Politecnico di Milano

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE
Corso di Laurea Magistrale in Space Engineering

Tesi di Laurea Magistrale

A Model-Based Thruster Fault Detection and Isolation Tool with Low Level Implementation in Basilisk Astrodynamics Simulation Framework

Relatore
Prof. James Douglas Biggs

Correlatore
Prof. Hanspeter Schaub

Candidato
Giulio Napolitano
Matr.878748

Anno Accademico 2019–2020

Acknowledgements

Un grande capitolo della mia vita si conclude qui. Tante cose sono successe durante questo periodo universitario, nel bene e nel male. Purtroppo te ne sei andato troppo presto per vedermi arrivare fin qui, papà, però ne hai visto almeno la parte iniziale. Hai visto lanciarmi in questa avventura e, come sempre, mi hai supportato senza battere ciglio. Grazie a te, mamma, sei la donna più forte che abbia mai conosciuto e non sai quanto mi sento fortunato che tu sia ancora qui con me a gioire, soffrire, pianificare, aiutare, sopportare. Grazie a nonna Grazia, zia Annamaria, zio Luigi, Dade, zia Antonella, zio Antonio, Francesco e Federico. Devo ringraziare voi tutti se sono arrivato fin qui, non avrei potuto desiderare una famiglia migliore di questa. E devo ringraziare anche voi, nonno Giulio e zia Jolanda, che purtroppo non avete potuto vedermi ora, ma mi avete accompagnato per tutta la vita. Grazie anche a zio Giovanni, zia Agata, Marzia e Malo, per essermi stati vicini nei momenti più difficili, ed averli condivisi con me. Grazie a tutti gli altri membri della famiglia, putroppo non posso nominarvi tutti qui, per il supporto che mi date.
I wanna thank Professor Biggs for supporting me during this work, and for giving me the chance to go to the USA where most of the research for the thesis has been done. I wanna thank Professor Schaub for helping me and letting me be part of his AVS lab for six months, where I learned so many skills that are and will be immeasurably useful for my career. I wanna thank the people from the lab for helping me and being so friendly with me from literally day one. From my period in Boulder I also wanna thank the people from Horizons, you guys made me feel so loved during those months. I know we're gonna see each others again.
Against all the odds I was also able to find you there, Vanessa, my love. Thank you for supporting and caring for me, you're an awesome person on top of being an awesome girlfriend. In Boulder we started a journey together and I can't wait to see what's coming for us. Gracias también a Lula, Omar, Brigitte, Christian, Sebastian y toda la familia de Vanessa, por aceptarme desde el principio y alojarme en Houston. Todos ustedes son gente maravillosa
Poi ci siete voi uagliù cioè mi avete letteralmente seguito tutti a Milano. Grazie Besio, Stefano, Ale, è stato come avere parte della famiglia con me anche lontano da casa. Grazie ai ragazzi delle ZdP (siete oggettivamente troppi da menzionare, ma se riconoscete l'acronimo ZdP allora ce l'ho con voi) che sono rimasti giù, e agli altri che piano piano stanno salendo su. Facciamo l'Arbostella a Milano. Grazie anche a tutti gli amici di Salerno Sara, Miriam, Guido, Antonella che mi avete letteralmente visto crescere e sostenuto da sempre. Grazie te Tina, per essere ed essere stata sempre li ad ascoltarmi, consigliarmi e volermi bene.

Grazie Nicola per gli anni passati insieme, abbiamo creato una sensazione di quotidianità che non pensavo potesse avvenire all'infuori di Salerno, facendomi sentire meno fuori sede. Grazie anche ad Anna per tutto il tempo assieme quando ci venivi a trovare e Monica per avermi supportato ed essere stata da subito affettuosa con me.
Grazie ai ragazzi di PdF Giovanni, Yle, Marti, Valeria, Alessia, Fede, Ezia, Irene, per tutte le uscite ed il tempo speso insieme lontani da casa. Grazie agli amici di CG Ale, Eli, Ste per aver condiviso tutte le belle (ed alcune stupide) memorie accumulate durante questo percorso. Grazie a Ila, Giulia, Alice, Vale, Alo, Renato, Davide, Edo, Andrea e tutti gli altri del gruppo Space. Essere con voi a Milano ha reso tutto più facile e bello. Spero, anzi sono sicuro che con voi non sia finita, oramai non siete più amici del poli, siete amici e basta.

