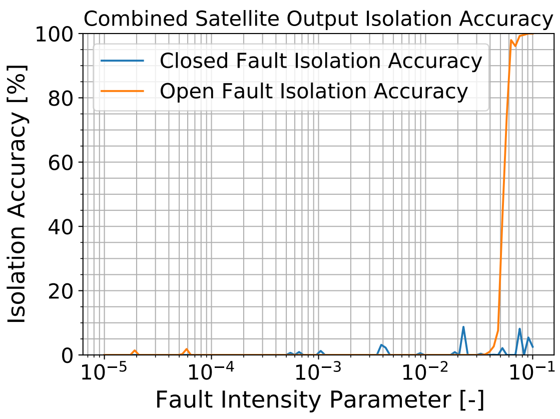


Figure 5.35: Combined network isolation accuracy versus fault intensity for closed thruster faults from the combined networks, in the low fault intensity scenario

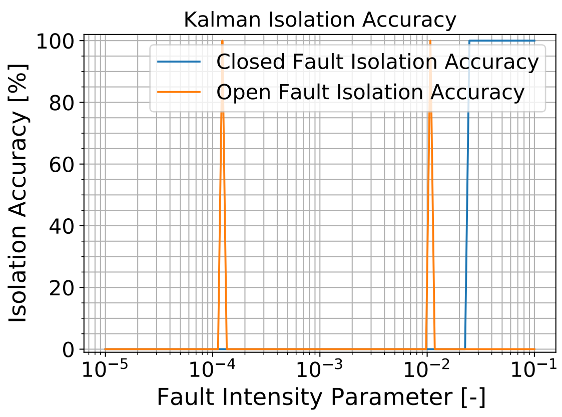


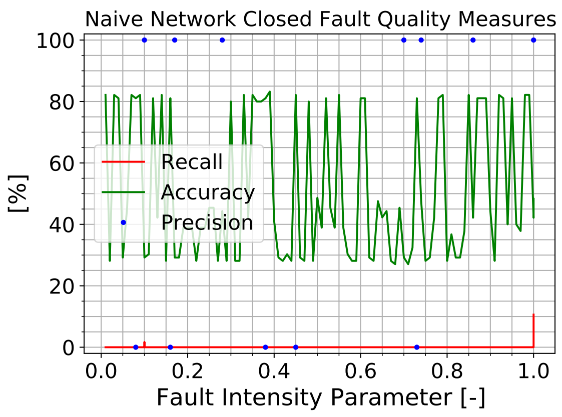
Figure 5.36: Kalman filter isolation accuracy versus fault intensity for open thruster faults from the combined network, in the low fault intensity scenario

The combined network shows a smooth decrease for the closed fault isolation until it reaches 0 at approximately 3e−2. The closed faults on the other hand continue the trend that was already visible in Figure 5.20 and reach 0% isolation accuracy at a fault intensity of 6e−2. Below these threshold, there are only a few outliers during which the faults are still correctly isolated, at least for a small fraction, i.e. only 1% to 10%, of the time.
The Kalman filter isolation performance can be seen in Figure 5.36. It can be seen that even at the lowest detectable fault intensities, the Kalman filter does not change its isolation behavior. Closed faults are correctly isolated until they become undetectable, while the open faults are incorrectly isolated, except for a few outliers. While the open faults are still detectable, the Kalman filter isolates them as belonging to the opposite thruster, as was mentioned in Section 5.2.2.

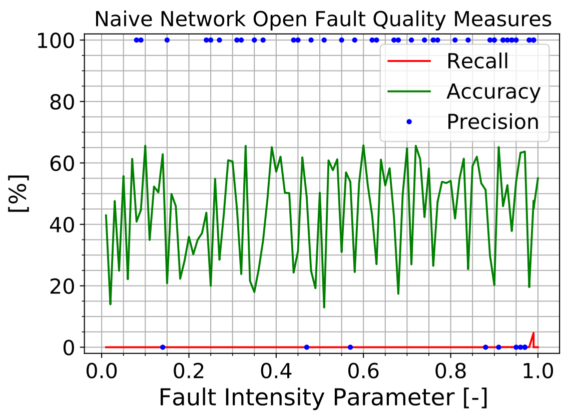
5.4.2. Decreased Thrust Capability

The main satellite characteristic affecting the control performances is the thrust output. In order to see the effect of a lower thrust on the FDI performance, the maximum possible thrust of the satellites was reduced from 4 N to 0.04 N, a factor of 100 lower. With regards to the FDI performance there are two effects to consider: the first is the overall lessened effect of the fault intensity parameter. As said parameter is a measure relating

the faulty condition to the nominal one (as described at in Section 3.6), reducing the effect of the nominal thruster value also reduces the effect of the fault for the same fault intensity parameter. For example, a fault intensity parameter of 0.5 for an open thruster fault in the regular 4 N case effectively means an additional 2 N of continuous thrust from the faulty thruster, while in this reduced case it only means an additional 0.02 N of thrust. This effect should reduce the performance of the FDI system in a similar way as the low intensity scenario shown above. Due to this, a relatively fine spacing is necessary to properly account for the reduction in the thrust and therefore this set also only contains faults in a single thruster.
The second effect is the change in the formation dynamics. Due to the lower thrust, the thrust times increase to deliver the same amount of ΔV. The average opening time per firing increases from 0.114 s to 11.3 s. While in the faultless case, this effect is not too large, since the thrust times are very small to begin with, the effect in the faulty case is more noticeable as the formation is less capable of accommodating an additional force disturbing the formation. This effect should increase the impact of any given fault, as recovering from it is more difficult.

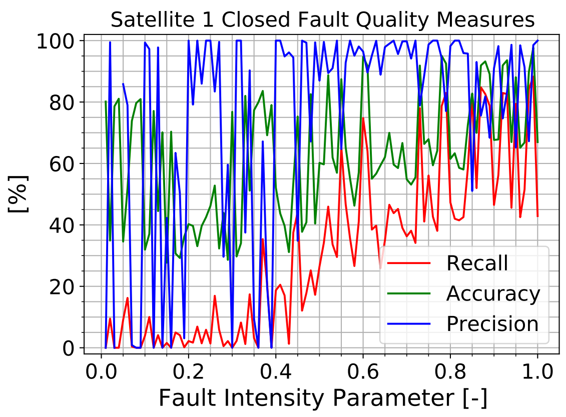


(a) Closed fault quality measures versus versus fault intensity for the low thrust scenario

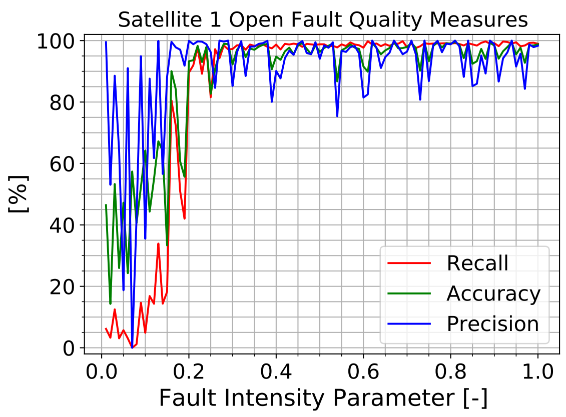


(b) Open fault quality measures versus fault intensity for the low thrust scenario

Figure 5.37: Naive network quality measures (accuracy, precision, recall) versus fault intensity for the low thrust scenario



(a) Closed fault quality measures versus fault intensity



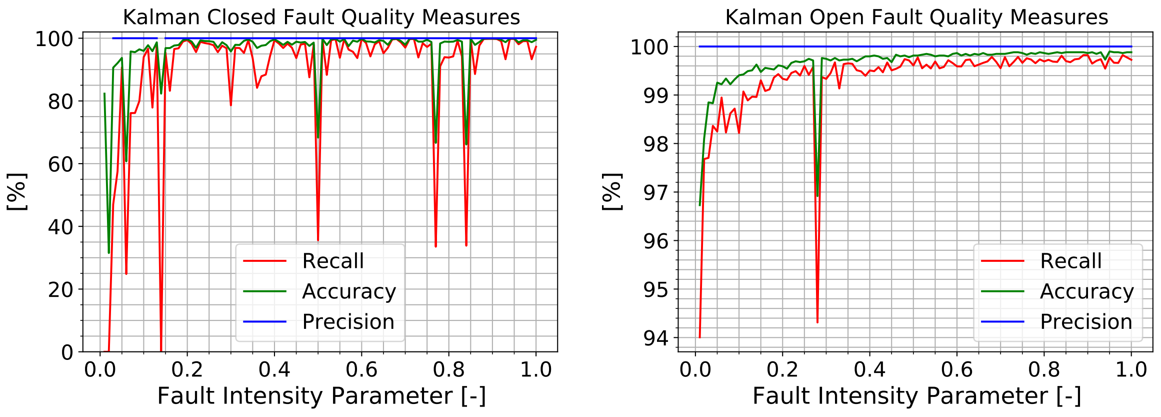
(b) Open fault quality measures versus fault intensity

Figure 5.38: Satellite 1 network quality measures (accuracy, precision, recall) versus fault intensity for the low thrust scenario

When analysing the detection performance, it was seen that the accuracy of the naive approach dropped to 47.7% while the recall dropped to 0.02%, meaning only a very small percentage of faults ever get detected here. Similar to the scenario with the low fault intensities, the precision rose, in this case to 97.2%. This behavior is not surprising as the overall intensity of the thrust decreased by a factor of 100, making them more

difficult to detect. The one simulation run in which a fault does get detected by the naive network it is after 2133 s.
The same pattern can not be seen in the individual network. Not only its accuracy, but also its precision and recall dropped. Comparing the two networks, the individual network has the lesser drop in performance, seemingly adapting better to the low thrust scenario. While it still manages to detect a large percentage of the faults (56.72%), the detection times for the closed faults vary wildly . The open faults show a more steady behavior, however most of the "detections" are registered before the fault has actually occurred, i.e. they are false positives. This was already seen in the behavior of the naive network in the regular dataset where due to the seemingly random fluctuations of the network output signal, it crossed the threshold for a significant amount of time which then registers as a detection.
The detection quality measures for the naive detection network can be seen in Figure 5.37, for closed and open faults respectively, while the individual network detection output can be seen in Figure 5.38.As expected, the naive network detects almost none of the faults and as such the accuracy which it does exhibit is a reflection of the proportion of faultless time steps among all time steps.

For the individual network an interesting behavior can be seen in Figure 5.38. Generally, the decreased thrust capability also reduces the lack of force acting on the satellite in absolute terms, but spreads it over a longer time. This seems to not affect the individual network much, considering that the decrease in recall towards the lower fault intensities is very similar compared to the performance on the regular dataset, which is shown in Figure 5.7a. While for the closed faults the recall drops steadily as the fault intensity decreases, this effect is much more sudden for the open faults (shown on the right). This can be explained by the fact that this fault does not change in its behavior (unlike the closed fault) and as such it is simply a 100 times less intense version of the faults. The point where the performance drops below 10% is at an intensity parameter of 0.1, which approximately corresponds to the same intensity where the drop-off in the low intensity scenario occurred.



(a) Kalman filter quality measures (accuracy, precision, recall) for closed faults versus fault intensity for the low thrust scenario

(b) Kalman filter quality measures (accuracy, precision, recall) for open faults versus fault intensity for the low thrust scenario

Figure 5.39: Kalman filter quality measures (accuracy, precision, recall) versus fault intensity for the low thrust scenario

The Kalman filter also shows a decrease in accuracy and recall (albeit much smaller), but not in precision. The performance of the filter versus the fault intensity can be seen in Figure 5.39. For the closed faults the performance shows an erratic behavior, due to some of the simulations going either completely undetected or having a very high detection time. For the open faults there is almost no change in the performance until the smallest tested fault intensity, which shows a drop in the recall and accuracy.
A similar behavior could also already be seen in Figure 5.34, which is not surprising considering the open faults with low thrust behave very similarly to the open faults with low fault intensity. However, the points where the drop off occur are not simply related by a factor of 100, as might be expected. In the low thrust scenario the first fault intensity parameter where the recall drops significantly is at 0.03. An equivalent fault intensity with the regular thruster is a 100 times smaller, putting the value at 3e−4. When compared to the fault intensity where detections ceased in the default case, which was at 3.9e−5, another order of magnitude is missing.

The decrease in recall indicates that only the time until fault detection increases. In fact, the median detection times for the Kalman increased to 67.0 s and 10.9 s for closed and open faults, respectively. This is a quite considerable increase for the open faults, but only a small increase for the closed faults. This could be due to the effects mentioned at the start of this section. While the fault impact is smaller in absolute terms, the relative effect of the closed faults also increases due to the longer thruster firing time. A more detailed view of the detection times can be seen in Figures 5.40 and 5.41. Interestingly, the time until fault detection for the closed thruster faults shown on the left is quite similar to the detection times in the regular dataset. This behavior is reflected in the recall and accuracy Figure 5.39a. The open thruster faults however show a clear and growing increase as the fault intensity decreases. Considering that the time until fault detection in the regular set was very consistent, this implies that the Kalman filter is starting to reach the lower limits of its detectability.

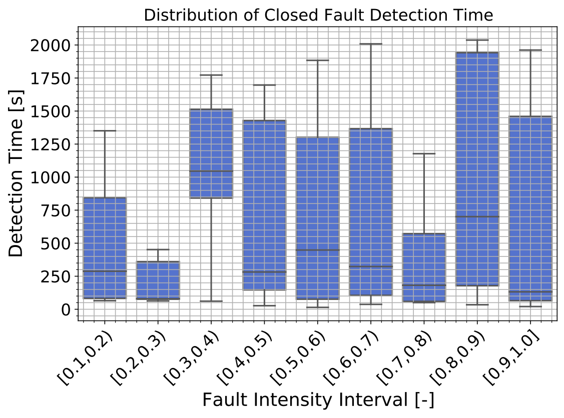


Figure 5.40: Box plot of Kalman FDI response Time to closed thruster faults, in the low thrust scenario

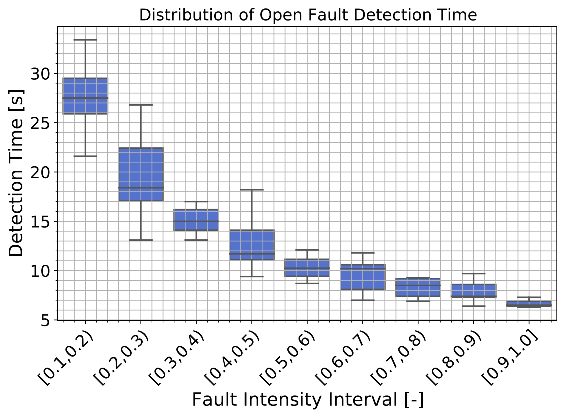


Figure 5.41: Box plot of Kalman FDI response Time to open thruster faults, in the low thrust scenario

The detection neural networks performed considerably worse. As the naive detection network was only able to detect a handful of faults at all, analyzing the detection times in detail is not very fruitful. Of the few fault it detected, the median time for the closed faults was 2725.0 s and for the open faults 3141.0 s.
The individual network, while detecting overall more faults, showed more false positives, indicated by the lower precision. This led to faults being 'detected' before the true fault occurrence, making the median detection times negative. For closed faults the individual network showed a median detection time -1528.5 s, while for open faults this value is at -929.0 s.

Similar to the low fault intensity scenario, the combined network output isolation accuracy can be seen in Figure 5.42, while the same for the Kalman filter can be seen in Figure 5.43.

For the combined neural networks a clear downward trend can be seen for the closed fault isolation accuracy as it reaches 0% at a fault intensity of 0.3. When taking into account that the thruster value is much lower, this made the closed faults actually more isolatable compared to regular simulation. The open fault isolation accuracy reaches this point at a much higher fault intensity, namely at approximately 0.9. Considering the parallels between the low thrust and low fault intensity effects for the open faults, the poor performance with regards to open faults is not surprising. However when comparing against the point where it reached 0% accuracy in the low intensity case ($3e{-}2$) against the low thrust equivalent fault intensity which is $0.9 \cdot 0.01 = 9e{-}3$ it can be seen that the lower thrust made the fault more detectable.
The Kalman filter in comparison shows very stable behavior, with only 3 outliers which break the pattern of perfect isolation for the closed and the false isolation of open faults. However, despite almost always mis-isolating open faults, when they are not correctly isolated they are always attributed to the other thruster of the thruster pair. The thruster pair isolation remains at 100% for the entire low thrust scenario.

5.4.3. Increased Noise Level

A few observations can be made when the noise level experienced by the sensors is increased by a factor of 10. First, the Kalman filter innovation has a greater variance, compared to the nominal case, as can be seen

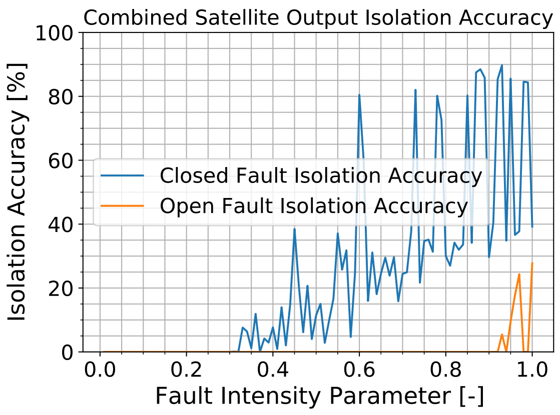


Figure 5.42: Combined network isolation accuracy versus fault intensity for closed thruster faults from the combined networks, in the low thrust scenario.

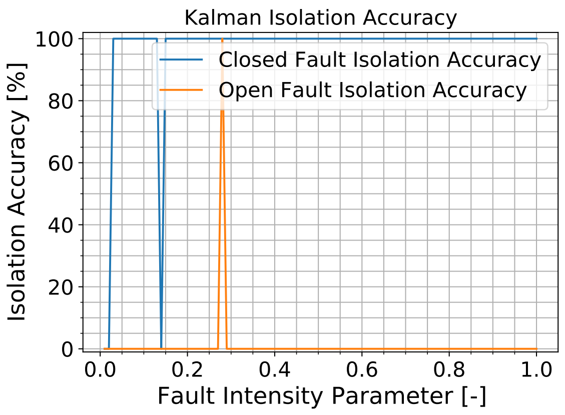


Figure 5.43: Kalman filter isolation accuracy versus fault intensity for open thruster faults from the combined network, in the low thrust scenario.

from Figure 5.44. Since the FDI based on the Kalman filter residual was only enabled after a certain time, the statistical tests on the signal picked up on many faults which were not in fact present due to higher variance.

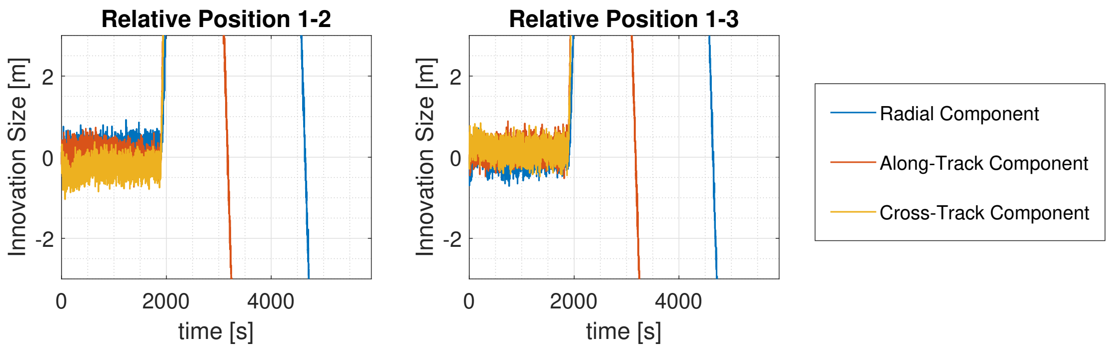


Figure 5.44: Kalman Innovation Vector in the high noise scenario.

This is the cause for the loss in precision of the Kalman filter (dropping from 100% to 69.5%) and the simultaneous increase in the recall, which rose to 100%. Since the Kalman filter now immediately starts detecting faults and never resets to a zero state, it always returns a detection. This raises the recall (no faulty state is missed anymore) but lowers the precision as any positive detection is no longer as reliable as before. Overall, the effect of this on the detection accuracy is a decrease to approximately 75% from 99% and a decrease from 45% to 30% in term of the isolation accuracy.
Furthermore, the detection time cannot be evaluated since the filter tests exceed their threshold soon after it is turned on, leading to negative detection times, similar to the networks. For the Kalman these median detections times are -1910.5 and -1532.4 for closed and open faults, respectively. As the fault inception time for the scenarios are spread over a half period of the orbit, with a 1000 s offset, the median time will be at 1000 + $\frac{1}{2} \cdot \frac{T}{2}$, where T is the orbital period. With an orbital period of 5902 s, this median time is 2475.5 s. Given that the Kalman FDI needs to settle for 1000 s, the expected median detection time if it immediately 'detects' a fault would be -1475.5 s. This of course only holds for the open faults, as the closed faults are not uniformly distributed. Given that the actual median for the open faults is 100 s less, this means that the true median for the fault distribution must be greater. Indeed, when investigating the overall median of the fault inception time, it was found to be 2567.3 s. This confirms that the Kalman filter detects a fault almost as soon as it is turned on.
The neural networks overall experienced a decrease in correct detections. Both detection networks experienced decreased in detection accuracy, but the behavior with respect to recall and precision differs between

the two. The naive network showed an increase in precision compared to the regular set (up to 86%) and a decrease in recall (down to 48%), whereas the opposite is true for the individual network. This network showed a precision decrease to 60% and a recall increase 99%. This pattern is similar to the Kalman filter and it can be interpreted as the network activating much more easily as a result of the increased noise. While this leads to an increase in recall (since fewer faults are missed) the precision drops because more false positives are recorded. Interestingly, the naive network which already showed this kind of behavior for the default set seems to be more difficult to activate, resulting in an increase in the precision and a decrease in recall. This is consistent with the hypothesis that the naive network 'memorized' the regular set to some extent and actually produces more valuable output in a slightly different dataset.
The detection times were also affected by the change in the noise. The naive network median detection times for closed and open faults are at -1307.0 and -692.0, both still negative indicating a false positive long before the actual fault occurrence. The individual network, which generally responded after the fault occurence now also has negative median detection times. For closed and open faults the individual network these values are -2054.0 and -1977.0.
The isolation quality also decreased. An example of the impact of the high noise on the confusion matrix can be seen in Figure 5.45, depicting the differences in the confusion matrix compared to the regular dataset of the individual network of satellite 4. Blue corresponds to an increase in the isolations of that particular field while red denotes a decrease. While all satellites showed a decrease in the correct isolations (i.e. the diagonal of the matrix), there was also a systematic decrease in the wrong isolations for satellites 1, 4 and 6 which for the regular dataset showed a similar pattern of isolations to the naive network, i.e. a consistent misisolation towards a particular thruster. this can be seen in the red streak in the column of the matrix shown in Figure 5.45. However, the incorrect isolations also increased, spread over the remaining thrusters. This led to an overall decrease in the isolation quality. The combined network output decreased in accuracy to 60.61%, while precision and recall dropped to 62.47% and 60.55%, respectively. The increase in incorrect isolations seems to not follow any pattern. It is also not quite clear, why the incorrect isolations of a single thruster systematically decreased.

5.4.4. Range-Based Navigation

In the training setup the noise of the measurement is Gaussian in every direction in the ECI frame, for both the relative position as well as the relative velocity. This is an approximation which does not hold for most navigation methods. For example, in a range-based navigation approach, the relative distance is measured as well as the relative position angle. This causes the noise to both be distributed differently compared to the simple model used in the simulation, as well as to be dependent on the distance, as the relative position is reconstructed from the relative distance and angle measurements. Even small changes in the input can in certain cases radically change the output of neural networks [34, 98], so a change in the noise size and distribution might produce interesting results.
This change in the navigation is simulated as follows. The true distance and angles are calculated from the satellites true relative position. Then, a bias is added to the distance and thw two relative angles, in addition to Gaussian noise. The standard deviation for the radial measurement was selected to be $1.1 \cdot 10^{-4}$ m, while the standard deviation for either angular measurement was $9.8 \cdot 10^{-5}$ radians [99, 100]. The size of the bias was selected to be equal to one standard deviation, randomly selected whether the bias is positive or negative. From this, Equation (5.1) is used to determine the relative position relative to the body frame, which then is transformed to the ECI frame by the known rotation matrix.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = r \cdot \begin{bmatrix} \sin(\theta)\cos(\psi) \\ \sin(\theta)\sin(\psi) \\ \cos(\theta) \end{bmatrix} \tag{5.1}$$

Where x, y an z are the distances in the body frame, r is the measured distance between the satellites, θ is the relative polar angle and ψ is the relative azimuthal angle.
This change overall leads to a decrease in the average error experienced by the navigation system, as shown in Figure 5.46. In addition, the noise is now no longer uncorrelated in the axes of the ECI coordinate system. Finally, the noise is now dependent on the distance, which changes quite drastically in some of the intense fault scenarios and therefore might influence the FDI performance. It should be noted that this only effects the relative position measurement and not the relative velocity measurement.

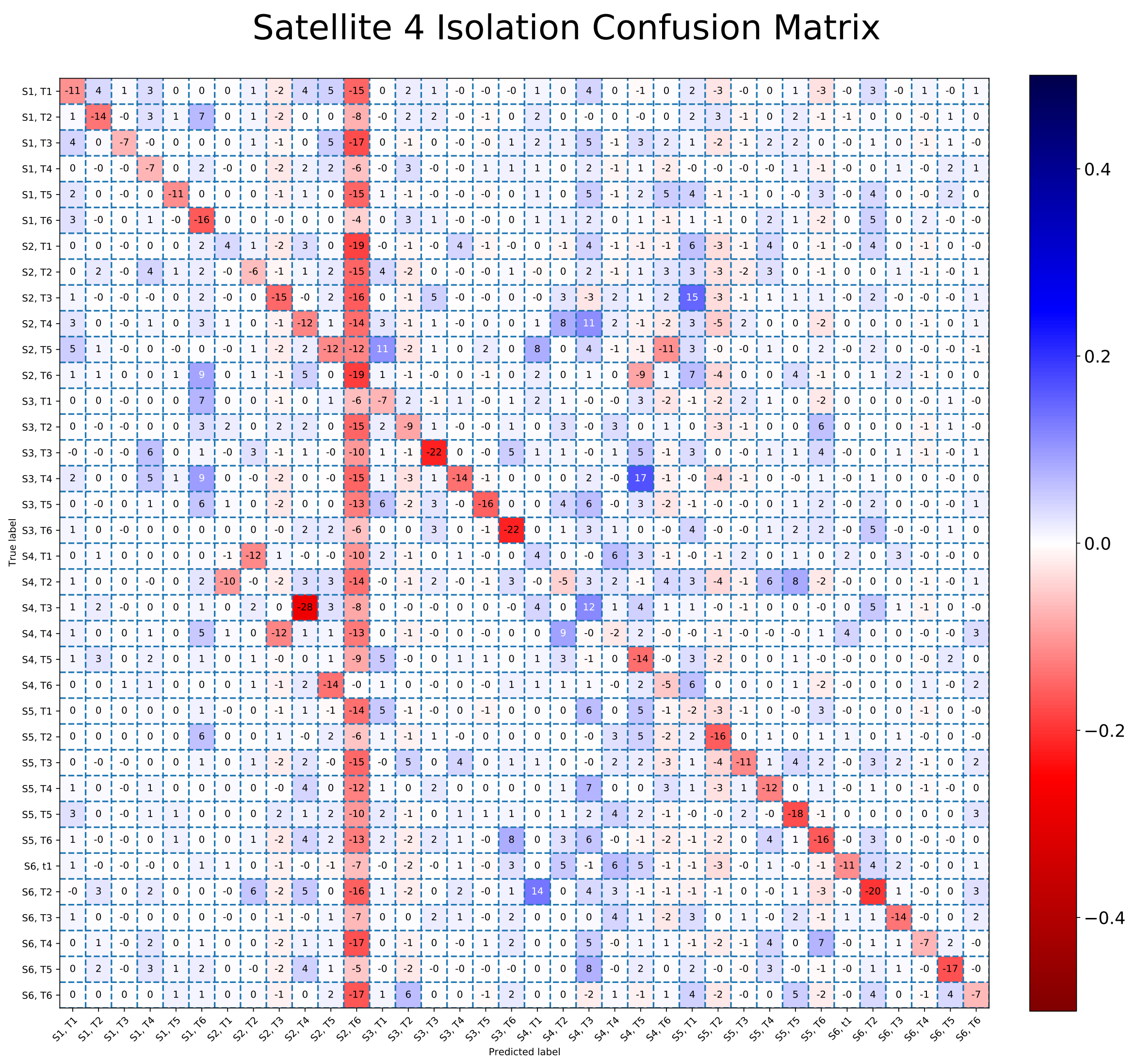


Figure 5.45: Satellite 4 Network Difference in confusion matrix in the increased noise level scenario, numbers in percent.

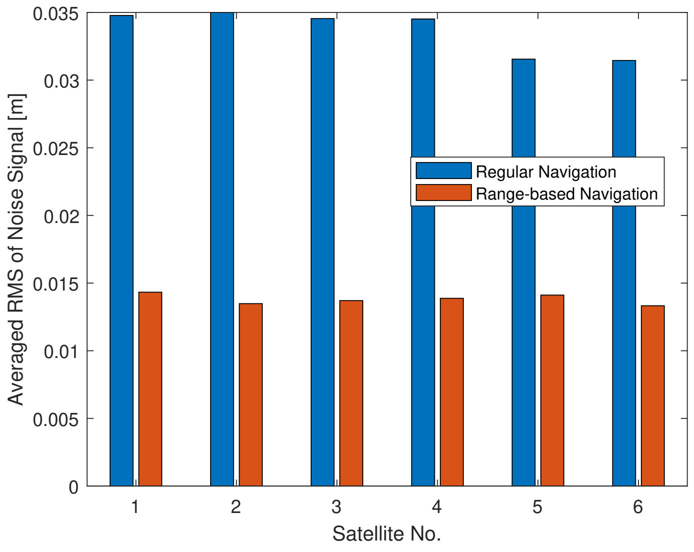


Figure 5.46: Comparison of average root mean square of the distance error of the changed navigation scenario compared to the default simulation.

The effect of this change for the detection networks is varied. The naive detection network experienced a 13 percent point drop to a detection accuracy of 67.33% and an almost 40 percent point drop in recall to 48.74%. while the precision of the naive network to 87.55%.
The individual isolation network shows a different pattern. While its overall accuracy dropped slightly to 80.29%, both the precision and recall rose slightly, rising to 99.94% and 64.94%. One might wonder how it is possible that both recall and precision increase, while the overall accuracy decreases. This can be explained by the fact that the scenarios do not include faultless runs, therefore the proportion of faultless states is smaller. If more correct states are identified as such, recall and precision can rise, but without taking the faultless states into account, these increases do not necessarily imply an increase in the accuracy as they do not incorporate the true negative rate directly.

The Kalman filter shows almost the same performance as for the default set. The accuracy is at 99.01% while the precision is still at 100% and finally the recall is at 98.17%, showing a minimal increase. This almost implies that the detection times are similar and by inspecting the response time, the median detection time for closed faults was found to be still 61.0 s, and for open faults it is still at 1.6 s.
The detection networks however show changes in the detection times. The naive detection network median detection time for closed faults is 3713.0 s, while the open fault median detection time is -609.5 s. This is particularly poor behavior as either the network has a false positive long before a fault, or a correct detection more than 10 times later than the centralized method. The individual network performs better in this regard. Its closed fault median detection time is at 223.0 s and the open fault median detection time is 10.0 s. This performance is not very different compared to the default case. In fact it present a slight improvement consistent with the improvement of the quality measures.

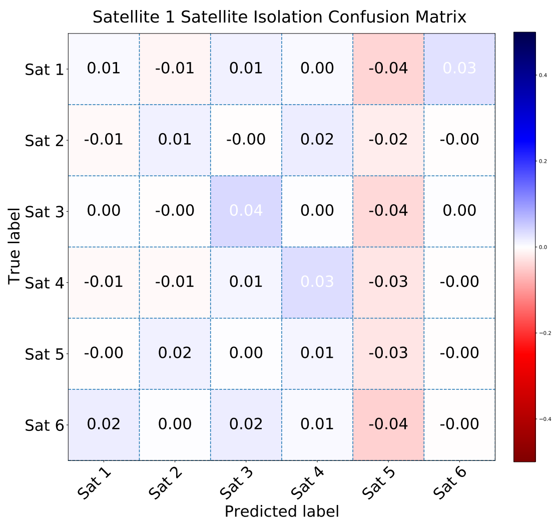
In term of the isolation capability, the overall effect is positive. The difference in the isolation confusion matrices for the individual networks with respect to satellite isolation can be seen in Figure 5.47 and the combined network output in Figure 5.48. The mean of the individual networks in terms of isolation accuracy rose to 66.97% while precision and recall rose to 75.66% and 67.05%, respectively, an average increase of approximately 3%. For the combined network this increase led to an accuracy of 78.82%, precision of 83.39% and 78.63% recall. In the confusion matrices it can be seen that to a large extent the increase in isolations is due to a decrease in isolations in the opposite satellite. However, for satellites 1, 4 and 5 there are also systematic decreases in the false isolations for one particular satellite. As was seen in Section 5.2.2, the naive network and some of the individual networks showed a bias towards one particular output. This seems to decrease with a change in the noise size and distribution. Considering that the combined isolation network already accounted for this bias using the confusion matrix, it is not surprising that this trend does not appear for this network. However, since the reason for this particular bias was unclear in the first place, the reason for its reduction due to the change in navigation is unknown.

5.4.5. Multiple Simultaneous Faults

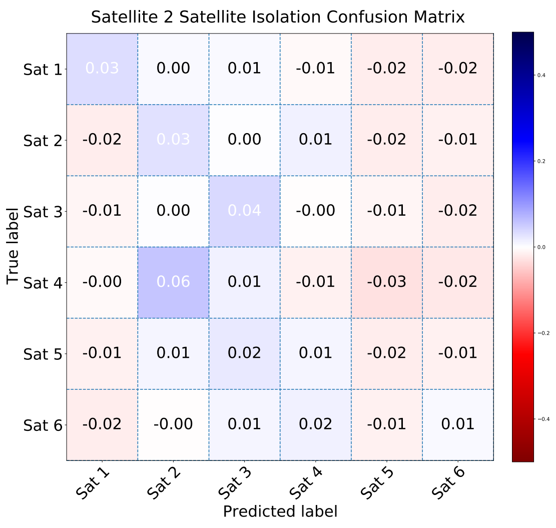
While the methods as described are not intended to diagnose multiple faults at once, their output can be analyzed to see if they could easily be extended to include multi-fault scenarios. For the Kalman filter this involves analysing the vector of likelihood tests and comparing relative magnitudes for the faults involved, whereas for the network the output vector can be evaluated.

The evaluation of the detection performance for this scenario is more straightforward. The accuracy of the naive detection network for the multi-fault case is at 85.43% with a precision and recall of 85.60% and 91.62%, respectively. Considering that this dataset only contains open faults which are generally easier to detect, this represents a performance increase in precision and accuracy, but a decrease in recall compared to the default set. This is surprising as the presence of a second fault intuitively should cause an increase in the recall. This implies that the network does not stay at or near its maximum output as often as was seen in the default case. The individual network performs very similarly to the default dataset when comparing the open fault detection performance. The accuracy is at 99.47% with a precision of 99.70% and a recall of 99.43%, which overall are slight decreases (less than half a percent point), which could simply be due to random variation.
The detection times are also very similar to the default case. The naive network again experiences many false positives, resulting in a negative median detection time of -528.5 s. The individual detection network performs better than the default case with a median detection time of 4.0 s. Interestingly, the Kalman filter also experienced similarly sized decreases to its detection performance. The accuracy of the Kalman filter is at 99.70%, the precision still at 100% while the recall dropped slightly at 99.48%.

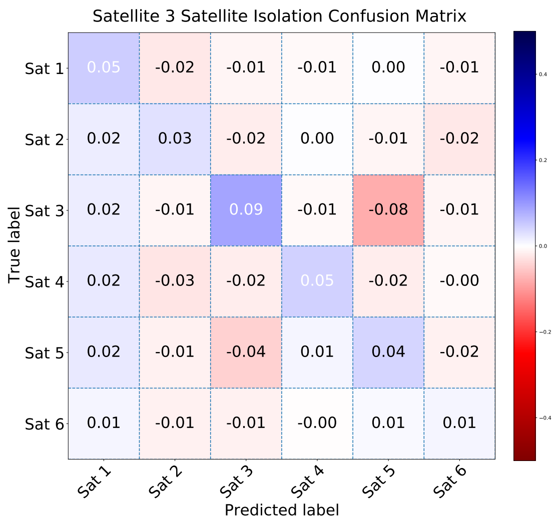
However, investigating the performance of the various isolation methods was the primary reason for creating this dataset. Due to the setup of this investigation, every simulation contains a fault in the first thruster of satellite 1. Therefore, in the confusion matrices presented in this section the vertical axis represents the location of the secondary fault, except for the very first label. The label for true fault in "S1, TP1" in the Kalman confusion matrix or "S1, T1" for the neural network confusion matrix refers to the time steps where the secondary fault has not yet occurred. In addition, the question of how to evaluate the quality measures (such as accuracy, precision and recall) now that there are more than one true fault state have to be answered. In this section, an isolation is considered correct if it isolates either of the two faulty thrusters after the occurrence of the secondary fault. For the precision and recall measures, this means a change of the categories which the isolations are divided into. Instead of calculating the isolation precision or recall for a single thruster fault category, it is calculated for the category "fault in thruster X OR thruster 1 in satellite 1".
First, a general analysis with regards to the selected most likely fault location, similar to the analysis on the default dataset, will be performed, followed by a more detailed analysis of the actual output of each of the methods.
The Kalman filter output is a vector of 36 tests, each of which is based on the fault signature of a specific fault in any of the 36 thrusters, as explained in Section 4.2.2. When a deviation from the mean of the vector signal occurs, these tests will differ from zero and once they exceed a certain threshold a fault detection is triggered. At the same time, the largest of these test values is the most likely location of the fault and is isolated.
The confusion matrix for the thruster pair isolation of the Kalman filter output can be seen in Figure 5.49. As the behavior of the Kalman filter on misisolating the open faults is known, the pair isolation confusion matrix is selected instead of the full isolation matrix.
As can be seen from Figure 5.49, the Kalman filter mainly isolates the fault that occurred first, i.e. the one in satellite 1 thruster 1. This can be seen by the vertical blue strip on the left of the matrix. The diagonal is also inhabited, with the most correct isolations occurring for the thruster pair 2 (i.e. thruster 3 and 4) on satellite 1. Interestingly, the thruster pairs belonging to satellite 6 are completely ignored. No isolations are made for that satellite.
In terms of its overall performance the isolation quality measures for this scenario are 5.16%,54.50% and 6.01% for accuracy, precision and recall, respectively. These low values are due to the known misisolation for the open fault case. However, all of these quality measures are considerably higher than in the singular open fault case. While a few outliers were also correctly isolated in the default dataset, it is not clear why this percentage increased with the addition of a second fault.



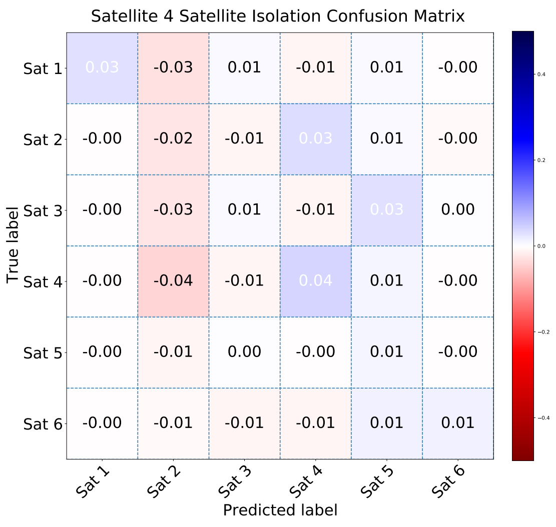
(a) Satellite 1 network difference in confusion matrix.



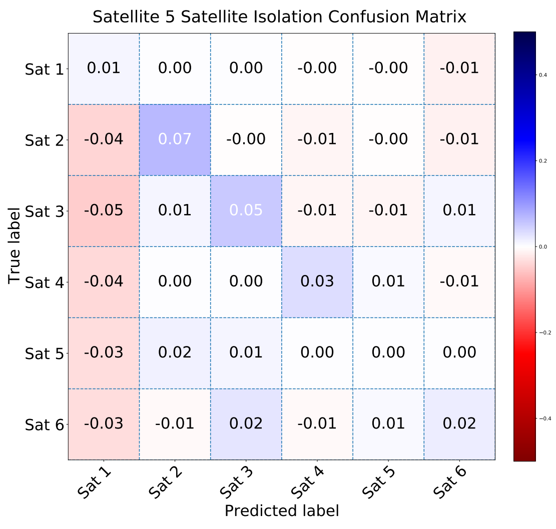
(b) Satellite 2 network difference in confusion matrix.



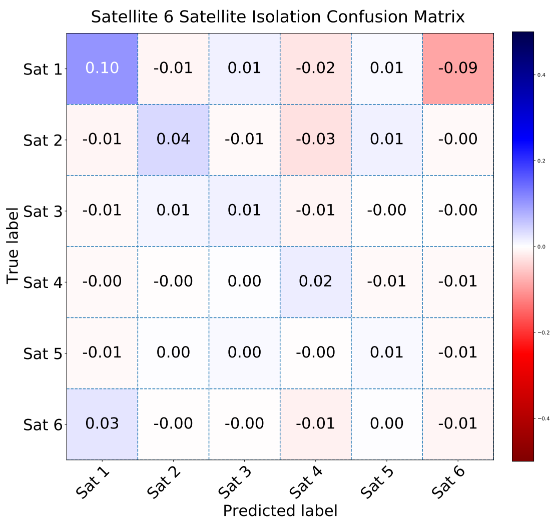
(c) Satellite 3 network difference in confusion Matrix.



(d) Satellite 4 network difference in confusion Matrix.

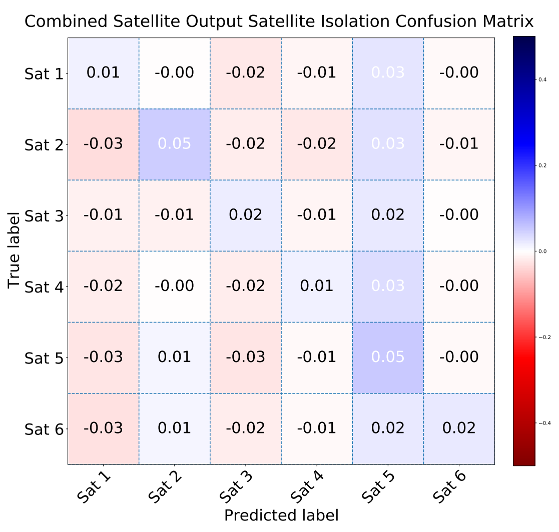


(e) Satellite 5 network difference in confusion matrix.



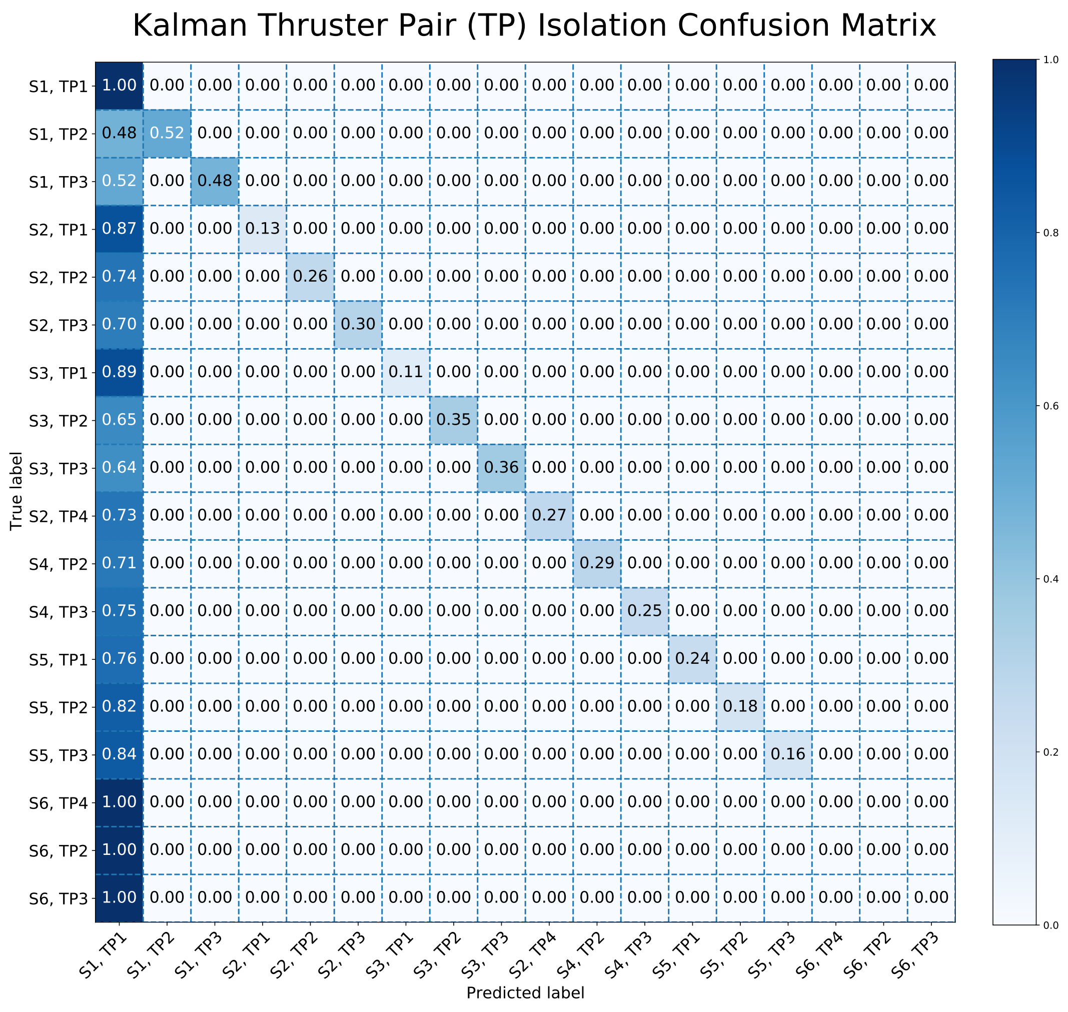
(f) Satellite 6 network difference in confusion matrix.

Figure 5.47: Satellite isolation differences in the confusion matrices for each individual network for the change in navigation scenario.



	Sat 1	Sat 2	Sat 3	Sat 4	Sat 5	Sat 6
Sat 1	0.01	-0.00	-0.02	-0.01	0.03	-0.00
Sat 2	-0.03	0.05	-0.02	-0.02	0.03	-0.01
Sat 3	-0.01	-0.01	0.02	-0.01	0.02	-0.00
Sat 4	-0.02	-0.00	-0.02	0.01	0.03	-0.00
Sat 5	-0.03	0.01	-0.03	-0.01	0.05	-0.00
Sat 6	-0.03	0.01	-0.02	-0.01	0.02	0.02

Figure 5.48: Combined network output difference in confusion matrix for satellite isolation.



	S1, TP1	S1, TP2	S1, TP3	S2, TP1	S2, TP2	S2, TP3	S3, TP1	S3, TP2	S3, TP3	S2, TP4	S4, TP2	S4, TP3	S5, TP1	S5, TP2	S5, TP3	S6, TP4	S6, TP2	S6, TP3
S1, TP1	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S1, TP2	0.48	0.52	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S1, TP3	0.52	0.00	0.48	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S2, TP1	0.87	0.00	0.00	0.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S2, TP2	0.74	0.00	0.00	0.00	0.26	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S2, TP3	0.70	0.00	0.00	0.00	0.00	0.30	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S3, TP1	0.89	0.00	0.00	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S3, TP2	0.65	0.00	0.00	0.00	0.00	0.00	0.00	0.35	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S3, TP3	0.64	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.36	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S2, TP4	0.73	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.27	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S4, TP2	0.71	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.29	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S4, TP3	0.75	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.25	0.00	0.00	0.00	0.00	0.00	0.00
S5, TP1	0.76	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.24	0.00	0.00	0.00	0.00	0.00
S5, TP2	0.82	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.18	0.00	0.00	0.00	0.00
S5, TP3	0.84	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.16	0.00	0.00	0.00
S6, TP4	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S6, TP2	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S6, TP3	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 5.49: Kalman filter confusion matrix for thruster pair isolation in the multiple fault scenario.

The isolation matrix of the combined network output approach can be seen in Figure 5.50. The response seems to be generally similar. The network mainly isolates the fault which occurred first and only selects the secondary fault for a select group of particular thrusters. For satellite 1 thruster 5, satellite 5 thruster 2 and satellite 6 thruster 2 the isolation recall is higher than 90%. Similar to the Kalman filter, the thrusters in satellite 6 are almost ignored in the isolation, except for the aforementioned thruster 2. However, the thruster on satellite 2 and 4 are isolated even less. Only the thrusters in satellite 1 itself are consistently isolated in addition to the original fault. The isolation quality measures are at 97.74%, 98.19%, 97.89% for accuracy, precision and recall. Considering that only open faults are represented in this dataset, this is a decrease of 1 to 2 percent points from the default set. This could be due to the fact that the network is faced with a new signature in the relative positioning data is has not been trained on.

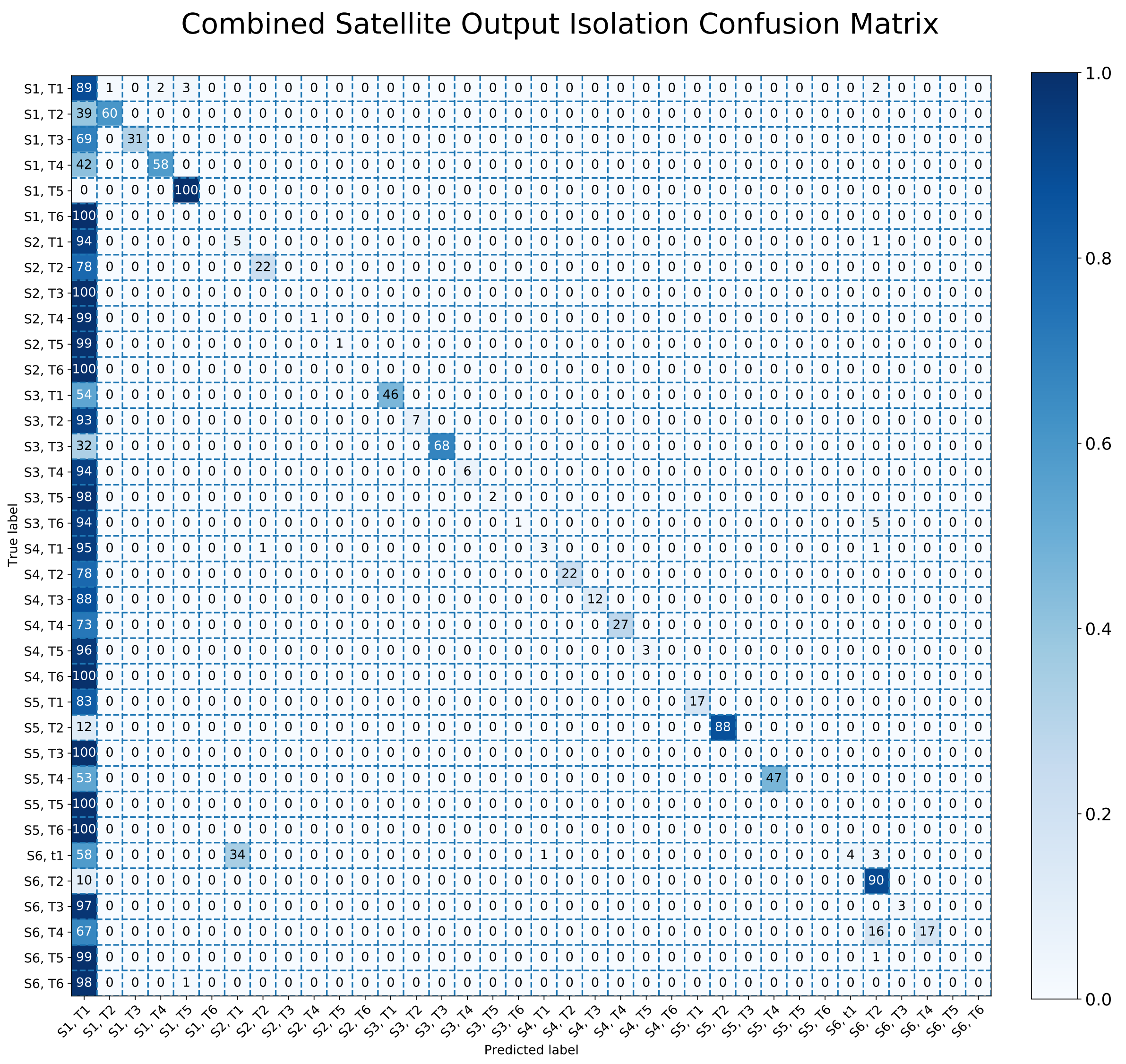


Figure 5.50: Combined network confusion matrix for the multiple fault scenario.

A more detailed analysis involves the components of the output vector of either method. This was done by not only considering the maximum value, but also the next highest components of the output vector. For the Kalman filter, it was found that on average 19.92% of the time the secondary faulty thruster was in the two highest components of the isolation vector. This could be due to the fact that when a fault occurs, the

component for the statistical test corresponding to that fault starts increasing quite fast. The longer the gap between the faults, the more time this value has to increase. Then, when the secondary fault occurs, the test for that fault starts to increase as well or if it had been increasing before it will do so more quickly. However, other components of the test vector also increase at the fault occurrence. They grow more slowly compared to a correct case, but they do have a head-start on the correct test of the secondary fault. This could be the reason for the relatively small percentage of times the secondary fault is within the two highest components of the test vector.
For the combined neural network approach the thruster which experienced a fault second was within the two largest components 75.25% of the time, a considerably larger percentage.

While this is better than guessing a secondary fault, it is not a reliable method to determine if a secondary fault has occurred. Indeed, it is not clear when to consider the second largest component or when to ignore it. In order to see if there is a distinct difference in the relative sizes of the largest component relative to the second largest, their ratio was determined for each time step and split into two groups: The first contains only the time steps with a single fault, while the second group contains all time steps with two simultaneous faults. Box plots of these ratios can be seen in Figures 5.51 and 5.52.

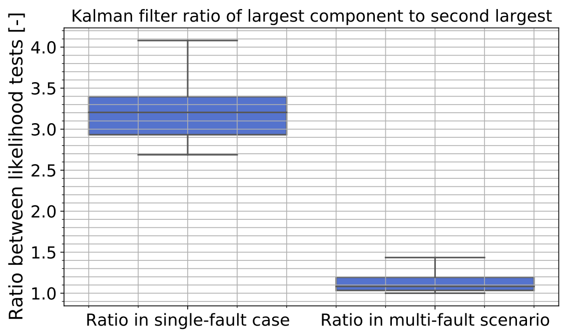


Figure 5.51: Box plot for Kalman filter ratios of largest output to second largest.

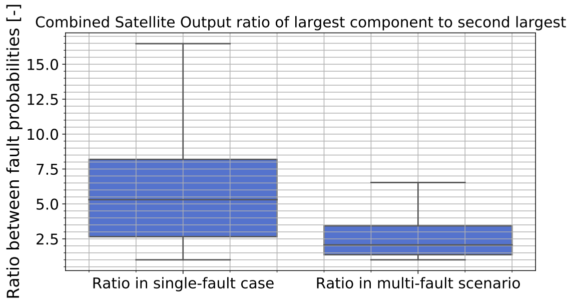


Figure 5.52: Box plot for combined network output ratios of largest output to second largest.

For the Kalman filter (Figure 5.51) there is a distinct difference in the ratios between the two cases, with a clear gap in between the two distributions. Given two time steps, it should be possible to distinguish between a case where there is a single fault as opposed to multiple faults. However, this evaluation should be repeated for more simulation runs, different fault types and intensities as well as fault locations before a definitive statement with regards to his performance can be made. This analysis also does not take into account a possible natural decrease of this ratio over time. As only the first 100 seconds after the occurrence of the first fault contain a single fault, a potential natural decrease over time could be hidden having a fixed time between the faults.
The combined neural network Figure 5.52 on the other hand has considerable overlap between the two cases. The lower quartile line for the single fault case lies below the upper quartile line of the multiple fault case. While it is still possible to say which of the two cases would be more probable given a ratio of the largest fault probabilities, a definite distinction can only be determined with much less confidence compared to the Kalman case.

Overall, while the Kalman filter has a better distinction between the single and multiple fault case by virtue of having a much lower ratio between the two largest components, the chance of isolating the secondary fault as well is much smaller. On the other hand, the combined neural networks show a lot more overlap in terms of the ratios between the single and multiple fault case but have a much higher chance of actually having the secondary fault be within the two largest components of the output vector.
It should also be mentioned here that it would be much easier to adapt the Kalman filter by simply adding another set of statistical tests where the tested deviation from the mean is the sum of the deviation of two thrusters. Re-training the neural networks to correctly put out two isolations would be much more time intensive, both in the generation of the datasets as well as the training time itself. Furthermore, the final activation function for the isolation networks, the so called 'soft-max' function, is not well suited for the isolation of two or more simultaneous faults. This is due to the tendency of this function to amplify differences in the

Table 5.7: Average accuracy (Acc.), precision (Prec.) and tecall (Rec.) for each detection network for all robustness scenarios, in percent

Dataset	Naive Network			Individual Network 1			Kalman		
	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.
Regular Dataset	80.56	76.47	86.81	81.73	99.37	62.88	99.09	100	98.00
Low Thrust	47.71	97.24	0.02	75.09	92.88	56.72	92.66	100	95.74
Low Intensity	45.18	94.16	0.08	57.04	98.27	22.09	73.18	100	49.67
High Noise	64.55	85.76	48.04	59.95	59.78	98.71	73.01	67.5	100
Navigation Change	67.33	87.55	48.74	80.29	99.94	64.94	99.01	100	98.17
Multiple Faults	85.43	85.60	91.62	99.47	99.70	99.43	99.70	100	99.48

final vector, leading to a single preferred output. A different activation function of the final layer would be more appropriate, e.g. a simple sigmoid function or the hyperbolic tangent. These function would yield better results in the multiple fault case as the output of a single neuron is not dependent on the activation value of the other neurons in that layer.

5.4.6. Robustness Conclusion

Through all the scenarios investigated here, a trend was very visible. The naive networks performed poorly, the individual networks performed much better and the Kalman filter was robust to almost any scenario that it was used in. A tabular comparison of the performance of the detection approaches can be seen in Table 5.7. As can be seen from the table, both network experience significant reductions in their performance due to the scenarios they were not trained on. The Kalman filter on the other hand shows very similar performance to the default dataset, with two exceptions: the performance drops at the lower fault intensities and in the high noise scenario. In the former case this is due to the detection limits of the filter, when the difference in the residual signal between the faulty and faultless state falls below the noise floor. For the latter this is due to the slower decay of the residual to the steady state. The slower decay means that the filter perceives the higher residual as a deviation from the known steady state mean and therefore the statistical tests start giving positive results. This is not the case in the default set as the Kalman filter is only turned on after a certain amount of time, when the signal has reached the steady state.
The naive neural network approach shows 15% to 35% decreases in accuracy for all scenarios except for the multiple fault scenario. These decreases are mainly due to a drop in the recall, i.e. less faults are positively detected. The precision of this network actually increases for all scenarios above the default set. Considering the amount of false positives that this network shows, it is likely that this increase in precision is due to the omission of the completely faultless cases, which can be a source of false positive detections.
The individual network a filter also has decreased accuracy in almost all scenarios. Again, this is mainly due to the drop in recall low intensity and low thrust scenarios as as fewer faults get detected. For the high noise, the same effect as for the Kalman filter occurs. The increase noise leads to more detections overall, which raises the recall, but also increased the amount of false positives, hence lowering the precision. There is similar or better performance on the range based navigation scenario as well the scenario involving multiple faults. This is not very surprising for either, as the range based navigation has an overall smaller error, while the multiple faults only contained open thruster faults, which were easier to detect for all detection methods.

The detection times for both the closed and open faults across all scenarios are shown in Table 5.8. As would be expected for both the lower maximum thrust and the low intensity scenarios, the detections decrease and the detection times increase. For the neural networks this means that the overall median is strongly affected by outliers or false positives, resulting in negative detection times for the individual network. The Kalman filter only experiences increases in the median detection time.
For the higher noise size scenario the detection times are in fact all negative. This is due to to higher excitation of the signal, whether it is the input to the networks or the residual from the Kalman filter, leading to more false positive detections.
It is quite surprising in fact that the naive network ever achieves a median detection time above 0. For both the low thrust and low fault intensity scenario, this is pure chance as only very few positive detection (whether false or true positive) are ever achieved. However, this is also the case for the range based navigation. Considering that the open thruster fault median detection time is still negative, however, this appears to also be a due to the partially random behavior of the naive network.

Table 5.8: Comparison of median detection time for all detectors over all robustness scenarios, in seconds.

Dataset	Naive Network		Individual Network 1		Kalman	
	Closed Fault	Open Fault	Closed Fault	Open Fault	Closed Fault	Open Fault
Regular Dataset	-1653.0	-1172.0	257.5	11.0	61.0	1.6
Low Thrust	n/a	452.5	-631.0	35.0	67.0	10.9
Low Intensity	2725.0	3141.0	-1528.5	-929.5	369.0	21.1
High Noise	-1307.0	-692.0	-2054.0	-1977.0	-1729.0	-1545.1
Navigation Change	3713.0	-609.5	223.0	10.0	61.0	1.6
Multiple Faults	n/a	-528.5	n/a	4.0	n/a	1.6

Table 5.9: Average isolation accuracy (Acc.), isolation precision (Prec.) and isolation recall (Rec.) for each isolation network, split by scenario, in percent; Individual networks are averaged.

Dataset	Naive Network			Individual Networks			Combined Networks			Kalman		
	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.
Regular Set	2.66	2.76	2.46	64.45	72.84	64.38	76.28	78.93	76.08	45.98	47.30	46.61
Low Thrust	0.00	0.00	0.00	5.84	2.87	5.84	8.70	2.94	8.70	45.33	50.00	45.33
Low Intensity	0.00	0.00	0.00	2.84	2.83	2.84	3.82	2.78	3.82	10.53	50.00	10.53
High Noise	2.46	2.77	1.27	53.25	56.61	53.09	60.61	62.47	60.55	30.86	33.02	44.30
Navigation Change	2.67	2.77	2.67	66.97	75.66	67.05	78.82	83.39	78.63	45.47	45.21	45.21
Multiple Faults	3.42	2.79	2.39	79.41	86.94	79.75	97.74	98.19	97.89	5.16	54.50	6.01

The individual network only has negative median detection time in the cases where few faults are detected in the first place, i.e. the scenarios with low fault impact, and the high noise scenario. For the range based navigation and the multiple fault scenario the detection times are mostly positive.
However, despite having good performance compared to the naive network, the Kalman filter shows much smaller detection times. Especially in the higher fault intensities, this fast response time is important in order to make sure that the appropriate response can be taken quickly to ensure the safety of the mission.

When considering the isolation capability, summarized in Table 5.9, the Kalman filter again proves to be very resilient against changes in the setup of the simulation. The comparatively low initial values for the regular dataset are due to the fact that open faults are almost always misisolated as the opposite thruster. So while the average detection of thrusters is lacking, the thruster pair isolation is perfect in every scenario except the higher noise floor. This is especially apparent in the multiple fault scenario which only contains open faults, which consistently get misisolated as the thruster opposite to the faulty one. However, these values are unusually high for the open faults, which are generally below 1% accuracy, precision and recall. It is not clear, why the presence of a second fault would increase the correct isolation rate for the Kalman filter.

Since the naive network only select one of two possible outputs, there is no performance to evaluate on the scenarios where the fault is always in the first thruster of satellite 1, as is the case for the low thrust and low fault intensity scenarios. It is clear that the network does not change its output based on the input and as such it is not analyzed further.
The individual naive networks show a severely decreased performance for the scenarios with low fault impact. In general it seems that the isolation is much more dependant on a clear fault signature than only the detection. The high noise scenario did not impact the fault isolation as much as the detection. This could be due to the fact that the detection network see a deviation from the standard signal to be a sign of a fault, leading to an activation. However, the isolation networks were already trained on the cases which included faults and therefore deviations from the faultless case. The higher noise still impacts the isolation quality by approximately 15% to 17% in all quality measures. In the navigation multiple failure scenario the performance increased on average, which for the former could be attributed to a decrease in the noise level. For the latter, the network has essentially two 'correct' choices, the initial and secondary fault. Since both are present in the input, it is sensible that the network has increased performance when either faulty thruster counts as a correct isolation.
The same trends as for the individual networks can be seen for the combined output, which is unsurprising as it is made up of the output of each individual network. The performance increase that is achieved by

combining them is however diminished for almost all scenarios except for the multiple fault case, where the combined network performs better by larger margin than for any other scenario.
The overall performing with regards to the robustness of the scenarios can be summarized as follows: In both detection (accuracy, precision, recall and detection time) and isolation, the Kalman filter exceeds the neural network based approaches in all categories. Only for open thruster faults can the combined neural network come close in terms of isolation quality.
What was also cemented is that the closed faults are both harder to detect and harder to isolate. This could be determined by their higher threshold for detectability and isolability for all methods, although the Kalman filter detected them for lower fault intensities compared to the networks.
Another observation that can be made is that for the default scenario, the detection threshold lies lower than the isolation threshold for both the individual neural network as well as the Kalman filter. This is however not the case for the low thrust, where both the isolation and detection for the closed faults specifically decrease together towards the limit. This is no issue for the Kalman filter as the detection limit also represent the isolation limit.

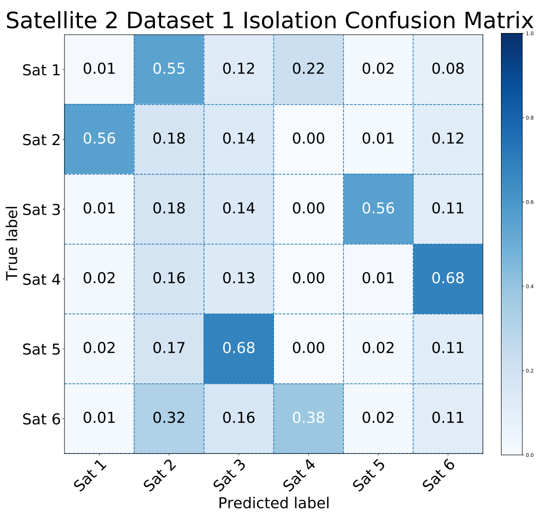
5.5. Response of Individual Networks to Mismatched Dataset

In Section 4.1.1, the possibility of a third training approach was mentioned, which was ultimately discarded due to time constraints. This approach envisioned training a single network for all satellites while transforming the input or the output or both such that the network can still properly learn the dependencies.
While such a network was not trained, an interesting behavior of the individual networks was found. When these networks were not given the data they were trained on (the data from that particular satellite) but that of satellite 1, there was a consistent mis-isolation. This can be seen in Figure 5.53, showing the confusion matrices on a satellite level for the satellites 2, 3, 4 and 6.

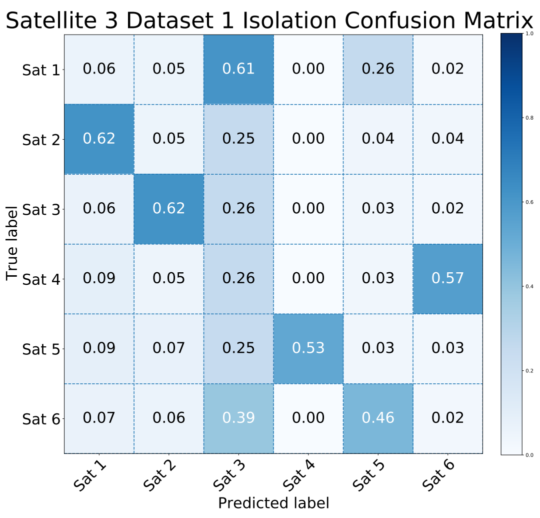
As can be seen in the figure, the isolation is not as consistent as when the networks are given the correct data. However, the presence of the patterns that can be see in the various networks suggest that if the output were adjusted, only a single network would need to be trained.
For example, when the output of the individual network of satellite 6 is inspected (see Figure 5.53d, there's a clear diagonal trend. That is to say, satellite 1 is most often misisolated as satellite 6, satellite 2 is misisolated as satellite 4 and so on. More specifically, this trend is not only satellite wide, but also seen on the thruster level. This can be seen in Figure 5.54.

More generally, a fault in satellite 2 (the first neighbor of satellite 1) will in most cases be isolated as the first neighbor of the satellite that is running the network. This can be seen for all the networks. The network of satellite 2 isolates a fault in satellite as belonging to satellite 1 (its first neighbor), the network of satellite 3 isolates is as belonging to 1 (its first neighbor) and so on. Furthermore, the faults in satellite 1 itself are mostly isolated as belonging to the satellite that runs the individual network, and satellite 6 (the one satellite 1 does not have a connection to) is isolated as the satellite the individual network does not have a connection to.
It appears as though Table 3.6 can be used as a cypher to predict the network output. As an example, the following procedure could be implemented in order to make use of this pattern. the network for satellite 6 could be trained not only on data from its relative measurements, but also with the data from all other satellites with their labels adjusted so the output matches the networks intended output. For example, the output network of satellite 6 when given data from satellite 1 should be inverted so that an isolation in satellite 6 thruster 1 is interpreted as a fault in satellite 1 thruster 1. Then, when the network is given that data during operations, the (incorrect) output of the network can be transformed to represent reality again.

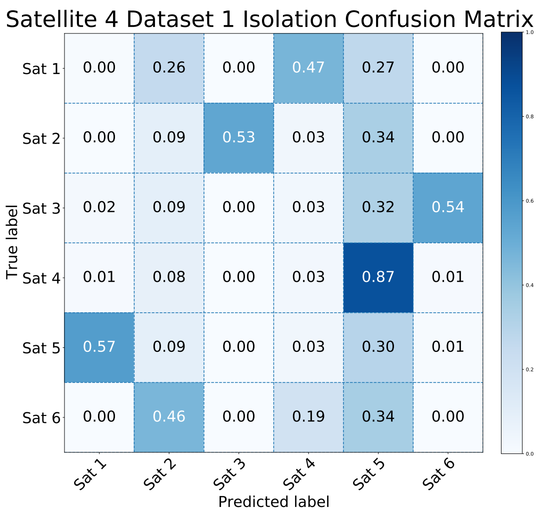
However, it is not clear how well this would work in practice as the network in question could also become confused from the input data. As such, this is a recommendation for further work.



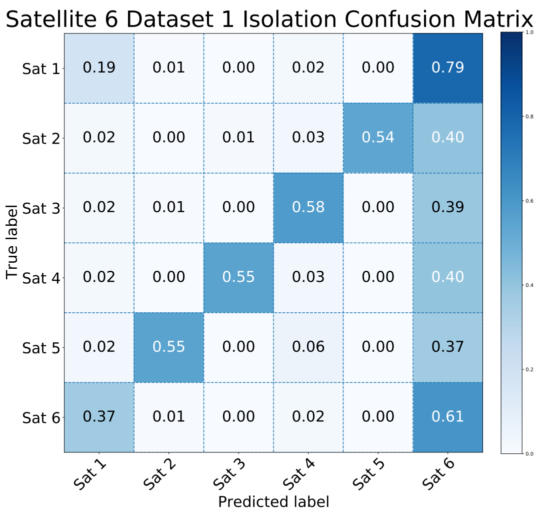
(a) Satellite 2 network confusion matrix.



(b) Satellite 3 network confusion matrix.



(c) Satellite 4 network confusion matrix.



(d) Satellite 6 network confusion matrix.

Figure 5.53: Satellite isolation confusion matrices of individual network when given data from satellite 1.

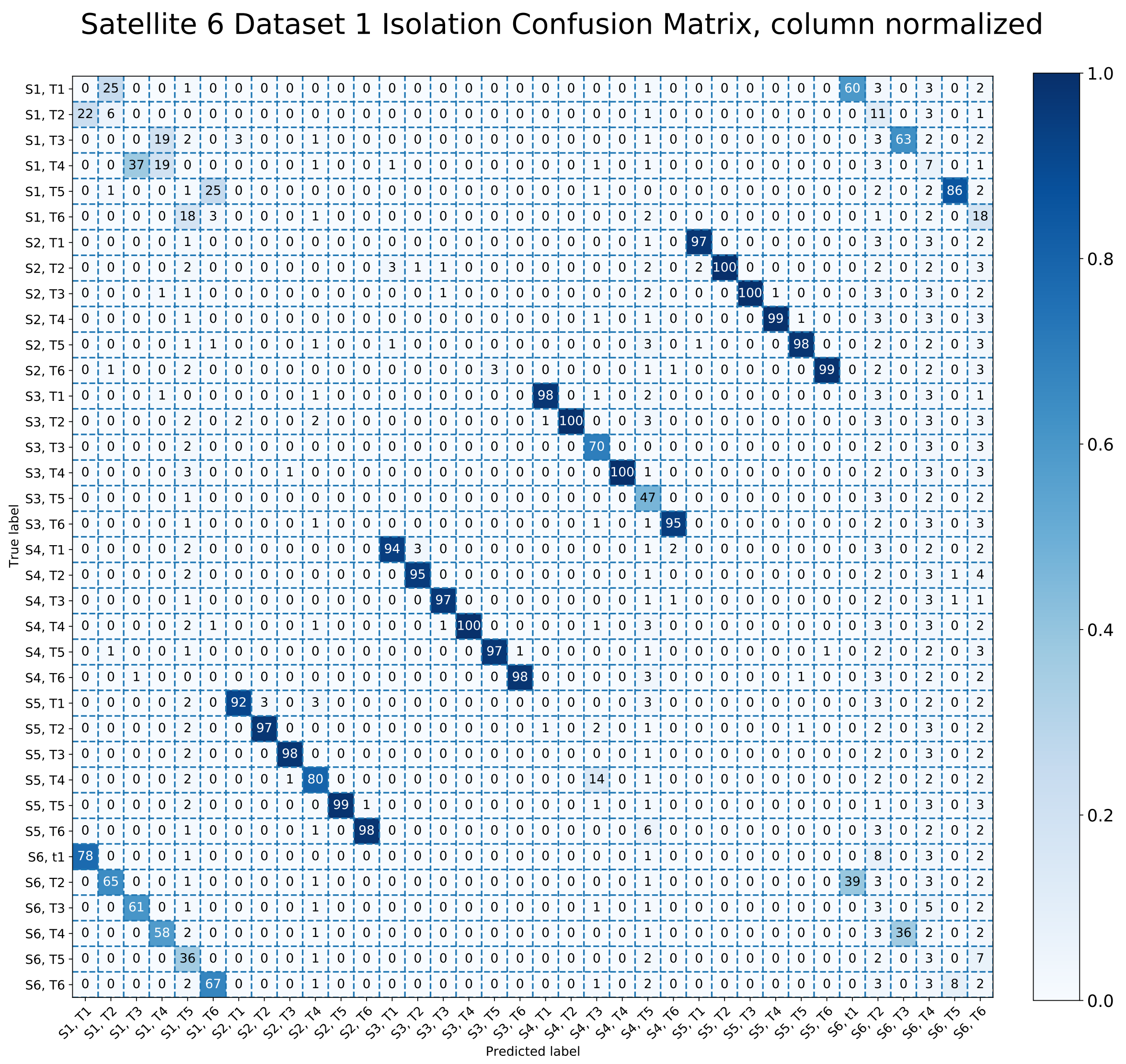


Figure 5.54: Satellite 6 column normalized confusion matrix when given data from satellite 1, numbers in percent.

5.6. Analysis Summary

Through all the evaluation techniques and criteria applied in this chapter it can be seen that the Kalman filter is the method with the better performance, across almost all categories. The Kalman filter has higher accuracy in detecting faults, lower detection times, perfect thruster pair isolation in the nominal cases, and overall good performance in the robustness scenarios.
First, the performance with regards to the detection capability is compared in Section 5.6.1, followed by the isolation capability comparison in Section 5.6.2. Furthermore, the performance in the robustness scenarios is discussed in Section 5.6.3.

5.6.1. Detection Comparison

The Kalman filter performed best with an accuracy of 99.09%. The neural networks on the other hand showed significantly worse performance with 80.56% and 81.73% for the naive and individual network, respectively. Similar differences could be seen in the recall. While the Kalman filter was able to detect 98.32% of all faults, this was only the case for 86.81% for the naive and only 62.88% for the individual detection network. In terms of precision however, the Kalman filter and the individual network are quite close, with a precision of 100 % for the filter and 99.37 % for the individual network. With these approaches a detection is almost certainly correct. The naive network on the other hand only reached a precision of 76.47%, meaning an incorrect detection in almost a quarter of the cases, making this method very unreliable.
Overall, the naive network appeared to be quite unstable given a set of input. In some cases it reacted as one would expect, with a generally low activation in the output layer followed by a sudden rise after the fault injection. However in many cases the naive network output hovered at very high values above 0.98 for the entire simulation duration. In these cases, the correct detections seem almost like pure chance rather than a response to a different input.
The individual network performed much better and more consistently. While overall its recall is lower, this is due to a much increased precision, making its output more reliable.
The Kalman filter performed best of the detection methods. Its detection output was consistent and swift, but more importantly there was not a single false positive detection. This makes the Kalman filter even more reliable than the individual network, even with the close to perfect recall.
For all methods a clear difference in performance could be seen based on the fault type. Closed valve faults were much more difficult to detect, while open faults were detected very consistently. The biggest difference is present for the individual network, which shows a recall of 99.76% for the open faults but only 21.57% for the closed faults. For the Kalman filter, this difference is only 3.5%. The naive network shows a difference in recall of approximately 27% between the fault types.
In terms of precision, the difference between the two fault types is not very large. Only the naive network shows a difference of approximately 10%, while the individual network shows only a 0.5% difference and the Kalman filter none whatsoever.

For the individual network and Kalman filter there was furthermore a trend visible based on the fault intensity. For the closed fault type, the detection quality of either approach decreases with decreasing fault intensity.
The limits of fault detectability for the networks were also at considerably higher fault intensities. The individual detection network dropped below 5% recall at a fault intensity of 0.1 for the closed faults and approximately 0.002 for open faults. The Kalman filter on the other hand was capable of detecting faults until fault intensities of 0.025 and $4e-5$, for closed and open faults respectively.

5.6.2. Isolation Comparison

In term of the isolation performance, the Kalman filter shows one of its weaknesses, that is almost always misisolating open thruster faults. However, for the closed faults the Kalman filter continues to have a very high performance, with quality measures above 99%.
The naive networks always produced one of two possible output, leading to very poor performance. This is most likely due to the the fact that the naive network received too different input data with the same output, leading to confusion during the training. The individual networks however show better performance. Especially the combined network output even exceeds 99.9% in all quality measures, when considering open faults. For the closed faults however, the performance is significantly worse, with only 49.61% accuracy, 54.19% precision and 48.63% recall.

The isolability also decreases with the lower fault intensities. For the Combined networks, the closed fault isolation accuracy dropped below 10% at a fault intensity of 0.1, while for open faults this occurred at a fault intensity of 0.05. For the Kalman filter the isolability limit coincides with the detectability limits mentioned above.

5.6.3. Robustness Comparison

In terms of robustness, the naive detection network also showed almost no detections in the scenarios where the fault impact is lower, either through low fault intensities or lower thrust values. It also showed decreased performance where one would expect better performance (and for the other methods do show better performance), such as the range based navigation scenario which entailed a net decrease in the noise. The naive isolation network failed to learn during training and as such is no better than random chance. This approach was not very successful.

The individual networks, while being better than the naive networks, still had some difficulties. The individual detection network is almost on par with the Kalman filter for the detection of open faults. For closed faults however, accuracy, precision and recall are considerably worse. While only 20% of closed faults are detected, more than 99% of the detections of the closed faults were accurate. The detection limits for the individual detection network were also at considerably higher fault intensities compared to the Kalman filter. The recall dropped below 5% at 0.23 for closed faults and $1.8e{-3}$ for the open faults. While the decline is gradual for the individual detection network and there are occasional correct detections even below this limit, the Kalman filter has a hard line separating the detectable and undetectable faults. These lie at 0.022 and $4e{-4}$ for closed and open faults, respectively.

The individual isolation networks show the same trend in the difference between open and closed faults. While the performance measures for the open faults all lay around 90%, the isolation accuracy and recall lay only around 35% for the closed faults.

Overall, the combination of the network output proved to be a significant increase in the isolation accuracy. The isolation on open faults is very good, with the mean isolation accuracy, isolation precision and isolation recall being above 99.9% for the open faults. The closed fault isolation performance also increased. However, while this performance is good on the open faults, it has to be considered that the isolation networks necessitate prior detection through a detection network. Therefore. the true performance will be worse, considering that the evaluation presented in this chapter assumed immediate detection. This effect will most likely be felt much more strongly for the closed faults, as the median detection time is much higher compared to the open fault detection time. As the limit of the isolability for the open fault is also at much higher fault intensities (as would be expected) than the corresponding detection limit, there should only be little difference if the combined system of detection networks and isolation network were to be tested on the open faults. However, the individual detection recall for closed faults drops below 10% much sooner than the corresponding isolation recall. This would lead to an even worse performance on the closed faults.

6 Conclusions and Recommendations

Having presented the results of the analysis in Chapter 5, the conclusions drawn from the analysis are presented in this chapter in Section 6.1, followed by a set of recommendations for further work and improvements in Section 6.2

6.1. Research Questions

From the previous chapters the research questions posed at the start of the thesis will be answered. First, the sub questions will be addressed before an answer of the main research question is attempted.

How does the detection and isolation performance (accuracy, precision and recall) compare between the centralized and distributed approaches?

The Kalman filter outperformed all networks by a considerable margin. The only downside of the Kalman filter implemented here is the misisolation of open faults as being faults of the opposite thruster. However, beside this disadvantage the performance was very good, with no false positive detections and no false isolation of a thruster pair.
Out of the tested distributed approaches, the individual networks performed the best, both for detection and isolation. The naive networks seemed to not learn the relation between the input and output data properly, as for the naive detection the output was very unstable, and the naive isolation network failed to learn any relation at all, merely isolating the same thruster for every time step. The most likely explanation for the difference in these two network approaches is the consistency of the training data. When the network receives very similar input data with different classification output, the network can get confused. This can lead to little to no training progress. Proper data pre-processing is key in getting the correct training behavior.
In addition, the combination of the individual isolation network output significantly increased the performance, showing more than 99% accuracy on the open faults. Each individual network showed weaknesses in isolating faults in itself and the satellite directly opposite in the formation, but these were removed due to the combination process.
The detectability and isolability limits were both considerably lower for the Kalman than for the neural networks with a difference of almost two orders of magnitude difference, showing a greater sensitivity to faults of the Kalman filter compared to the neural networks.

How quickly and consistently do the methods respond to the faults?

The response times for the naive network were almost always negative due to the large amunt of false negative detections. As such, it cannot be said that the naive network *responded* to the fault at all. The consistency of this network was also lacking, showing little distinction between a faultless and faulty categorization.
The individual network on the other hand did show proper responses, at least for the higher fault intensities. However, with a median detection time of 257.5 s and 11.0 s for closed and open faults, respectively, it took much longer to respond compared to the Kalman filter, whose median detection times are 61.0 s and 1.6 s. Both the Kalman filter and the individual network show consistent responses to the open faults, while for the closed faults there are intermittent dropouts in the network, due to the effect of the fault disappearing.

How robust are the FDI methods to scenarios they were not designed for?

The Kalman filter showed very consistent results for the various scenarios it was subjected to. The only significant drops in performance were for the low intensity scenario for which the Kalman filter eventually reached its detection limit and the high noise scenario. The drop in performance in the latter case is unsurprising considering that the statistical testing on the residual assumes a certain variance which is no longer the case for a higher noise floor, which led to premature detections. However, it should be said that after the fault was injected, the isolation soon consistently returned to the faulty thruster pair.
The robustness of the neural networks showed quite poor robustness. For the low intensity faults as well the low thrust scenario the detection and isolation capability dropped significantly, with the naive detection network detection barely any faults correctly (0.02%), and the individual detection network experiencing a drop in recall of 40% for the low intensity faults. The high noise also affected the individual network similarly to the Kalman filter, indicating a response based on the size of the relative positioning vector.
In terms of isolation robustness the networks showed a similar pattern, with very low performance for low intensity faults and a drop for the higher noise floor.
Overall, the Kalman filter responded much more robustly to the scenarios compared to the neural networks.

What is the influence of model uncertainties and disturbances on the detection and isolation?

Due to the setup of the data generation and training, this could not be investigated. Generating the data required optimized code and therefore the simulation was adapted to provide sufficiently accurate results. Increasing the accuracy of the model further or including the disturbances would change the relative positioning by less than 6e−3 m, whereas the simulated measurement noise already has a standard deviation more than an order of magnitude greater. The effect of model uncertainties could only be seen if the simulated relative navigation would be considerably more accurate.

What is the effect of distributing the FDI system on the computation time, bandwidth usage and time until fault detection

As can be seen in the response time, the distributed approaches take up to five times longer to respond to faults compared to the Kalman filter. However, there is less of a communication requirement as the isolation networks, whose output are shared, are only activated after the detection of a fault, whereas the Kalman filter requires a constant exchange of information.
The computation time is also considerably less for the networks than for the Kalman filter, as the only computation necessary is a series of matrix operations and application of simple non-linear functions, which can be performed very quickly. The isolation networks take approximately 550 ms for a full batch of 4096 sets of input data, averaging to 0.13 ms for a single input evaluation. The Kalman filter, on the other hand, takes 8.96 ms for a single evaluation. However, it should be said that this comparison used times from tensorflow compared to MATLAB, which have differing levels of code optimization. Nonetheless, this gives an indication that the neural networks are indeed very fast in computation.
Furthermore, the use of on-line matrix inversion is avoided in the Kalman filter, opting instead to use the steady state matrices, which significantly speeds up the computation. This is not to mention the increased computational cost if the Kalman filter were to be adapted to deal with the errors due to the linearization of the non-linear relative motion.

Having addressed the sub question, the main question can be answered.

How does a distributed, neural-network based FDI architecture for thruster faults compare to a centralized model-based architecture in terms of detection capability, isolation capability and robustness in a close formation flying mission?

This main question can be answered by summarizing the answers to the sub-questions listed above and through the various differences that were explored in Chapter 5. The naive approaches were not able to reliably perform the functions of an FDI system as the naive detection network had a false positive rate of almost 25% of all positive detections, while the naive isolation network only put out a single thruster consistently. But while the individual network approaches were capable of reliably detecting and isolating faults, they showed significantly worse performance compared to the Kalman filter approach, especially for the more complicated closed valve fault. In addition, the neural networks were not particularly robust to changes in

the parameters which affect the relative positioning.
However, from this it cannot be concluded that the Kalman filter (or another model-based method) is categorically preferable to the neural network approach. Especially for open faults the individual detection network together with the combination of the isolation networks had comparable performance and would require less computing time. In addition, the centralized Kalman filter requires constantly sharing the relative positioning data and therefore requires more bandwidth. This need for constant information exchange also makes the system less robust if that link breaks down due to a fault in the telecommunication system. If one or more communication links become unavailable, the Kalman filter will not be able to handle the missing measurements and therefore not be able to perform any FDI.
On the other hand, if the distributed approach is implemented, and a communication link is broken, the remaining satellites can still combine the probability output that they do have access to in order to come to a better diagnosis. If all links break down, but the relative positioning measurements are still available, the satellite can still use the results from the on-board individual network, albeit with worse performance than the Kalman filter.

In conclusion, the currently implemented neural networks show worse performance than the centralized method, but are comparable in performance in certain situation and even have certain advantages over the centralized method. When the fault is difficult to implement in a state-space model, the neural network might be able to outperform the model-based approaches.

6.2. Recommendations

In this section, recommendations for further work and possible improvements to the presented work are given. First, recommendations regarding the FDI methodology are presented in Section 6.2.1 and Section 6.2.2, followed by improvements to the numerical simulation in Section 6.2.3. Finally, recommendations for the analysis of the networks are given in Section 6.3

6.2.1. Neural Networks

Several aspects of the neural network side of the thesis can be improved upon. These recommendation can be broadly split into comments on the training platform, Google Colab, and comments on the structure of the networks.

Training Platform Due to various restrictions of the platform used to train the network, only a small fraction of the data that was generated for training could actually be used. This has affected the results as a larger dataset tends to yield better results, as shown in Chapter 5.
Furthermore, the various hyper-parameters of the networks (number of layers, number of units, learning rate, etc), while investigated, were not optimized for the problem at hand. If a faster training process could be achieved (e.g. through the use of a super-computer), it is possible that a much better set of network and training parameters could be found.
The use of Google Colab as a platform can also not be recommended for a project such as this. From the experiences with this service, only the training of a single network with a small amount of layers and units per layer can be done with any sort of consistency. Frequent crashes of the virtual machine on Google's servers due to disconnects or running out of ram (sometimes for inexplicable reasons) disrupt training and mean having to restart at the last completed epoch, which in this project involved losing a few minutes of progress at best, to losing up to two hours at worst. In addition, the use of hardware accelerators (TPUs or GPUs) is capped for a regular user, while a professional subscription is only available for users in North America. This results in days where no hardware accelerator is allocated, making training effectively impossible as the time required for a single epoch can exceed 4 hours on a regular CPU with even just a small fraction of the dataset. Combining this with the regular disruptions due to disconnects or RAM issues and this can extend the required training time by days.

Network structure While a network based on the Long Short-Term Memory units was presented due to their use in time-series modeling, there are many other types of neural networks architectures which could be suited for this purpose. More general Recurrent Neural Network (RNN) structures have also shown to provide useful results. Furthermore, a multitude of variants exist, each with their own benefits and shortcomings which makes them suited for specific problems. A more suitable set of parameters could investigated to improve the performance of the network approach. This includes increasing the number of layers (if the computational resources allow) or using a different combination of recurrent and non-recurrent layer types.

6.2.2. Kalman Filter

The Kalman Filter implemented here, as was seen in Section 3.5.4, still suffers from errors due to the non-linearity. While this most likely made little difference for the noise level considered in the analysis, it would become significant if the noise level drops significantly.
Implementing and testing a filter better capable of dealing with these non-linearities such as an Extended Kalman Filter (EKF) or an Unscented Kalman Filter (UKF) would be preferable in that case.
Furthermore, a comparison between the distributed neural networks and a distributed filter would make for a easier evaluation of the difference between knowledge-based and model-based methods.

6.2.3. Simulation

The simulation of the satellite formation is improvable in various ways. A general simulator of any satellite formations could be very useful in their design or qualitative study. As of right now, only octahedral virtual structure type formations are to usable, with easy extension to other geometric shapes. However, passive formations, deep space formations or more complicated formation types would be difficult to implement with the current structure. In addition, the simulation only considers circular orbits around the Earth. However, many satellite formations, are on elliptical orbits (especially the virtual structure type) or in the vicinity of a Lagrange point. Also, ideas exist for formations around other bodies in the solar system. Adjusting the simulation for different orbits would take significant effort, but could be useful for future research.
Apart from expanding the functionality of the simulation, the runtime could be optimized, as a simulation over a single orbital period can take from on average 13 seconds when using the parallel computing features of MATLAB to up to 40 seconds. An additional avenue for optimization is the gravity field calculator which still takes a significant amount of time during the simulation. Another approach would be to use a lower-level programming language such as C or C++, which would also allow for integration with Tensorflow directly in order to enable the use of online learning.

6.3. Analysis

One practical limitation of using the Colab environment is that the evaluation of the individual networks on the validation part of the dataset takes a considerable amount of time and is subject to the same issues as with training the networks that are described above. Due to this, determining the output of the networks was performed separately from the subsequent evaluation and creation of the plots. However, the output of the individual isolation network is a 1 by 36 vector of floating point numbers, which makes it infeasible to save the entire output of the network to a file and then perform the analysis, as the file-size would simply be too big. Only the maximum value of the network (i.e. the thruster the network identifies as faulty) was saved to a file. A consequence of this is that more in-depth analysis on the network output could not be performed and there is no distinction being made between a network output where the maximum value was 0.3 and an output where the maximum value is 0.99, despite there being a large difference in confidence of the network. A more in depth analysis of the isolation network output could give more insight into the behavior and possible issues resulting from the way the network was trained.
Furthermore, in the analysis it was seen that the individual networks, when given the data of a satellite they were not trained on, systematically miscategorized the fault location, corresponding to their difference in neighbor ordering. This could be used to reduce the training effort as only a single network needs to be trained, if the fault label is consequently adjusted. Potentially, this could also increase the network performance as it is trained with more diverse data.

6.3.1. Comparison Case

While the Kalman filter is a standard method for FDI purposes one of the features of the Kalman filter is the assumption of known control input to the system. This gives it an edge compared to neural network based methods, which do not have the control input as a network input. While the Kalman filter also assumes additional information in the form of the state and measurement equation, which are used in the design of the filter, the network should be able to extract certain aspects of the dynamic system model through the relationship of input to output data. However, the control input is not present in the networks.

A possible remedy for this problem is to consider unknown-input-observers, which can work without having completel knowledge of the control-input. Alternatively, the commanded control input could be another input to the network. Training with this additional input compared to the networks presented in Section 4.1 could give insight into what information the network draws from the relative positioning and velocities alone.

A
Additional Confusion Matrices

In this chapter, the additional confusion matrices for the individual network of satellites 2 to 5 can be found.

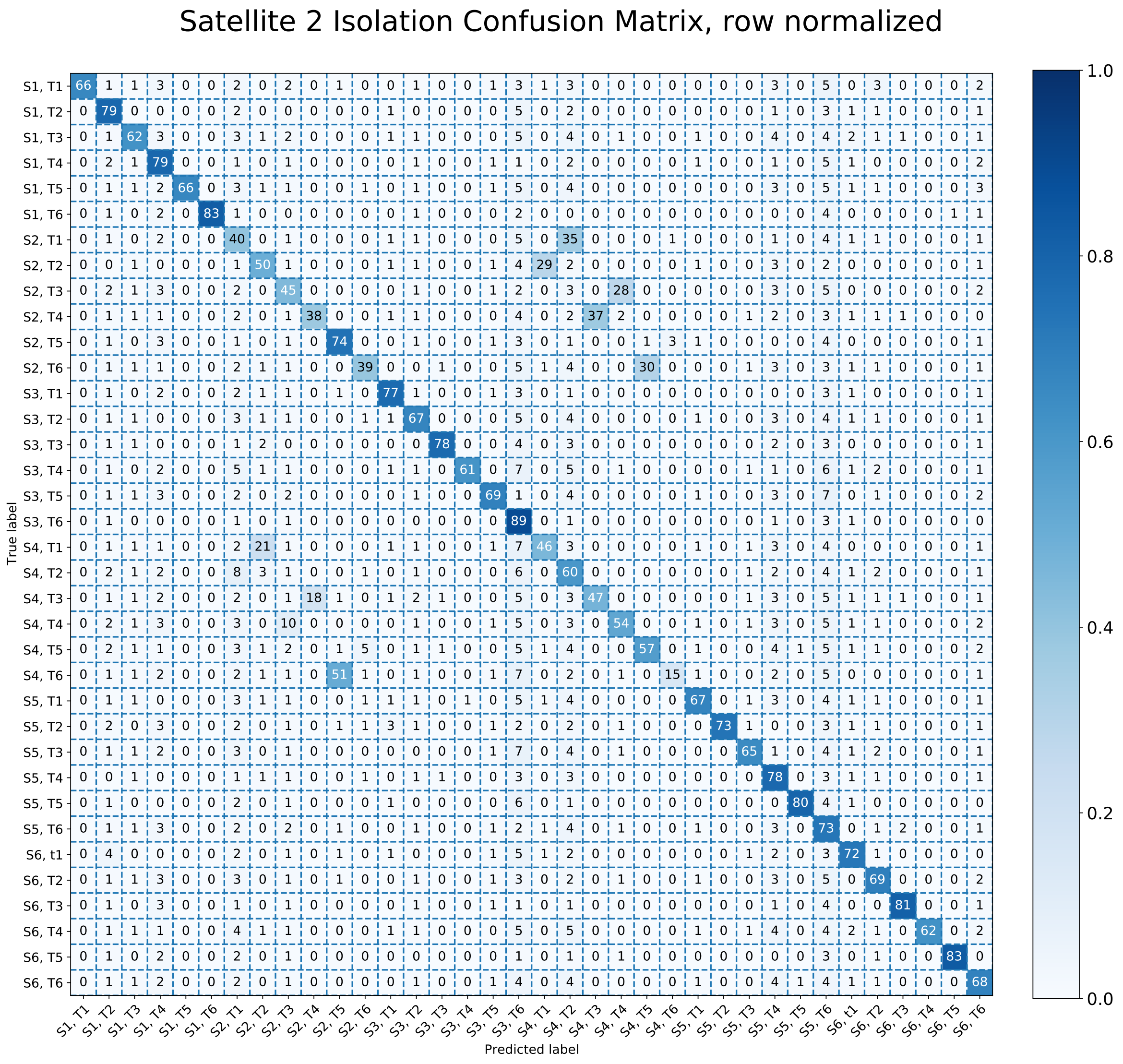
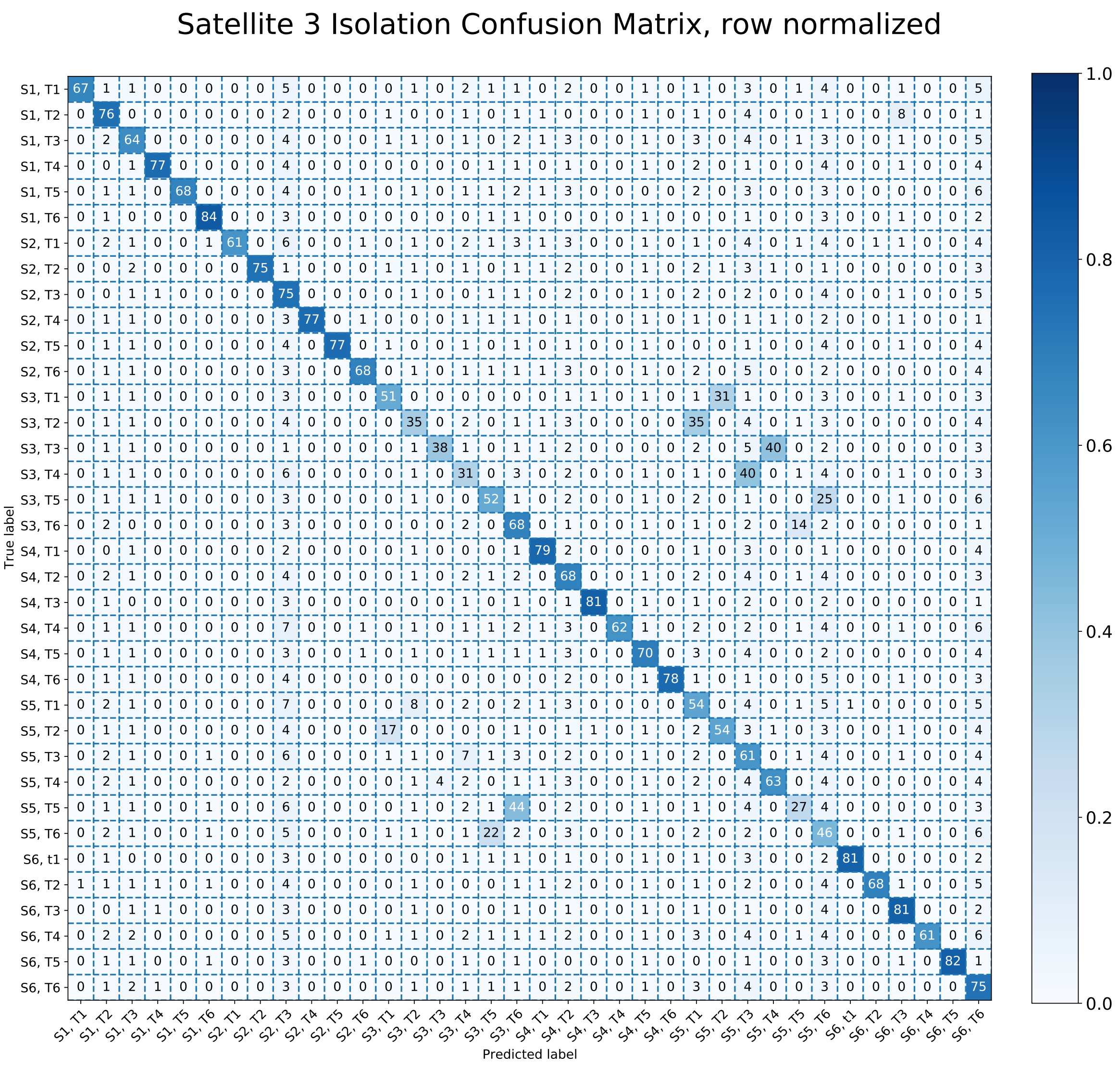
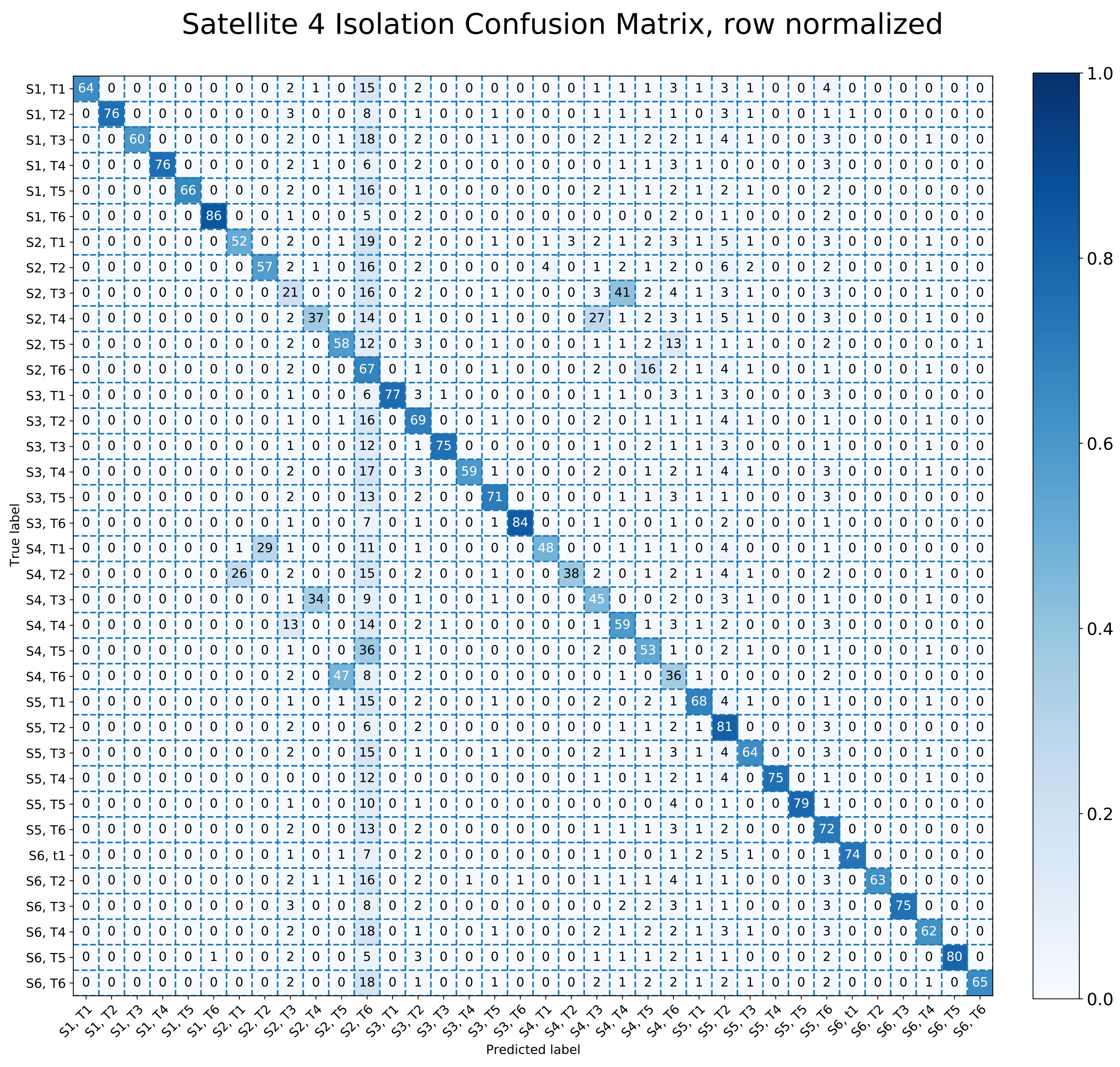


Figure A.1: Satellite 2 isolation confusion matrix for thruster isolation.



True label \ Predicted label	S1, T1	S1, T2	S1, T3	S1, T4	S1, T5	S1, T6	S2, T1	S2, T2	S2, T3	S2, T4	S2, T5	S2, T6	S3, T1	S3, T2	S3, T3	S3, T4	S3, T5	S3, T6	S4, T1	S4, T2	S4, T3	S4, T4	S4, T5	S4, T6	S5, T1	S5, T2	S5, T3	S5, T4	S5, T5	S5, T6	S6, t1	S6, T2	S6, T3	S6, T4	S6, T5	S6, T6
S1, T1	67	1	1	0	0	0	0	0	5	0	0	0	0	1	0	2	1	1	0	2	0	0	1	0	1	0	3	0	1	4	0	0	1	0	0	5
S1, T2	0	76	0	0	0	0	0	0	2	0	0	0	1	0	0	1	0	1	1	0	0	0	1	0	1	0	4	0	0	1	0	0	8	0	0	1
S1, T3	0	2	64	0	0	0	0	0	4	0	0	0	1	1	0	1	0	2	1	3	0	0	1	0	3	0	4	0	1	3	0	0	1	0	0	5
S1, T4	0	0	1	77	0	0	0	0	4	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	2	0	1	0	0	4	0	0	1	0	0	4
S1, T5	0	1	1	0	68	0	0	0	4	0	0	1	0	1	0	1	1	2	1	3	0	0	0	0	2	0	3	0	0	3	0	0	0	0	0	6
S1, T6	0	1	0	0	0	84	0	0	3	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	1	0	0	3	0	0	1	0	0	2
S2, T1	0	2	1	0	0	1	61	0	6	0	0	1	0	1	0	2	1	3	1	3	0	0	1	0	1	0	4	0	1	4	0	1	1	0	0	4
S2, T2	0	0	2	0	0	0	0	75	1	0	0	0	1	1	0	1	0	1	1	2	0	0	1	0	2	1	3	1	0	1	0	0	0	0	0	3
S2, T3	0	0	1	1	0	0	0	0	75	0	0	0	0	1	0	0	1	1	0	2	0	0	1	0	2	0	2	0	0	4	0	0	1	0	0	5
S2, T4	0	1	1	0	0	0	0	0	3	77	0	1	0	0	0	1	1	1	0	1	0	0	1	0	1	0	1	1	0	2	0	0	1	0	0	1
S2, T5	0	1	1	0	0	0	0	0	4	0	77	0	1	0	0	1	0	1	0	1	0	0	1	0	0	0	1	0	0	4	0	0	1	0	0	4
S2, T6	0	1	1	0	0	0	0	0	3	0	0	68	0	1	0	1	1	1	1	3	0	0	1	0	2	0	5	0	0	2	0	0	0	0	0	4
S3, T1	0	1	1	0	0	0	0	0	3	0	0	0	51	0	0	0	0	0	0	1	1	0	1	0	1	31	1	0	0	3	0	0	1	0	0	3
S3, T2	0	1	1	0	0	0	0	0	4	0	0	0	0	35	0	2	0	1	1	3	0	0	0	0	35	0	4	0	1	3	0	0	0	0	0	4
S3, T3	0	1	1	0	0	0	0	0	1	0	0	0	0	1	38	1	0	1	1	2	0	0	0	0	2	0	5	40	0	2	0	0	0	0	0	3
S3, T4	0	1	1	0	0	0	0	0	6	0	0	0	0	1	0	31	0	3	0	2	0	0	1	0	1	0	40	0	1	4	0	0	1	0	0	3
S3, T5	0	1	1	1	0	0	0	0	3	0	0	0	0	1	0	0	52	1	0	2	0	0	1	0	2	0	1	0	0	25	0	0	1	0	0	6
S3, T6	0	2	0	0	0	0	0	0	3	0	0	0	0	0	0	2	0	68	0	1	0	0	1	0	1	0	2	0	14	2	0	0	0	0	0	1
S4, T1	0	0	1	0	0	0	0	0	2	0	0	0	0	1	0	0	0	1	79	2	0	0	0	0	1	0	3	0	0	1	0	0	0	0	0	4
S4, T2	0	2	1	0	0	0	0	0	4	0	0	0	0	1	0	2	1	2	0	68	0	0	1	0	2	0	4	0	1	4	0	0	0	0	0	3
S4, T3	0	1	0	0	0	0	0	0	3	0	0	0	0	0	0	1	0	1	0	1	81	0	1	0	1	0	2	0	0	2	0	0	0	0	0	1
S4, T4	0	1	1	0	0	0	0	0	7	0	0	1	0	1	0	1	1	2	1	3	0	62	1	0	2	0	2	0	1	4	0	0	1	0	0	6
S4, T5	0	1	1	0	0	0	0	0	3	0	0	1	0	1	0	1	1	1	1	3	0	0	70	0	3	0	4	0	0	2	0	0	0	0	0	4
S4, T6	0	1	1	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	2	0	0	1	78	1	0	1	0	0	5	0	0	1	0	0	3
S5, T1	0	2	1	0	0	0	0	0	7	0	0	0	0	8	0	2	0	2	1	3	0	0	0	0	54	0	4	0	1	5	1	0	0	0	0	5
S5, T2	0	1	1	0	0	0	0	0	4	0	0	0	17	0	0	0	0	1	0	1	1	0	1	0	2	54	3	1	0	3	0	0	1	0	0	4
S5, T3	0	2	1	0	0	1	0	0	6	0	0	0	1	1	0	7	1	3	0	2	0	0	1	0	2	0	61	0	1	4	0	0	1	0	0	4
S5, T4	0	2	1	0	0	0	0	0	2	0	0	0	0	1	4	2	0	1	1	3	0	0	1	0	2	0	4	63	0	4	0	0	0	0	0	4
S5, T5	0	1	1	0	0	1	0	0	6	0	0	0	0	1	0	2	1	44	0	2	0	0	1	0	1	0	4	0	27	4	0	0	0	0	0	3
S5, T6	0	2	1	0	0	1	0	0	5	0	0	0	1	1	0	1	22	2	0	3	0	0	1	0	2	0	2	0	0	46	0	0	1	0	0	6
S6, t1	0	1	0	0	0	0	0	0	3	0	0	0	0	0	0	1	1	1	0	1	0	0	1	0	1	0	3	0	0	2	81	0	0	0	0	2
S6, T2	1	1	1	1	0	1	0	0	4	0	0	0	0	1	0	0	0	1	1	2	0	0	1	0	1	0	2	0	0	4	0	68	1	0	0	5
S6, T3	0	0	1	1	0	0	0	0	3	0	0	0	0	1	0	0	0	1	0	1	0	0	1	0	1	0	1	0	0	4	0	0	81	0	0	2
S6, T4	0	2	2	0	0	0	0	0	5	0	0	0	1	1	0	2	1	1	1	2	0	0	1	0	3	0	4	0	1	4	0	0	0	61	0	6
S6, T5	0	1	1	0	0	1	0	0	3	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1	0	0	3	0	0	1	0	82	1
S6, T6	0	1	2	1	0	0	0	0	3	0	0	0	0	1	0	1	1	1	0	2	0	0	1	0	3	0	4	0	0	3	0	0	0	0	0	75

Figure A.2: Satellite 3 isolation confusion matrix for thruster isolation.



	S1, T1	S1, T2	S1, T3	S1, T4	S1, T5	S1, T6	S2, T1	S2, T2	S2, T3	S2, T4	S2, T5	S2, T6	S3, T1	S3, T2	S3, T3	S3, T4	S3, T5	S3, T6	S4, T1	S4, T2	S4, T3	S4, T4	S4, T5	S4, T6	S5, T1	S5, T2	S5, T3	S5, T4	S5, T5	S5, T6	S6, t1	S6, T2	S6, T3	S6, T4	S6, T5	S6, T6
S1, T1	64	0	0	0	0	0	0	0	2	1	0	15	0	2	0	0	0	0	0	0	1	1	1	3	1	3	1	0	0	4	0	0	0	0	0	0
S1, T2	0	76	0	0	0	0	0	0	3	0	0	8	0	1	0	0	1	0	0	0	1	1	1	1	0	3	1	0	0	1	1	0	0	0	0	0
S1, T3	0	0	60	0	0	0	0	0	2	0	1	18	0	2	0	0	1	0	0	0	2	1	2	2	1	4	1	0	0	3	0	0	0	1	0	0
S1, T4	0	0	0	76	0	0	0	0	2	1	0	6	0	2	0	0	0	0	0	0	0	1	1	3	1	0	0	0	0	3	0	0	0	0	0	0
S1, T5	0	0	0	0	66	0	0	0	2	0	1	16	0	1	0	0	0	0	0	0	2	1	1	2	1	2	1	0	0	2	0	0	0	0	0	0
S1, T6	0	0	0	0	0	86	0	0	1	0	0	5	0	2	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	2	0	0	0	0	0	0
S2, T1	0	0	0	0	0	0	52	0	2	0	1	19	0	2	0	0	1	0	1	3	2	1	2	3	1	5	1	0	0	3	0	0	0	1	0	0
S2, T2	0	0	0	0	0	0	0	57	2	1	0	16	0	2	0	0	0	0	4	0	1	2	1	2	0	6	2	0	0	2	0	0	0	1	0	0
S2, T3	0	0	0	0	0	0	0	0	21	0	0	16	0	2	0	0	1	0	0	0	3	41	2	4	1	3	1	0	0	3	0	0	0	1	0	0
S2, T4	0	0	0	0	0	0	0	0	2	37	0	14	0	1	0	0	1	0	0	0	27	1	2	3	1	5	1	0	0	3	0	0	0	1	0	0
S2, T5	0	0	0	0	0	0	0	0	2	0	58	12	0	3	0	0	1	0	0	0	1	1	2	13	1	1	1	0	0	2	0	0	0	0	0	1
S2, T6	0	0	0	0	0	0	0	0	2	0	0	67	0	1	0	0	1	0	0	0	2	0	16	2	1	4	1	0	0	1	0	0	0	1	0	0
S3, T1	0	0	0	0	0	0	0	0	1	0	0	6	77	3	1	0	0	0	0	0	1	1	0	3	1	3	0	0	0	3	0	0	0	0	0	0
S3, T2	0	0	0	0	0	0	0	0	1	0	1	16	0	69	0	0	1	0	0	0	2	0	1	1	1	4	1	0	0	1	0	0	0	1	0	0
S3, T3	0	0	0	0	0	0	0	0	1	0	0	12	0	1	75	0	0	0	0	0	1	0	2	1	1	3	0	0	0	1	0	0	0	1	0	0
S3, T4	0	0	0	0	0	0	0	0	2	0	0	17	0	3	0	59	1	0	0	0	2	0	1	2	1	4	1	0	0	3	0	0	0	1	0	0
S3, T5	0	0	0	0	0	0	0	0	2	0	0	13	0	2	0	0	71	0	0	0	0	1	1	3	1	1	0	0	0	3	0	0	0	0	0	0
S3, T6	0	0	0	0	0	0	0	0	1	0	0	7	0	1	0	0	1	84	0	0	1	0	0	1	0	2	0	0	0	1	0	0	0	0	0	0
S4, T1	0	0	0	0	0	0	1	29	1	0	0	11	0	1	0	0	0	0	48	0	0	1	1	1	0	4	0	0	0	1	0	0	0	0	0	0
S4, T2	0	0	0	0	0	0	26	0	2	0	0	15	0	2	0	0	1	0	0	38	2	0	1	2	1	4	1	0	0	2	0	0	0	1	0	0
S4, T3	0	0	0	0	0	0	0	0	1	34	0	9	0	1	0	0	1	0	0	0	45	0	0	2	0	3	1	0	0	1	0	0	0	1	0	0
S4, T4	0	0	0	0	0	0	0	0	13	0	0	14	0	2	1	0	0	0	0	0	1	59	1	3	1	2	0	0	0	3	0	0	0	0	0	0
S4, T5	0	0	0	0	0	0	0	0	1	0	0	36	0	1	0	0	0	0	0	0	2	0	53	1	0	2	1	0	0	1	0	0	0	1	0	0
S4, T6	0	0	0	0	0	0	0	0	2	0	47	8	0	2	0	0	0	0	0	0	0	1	0	36	1	0	0	0	0	2	0	0	0	0	0	0
S5, T1	0	0	0	0	0	0	0	0	1	0	1	15	0	2	0	0	1	0	0	0	2	0	2	1	68	4	1	0	0	1	0	0	0	1	0	0
S5, T2	0	0	0	0	0	0	0	0	2	0	0	6	0	2	0	0	0	0	0	0	0	1	1	2	1	81	0	0	0	3	0	0	0	0	0	0
S5, T3	0	0	0	0	0	0	0	0	2	0	0	15	0	1	0	0	1	0	0	0	2	1	1	3	1	4	64	0	0	3	0	0	0	1	0	0
S5, T4	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0	1	0	1	2	1	4	0	75	0	1	0	0	0	1	0	0
S5, T5	0	0	0	0	0	0	0	0	1	0	0	10	0	1	0	0	0	0	0	0	0	0	0	4	0	1	0	0	79	1	0	0	0	0	0	0
S5, T6	0	0	0	0	0	0	0	0	2	0	0	13	0	2	0	0	0	0	0	0	1	1	1	3	1	2	0	0	0	72	0	0	0	0	0	0
S6, t1	0	0	0	0	0	0	0	0	1	0	1	7	0	2	0	0	0	0	0	0	1	0	0	1	2	5	1	0	0	1	74	0	0	0	0	0
S6, T2	0	0	0	0	0	0	0	0	2	1	1	16	0	2	0	1	0	1	0	0	1	1	1	4	1	1	0	0	0	3	0	63	0	0	0	0
S6, T3	0	0	0	0	0	0	0	0	3	0	0	8	0	2	0	0	0	0	0	0	0	2	2	3	1	1	0	0	0	3	0	0	75	0	0	0
S6, T4	0	0	0	0	0	0	0	0	2	0	0	18	0	1	0	0	1	0	0	0	2	1	2	2	1	3	1	0	0	3	0	0	0	62	0	0
S6, T5	0	0	0	0	0	1	0	0	2	0	0	5	0	3	0	0	0	0	0	0	1	1	1	2	1	1	0	0	0	2	0	0	0	0	80	0
S6, T6	0	0	0	0	0	0	0	0	2	0	0	18	0	1	0	0	1	0	0	0	2	1	2	2	1	2	1	0	0	2	0	0	0	1	0	65

Figure A.3: Satellite 4 isolation confusion matrix for thruster isolation.

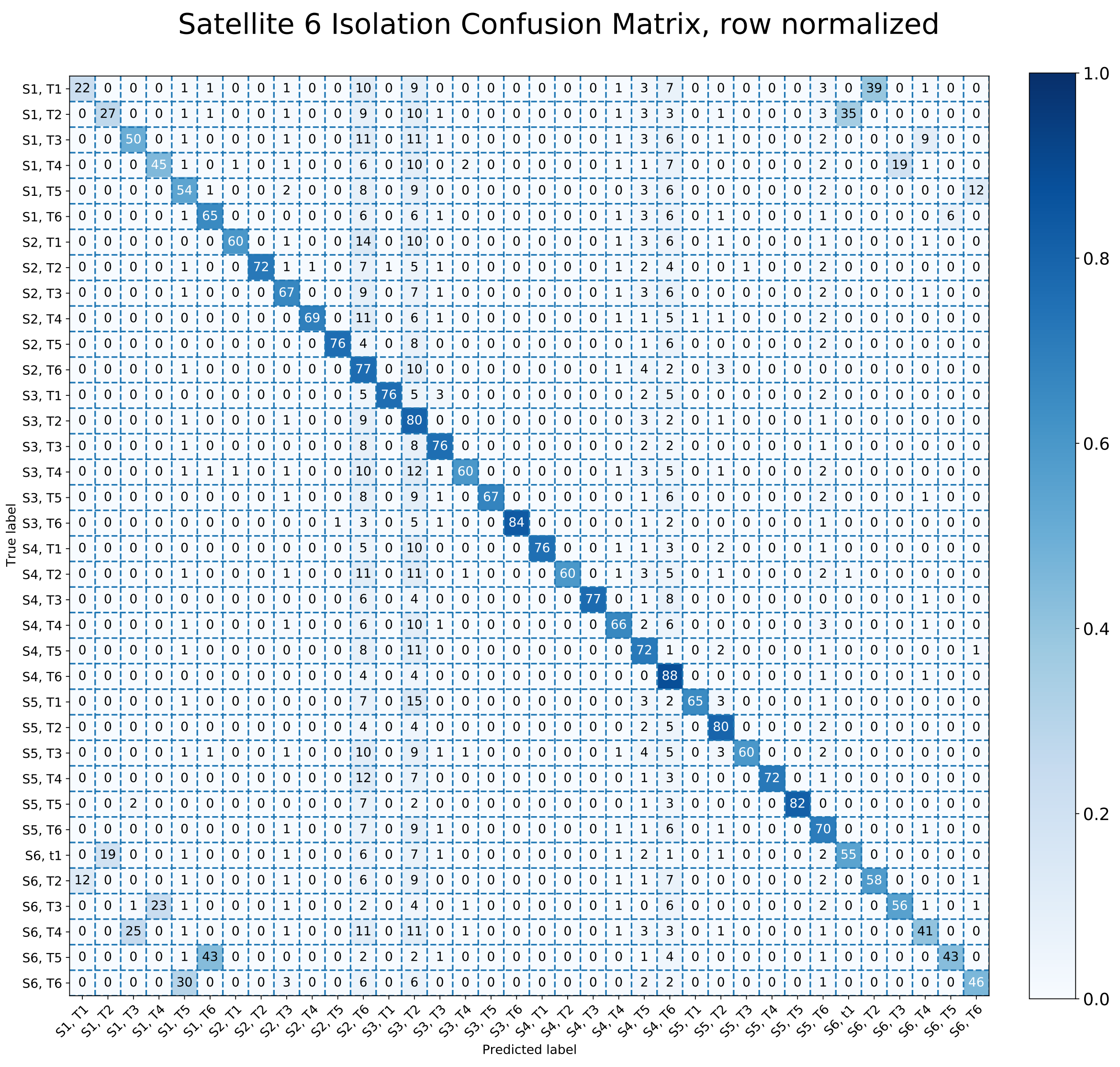


Figure A.4: Satellite 5 isolation confusion matrix for thruster isolation.

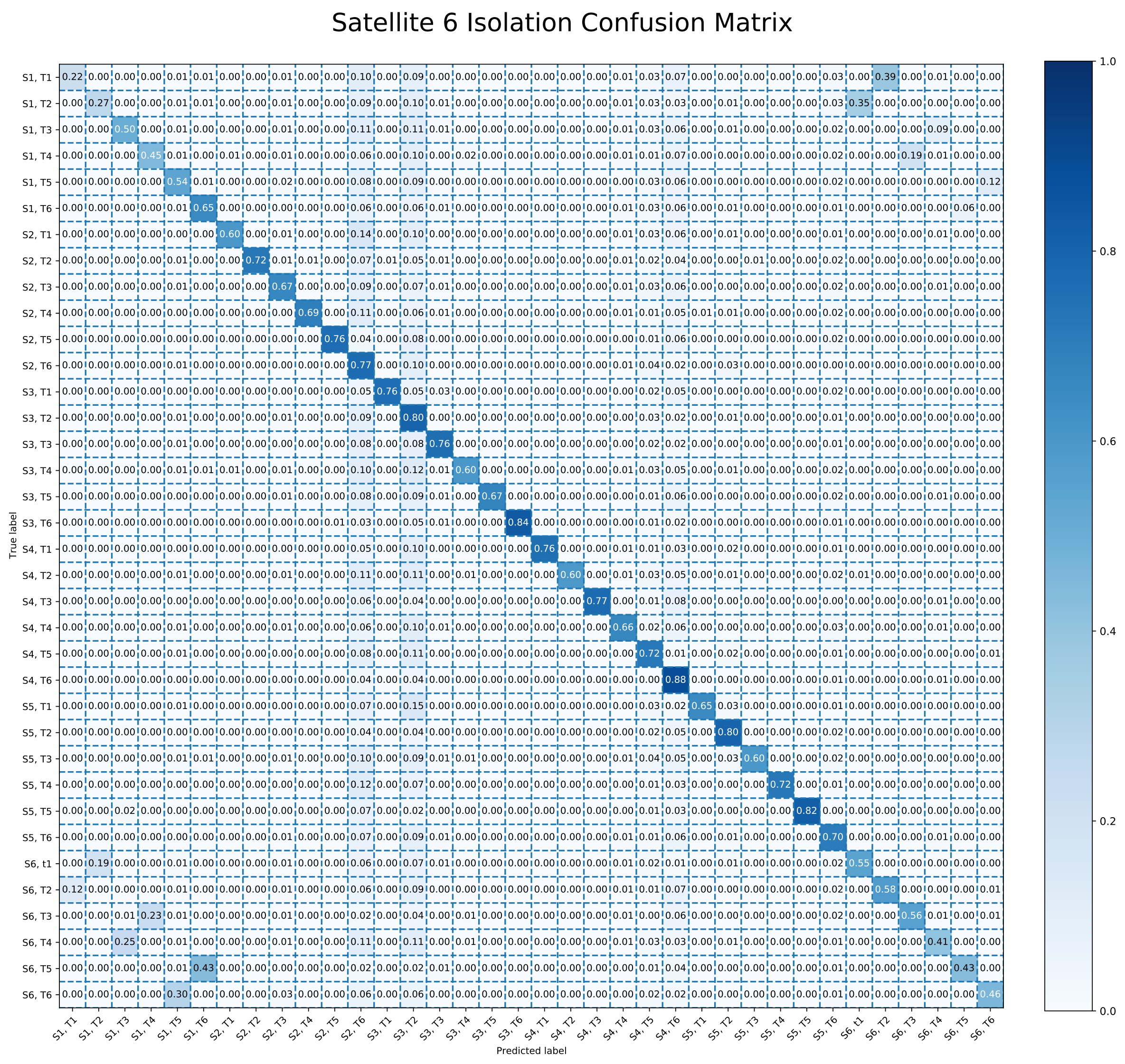


Figure A.5: Satellite 6 isolation confusion matrix for thruster isolation.

B Project Code

The entirety of the code written for this thesis can be found at
`https://github.com/MHenkel6/thesis-project-code/`.

B.1. Neural Network Code

This appendix contains some of the code for the verification of the neural networks.
A large percentage of the code for training the networks was re-purposed from code written by Sander Voss for his MSc Thesis "Application of Deep Learning for Spacecraft Fault Detection and Isolation" [92]. The code in this section requires the use of Tensorflow version 1.15.0, and might behave unexpectedly if a different version is used.

```python
## Verification of TFRecord writer
# Setup

"""Package Import and Dependencies"""
import tensorflow as tf
import tensorflow.train as tft
import numpy as np
import pandas as pd
import io
import os
import shutil as sh
import pickle
from pathlib import Path
import random
# Authentication for Managing Data
from google.colab import drive
drive.mount('/content/drive')
tf.enable_eager_execution()

rootPath = '/content/drive/My Drive/'
register = np.zeros(1)
while not np.any(register):
    try:
        with open(rootPath + 'DataRaw/Detection/Training/DataNoFault4N_32.csv','r') as f:
            register = np.genfromtxt(f,delimiter = ",")
    except:
        pass
np.shape(register)

"""Function Definitions"""
```

```python
def process(fileName):
    # Load correct File
    with tf.io.gfile.GFile(fileName,'r') as f:
        data = np.genfromtxt(f,delimiter = ",")
    # Seperate Relative Position & Velocity data
    info = data[0,:]
    settleIndex = 300
    faultTime = np.ceil(info[0])
    if faultTime<1:
        faultTime = 1e10
    faultSat = info[1]
    faultThruster = info[2]
    faultType = info[3]

    posvelData = data[1:,:]
    noRows = np.size(posvelData, 0)//24 - settleIndex
    noCols = 6
    dataSat1 = np.zeros([noRows, 4*noCols])
    dataSat2 = np.zeros([noRows, 4*noCols])
    dataSat3 = np.zeros([noRows, 4*noCols])
    dataSat4 = np.zeros([noRows, 4*noCols])
    dataSat5 = np.zeros([noRows, 4*noCols])
    dataSat6 = np.zeros([noRows, 4*noCols])

    faultLabel = np.zeros([noRows, 1])
    faultLabel[np.where(np.arange(noRows)>faultTime-settleIndex)] = 1
    settleOffset = settleIndex * 24
    dataSat1[:,0*noCols:1*noCols] = posvelData[settleOffset + 0::24,0:noCols]
    dataSat1[:,1*noCols:2*noCols] = posvelData[settleOffset + 1::24,0:noCols]
    dataSat1[:,2*noCols:3*noCols] = posvelData[settleOffset + 2::24,0:noCols]
    dataSat1[:,3*noCols:4*noCols] = posvelData[settleOffset + 3::24,0:noCols]

    dataSat2[:,0*noCols:1*noCols] = posvelData[settleOffset + 4::24,0:noCols]
    dataSat2[:,1*noCols:2*noCols] = posvelData[settleOffset + 5::24,0:noCols]
    dataSat2[:,2*noCols:3*noCols] = posvelData[settleOffset + 6::24,0:noCols]
    dataSat2[:,3*noCols:4*noCols] = posvelData[settleOffset + 7::24,0:noCols]

    dataSat3[:,0*noCols:1*noCols] = posvelData[settleOffset + 8::24,0:noCols]
    dataSat3[:,1*noCols:2*noCols] = posvelData[settleOffset + 9::24,0:noCols]
    dataSat3[:,2*noCols:3*noCols] = posvelData[settleOffset + 10::24,0:noCols]
    dataSat3[:,3*noCols:4*noCols] = posvelData[settleOffset + 11::24,0:noCols]

    dataSat4[:,0*noCols:1*noCols] = posvelData[settleOffset + 12::24,0:noCols]
    dataSat4[:,1*noCols:2*noCols] = posvelData[settleOffset + 13::24,0:noCols]
    dataSat4[:,2*noCols:3*noCols] = posvelData[settleOffset + 14::24,0:noCols]
    dataSat4[:,3*noCols:4*noCols] = posvelData[settleOffset + 15::24,0:noCols]

    dataSat5[:,0*noCols:1*noCols] = posvelData[settleOffset + 16::24,0:noCols]
    dataSat5[:,1*noCols:2*noCols] = posvelData[settleOffset + 17::24,0:noCols]
    dataSat5[:,2*noCols:3*noCols] = posvelData[settleOffset + 18::24,0:noCols]
    dataSat5[:,3*noCols:4*noCols] = posvelData[settleOffset + 19::24,0:noCols]

    dataSat6[:,0*noCols:1*noCols] = posvelData[settleOffset + 20::24,0:noCols]
    dataSat6[:,1*noCols:2*noCols] = posvelData[settleOffset + 21::24,0:noCols]
    dataSat6[:,2*noCols:3*noCols] = posvelData[settleOffset + 22::24,0:noCols]
    dataSat6[:,3*noCols:4*noCols] = posvelData[settleOffset + 23::24,0:noCols]

    sats = np.arange(6)
    sats = np.repeat(sats,noRows)
    sats = sats.reshape(-1,1)
    time = np.arange(settleIndex,noRows+settleIndex)

    time = np.tile(time,6).reshape(-1,1)

    data = np.concatenate((dataSat1,dataSat2,dataSat3,dataSat4,dataSat5,dataSat6),0)
    labels = np.concatenate((faultLabel,faultLabel,faultLabel,faultLabel,faultLabel,faultLabel
        ),0)
    data = np.concatenate((data,labels,time,sats),axis = 1)

    return data # ds3.map(lambda a,b,c: (a,b)), ds3.map(lambda a,b,c: c)

def create_tfrecord(filePath,fileName, data):
    # Create tfrecord

    header = ['x1','y1','z1','vx1','vy1','vz1',
              'x2','y2','z2','vx2','vy2','vz2',
              'x3','y3','z3','vx3','vy3','vz3',
              'x4','y4','z4','vx4','vy4','vz4',
              'label','time','sat']
    # Create dict
    x1  = tft.Feature(float_list = tft.FloatList(value = data[:,0]))
    y1  = tft.Feature(float_list = tft.FloatList(value = data[:,1]))
    z1  = tft.Feature(float_list = tft.FloatList(value = data[:,2]))
    vx1 = tft.Feature(float_list = tft.FloatList(value = data[:,3]))
    vy1 = tft.Feature(float_list = tft.FloatList(value = data[:,4]))
    vz1 = tft.Feature(float_list = tft.FloatList(value = data[:,5]))

    x2  = tft.Feature(float_list = tft.FloatList(value = data[:,6]))
    y2  = tft.Feature(float_list = tft.FloatList(value = data[:,7]))
    z2  = tft.Feature(float_list = tft.FloatList(value = data[:,8]))
    vx2 = tft.Feature(float_list = tft.FloatList(value = data[:,9]))
    vy2 = tft.Feature(float_list = tft.FloatList(value = data[:,10]))
    vz2 = tft.Feature(float_list = tft.FloatList(value = data[:,11]))

    x3  = tft.Feature(float_list = tft.FloatList(value = data[:,12]))
    y3  = tft.Feature(float_list = tft.FloatList(value = data[:,13]))
    z3  = tft.Feature(float_list = tft.FloatList(value = data[:,14]))
    vx3 = tft.Feature(float_list = tft.FloatList(value = data[:,15]))
    vy3 = tft.Feature(float_list = tft.FloatList(value = data[:,16]))
    vz3 = tft.Feature(float_list = tft.FloatList(value = data[:,17]))

    x4  = tft.Feature(float_list = tft.FloatList(value = data[:,18]))
    y4  = tft.Feature(float_list = tft.FloatList(value = data[:,19]))
    z4  = tft.Feature(float_list = tft.FloatList(value = data[:,20]))
    vx4 = tft.Feature(float_list = tft.FloatList(value = data[:,21]))
    vy4 = tft.Feature(float_list = tft.FloatList(value = data[:,22]))
    vz4 = tft.Feature(float_list = tft.FloatList(value = data[:,23]))

    label = tft.Feature(int64_list = tft.Int64List(value = data[:,24].astype(int)))
    time  = tft.Feature(int64_list = tft.Int64List(value = data[:,25].astype(int)))
    sats  = tft.Feature(int64_list = tft.Int64List(value = data[:,26].astype(int)))



    feature_dict = {'x1':x1,'y1':y1,'z1':z1,'vx1':vx1,'vy1':vy1,'vz1':vz1,
                    'x2':x2,'y2':y2,'z2':z2,'vx2':vx2,'vy2':vy2,'vz2':vz2,
                    'x3':x3,'y3':y3,'z3':z3,'vx3':vx3,'vy3':vy3,'vz3':vz3,
                    'x4':x4,'y4':y4,'z4':z4,'vx4':vx4,'vy4':vy4,'vz4':vz4,
                    'label':label,'time':time,'sats':sats}
    features = tft.Features(feature = feature_dict)
    DataExample = tft.Example(features = features)

    with tf.python_io.TFRecordWriter(filePath+fileName) as writer:
        writer.write(DataExample.SerializeToString())
    return

def decode_TFRecord(exampleProto):
# Read TFRecord file
    # Define features
    featureDescription = {
        'x1': tf.VarLenFeature(dtype=tf.float32),
        'y1': tf.VarLenFeature(dtype=tf.float32),
        'z1': tf.VarLenFeature(dtype=tf.float32),
        'vx1': tf.VarLenFeature(dtype=tf.float32),
        'vy1': tf.VarLenFeature(dtype=tf.float32),
        'vz1':tf.VarLenFeature(dtype=tf.float32),
        'x2': tf.VarLenFeature(dtype=tf.float32),
        'y2': tf.VarLenFeature(dtype=tf.float32),
        'z2': tf.VarLenFeature(dtype=tf.float32),
        'vx2': tf.VarLenFeature(dtype=tf.float32),
        'vy2': tf.VarLenFeature(dtype=tf.float32),
        'vz2':tf.VarLenFeature(dtype=tf.float32),
        'x3': tf.VarLenFeature(dtype=tf.float32),
        'y3': tf.VarLenFeature(dtype=tf.float32),
        'z3': tf.VarLenFeature(dtype=tf.float32),
        'vx3': tf.VarLenFeature(dtype=tf.float32),
        'vy3': tf.VarLenFeature(dtype=tf.float32),
        'vz3': tf.VarLenFeature(dtype=tf.float32),
        'x4': tf.VarLenFeature(dtype=tf.float32),
        'y4': tf.VarLenFeature(dtype=tf.float32),
        'z4': tf.VarLenFeature(dtype=tf.float32),
        'vx4': tf.VarLenFeature(dtype=tf.float32),
        'vy4': tf.VarLenFeature(dtype=tf.float32),
        'vz4': tf.VarLenFeature(dtype=tf.float32),
        'label': tf.VarLenFeature(dtype=tf.int64),
        'time': tf.VarLenFeature(dtype=tf.int64),
        'sats': tf.VarLenFeature(dtype=tf.int64)}

    # Extract features from serialized data
    return  tf.io.parse_single_example(exampleProto, featureDescription)

filePathCheck = rootPath + 'DataRaw/Detection/Training/'
fileName = "DataOpenFault4N_66_404.csv"

# Read Data from Test file
data = process(filePathCheck + fileName)
# Create TFRecord file
filePathWrite = rootPath + "Colab Notebooks/Verification/"
fileNameCheck = 'Test_0.tfrecord'
create_tfrecord(filePathWrite, fileNameCheck,data)

# Read Created TFRecord File
readSet = tf.data.TFRecordDataset(filePathWrite+ fileNameCheck)
# Define features
read_features = {
    'x1': tf.VarLenFeature(dtype=tf.float32),
    'y1': tf.VarLenFeature(dtype=tf.float32),
    'z1': tf.VarLenFeature(dtype=tf.float32),
    'vx1': tf.VarLenFeature(dtype=tf.float32),
    'vy1': tf.VarLenFeature(dtype=tf.float32),
    'vz1':tf.VarLenFeature(dtype=tf.float32),
    'x2': tf.VarLenFeature(dtype=tf.float32),
    'y2': tf.VarLenFeature(dtype=tf.float32),
    'z2': tf.VarLenFeature(dtype=tf.float32),
    'vx2': tf.VarLenFeature(dtype=tf.float32),
    'vy2': tf.VarLenFeature(dtype=tf.float32),
    'vz2':tf.VarLenFeature(dtype=tf.float32),
    'x3': tf.VarLenFeature(dtype=tf.float32),
    'y3': tf.VarLenFeature(dtype=tf.float32),
    'z3': tf.VarLenFeature(dtype=tf.float32),
    'vx3': tf.VarLenFeature(dtype=tf.float32),
    'vy3': tf.VarLenFeature(dtype=tf.float32),
    'vz3':tf.VarLenFeature(dtype=tf.float32),
    'x4': tf.VarLenFeature(dtype=tf.float32),
    'y4': tf.VarLenFeature(dtype=tf.float32),
    'z4': tf.VarLenFeature(dtype=tf.float32),
    'vx4': tf.VarLenFeature(dtype=tf.float32),
    'vy4': tf.VarLenFeature(dtype=tf.float32),
    'vz4':tf.VarLenFeature(dtype=tf.float32),
    'label': tf.VarLenFeature(dtype=tf.int64),
    'time': tf.VarLenFeature(dtype=tf.int64),
    'sats': tf.VarLenFeature(dtype=tf.int64)}

# Extract features from serialized data
for s in readSet.take(1):
    feature = tf.parse_single_example(s,features = read_features)

# Print features
x1read = tf.sparse.to_dense(feature['x1']).numpy()
x1write = data[:,0]
print(np.sum(np.abs(x1read-x1write)/len(x1read)))
```

Verification of Dataset Pre-processing Only the relevant section (i.e. the logic for checking the output) is shown here, the full script can be seen in the github project linked above.

```
fileListDataset = tf.data.TFRecordDataset(listdir)
decodedDataset = fileListDataset.map(decode_TFRecord)
processedDataset = decodedDataset.flat_map(preprocess)

datalabels = ['x1','y1','z1','vx1' ,'vy1','vz1','x2','y2','z2','vx2','vy2','vz2','x3',
        'y3','z3','vx3','vy3','vz3','x4','y4','z4','vx4','vy4','vz4','label','time','sats']

# Take data and labels of entire run (just the decoded Dateset)
npdataFull = np.zeros([33612,27])
for data in decodedDataset.take(-1):
    for number, label in enumerate(datalabels):
        npdataFull[:,number] = tf.sparse.to_dense(data[label]).numpy()

# Take data from the processed Dataset and compare if it matches with the correct
# spot in the full set
i = 0
indexList = []
incorrectLabelList = []
for data,label in processedDataset.take(-1):
    if i < 33612 - 51:
        npdata = data.numpy()
        diff = npdataFull[i:i+50,0:24] - npdata
        while np.abs(np.sum(diff)) > 1e-9 :
            indexList.append(i)
            i += 1
            diff = npdataFull[i:i+50,0:24] - npdata
        if np.abs(np.sum(diff)) < 1e-9 :
            if not label.numpy() == npdataFull[i,-3]:
                print(np.sum(diff))
                print(label.numpy(),npdataFull[i,-3])
                incorrectLabelList.append(i)
        i += 1
print(indexList[:10])
if len(indexList) == 250 and len(incorrectLabelList) == 0:
    print("Slicing Preprocessing Successful")
```

Verification of Individual Satellite Data Selection Similar to the above, only the relevant section is shown here.

```

# Check Satellite Selection
satView = 1
satProcessedDataset = decodedDataset.flat_map(lambda x: preprocDetInd(x,50,satView))
i0 = np.where(npdataFull[:,-1] == satView)[0][0]
i = i0
indexList = []

for data,label in satProcessedDataset.take(-1):
    if i < i0 + 33612//6:
        npdata = data.numpy()
        diff = npdataFull[i:i+50,0:24] - npdata
        while np.abs(np.sum(diff)) > 1e-9 and i < i0 + 33612//6:
            indexList.append(i)
            i = i+1
            diff = npdataFull[i:i+50,0:24] - npdata

    i += 1
print(indexList[:10])
if len(indexList) == 50:
    print("Satellite Selection Successful")
```

B.2. Formation Simulation Code

This appendix contains the definitions of the `formation` and `spacecraft` class in MATLAB, used in the simulation of the formation. The main scripts used to run the simulation are not included here due to their length but can be found in the "Matlab - Formation Simulation" folder in `https://github.com/MHenkel6/thesis-project-code/`.

```
classdef formation < handle
    %FORMATION Class containing the formation as a whole and relevant
    %methods
    properties
        spacecraftArray; % Array containing spacecraft objects in formation
        nSpacecraft; % no. of spacecraft
        formationSize;  % distance between satellite
        nImpulse; % number of impulses used to control orbit

        % Center Orbit parameters
        orbitParam; % Array to jointly hold orbital parameters
        a; % Semi-major axis [m]
        e; % Eccentricity [-]
        inc; % Inclination [rad]
        O; % RAAN [rad]
        o; % Argument of periapsis [rad]
        nu; % True anomaly [rad]
        truLon; % True Longitude [rad]
        argLat; % Argument of latitude [rad]
        lonPer; % Longitude of Periapsis [rad]
        p; % Semi-latus rectum [m]

        % Control parameteers
        controltype; % Continuous (1) vs discrete (2) control system
        disctype; % Discretization type if discrete control system

        % Navigation parameters
        navType = 1;  % Cartesian (1) or Spharical (2) noise and bias

        % Propagation parameters
        disturbancesOn = 1; % Boolean to en/disable disturbanceForces
    end

    methods
        function obj = formation(nSpacecraft,type,center,velCenter,dist,qOrientation,
            controltype,disctype,spacecraftParameters,residualCovariance,residualMean,navType,
            navArray)
            %FORMATION Construct an instance of this class
            %   Inputs:
            %   nSpacecraft             = number of spacecraft in formation
            %   type                    = Formation type
            %   center                  = position of geometric center of Formation in ECI
            %   velCenter               = velocity of geometric center of Formation in ECI
            %   dist                    = size parameter of formation (distance between
            %                             sats)
            %   qOrientation            = orientation quaternion relative to ECI
            %   controlType             = continuous vs discrete control
            %                             method
            %   discType                = discretization type
            %                             (ZOH,FOH,Impulse)
            %   spacecraftParameters    = array of mass, size, moment of
            %                             inertia and other relevant
            %                             spacecraft parameters, see
            %                             spacracraft class
            %   residualCovariance      = Covariance matrix of residual
            %                             vector in the faultless case
            %                             (determined from the
            %                             simulation)
            %   residualMean            = mean of the residual vector

            % Construct 6 spacecraft objects
            obj.nSpacecraft = nSpacecraft;
            obj.formationSize = dist;
            % If octahedron type. get 6 equidistant points from center
            switch type
                case 'octahedron'
                    distCenter = dist*sqrt(2)/2; %distance from center
                    distArray = [0,0,distCenter;
                                 distCenter,0,0;
                                 0,distCenter,0;
                                 -distCenter,0,0;
                                 0 -distCenter,0;
                                 0,0,-distCenter];
                case 'single'
                    distArray = [0,0,0];
                otherwise
                    warning('Unsupported Formation Type')
            end
            % Reorient positions to align with formation frame
            distArrayRot =  quatrotate(qOrientation,distArray);
            positions = center + distArrayRot+1*randn(6,3);

            % Reference Orbit
            [a,e,inc,O,o,nu,truLon,argLat,lonPer,p] = rv2orb(center',velCenter');
            obj.a = a;
            obj.e = e;
            obj.inc = inc;
            obj.O = O;
            obj.o = o;
            obj.nu = nu;
            obj.truLon = truLon;
            obj.argLat = argLat;
            obj.lonPer = lonPer;
            obj.p = p;
            obj.orbitParam = [a,e,inc,O,o,nu,truLon,argLat,lonPer,p];
            % Create Spacecraft
            sats=[];
            for inc = 1:nSpacecraft
                 % Fixed attitude
                attQuat = qOrientation';
                spinVec = zeros([3,1]);
                spinRate = 0;
                formationParameters = [inc,dist,qOrientation,obj.orbitParam];
                sp   = spacecraft(positions(inc,:),velCenter,attQuat,spinRate*spinVec,...
                                 spacecraftParameters,formationParameters);
                sats = [sats(:)',sp];
            end


            % Give each spacecraft its neighbors
            % Neighbor sat 1
            sats(1).spacecraftNeighbor1 = sats(2);
            sats(1).spacecraftNeighbor2 = sats(3);
            sats(1).spacecraftNeighbor3 = sats(4);
            sats(1).spacecraftNeighbor4 = sats(5);

            sats(1).n1Offset = -(sats(2).centerOffset - sats(1).centerOffset);
            sats(1).n2Offset = -(sats(3).centerOffset - sats(1).centerOffset);
            sats(1).n3Offset = -(sats(4).centerOffset - sats(1).centerOffset);
            sats(1).n4Offset = -(sats(5).centerOffset - sats(1).centerOffset);
            % Neighbor sat 2
            sats(2).spacecraftNeighbor1 = sats(1);
            sats(2).spacecraftNeighbor2 = sats(5);
            sats(2).spacecraftNeighbor3 = sats(6);
            sats(2).spacecraftNeighbor4 = sats(3);

            sats(2).n1Offset = -(sats(1).centerOffset - sats(2).centerOffset);
            sats(2).n2Offset = -(sats(5).centerOffset - sats(2).centerOffset);
            sats(2).n3Offset = -(sats(6).centerOffset - sats(2).centerOffset);
            sats(2).n4Offset = -(sats(3).centerOffset - sats(2).centerOffset);
            % Neighbor sat 3
            sats(3).spacecraftNeighbor1 = sats(1);
            sats(3).spacecraftNeighbor2 = sats(2);
            sats(3).spacecraftNeighbor3 = sats(6);
            sats(3).spacecraftNeighbor4 = sats(4);

            sats(3).n1Offset = -(sats(1).centerOffset - sats(3).centerOffset);
            sats(3).n2Offset = -(sats(2).centerOffset - sats(3).centerOffset);
            sats(3).n3Offset = -(sats(6).centerOffset - sats(3).centerOffset);
            sats(3).n4Offset = -(sats(4).centerOffset - sats(3).centerOffset);
            % Neighbor sat 4
            sats(4).spacecraftNeighbor1 = sats(3);
            sats(4).spacecraftNeighbor2 = sats(6);
            sats(4).spacecraftNeighbor3 = sats(5);
            sats(4).spacecraftNeighbor4 = sats(1);

            sats(4).n1Offset = -(sats(3).centerOffset - sats(4).centerOffset);
            sats(4).n2Offset = -(sats(6).centerOffset - sats(4).centerOffset);
            sats(4).n3Offset = -(sats(5).centerOffset - sats(4).centerOffset);
            sats(4).n4Offset = -(sats(1).centerOffset - sats(4).centerOffset);
            % Neighbor sat 5
            sats(5).spacecraftNeighbor1 = sats(4);
            sats(5).spacecraftNeighbor2 = sats(6);
            sats(5).spacecraftNeighbor3 = sats(2);
            sats(5).spacecraftNeighbor4 = sats(1);

            sats(5).n1Offset = -(sats(4).centerOffset - sats(5).centerOffset);
            sats(5).n2Offset = -(sats(6).centerOffset - sats(5).centerOffset);
            sats(5).n3Offset = -(sats(2).centerOffset - sats(5).centerOffset);
            sats(5).n4Offset = -(sats(1).centerOffset - sats(5).centerOffset);
            % Neighbor sat 6
            sats(6).spacecraftNeighbor1 = sats(5);
            sats(6).spacecraftNeighbor2 = sats(4);
            sats(6).spacecraftNeighbor3 = sats(3);
            sats(6).spacecraftNeighbor4 = sats(2);

            sats(6).n1Offset = -(sats(5).centerOffset - sats(6).centerOffset);
            sats(6).n2Offset = -(sats(4).centerOffset - sats(6).centerOffset);
            sats(6).n3Offset = -(sats(3).centerOffset - sats(6).centerOffset);
            sats(6).n4Offset = -(sats(2).centerOffset - sats(6).centerOffset);
            % Assign satellite array to formation property
            obj.spacecraftArray = sats;

            obj.controltype = controltype;
            obj.disctype = disctype;
            if exist('navType')
                obj.navType = navType;
            else
                obj.navType = 1;
            end

            if navType == 2
                rangeNoiseSize = navArray(1);
                rangeBiasSize = navArray(2);
                angleNoiseSize = navArray(3);
                angleBiasSize = navArray(4);
                for sc = obj.spacecraftArray
                    sc.rangeBias = (2*randi(2,4,1)-3)*rangeBiasSize;
                    sc.rangeNoise = rangeNoiseSize;
                    sc.angleBias = (2*randi(2,4,2)-3)*angleBiasSize;
                    sc.angleNoise =  angleNoiseSize;
                end
            end
            % Give spacecraft 1 the residual Covariance for FDI function
            obj.spacecraftArray(1).residualMean = residualMean;
            % Finish off with inital measurements and control commands
            for ii = 1:6
                obj.spacecraftArray(ii).navigation(obj.navType);
                obj.spacecraftArray(ii).controlCommand(0,obj.controltype,obj.disctype);
                obj.spacecraftArray(ii).thrustAlloc(0);
                cF = obj.spacecraftArray(ii).controlForce(0);
            end
        end

        function rk4Prop(obj,time,dt)
```

```
            %rk4Prop Propagate the states of the formation
            %   Main function to propagate the state of the entire
            %   formation by one time step dt

            % nSpacecraft x 6 Matrix containing state derivatives for
            % all spacecraft
            devArray = zeros(obj.nSpacecraft,6);
            KalmanMeasurement = zeros(6,24);
            KalmanControlStates = zeros(1,3*6);
            % Determine state changes for each satellite
            for i = 1:obj.nSpacecraft
                spacecraft =  obj.spacecraftArray(i);
                %spacecraft.guidance(time);

                % Perform relative and absolute measurements
                spacecraft.navigation(obj.navType);
                % Determine thruster output
                spacecraft.controlCommand(time,obj.controltype,obj.disctype);
                % Determine thruster opening time based on commanded input
                spacecraft.thrustAlloc(time);
                % Determine actual thrust exerted by thrusters in ECI frame
                cF = spacecraft.controlForce(time);

                % Gather measurements and control states for the Kalman
                % filter
                KalmanMeasurement(i,:) = reshape(spacecraft.relEstHill',1,24);
                KalmanControlStates(1,(i-1)*3+1:i*3) = spacecraft.cAccHill;

                % Runge-Kutta 4 Integration Scheme
                dev1 = dynamics(time, spacecraft.position,spacecraft.velocity,...
                                spacecraft.mass,cF,obj.disturbancesOn);
                dev2 = dynamics(time+dt/2, spacecraft.position+dt/2*dev1(1:3),...
                                spacecraft.velocity+dt/2*dev1(4:6),...
                                spacecraft.mass,cF,obj.disturbancesOn);
                dev3 = dynamics(time+dt/2, spacecraft.position+dt/2*dev2(1:3),...
                                spacecraft.velocity+dt/2*dev2(4:6),...
                                spacecraft.mass,cF,obj.disturbancesOn);
                dev4 = dynamics(time+dt, spacecraft.position+dt*dev3(1:3),...
                                spacecraft.velocity+dt*dev3(4:6),...
                                spacecraft.mass,cF,obj.disturbancesOn);
                devTotal = dt*(dev1 + 2*dev2 + 2*dev3 + dev4)/6;
                devArray(i,:) = devTotal;

                % Determine change in attitude and rotation
                rotState = [spacecraft.attitude;
                            spacecraft.spin];
                rotDev1 = rotDynamics(rotState,spacecraft.inertia);
                rotDev2 = rotDynamics(rotState+rotDev1*dt/2,spacecraft.inertia);
                rotDev3 = rotDynamics(rotState+rotDev2*dt/2,spacecraft.inertia);
                rotDev4 = rotDynamics(rotState+rotDev3*dt,spacecraft.inertia);

                rotDevTotal = dt*(rotDev1 + 2*rotDev2 + 2*rotDev3 + rotDev4)/6;
                spacecraft.updateAtt(rotDevTotal);
            end
            % Kalman filer update
            % Use all relative Measurements in ECI to get proper Hill frame

            % Reorder Measurements to be in correct order
            % Correct Order: [s12,s13,s14,s15,s23,s34,s45,s52,s65,s64,s63,s62, ...
            %                 s21,s31,s41,s51,s32,s43,s54,s25,s56,s46,s36,s26]
            % Current Order:
```

```
            %                 [s12,s13,s14,s15,s21,s25,s26,s23,s31,s32,s36,s34,...
            %                  s43,s46,s45,s41,s54,s56,s52,s51,s65,s64,s63,s62

            KalmanMeasurementCorrect = zeros(1,144);
            KalmanMeasurementCorrect(1:24)      = KalmanMeasurement(1,:);
            KalmanMeasurementCorrect(25:30)     = KalmanMeasurement(2,19:24);
            KalmanMeasurementCorrect(31:36)     = KalmanMeasurement(3,19:24);
            KalmanMeasurementCorrect(37:42)     = KalmanMeasurement(4,13:18);
            KalmanMeasurementCorrect(43:48)     = KalmanMeasurement(5,13:18);
            KalmanMeasurementCorrect(49:72)     = KalmanMeasurement(6,:);

            KalmanMeasurementCorrect(73:78)     = KalmanMeasurement(2,1:6);
            KalmanMeasurementCorrect(79:84)     = KalmanMeasurement(3,1:6);
            KalmanMeasurementCorrect(85:90)     = KalmanMeasurement(4,19:24);
            KalmanMeasurementCorrect(91:96)     = KalmanMeasurement(5,19:24);
            KalmanMeasurementCorrect(97:102)    = KalmanMeasurement(3,7:12);
            KalmanMeasurementCorrect(103:108)   = KalmanMeasurement(4,1:6);
            KalmanMeasurementCorrect(109:114)   = KalmanMeasurement(5,1:6);
            KalmanMeasurementCorrect(115:120)   = KalmanMeasurement(2,7:12);
            KalmanMeasurementCorrect(121:126)   = KalmanMeasurement(5,7:12);
            KalmanMeasurementCorrect(127:132)   = KalmanMeasurement(4,7:12);
            KalmanMeasurementCorrect(133:138)   = KalmanMeasurement(3,13:18);
            KalmanMeasurementCorrect(139:144)   = KalmanMeasurement(2,13:18);

            % Perform Kalman filter update
            obj.spacecraftArray(1).kalmanUpdate(KalmanMeasurementCorrect,KalmanControlStates);
            %obj.spacecraftArray(1).extendedKalmanUpdate(time,KalmanMeasurementCorrect,
                KalmanControlStates);

            % Perform FDI once residual signal has settle (~1000 seconds
            if time>1000
                obj.spacecraftArray(1).fdi(time)
            end

            % Update all positions
            for i = 1:obj.nSpacecraft
                obj.spacecraftArray(i).updatePos(devArray(i,:));
            end
            % For output purposes, another measurement is taken
            for ii = 1:obj.nSpacecraft
                obj.spacecraftArray(ii).navigation(obj.navType);
            end
        end

        function states = getStates(obj)
            % Return true position and velocity of the formation in ECI
            states = zeros(obj.nSpacecraft,6);
            for i = 1:obj.nSpacecraft
                states(i,:) = obj.spacecraftArray(i).getState();
            end
        end

        function [relState,relStateHill] = getRelStates(obj, zeroed)
            % Return the relative measurements for each member of the
            % formation, both in ECI and in the HILL frame
            relState = zeros(4*obj.nSpacecraft,6);
            relStateHill = zeros(4*obj.nSpacecraft,6);
            for i = 1:6
                sc = obj.spacecraftArray(i);
                relEst = sc.relativeEst;

                relDiff = relEst - [sc.n1Offset,0,0,0;
                                    sc.n2Offset,0,0,0;
                                    sc.n3Offset,0,0,0;
                                    sc.n4Offset,0,0,0];
                if zeroed
                    relState((i-1)*4+1:i*4,:) = relDiff;
                else
                    relState((i-1)*4+1:i*4,:) = relEst;
                end
                relStateHill((i-1)*4+1:i*4,:) = sc.relEstHill;
            end
        end
        function [absMeas] = getAbsoluteMeasurement(obj)
            % Return measured position and velocity of the formation in ECI
            absMeas = zeros(6,6);
            for ii = 1:6
                absMeas(ii,:) = [obj.spacecraftArray(ii).positionEst,...
                                 obj.spacecraftArray(ii).velocityEst];
            end
        end
        function [command, err] = getControlCommands(obj)
            % Return current commanded control acceleration
            command = zeros(6,3);
            err = zeros(6,6);
            for ii = 1:6
                sc = obj.spacecraftArray(ii);
                command(ii,:) = sc.cAccHill;% sc.cImpulseHill;%
                err(ii,:) = sc.cErr; % control err;
            end

        end
        function setFault(obj,faultTime,satelliteNo,thrusterNo,faultType,faultParam)
            % Set faultvector of selected satellite and thruster to
            % faultparameter
            faultySat = obj.spacecraftArray(satelliteNo);
            faultySat.faultTime = faultTime;

            if faultType == 1 % Closed thruster fault
                faultySat.faultVectorClosed(thrusterNo) = 1-faultParam;
            elseif faultType == 2 % Open thruster
                faultySat.faultVectorOpen(thrusterNo) = faultParam;
            end
        end

        %% Following functions mainly used in the optimization of the formation
        function setStates(obj,center,size,qOrientation,velCenter)
            % Set state of formation for optimization
            distCenter = size*sqrt(2)/2; %distance from center
            distArray = [0,0,distCenter;
                         distCenter,0,0;
                         0,distCenter,0;
                         -distCenter,0,0;
                         0 -distCenter,0;
                         0,0,-distCenter];
            distArrayRot =  quatrotate(qOrientation,distArray);
            positions = center + distArrayRot;
            it = 1 ;
            for sat = obj.spacecraftArray
                sat.setState([positions(it,:),velCenter]);
            end
        end
```

```

%         function [deltaVs, positions] = guideOpt(obj,dt,T)
%             % Calculate deltaV required to keep current formation for 1
%             % orbit
%             % Returns required DeltaVs per satellite and the positions wrt
%             % the center of the formation.
%
%             % Propagate orbit for one revolution with Keplerian dynamics
%             % only
%             for t = 0:dt:T
%                 devArray = zeros(obj.nSpacecraft,6);
%                 for i = 1:obj.nSpacecraft
%                     spacecraft =  obj.spacecraftArray(i);
%                     dev1 = dynamicsKepler(spacecraft.position,spacecraft.velocity);
%                     dev2 = dynamicsKepler(spacecraft.position+dt/2*dev1(1:3) ,...
%                                           spacecraft.velocity+dt/2*dev1(4:6));
%                     dev3 = dynamicsKepler(spacecraft.position+dt/2*dev2(1:3) ,...
%                                           spacecraft.velocity+dt/2*dev2(4:6));
%                     dev4 = dynamicsKepler(spacecraft.position+dt*dev3(1:3) ,...
%                                           spacecraft.velocity+dt*dev3(4:6));
%
%                     devTotal = dt*(dev1 + 2*dev2 + 2*dev3 + dev4)/6;
%                     devArray(i,:) = devTotal;
%                 end
%                 % Update all positions
%                 for i = 1:obj.nSpacecraft
%                     obj.spacecraftArray(i).updatePos(devArray(i,:));
%                 end
%             end
%             deltaVs = 0;
%             positions = 0;
%         end

    end
end
```

```
classdef spacecraft < handle
    properties
        position; % Position in ECI in m, 3 doubles
        velocity; % velocity in ECI in m/s 3 doubles
        attitude; % attitude quaternion in ECI
        spin; % rotation rate in rad/s

        positionEst; % estimated position in ECI in m, 3 doubles
        velocityEst; % estimated velocity in ECI in m, 3 doubles
        attitudeEst; % estimated attitude quaternion in ECI

        relativeEst; % estimated/measured relative positions [m]
        relEstHill;  % Estimated distance,velocity in Hill frame [m, m/s]

        posSelfBias; % position bias for absolute position determination in m
        velSelfBias; % velocity bias for absolute velocity determination in m
        posRelBias; % position bias for relative position determination in m
        velRelBias; % position bias for relative velocity determination in m

        posSelfNoise; % standard deviation of absolute position noise [m]
        velSelfNoise; % standard deviation of absolute velocity noise [m/s]
        posRelNoise; % standard deviation of relative position noise [m]
        velRelNoise; % standard deviation of relative velocity noise [m/s]

        rangeBias; % Bias in range measurement for robustness cases [m]
        angleBias; % Bias in angle measurement for robustness cases [rad]

        rangeNoise; % standard deviation in range measurement for robustness cases [m]
        angleNoise; % standard deviation in angle measurement for robustness cases [rad]

        mass = 100 ; % satellite mass [kg]
        inertia = [100,0,0;
                   0,100,0;
                   0,0,100]; % Inertia Tensor
        dim = 1;% satellite dimensions in m


        % Formation Info
        centerOffset; % Distance to center of ideal formation [m]
        formationNo; % Satellite number in formation
        formationOrientation; % Orientation quaternion of the formation
        formationSize; % Distance between adjacent satellites [m]
        formationCenterOrbit; % Orbital parameters of center reference orbit

        %Thruster Parameters
        minImpulse = 0.0001; % minimum inpulse bit
        isp = 200; % specific impulse in s
        thrust = 4; % Maximum thrust force of satellite thruster [4]
        Tconfig = [1,-1,0,0  ,0,0  ;
                   0,0  ,1,-1,0,0  ;
                   0,0  ,0,0  ,1,-1];% thruster configuration matrix
        faultVectorClosed = ones(6,1);
        faultVectorOpen = zeros(6,1);
        % boolean to adjust fault time to after first firing of affected thruster
        firstThrust = false;
        % Occurenf of fault time
        faultTime = 1e9;

        % Pseudo-inverse of the thruster configuration
        TconfigInv = [0.5,0    ,0    ;
                      -0.5,0    ,0    ;
                       0   ,0.5  ,0    ;
                       0   ,-0.5,0    ;
                       0   ,0    ,0.5  ;
                       0   ,0    ,-0.5]


        thrustInterval; % Time in between beginnings of burn windows [s]
        burnTime; % Duration of burn window [s]
        accumDV = 0; % accumulated DV over orbit [m/s]
        maxBurnTime = 0; % maximum obverved burn time;
        thrusterOpeningCount = zeros(6,1); % Amount of times each thruster has opened
        thrusterOpeningTime = zeros(6,1); % Cumulative time of each thruster opening [s]
        spentProp = 0; % Amount of fuel expelled during burns [kg]
        % Satellite neighbors (spacecraft object and offset in ideal
        % conditions)
        spacecraftNeighbor1;
        n1Offset;
        spacecraftNeighbor2;
        n2Offset;
        spacecraftNeighbor3;
        n3Offset;
        spacecraftNeighbor4;
        n4Offset;

        %Navigation Matrices
        THillECI; % Transformation matrix from Hill to ECI frame
        Tdot; % Derivative of THillECI

        % Controller Parameters
        Kcont; % Continuous LQR controller gain
        KZOH; % Discrete LQR controller gain using Zero−Order−Hold Discretization
        Kimp; % Discrete LQR controller gain using Impulse−Method Discretization

        % Current Control Commands
        cImpulse; % Commanded control impulse in ECI
        cImpulseHill; % Commanded control impulse in Hill Frame
        cFECI; % Actual force output from thruster in ECI [N]
        cAccHill; % Commanded acceleration in Hill Frame [m/s^2]
        cErr; % Control error [m,m/s]
        opTimes; % Opening time vector for thruster allocation [s]
        thrustAllocComplete = 0; % Boolean to indicate thruster allocation
        burnFraction; % vector to scale down thrust
        pastControl;
        % Kalman filter Matrices

        A; % State matrix
        B; % Input Matrix
        C; % Measurement Matrix
        D; % Feed−through Matrix

        Q; % Covariance matrix of system noise

        R; % Covariance matrix of measurement noise
        Kk; % Kalman gain
        P;% Solution to Discrete Ricatti Equation from Kalman filter
        M; % Innovation gain

        kalmanEstimate; % Current Kalman estimate of formation state
        kalmanResidual; % Current Kalman residual
        ErrCov; % Error Covariance Matrix
        alphLow = 0.025; % Low pass filter parameter
        kalLowPass; % Kalman filter passed through discrete low pass filter


        % Extended Kalman filter Matrices
        Bf;
        extQ;
        extKalmanEstimate;
        extKalmanResidual;
        extErrCov;

        % Relative state and absolute state from last time step
        lastRelativeEst;
        lastPositionEst;
        lastVelocityEst;

        % Fault Detection  Variables
        residualCovariance; % Covariance Matrix of residual/innovation
                            % vector in the faultless case
        Qinv; % Inverse of residual Covariance, saved here for increased speed
        residualMean; % Mean of residual vector
        faultThreshold; % Threshold before fault alarm is issued
        gk; % Cumulative sum of Log−likelihood ratio, recursively computed
        detect; % Boolean to indicate detection
        isolate; % Index of detected faulty thruster
        isoCounter; % Counter to indicate successive identifications
        faultDetTime; % time at which fault was detected [s]
        faultDetected; % Boolean to indicate if fault was found at all
        % FDI matrix
        tempMatrix; % Temporary matrix saved here for reduced computing effort
    end

    methods
        function obj = spacecraft(position,velocity, attitude,spin,...
                                    spacecraftParameters,formationParameters)
            %Constructor
            obj.position = position;
            obj.velocity = velocity;
            obj.attitude = attitude;
            obj.spin = spin;


            % Unpack and assign parameters
            obj.mass = spacecraftParameters{1};
            obj.dim = spacecraftParameters{2};
            obj.inertia = spacecraftParameters{3};
            obj.thrustInterval = spacecraftParameters{4};
            obj.burnTime =  spacecraftParameters{5};
            obj.thrust = spacecraftParameters{6};
            obj.isp = spacecraftParameters{7};
            obj.minImpulse = 1e-3*spacecraftParameters{8};
            selfBiasSize = spacecraftParameters{9};
            relBiasSize  = spacecraftParameters{10};
            velBiasSize  = spacecraftParameters{11};
            obj.posSelfNoise = spacecraftParameters{12};
            obj.velSelfNoise = spacecraftParameters{14};
            obj.posRelNoise = spacecraftParameters{13};
            obj.velRelNoise = spacecraftParameters{14};
            alpha1 = spacecraftParameters{15};
            alpha2 = spacecraftParameters{16};

            % Bias vectors
            obj.posSelfBias = random_unit_vector()'*selfBiasSize;
            obj.velSelfBias = random_unit_vector()'*velBiasSize;
            obj.posRelBias  = random_unit_vector()'*relBiasSize;
            obj.velRelBias  = random_unit_vector()'*velBiasSize;

            % Range Navigation biases and noises
            %obj.rangeBias = (2*(randi(2)-1)-1)*Constants.rangeBiasSize;
            %obj.angleBias = [cos(2*pi*rand()),sin(2*pi*rand())]*Constants.angleBiasSize;

            % Unpack and assign formation Parameters
            formationNo = formationParameters(1);
            formationSize = formationParameters(2);
            formationOrientation = formationParameters(3:6);
            obj.formationNo = formationNo;
            obj.formationSize = formationSize;
            obj.formationOrientation = formationParameters(3:6);
            obj.formationCenterOrbit = formationParameters(7:16);
            switch formationNo
                case 1
                    offset = -[0,0,formationSize*sqrt(2)/2];
                case 2
                    offset = -[formationSize*sqrt(2)/2,0,0];
                case 3
                    offset = -[0,formationSize*sqrt(2)/2,0];
                case 4
                offset = -[ -formationSize*sqrt(2)/2,0,0];
            case 5
                offset = -[0 -formationSize*sqrt(2)/2,0];
            case 6
                offset = -[0,0,-formationSize*sqrt(2)/2];
        end
        obj.centerOffset = quatrotate(formationOrientation,offset);

        a = obj.formationCenterOrbit(1);
        n = sqrt(Constants.muEarth/a^3);

        % Inclusion of J2 effect
        J2 = Constants.J2;
        r = a;
        inc = obj.formationCenterOrbit(3);
        s = (3*J2*Constants.graviPar.Re^2 / (8*r^2)) * (1 + 3*cos(2*inc));
        c = sqrt(1 + s);
        %Test without j2 compensation
        %c = 1;
        % Determine Continuous LQR gain
        A = [0              ,0,0               ,1      ,0     ,0;
             0              ,0,0               ,0      ,1     ,0;
             0              ,0,0               ,0      ,0     ,1;
            (5*c^2-2)*n^2,0,0               ,0      ,2*n*c,0;
             0              ,0,0               ,-2*n*c,0     ,0;
             0              ,0,-(3*c^2-2)*n^2,0      ,0     ,0];


        B = [0,0,0;
             0,0,0;
             0,0,0;
             1,0,0;
             0,1,0;
             0,0,1];

        contSys = ss(A,B,eye(6,6),0);
        Qcont = zeros(6);
        Qcont(1:3,1:3) = alpha1 * eye(3);
        Qcont(4:6,4:6) = alpha2 * eye(3);
        Qdisc = zeros(6);
        Qdisc(1:3,1:3) = alpha1 * eye(3);
        Qdisc(4:6,4:6) = alpha2 * eye(3);
        %{
        Q(1,1) = 0.5;
        Q(2,2) = 0.5;
        Q(3,3) = 0.5;

        Q(4,4) = 1;
        Q(5,5) = 1;
        Q(6,6) = 1;
        %}
        Rcont = eye(3,3);
        Rdisc = 1*eye(3,3);
        E = eye(6,6);
        [S,~,~,~] = icare(A,B,Qcont,Rcont,0,E,0);
        obj.Kcont = Rcont\transpose(B)*S;

        % Determine Discrete LQR gain using Zero Order Hold ZOH
        discSysZOH = c2d(contSys,obj.thrustInterval,'zoh');
        AdZOH = discSysZOH.A;
        BdZOH = discSysZOH.B;
```

```
            SdZOH = idare(AdZOH,BdZOH,Qdisc,Rdisc,0,E);
            obj.KZOH = (Rdisc+transpose(BdZOH)*SdZOH*BdZOH)\transpose(BdZOH)*SdZOH*AdZOH;%;-R\
                transpose(BdZOH)*SdZOH;
            % Determine Discrete LQR using Impulse method
            discSysImp = c2d(contSys,obj.thrustInterval,'impulse');
            AdImp = discSysImp.A;
            BdImp = discSysImp.B;
            SdImp = idare(AdImp,BdImp,Qdisc,Rdisc,0,E);
            obj.Kimp = (Rdisc+transpose(BdImp)*SdImp*BdImp)\transpose(BdImp)*SdImp*AdImp;%
                Rdisc\transpose(BdImp)*SdImp;%

            % Give Formation Satellite 1 a Kalman filter
            if formationNo == 1
                A4 = blkdiag(A,A,A,A);
                Aformation = blkdiag(A4,A4,A4);
                zerosB = zeros(size(B));
                Bformation = [-B     , B      , zerosB, zerosB, zerosB, zerosB;
                              -B     , zerosB, B      , zerosB, zerosB, zerosB;
                              -B     , zerosB, zerosB, B      , zerosB, zerosB;
                              -B     , zerosB, zerosB, zerosB, B      , zerosB;
                              zerosB,-B      , B      , zerosB, zerosB, zerosB;
                              zerosB, zerosB,-B      , B      , zerosB, zerosB;
                              zerosB, zerosB, zerosB, -B     , B      , zerosB;
                              zerosB, B      , zerosB, zerosB,-B      , zerosB;
                              zerosB, zerosB, zerosB, zerosB, B      ,-B;
                              zerosB, zerosB, zerosB, B      , zerosB,-B;
                              zerosB, zerosB, B      , zerosB, zerosB,-B;
                              zerosB, B      , zerosB, zerosB, zerosB,-B];
                Cformation =[eye(72);-eye(72)];
                G = eye(length(Aformation));
                H = [eye(length(Aformation));eye(length(Aformation))];
                kSys = ss(Aformation,[Bformation,G],Cformation,[[zeros(size(Bformation));zeros
                    (size(Bformation))],H]);
                dSys = c2d(kSys,Constants.dt,'zoh');
                Q = blkdiag(1e-12*eye(3),1e-12*eye(3));

                Q = blkdiag(Q,Q,Q,Q,...
                            Q,Q,Q,Q,....
                            Q,Q,Q,Q);

                R = zeros(6,6);
                R(1:3,1:3) = Constants.posRelNoise^(2)*eye(3);
                R(4:6,4:6) = Constants.velRelNoise^(2)*eye(3);


                R = blkdiag(R,R,R,R,...
                            R,R,R,R,...
                            R,R,R,R,...
                            R,R,R,R,...
                            R,R,R,R,...
                            R,R,R,R);
                P = idare(dSys.A',dSys.C',Q,R,0,eye(size(dSys.A)));
                Man = P*dSys.C'/(dSys.C*P*dSys.C'+R);

                obj.P = P;
                obj.Kk = Man;
                obj.A = dSys.A;
                obj.B = dSys.B;
                obj.Bf = Bformation;
                obj.C = dSys.C;
                obj.D = dSys.D;
                obj.Q = Q;
                obj.Qinv= inv(dSys.C*P*dSys.C'+R);
                obj.extQ = Q;
                obj.R = R;

                obj.gk = zeros([36,1]);
                obj.faultThreshold = 25;
                obj.detect = 0 ;
                obj.isolate = 0;
                obj.faultDetTime = 0;
                obj.faultDetected = 0;
                % The temp matrix is used in the calculation of the
                % residual-fault transfer function and only calculated here
                % to save computation time
                obj.tempMatrix = obj.C/(eye(size(obj.A)) - obj.A + obj.A*obj.Kk*obj.C);
            end
        end

        function updatePos(obj, stateChange)
            obj.position = obj.position + stateChange(1:3);
            obj.velocity = obj.velocity + stateChange(4:6);
        end

        function updateAtt(obj,attChange)
            obj.attitude = (obj.attitude + attChange(1:4))/(norm(obj.attitude + attChange(1:4)
                ));
            obj.spin = obj.spin + attChange(5:7);
        end

        function state = getState(obj)
            state = [obj.position,obj.velocity];
        end

        function setState(obj,state)
            obj.position = state(1:3);
            obj.velocity = state(4:6);
        end

        % Navigation function
        % access actual position, add noise and bias
        function [posEst,velEst] = selfNav(obj)
            posEst = obj.position + obj.posSelfBias + obj.posSelfNoise*randn([1,3]);
            velEst = obj.velocity + obj.velSelfBias + obj.velSelfNoise*randn([1,3]);
        end

        % Relative navigation function
        % given another spacecraft, subtract positions, add noise and bias
        function relEst = relNav(obj)
            relDist1 = obj.spacecraftNeighbor1.position - obj.position ...
                       + obj.posRelBias + obj.posRelNoise * randn([1,3]);
            relVel1  = obj.spacecraftNeighbor1.velocity - obj.velocity ...
                       + obj.velRelBias + obj.velRelNoise * randn([1,3]);
            relDist2 = obj.spacecraftNeighbor2.position - obj.position ...
                       + obj.posRelBias + obj.posRelNoise * randn([1,3]);
            relVel2  = obj.spacecraftNeighbor2.velocity - obj.velocity ...
                       + obj.velRelBias + obj.velRelNoise * randn([1,3]);
            relDist3 = obj.spacecraftNeighbor3.position - obj.position ...
                       + obj.posRelBias + obj.posRelNoise * randn([1,3]);
            relVel3  = obj.spacecraftNeighbor3.velocity - obj.velocity ...
                       + obj.velRelBias + obj.velRelNoise * randn([1,3]);
            relDist4 = obj.spacecraftNeighbor4.position - obj.position ...
```

```
                        + obj.posRelBias + obj.posRelNoise * randn([1,3]);
            relVel4  = obj.spacecraftNeighbor4.velocity - obj.velocity ...
                        + obj.velRelBias + obj.velRelNoise * randn([1,3]);
            relEst = [relDist1,relVel1;
                      relDist2,relVel2;
                      relDist3,relVel3;
                      relDist4,relVel4;];
        end

        function relEst = relNavDistAngle(obj)
            qAtt = obj.attitude;

            rVec1 =  quatrotate(qAtt',(obj.spacecraftNeighbor1.position - obj.position));
            rVec2 =  quatrotate(qAtt',(obj.spacecraftNeighbor2.position - obj.position));
            rVec3 =  quatrotate(qAtt',(obj.spacecraftNeighbor3.position - obj.position));
            rVec4 =  quatrotate(qAtt',(obj.spacecraftNeighbor4.position - obj.position));

            r = euclidnorm(rVec1) + obj.rangeBias(1) + obj.rangeNoise * randn(1);
            theta = acos(rVec1(3)/euclidnorm(rVec1)) + obj.angleBias(1,1) +obj.angleNoise *
                randn(1);
            psi = atan2(rVec1(2),rVec1(1)) + obj.angleBias(1,2) + obj.angleNoise * randn(1);
            relDist1 = quatrotate(quatinv(qAtt'), r*[sin(theta)*cos(psi),sin(theta)*sin(psi),
                cos(theta)]);

            relVel1  = obj.spacecraftNeighbor1.velocity - obj.velocity ...
                        + obj.velRelBias + obj.velRelNoise * randn([1,3]);

            r = euclidnorm(rVec2) + obj.rangeBias(2) + obj.rangeNoise * randn(1);
            theta = acos(rVec2(3)/euclidnorm(rVec2)) + obj.angleBias(2,1) + obj.angleNoise *
                randn(1);
            psi = atan2(rVec2(2),rVec2(1)) + obj.angleBias(2,2) + obj.angleNoise * randn(1);
            relDist2 = quatrotate(quatinv(qAtt'), r*[sin(theta)*cos(psi),sin(theta)*sin(psi),
                cos(theta)]);
            relVel2  = obj.spacecraftNeighbor2.velocity - obj.velocity ...
                        + obj.velRelBias + obj.velRelNoise * randn([1,3]);

            r = euclidnorm(rVec3) + obj.rangeBias(3) + obj.rangeNoise * randn(1);
            theta = acos(rVec3(3)/euclidnorm(rVec3)) + obj.angleBias(3,1) + obj.angleNoise *
                randn(1);
            psi = atan2(rVec3(2),rVec3(1)) + obj.angleBias(3,2) + obj.angleNoise * randn(1);
            relDist3 = quatrotate(quatinv(qAtt'), r*[sin(theta)*cos(psi),sin(theta)*sin(psi),
                cos(theta)]);
            relVel3  = obj.spacecraftNeighbor3.velocity - obj.velocity ...
                        + obj.velRelBias + obj.velRelNoise * randn([1,3]);

            r = euclidnorm(rVec4) + obj.rangeBias(4) + obj.rangeNoise * randn(1);
            theta = acos(rVec4(3)/euclidnorm(rVec4)) + obj.angleBias(4,1) + obj.angleNoise *
                randn(1);
            psi = atan2(rVec4(2),rVec4(1)) + obj.angleBias(4,2) + obj.angleNoise * randn(1);

            relDist4 = quatrotate(quatinv(qAtt'), r*[sin(theta)*cos(psi),sin(theta)*sin(psi),
                cos(theta)]);
            relVel4  = obj.spacecraftNeighbor4.velocity - obj.velocity ...
                        + obj.velRelBias + obj.velRelNoise * randn([1,3]);
            relEst = [relDist1,relVel1;
                      relDist2,relVel2;
                      relDist3,relVel3;
                      relDist4,relVel4;];

        end
        function navigation(obj,navType)
```

```
            obj.lastPositionEst = obj.positionEst;
            obj.lastVelocityEst = obj.velocityEst;
            obj.lastRelativeEst = obj.relativeEst;

            [pos,vel] = obj.selfNav();
            obj.positionEst = pos;
            obj.velocityEst = vel;
            if navType == 1
                obj.relativeEst = obj.relNav();
            elseif navType == 2
                obj.relativeEst = obj.relNavDistAngle();
            end

            [a,e,inc,O,~,~,~,argLat,~] = rv2orb(pos',vel');
            n = sqrt(Constants.muEarth/(a^3*(1-e^2)^3))*(1+e*cos(argLat))^2;

            x = pos/norm(pos);
            z = cross(x,vel/norm(vel));
            y = cross(z,x)/(norm(cross(z,x)));
            obj.THillECI = [x;
                            y;
                            z];
            S = n*skewSym([0,0,1]);
            obj.Tdot = -S*obj.THillECI;%n*rotZdot(argLat)*rotX(inc)*rotZ(O);

            % Convert to Hill frame

            relPosHill = obj.THillECI*obj.relativeEst(:,1:3)';

            %relVelHill = THillECI*obj.relativeEst(:,4:6)'-S*THillECI*obj.relativeEst(:,1:3)';
            relVelHill = obj.THillECI*obj.relativeEst(:,4:6)'+obj.Tdot*obj.relativeEst(:,1:3)
                ';
            posSelf = repmat(pos',[1,4]);
            velSelf = repmat(vel',[1,4]);
            posRel = posSelf+obj.relativeEst(:,1:3)';
            velRel = velSelf+obj.relativeEst(:,4:6)';
            obj.relEstHill = [relPosHill',relVelHill'];
        end
        % Kalman filter update
        function kalmanUpdate(obj,measurement,controlstates)
            % Project state into the present based on past state and control
            % input
            if ~ length(obj.kalmanEstimate) == 0
                estimatePriori = obj.A * obj.kalmanEstimate + obj.B(:,1:18) * obj.pastControl
                    ';%controlstates';
                % Propagate error covariance matrix
                obj.ErrCov = obj.A*obj.ErrCov*obj.A'+obj.Q;
                % Compute Kalman gain
                %K = obj.ErrCov*obj.C'/(obj.C*obj.ErrCov*obj.C'+obj.R);
                K = obj.Kk;
                % Update estimate using measurement
                obj.kalmanResidual =  (measurement'-(obj.C*estimatePriori +obj.D(:,1:18)*
                    controlstates'));
                obj.kalmanEstimate = estimatePriori + K * obj.kalmanResidual;
                % Update Error Covariance Matrix
                I = eye(size(obj.A));
                obj.ErrCov = (I - K * obj.C)*obj.ErrCov*(I -K * obj.C)' + K*obj.R*K';
                % Low Pass Filter
                obj.kalLowPass = obj.alphLow * obj.kalmanResidual' + (1-obj.alphLow) * obj.
                    kalLowPass;
                obj.pastControl = controlstates;
            else
                obj.kalmanEstimate = 0.5*(measurement(1,1:72)-measurement(1,73:end))';
                obj.kalmanResidual = zeros(size(measurement))';
                obj.kalLowPass = zeros(size(measurement));
                obj.pastControl = controlstates;
                obj.ErrCov = obj.Q;
            end
        end

        % Extended Kalman filter update DEPRECATED DONT USE
        function extendedKalmanUpdate(obj,time,measurement,controlstates)
            % Project state into future based on past state and control
            % input
            state = obj.lastRelativeEst;
            satellitePosECI = obj.lastPositionEst;
            satelliteVelECI = obj.lastVelocityEst;
            posCenterECI = satellitePosECI + mean(state(:,1:3),1);
            velCenterECI = satelliteVelECI + mean(state(:,4:6),1);
            [a,e,~,~,~,~,~,argLat,~] = rv2orb(posCenterECI',velCenterECI');

            if ~ length(obj.extKalmanEstimate) == 0
                % Propagate State with nonlinear function
                Alin = expm(dynJac(obj.extKalmanEstimate,posCenterECI,velCenterECI,a,e,argLat
                    )*Constants.dt);

                estimatePriori = Alin*obj.extKalmanEstimate + obj.B(1:72,1:18)*controlstates';
                % Linearly approximate function
                % Jacobian
                % Propagate error covariance matrix
                obj.extErrCov = Alin*obj.extErrCov*Alin'+obj.Q;
                % Compute Kalman gain
                K = obj.extErrCov*obj.C'/(obj.C*obj.extErrCov*obj.C'+obj.R);
                %K = obj.Kk;
                % Update estimate using measurement
                observation =  obj.C*estimatePriori;
                obj.extKalmanResidual = (measurement'- observation);
                obj.extKalmanEstimate = (estimatePriori +  K * obj.extKalmanResidual);
                % Update Error Covariance Matrix
                I = eye(size(Alin));
                obj.extErrCov = (I - K * obj.C)*obj.extErrCov*(I - K *obj.C)' + K*obj.R*K';
            else
                obj.extKalmanEstimate = 0.5*(measurement(1,1:72)-measurement(1,73:end))';
                obj.extKalmanResidual = zeros(size(measurement));
                obj.extErrCov = obj.Q;
            end
        end

        function fdi(obj,time)
            % Perform a recursive formulation of a GLR test for various
            % 6*n fault scenarios
            residual = obj.kalmanResidual';%-obj.kalLowPass;
            I = eye(72);
            TECIBody =myQuat2dcm(quatinv(obj.formationOrientation));
            TConf2Acc = 1/obj.mass*obj.THillECI*TECIBody*obj.Tconfig;
            TConfStack = blkdiag(TConf2Acc,TConf2Acc,TConf2Acc,...
                                 TConf2Acc,TConf2Acc,TConf2Acc);
            Ff = obj.B(:,1:18)*TConfStack;

            Du = obj.D(:,1:18)*TConfStack;
            K = obj.Kk;

            Mat = -obj.tempMatrix*(Ff-obj.A*K*Du)-Du;
            nf = 6*6;

            for ii = 1:nf
                fVec = zeros([nf,1]);
                fVec(ii) = 0.0005;
                mu = Mat*fVec;
                sz = (mu-obj.residualMean')'*obj.Qinv*(residual'-1/2*(mu+obj.residualMean'));

                obj.gk(ii) = max(0,obj.gk(ii) + sz);
            end
            if any(obj.gk>obj.faultThreshold)
                obj.detect = 1;
                [~,obj.isolate] = max(obj.gk);
                obj.isoCounter = obj.isoCounter + 1;
                if ~obj.faultDetected
                    obj.faultDetTime = time;
                    obj.faultDetected = true;
                end

            end
        end

        % Guidance Law
        % determine position to be in for current time/trajectory to follow
        function deltaV = guidance(obj,time)
            if mod(time,obj.thrustInterval) == 0
                debugGuidance = 1;
                if debugGuidance
                    a       = obj.formationCenterOrbit(1);
                    e       = obj.formationCenterOrbit(2);
                    inc     = obj.formationCenterOrbit(3);
                    O       = obj.formationCenterOrbit(4);
                    o       = obj.formationCenterOrbit(5);
                    nu      = obj.formationCenterOrbit(6);
                    truLon  = obj.formationCenterOrbit(7);
                    argLat  = obj.formationCenterOrbit(8);
                    lonPer  = obj.formationCenterOrbit(9);
                    p       = obj.formationCenterOrbit(10);
                    [pos,~] = keplerEQsolve(a,e,inc,O,o,nu,truLon,argLat,lonPer,p,time+obj.
                        thrustInterval);
                else
                    %Test propagation using virtual center
                    state = obj.relNav();
                    stateR = reshape(state,6,4);
                    center = obj.selfNav()' + mean(stateR(1:3,:),2);
                    inc = obj.formationCenterOrbit(3);
                    velMag = sqrt(Constants.muEarth/norm(center));
                    centerDir = center/norm(center);
                    nDir = [0,-sin(inc),cos(inc)];
                    velDir = cross(nDir,centerDir);
                    velCenter = velMag*velDir;
                    [a,e,inc,O,o,nu,truLon,argLat,lonPer,p] = rv2orb(center,velCenter');
                    [pos,~] = keplerEQsolve(a,e,inc,O,o,nu,truLon,argLat,lonPer,p,obj.
                        thrustInterval);
                end

                posNext = pos+obj.centerOffset;
                [v1B,v2B] = lambertBook(obj.selfNav(),posNext,obj.thrustInterval,'pro');
                deltaV = v1B-obj.velocity;
                obj.accumDV = obj.accumDV +norm(deltaV);
                obj.velocity = v1B;
            end

        end

        % Control Law
        % Determine control impulse based on estimated position and
        % position/trajectory
        function controlCommand(obj, time,controltype,disctype)
            % Check if we have a form of discrete control (disctype =|= 0)
            if disctype ~= 0
                % Check if we are outside a burn window
                if mod(time,obj.thrustInterval)>obj.burnTime
                    obj.cImpulse = [0;0;0];
                    obj.thrustAllocComplete = 0;
                    return
                elseif any(obj.cImpulse)% Check if we already were in this burn window
                    return
                end
            end

            switch controltype
                case 1
                    err =  obj.errTrackedCenter(time);
                case 2
                    err =  obj.errVirtualCenter(time);
            end

            % Select control gain according to discretization method
            % 0     =     Continuous
            % 1     =     Zero Order Hold (ZOH)
            % 2     =     Impulse (imp)
            switch disctype
                case 0
                    K = obj.Kcont;
                case 1
                    K = obj.KZOH;
                case 2
                    K = obj.Kimp;
            end

            cImpulse = -K*err';
            obj.cImpulseHill = cImpulse;

            % Transform control force to ECI frame
            state = obj.relativeEst;
            %stateR = reshape(state,6,4);
            satellitePosECI = obj.positionEst;
            velECI = obj.velocityEst;
            posCenterECI = satellitePosECI + mean(state(:,1:3),1);
            % Estimate center velocity
            velCenterECI = velECI + mean(state(:,4:6),1);
            % Transform error into Hill Frame

            % INCORRECT TRANSFORMATION, DEPRECATED only used for continutiy
            x = velCenterECI/norm(velCenterECI);
            z = posCenterECI/norm(posCenterECI);
            y = cross(z,x)/(norm(cross(z,x)));
            THillECI_Incorrect = [x;
                               y;
                               z];
            cImpulse = THillECI_Incorrect\cImpulse;
            %cImpulse = obj.THillECI\cImpulse;

            if disctype>0
                cImpulse = cImpulse*obj.thrustInterval*0.8;
            end
            obj.cImpulse = cImpulse;
            obj.accumDV = obj.accumDV + norm(cImpulse);
        end

        % Thruster allocation
        % Given control force vector, determine which thrusters should fire
        % for how long
        function thrustAlloc(obj,time)
            if any(obj.cImpulse) && ~obj.thrustAllocComplete
                obj.thrustAllocComplete = 1;
                impulseECI = obj.cImpulse;
                % Determine current attitude
                qAtt = obj.attitude();
                % Transform impulse into body frame
                impulseBody = quatrotate(qAtt',impulseECI');
                % Calculate opening times
                opening = 1/obj.thrust*obj.TconfigInv*obj.mass*impulseBody';
                % Negative opening times are added to other thruster
                if opening(1)<0
                    opening(2) = opening(2)-opening(1);
                    opening(1) = 0;
                end
                if opening(2)<0
                    opening(1) = opening(1)-opening(2);
                    opening(2) = 0;
                end
                if opening(3)<0
                    opening(4) = opening(4)-opening(3);
                    opening(3) = 0;
                end
                if opening(4)<0
                    opening(3) = opening(3)-opening(4);
                    opening(4) = 0;
                end
                if opening(5)<0
                    opening(6) = opening(6)-opening(5);
                    opening(5) = 0;
                end
                if opening(6)<0
                    opening(5) = opening(5)-opening(6);
                    opening(6) = 0;
                end
                obj.thrusterOpeningCount(opening>0) = obj.thrusterOpeningCount(opening>0) + 1;
                obj.thrusterOpeningTime = obj.thrusterOpeningTime + opening;
                obj.spentProp = obj.spentProp + sum(opening)*obj.thrust/(obj.isp*9.81);
                obj.opTimes = time+opening;
                obj.maxBurnTime = max(obj.maxBurnTime,max(opening));
            end
        end

        % Determine force based on opening times
        function cFECI = controlForce(obj,time)
            remainingBurntime = max(obj.opTimes-time,0);
            minOpTime = obj.minImpulse/obj.thrust;
            for ii = 1:6
```

```
                if remainingBurntime(ii) > 0
                    if remainingBurntime(ii) < minOpTime
                        remainingBurntime(ii) = minOpTime;
                    end
                end
            end
            thrusterOpening = remainingBurntime>0;
            burnFraction = min(abs(remainingBurntime)/Constants.dt,1);
            obj.burnFraction = burnFraction;
            if time > obj.faultTime
                % Determine
                thrusterForce = max(obj.thrust*obj.faultVectorClosed.*burnFraction.*thrusterOpening,...
                                obj.thrust*obj.faultVectorOpen);
                if any(1-obj.faultVectorClosed)
                    faultyThrusterActivation = find(1-obj.faultVectorClosed) == find(thrusterOpening);

                    if any(faultyThrusterActivation) && ~ obj.firstThrust
                        obj.faultTime = time;
                        obj.firstThrust = true;
                    end
                end
            else
                thrusterForce = obj.thrust*burnFraction.*thrusterOpening;
            end
            cFBody = obj.Tconfig*thrusterForce;
            cFECI = quatrotate(quatinv(obj.attitude'),cFBody');
            obj.cFECI = cFECI;
            % For Kalman filter purposes, get commanded force output in
            % Hill frame
            thrusterForceFaultless = obj.thrust*burnFraction.*thrusterOpening;
            cFfaultless = obj.Tconfig*thrusterForceFaultless;
            cFECIfaultless = quatrotate(quatinv(obj.attitude'),cFfaultless');
            obj.cAccHill = obj.THillECI * cFECIfaultless'/obj.mass;

            deltaMass = sum(thrusterForce)/(obj.isp*9.81);
            obj.mass = obj.mass-deltaMass;
        end
        %
        function err = errVirtualCenter(obj,time)

            state = obj.relativeEst;
            satellitePosECI = obj.positionEst;
            velECI = obj.velocityEst;
            posCenterECI = satellitePosECI + mean(state(:,1:3),1);
            % Estimate center velocity
            velCenterECI = velECI + mean(state(:,4:6),1);
            [a,~,inc,O,~,theta,~,argLat,~] = rv2orb(posCenterECI',velCenterECI');
            n = sqrt(Constants.muEarth/a^3);
            realCenterECI = satellitePosECI + obj.centerOffset;
            % Transform error into Hill Frame
            % NOTE: INCORRECT TRANSFORMATION, DEPRECATED, used only for
            % consistency with Data generation method. CORRECT
            % transformations are given by obj.THillECI and obj.Tdot
            x = velCenterECI/norm(velCenterECI);
            z = posCenterECI/norm(posCenterECI);
            y = cross(z,x)/(norm(cross(z,x)));
            THillECI_Incorrect = [x;
                            y;
                            z];
            Tdot_Incorrect = n*rotX(0.5*pi)*rotZdot(theta)*rotZ(0.5*pi)*rotX(inc)*rotZ(O);

            vHill = THillECI_Incorrect*(velECI-velCenterECI)'-Tdot_Incorrect*
                THillECI_Incorrect*(satellitePosECI-posCenterECI)';
            %vHill = obj.THillECI*(velECI-velCenterECI)'+obj.Tdot*(satellitePosECI-
                posCenterECI)';

            vRef = Tdot_Incorrect*THillECI_Incorrect*obj.centerOffset';
            %vRef = -obj.Tdot*obj.centerOffset';
            err(1:3) = THillECI_Incorrect*(realCenterECI - posCenterECI)';
            %err(1:3) = obj.THillECI*(realCenterECI - posCenterECI)';
            err(4:6) = vHill-vRef;

            obj.cErr = err;
        end

        function err= errTrackedCenter(obj,time)
            a       = obj.formationCenterOrbit(1);
            e       = obj.formationCenterOrbit(2);
            inc     = obj.formationCenterOrbit(3);
            O       = obj.formationCenterOrbit(4);
            o       = obj.formationCenterOrbit(5);
            nu      = obj.formationCenterOrbit(6);
            truLon  = obj.formationCenterOrbit(7);
            argLat  = obj.formationCenterOrbit(8);
            lonPer  = obj.formationCenterOrbit(9);
            p       = obj.formationCenterOrbit(10);
            [posCenterECI,velCenterECI,theta] = keplerEQsolve(a,e,inc,O,o,nu,truLon,argLat,
                lonPer,p,time);
            satellitePosECI = obj.positionEst;
            velECI = obj.velocityEst;
            n = sqrt(Constants.muEarth/a^3);
            %{
            state = obj.relNav();
            stateR = reshape(state,6,4);
            realCenterECI = obj.position + mean(stateR(1:3,:),2)';
            %}
            realCenterECI = satellitePosECI + obj.centerOffset;
            inc = obj.formationCenterOrbit(3);
            % Transform error into Hill Frame

            vHill = obj.THillECI*(velECI-velCenterECI)'-obj.Tdot*obj.THillECI*(satellitePosECI
                -posCenterECI)';
            vRef = obj.Tdot*obj.THillECI*obj.centerOffset';
            err(1:3) = obj.THillECI*(realCenterECI - posCenterECI)';
            err(4:6) = vHill-vRef;

        end
    end
end
```

Bibliography

[1] G. Di Mauro, M. Lawn, and R. Bevilacqua. Survey on Guidance Navigation and Control Requirements for Spacecraft Formation-Flying Missions. *Journal of Guidance, Control, and Dynamics*, 41(3):581–602, 2017. ISSN 0731-5090. doi: 10.2514/1.g002868.

[2] Michael Tillerson, Louis Breger, and Jonathan P How. Distributed coordination and control of formation flying spacecraft. In *Proc. Amer. Control Conf*, volume 2, pages 1740–1745, 2003.

[3] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. A fully trainable network with RNN-based pooling. *Neurocomputing*, 338:72–82, apr 2019. ISSN 18728286. doi: 10.1016/j.neucom.2019.02.004.

[4] Will Campbell (2020). Earth-sized sphere with topography. URL `https://www.mathworks.com/matlabcentral/fileexchange/27123-earth-sized-sphere-with-topography`. [Online; accessed 04.01.20].

[5] Christopher Riggio. What's the deal with accuracy, precision, recall and f1?, November 2019. URL `https://towardsdatascience.com/whats-the-deal-with-accuracy-precision-recall-and-f1-f5d8b4db1021`. [Online; accessed 05.07.20].

[6] Marco D'Errico. *Distributed space missions for earth system monitoring*, volume 31. Springer Science & Business Media, 2012.

[7] Inseok Hwang, Sungwan Kim, Youdan Kim, and Chze Eng Seah. A survey of fault detection, isolation, and reconfiguration methods. *IEEE transactions on control systems technology*, 18(3):636–653, may 2010. ISSN 1063-6536. doi: 10.1109/TCST.2009.2026285.

[8] Mogens Blanke, Michel Kinnaert, Jan Lunze, Marcel Staroswiecki, and J Schröder. *Diagnosis and fault-tolerant control*, volume 3. Springer, 2015.

[9] J.-M. Jazequel and B. Meyer. Design by contract: the lessons of Ariane. *Computer*, 30(1):129–130, 1997. ISSN 00189162. doi: 10.1109/2.562936.

[10] Rahat Iqbal, Tomasz Maniak, Faiyaz Doctor, and Charalampos Karyotis. Fault Detection and Isolation in Industrial Processes Using Deep Learning Approaches. *IEEE Transactions on Industrial Informatics*, pages 1–1, 2019. ISSN 1551-3203. doi: 10.1109/TII.2019.2902274.

[11] Azzeddine Bakdi and Abdelmalek Kouadri. An improved plant-wide fault detection scheme based on PCA and adaptive threshold for reliable process monitoring: Application on the new revised model of Tennessee Eastman process. *Journal of Chemometrics*, 32(5):e2978, may 2018. ISSN 08869383. doi: 10.1002/cem.2978.

[12] Parisa Yazdjerdi and Nader Meskin. Actuator fault detection and isolation of differential drive mobile robots using multiple model algorithm. In *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 0439–0443. IEEE, apr 2017. ISBN 978-1-5090-6465-6. doi: 10.1109/CoDIT.2017.8102631.

[13] Qiao-Ning Xu, Kok-Meng Lee, Hua Zhou, and Hua-Yong Yang. Model-Based Fault Detection and Isolation Scheme for a Rudder Servo System. *IEEE Transactions on Industrial Electronics*, 62(4):2384–2396, apr 2015. ISSN 0278-0046. doi: 10.1109/TIE.2014.2361795.

[14] Ron J. Patton, Faisal J. Uppal, Silvio Simani, and Bernard Polle. Robust FDI applied to thruster faults of a satellite system. *Control Engineering Practice*, 18(9):1093–1109, sep 2010. ISSN 0967-0661. doi: 10.1016/J.CONENGPRAC.2009.04.011.

[15] Daniel P Scharf, Fred Y Hadaegh, and Scott R Ploen. A survey of spacecraft formation flying guidance and control (part I): Guidance. 2003.

[16] S A Fuselier, W S Lewis, C Schiff, R Ergun, J L Burch, S M Petrinec, and K J Trattner. Magnetospheric multiscale science mission profile and operations. *Space Science Reviews*, 199(1-4):77–103, 2016.

[17] B. D. Tapley, S. Bettadpur, M. Watkins, and C. Reigber. The gravity recovery and climate experiment: Mission overview and early results. *Geophysical Research Letters*, 31(9):1–4, 2004. ISSN 00948276. doi: 10.1029/2004GL019920.

[18] Christine Edwards-Stewart. NASA's GRAIL Spacecraft Formation Flight, End of Mission Results, and Small-Satellite Applications. *Annual AIAA/USU Conference on Small Satellites*, pages SSC13–X–2, 2013.

[19] Markus Landgraf and Agnes Mestreau-Garreau. Formation flying and mission design for Proba-3. *Acta Astronautica*, 82(1):137–145, jan 2013. ISSN 0094-5765. doi: 10.1016/j.actaastro.2012.03.028.

[20] Lorenzo Tarabini Castellani, Jesús Salvador Llorente, José Mar\'\ia Fernández Ibarz, Mercedes Ruiz, Agnes Mestreau-Garreau, Alexander Cropp, and Andrea Santovincenzo. PROBA-3 mission. *International Journal of Space Science and Engineering*, 1(4):349–366, 2013.

[21] S. Ghasemi and K. Khorasani. Distributed fault detection and isolation in formation flight of satellites using extended kalman filters. In *International Conference on Advances in Engineering and Technology (ICAET)*, pages 541–548, 2014.

[22] S.M. Azizi and K. Khorasani. A distributed Kalman filter for actuator fault estimation of deep space formation flying satellites. In *2009 3rd Annual IEEE Systems Conference*, pages 354–359. IEEE, mar 2009. ISBN 978-1-4244-3462-6. doi: 10.1109/SYSTEMS.2009.4815826.

[23] Amitabh Barua and Khashayar Khorasani. Intelligent model-based hierarchical fault diagnosis for satellite formations. In *2007 IEEE International Conference on Systems, Man and Cybernetics*, pages 3191–3196. IEEE, oct 2007. ISBN 978-1-4244-0990-7. doi: 10.1109/ICSMC.2007.4413703.

[24] Arturo Valdes, K. Khorasani, and Liying Ma. Dynamic Neural Network-Based Fault Detection and Isolation for Thrusters in Formation Flying of Satellites. In *ISNN 2009: Advances in Neural Networks*, pages 780–793, 2009. doi: 10.1007/978-3-642-01513-7_85.

[25] Filippo Arrichiello, Alessandro Marino, and Francesco Pierri. Observer-Based Decentralized Fault Detection and Isolation Strategy for Networked Multirobot Systems. *IEEE Transactions on Control Systems Technology*, 23(4):1465–1476, jul 2015. ISSN 1063-6536. doi: 10.1109/TCST.2014.2377175.

[26] Amin Azarshab and Mehdi Shahbazian. Distributed fault detection in large-scale systems using hybrid extended information filter. *Chemical Engineering Research and Design*, 132:285–294, apr 2018. ISSN 0263-8762. doi: 10.1016/J.CHERD.2017.12.034.

[27] Jiantao Shi, Donghua Zhou, Yuhao Yang, Jun Sun, and Yi Chen. Distributed fault detection for formation of multi-agent systems. In *2017 Chinese Automation Congress (CAC)*, pages 4134–4139. IEEE, oct 2017. ISBN 978-1-5386-3524-7. doi: 10.1109/CAC.2017.8243505.

[28] Elizabeth Gibney. Google AI algorithm masters ancient game of Go, jan 2016. ISSN 14764687. URL `http://www.nature.com/news/google-ai-algorithm-masters-ancient-game-of-go-1.19234`.

[29] Sebastian Risi and Mike Preuss. Behind DeepMind's AlphaStar AI that Reached Grandmaster Level in StarCraft II: Interview with Tom Schaul, Google DeepMind, mar 2020. ISSN 16101987. URL `https://doi.org/10.1007/s13218-020-00642-1`.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[31] Hadi Shahnazari, Prashant Mhaskar, John M. House, and Timothy I. Salsbury. Modeling and fault diagnosis design for HVAC systems using recurrent neural networks. *Computers & Chemical Engineering*, 126:189–203, jul 2019. ISSN 00981354. doi: 10.1016/j.compchemeng.2019.04.011.

[32] P. Boi and A. Montisci. A Neural Based Approach and Probability Density Approximation for Fault Detection and Isolation in Nonlinear Systems. In *ANN 2011, AIAI 2011: Engineering Applications of Neural Networks*, pages 296–305, 2011. doi: 10.1007/978-3-642-23957-1_34.

[33] K. Khorasani and H. A. Talebi. A Neural Network-Based Multiplicative Actuator Fault Detection and Isolation of Nonlinear Systems. *IEEE Transactions on Control Systems Technology*, 21(3):842–851, may 2013. ISSN 1063-6536. doi: 10.1109/TCST.2012.2186634.

[34] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial Attacks on Neural Network Policies. In *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*. International Conference on Learning Representations, ICLR, feb 2017.

[35] Mark W Spong, Seth Hutchinson, Mathukumalli Vidyasagar, and Others. *Robot modeling and control*. 2006.

[36] Shiyu Zhao. Time derivative of rotation matrices: A tutorial. *arXiv preprint arXiv:1609.06088*, 2016.

[37] Daniel Chavez Clemente and Ella M. Atkins. Optimization of a Tetrahedral Satellite Formation. *Journal of Spacecraft and Rockets*, 42(4):699–710, jul 2005. ISSN 0022-4650. doi: 10.2514/1.9776.

[38] J. S. Llorente, A. Agenjo, C. Carrascosa, C. De Negueruela, A. Mestreau-Garreau, A. Cropp, and A. Santovincenzo. PROBA-3: Precise formation flying demonstration mission. *Acta Astronautica*, 82(1):38–46, 2013. ISSN 00945765. doi: 10.1016/j.actaastro.2012.05.029.

[39] Gerhard Krieger, Alberto Moreira, Hauke Fiedler, Irena Hajnsek, Marian Werner, Marwan Younis, and Manfred Zink. TanDEM-X : A Satellite Formation for High-Resolution SAR Interferometry. *IEEE Transactions on Geoscience and Remote Sensing*, 45(11):3317–3341, 2007. ISSN 0196-2892. doi: 10.1109/TGRS.2007.900693.

[40] Gerhard Krieger, Manfred Zink, Markus Bachmann, Benjamin Bräutigam, Daniel Schulze, Michele Martone, Paola Rizzoli, Ulrich Steinbrecher, John Walter Antony, Francesco De Zan, Others, John Walter Antony, Francesco De Zan, Irena Hajnsek, Kostas Papathanassiou, Florian Kugler, Marc Rodriguez Cassola, Marwan Younis, Stefan Baumgartner, Paco López-Dekker, Pau Prats, and Alberto Moreira. TanDEM-X: A radar interferometer with two formation-flying satellites. *Acta Astronautica*, 89:83–98, 2013. ISSN 00945765. doi: 10.1016/j.actaastro.2013.03.008.

[41] Staffan Persson, Per Bodin, Eberhard Gill, Jon Harr, and John Jörgensen. PRISMA–an autonomous formation flying mission. In *ESA Small Satellite Systems and Services Symposium (4S), Sardinia, Italy*, pages 25–29, 2006.

[42] Nathan G. Orr, Jesse K. Eyer, Benoit P. Larouche, and Robert E. Zee. Precision formation flight: The CanX-4 and CanX-5 dual nanosatellite mission. *European Space Agency, (Special Publication) ESA SP*, (660 SP), 2008. ISSN 03796566.

[43] Grant Bonin, Niels Roth, Scott Armitage, Josh Newman, Ben Risi, and Robert E Zee. CanX–4 and CanX–5 Precision Formation Flight: Mission Accomplished! 2015.

[44] Niels H Roth, Ben Risi, C Grant, and R Zee. Flight Results from the CanX-4 and CanX-5 Formation Flying Mission. In *ESA and CNES: The 4S Symposium*, 2016.

[45] Michael Kirschner, Oliver Montenbruck, and Srinivas Bettadpur. Flight dynamics aspects of the GRACE formation flying. In *2nd International Workshop on Satellite Constellations and Formation Flying*, pages 19–20, 2001.

[46] Greg Holt, Thomas Campbell, and E Glenn Lightsey. GPS, distributed communications, and thruster experiments on the University of Texas FASTRAC mission. In *AMSAT 22nd Annual Space Symposium*, 2004.

[47] Sebastián Muñoz, Richard W Hornbuckle, and E Glenn Lightsey. FASTRAC Early Flight Results. *Journal of Small Satellites*, 1(2):49–61, 2012.

[48] Craig Underwood, Sergio Pellegrino, Vaios J. Lappas, Christopher P. Bridges, and John Baker. Using CubeSat/micro-satellite technology to demonstrate the Autonomous Assembly of a Reconfigurable Space Telescope (AAReST). *Acta Astronautica*, 114:112–122, sep 2015. ISSN 0094-5765. doi: 10.1016/J.ACTAASTRO.2015.04.008.

[49] Christopher W.T. Roscoe, Jason J. Westphal, and Ehson Mosleh. Overview and GNC design of the CubeSat Proximity Operations Demonstration (CPOD) mission. *Acta Astronautica*, 153:410–421, dec 2018. ISSN 0094-5765. doi: 10.1016/J.ACTAASTRO.2018.03.033.

[50] N. Rando, A. Lyngvi, D. Lumb, P. Verhoeve, A. Parmar, A. Peacock, Philippe Gondoin, M. Bavdaz, and D. de Wilde. ESA study of XEUS, a potential follow-on to XMM-Newton. In Errico Armandillo, Josiane Costeraste, and Nikos Karafolas, editors, *International Conference on Space Optics — ICSO 2006*, volume 10567, page 45. SPIE, nov 2017. ISBN 9781510616233. doi: 10.1117/12.2308079.

[51] Sara Seager, Margaret Turnbull, William Sparks, Mark Thomson, Stuart B. Shaklan, Aki Roberge, Marc Kuchner, N. Jeremy Kasdin, Shawn Domagal-Goldman, Webster Cash, Keith Warfield, Doug Lisman, Dan Scharf, David Webb, Rachel Trabert, Stefan Martin, Eric Cady, and Cate Heneghan. The Exo-S probe class starshade mission. volume 9605, page 96050W. International Society for Optics and Photonics, sep 2015. doi: 10.1117/12.2190378.

[52] Jørgen Christensen-Dalsgaard, Kenneth G Carpenter, Carolus J Schrijver, Margarita Karovska, and the Si Team. The Stellar Imager (SI) – A Mission to Resolve Stellar Surfaces, Interiors, and Magnetic Activity. *Journal of Physics: Conference Series*, 271(1):012085, jan 2011. ISSN 1742-6596. doi: 10.1088/1742-6596/271/1/012085.

[53] Giri P. Subramanian, Rebecca Foust, Derek Chen, Stanley Chan, Younes Taleb, Dayne L. Rogers, Jobin Kokkat, Saptarshi Bandyopadhyay, Daniel Morgan, Soon-Jo Chung, and Fred Hadaegh. Information-Driven Systems Engineering Study of a Formation Flying Demonstration Mission using Six CubeSats. In *53rd AIAA Aerospace Sciences Meeting*, Reston, Virginia, jan 2015. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-343-8. doi: 10.2514/6.2015-2043.

[54] G. W. Hill. Researches in the Lunar Theory. *American Journal of Mathematics*, 1(1):5, 1878. ISSN 00029327. doi: 10.2307/2369430.

[55] W. H. Clohessy. Terminal Guidance System for Satellite Rendezvous. *Journal of the Aerospace Sciences*, 27(9):653–658, sep 1960. ISSN 1936-9999. doi: 10.2514/8.8704.

[56] David A Vallado. *Fundamentals of astrodynamics and applications*, volume 12. Springer Science & Business Media, 2001.

[57] Samuel A. Schweighart and Raymond J. Sedwick. High-fidelity linearized J2 model for satellite formation flight. *Journal of Guidance, Control, and Dynamics*, 25(6):1073–1080, may 2002. ISSN 15333884. doi: 10.2514/2.4986.

[58] Y. Kozai, Kozai, and Y. New Determination of Zonal Harmonics Coefficients of the Earth's Gravitational Potential. Technical report, 1964. URL `https://ui.adsabs.harvard.edu/abs/1964SAOSR.165.....K/abstract`.

[59] Daniel P Scharf, Fred Y Hadaegh, and Scott R Ploen. A survey of spacecraft formation flying guidance and control. part ii: control. In *Proceedings of the 2004 American control conference*, volume 4, pages 2976–2985. IEEE, 2004.

[60] Guo Ping Liu and Shijie Zhang. A Survey on Formation Control of Small Satellites. In *Proceedings of the IEEE*, volume 106, pages 440–457. Institute of Electrical and Electronics Engineers Inc., mar 2018. doi: 10.1109/JPROC.2018.2794879.

[61] Elbert Hendricks, Ole Jannerup, and Paul Haase Sørensen. *Linear systems control: deterministic and stochastic methods*. Springer, 2008.

[62] Kyle Alfriend, Srinivas Rao Vadali, Pini Gurfil, Jonathan How, and Louis Breger. *Spacecraft formation flying: Dynamics, control and navigation*, volume 2. Elsevier, 2009.

[63] Frank L Lewis, Draguna Vrabie, and Vassilis L Syrmos. *Optimal control*. John Wiley & Sons, 2012.

[64] Neha Yadav, Anupam Yadav, and Manoj Kumar. *An Introduction to Neural Network Methods for Differential Equations*. SpringerBriefs in Applied Sciences and Technology. Springer Netherlands, Dordrecht, 2015. ISBN 978-94-017-9815-0. doi: 10.1007/978-94-017-9816-7.

[65] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:, 2015.

[66] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, apr 2018. ISSN 0278-3649. doi: 10.1177/0278364917710318.

[67] Pavol Bezak, Pavol Bozek, and Yuri Nikitin. Advanced Robotic Grasping System Using Deep Learning. *Procedia Engineering*, 96:10–20, jan 2014. ISSN 1877-7058. doi: 10.1016/J.PROENG.2014.12.092.

[68] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In *Neural Networks: Tricks of the Trade*, pages 9–48. Springer, Berlin, Heidelberg, 2012. doi: 10.1007/978-3-642-35289-8_3.

[69] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, nov 1997. ISSN 08997667. doi: 10.1162/neco.1997.9.8.1735.

[70] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, 6(2):107–116, nov 1998. ISSN 02184885. doi: 10.1142/S0218488598000094.

[71] Charles K Chui, Guanrong Chen, and Others. *Kalman filtering*. Springer, 2017.

[72] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[73] C R Tooley, R K Black, B P Robertson, J M Stone, S E Pope, and G T Davis. The magnetospheric multiscale constellation. *Space Science Reviews*, 199(1-4):23–76, 2016.

[74] arianegroup. Hydrazine thruster brochure. Brochure. URL `https://www.space-propulsion.com/brochures/hydrazine-thrusters/hydrazine-thrusters.pdf`. [Online; accessed 15.01.20].

[75] Jose Guzman and Conrad Schiff. A Preliminary Study for a Tetrahedron Formation: Quality Factors and Visualization. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Reston, Virigina, aug 2002. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-124-3. doi: 10.2514/6.2002-4637.

[76] Karel F Wakker. Fundamentals of astrodynamics. 2015.

[77] Norman W Peddie. International geomagnetic reference field. *Journal of geomagnetism and geoelectricity*, 34(6):309–326, 1982.

[78] Nikolaos K Pavlis, Simon A Holmes, Steve C Kenyon, and John K Factor. The development and evaluation of the Earth Gravitational Model 2008 (EGM2008). *Journal of geophysical research: solid earth*, 117 (B4), 2012.

[79] Howard D Curtis. *Orbital mechanics for engineering students*. Butterworth-Heinemann, 2013.

[80] David R. Williams. Earth fact sheet. URL `https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html`. [Online; accessed 05.12.19].

[81] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, may 2018. ISSN 00313203. doi: 10.1016/j.patcog.2017.10.013.

[82] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale Video Classification with Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun 2014.

[83] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, pages 1097–1105, 2012.

[84] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-Aware neural language models. In *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pages 2741–2749. AAAI press, 2016. ISBN 9781577357605.

[85] Jerome Connor and Les Atlas. Recurrent neural networks and time series prediction. In *Proceedings. IJCNN-91-Seattle: International Joint Conference on Neural Networks*, pages 301–306. Publ by IEEE, 1991. ISBN 0780301641. doi: 10.1109/ijcnn.1991.155194.

[86] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[87] Yunpeng Pan and Jun Wang. Model predictive control of unknown nonlinear dynamical systems based on recurrent neural networks. *IEEE Transactions on Industrial Electronics*, 59(8):3089–3101, aug 2012. ISSN 02780046. doi: 10.1109/TIE.2011.2169636.

[88] Heidar A Talebi, Farzaneh Abdollahi, Rajni V Patel, and Khashayar Khorasani. *Neural Network-Based State Estimation of Nonlinear Systems*. Springer, 2010.

[89] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning. may 2015.

[90] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International conference on machine learning*, pages 2342–2350, 2015.

[91] Monica Bianchini and Franco Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE Transactions on Neural Networks and Learning Systems*, 25(8):1553–1565, 2014. ISSN 21622388. doi: 10.1109/TNNLS.2013.2293637.

[92] Sander Voss. Application of Deep Learning for Spacecraft Fault Detection and Isolation. 2019.

[93] Renata Furtuna, Silvia Curteanu, and Florin Leon. Multi-objective optimization of a stacked neural network using an evolutionary hyper-heuristic. *Applied Soft Computing Journal*, 12(1):133–144, jan 2012. ISSN 15684946. doi: 10.1016/j.asoc.2011.09.001.

[94] Ekaba Bisong. *Google Colaboratory*, pages 59–64. Apress, Berkeley, CA, 2019. ISBN 978-1-4842-4470-8. doi: 10.1007/978-1-4842-4470-8_7.

[95] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. Robot motor skill coordination with EM-based reinforcement learning. In *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pages 3232–3237, 2010. ISBN 9781424466757. doi: 10.1109/IROS.2010.5649089.

[96] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, dec 2015.

[97] Rolf Isermann. Model-based fault-detection and diagnosis – status and applications. *Annual Reviews in Control*, 29(1):71–85, jan 2005. ISSN 1367-5788. doi: 10.1016/J.ARCONTROL.2004.12.002.

[98] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, dec 2014.

[99] Shinwon Jung, Sang-Young Park, Han-Earl Park, Chan-Deok Park, Seung-Woo Kim, and Yoon-Soo Jang. Real-time determination of relative position between satellites using laser ranging. *Journal of Astronomy and Space Sciences*, 29(4):351–362, 2012.

[100] Jongwoo Lee, Sang Young Park, and Dae Eun Kang. Relative navigation with intermittent laser-based measurement for spacecraft formation flying. *Journal of Astronomy and Space Sciences*, 35(3):163–173, 2018.