Milano, Giugno 2020 G. N.

Per te, papà

Contents

List of Figures

Abstract

The problem of Fault Detection and Isolation (FDI), is of much concern among the control engineering community. In addition, the space industry is pushing towards more compact spacecraft solutions (e.g. Cubesats) that have the advantage of being relatively low cost and easily configurable, posing however new constraints in terms of mass and power available onboard. The FDI problem is directly influenced by these new trends, that limit the amount of sensors carriable on board, forcing the community to come up with innovative ways to guarantee the fulfillment of those new requirements. With this in mind, the thesis carries out the development of a tool for detecting and isolating specific kinds of faults, that can occur in thruster-controlled space missions' maneuvers, making use of only accelerometer and gyroscope information. This kind of approach is particularly suitable for smaller spacecraft which, having more stringent requirements in terms of mass and power consumption, can benefit from the minimal number of sensors needed. In addition, this tool can be used soon after the spacecraft is released by the carrier, when not all the sensors are active, and usually information comes from simple sensor like an IMU unit. The method consists in three-stages (one for fault detection, two for fault identification), and combines simple sensor information monitoring with model-based approaches, leading to the design of multiple Nonlinear Unknown Input Observers. The implementation is carried out using Basilisk, the astrodynamics simulation software developed by the University of Colorado AVS Lab and the Laboratory for Atmospheric and Space Physics (LASP). It makes use of Python's ease of scripting and reconfiguration, together with the execution speed of low level programmed C/C++ modules, making the developed algorithms faster with respect to classical MATLAB simulations, and directly usable on spacecraft's onboard computers.

Sommario

Il problema di Fault Detection and Identification (FDI) è di grande interesse nel campo di ingegneria del controllo. In più, l'industria spaziale si sta indirizzando verso l'utilizzo di satelliti sempre più compatti (come ad esempio i Cubesat). Essi hanno il vantaggio di essere relativamente economici e facilmente configurabili, creando tuttavia nuovi vincoli in termini di massa e consumo disponibile a bordo. La FDI è direttamente influenzata da questi nuovi trends, che limitano la quantità di sensori trasportabili a bordo, forzando quindi gli ingegneri a trovare nuove innovative soluzioni che garantiscano l'adempimento a questi nuovi vincoli. Tenendo conto di tutto ciò, la tesi propone il design di un nuovo metodo per captare ed isolare specifici tipi di malfunzionamenti che possono avvenire durante manovre nello spazio che utilizzano propulsori di tipo "Cold Gas Thrusters", facendo uso esclusivamente di misurazioni di accelerometri e giroscopi. Questo tipo di approccio è particolarmente utile per piccoli veicoli che, avendo vincoli di massa e consumo elettrico piu stringenti, beneficiano della possibilità di utilizzare solo questi due sensori. Ancora, questo strumento può essere utilizzato appena dopo il rilasio del satellite da parte del vettore, quando non tutti i sensori sono ancora attivi, e le misurazioni arrivano solo da sensori quali IMU. Il metodo consiste in tre stadi (uno per fault detection e due per fault identification), e combina il semplice monitoramento dell'accelerazione con tecniche "Model-Based", portando al design di Nonlinear Unknown Input Observers. L'implementazione è portata a termine in Basilisk, il software di simulazione astrodinamica creato nell'Università del Colorado Boulder, dall'AVS Laboratory ed il Laboratory for Atmospheric and Space Physics (LASP). Questo software sfrutta la semplicità di riconfigurazione e scripting di Python, insieme con la velocità di esecuzione di moduli codificati nei linguaggi di basso livello C e C++. Ciò rende gli algoritmi qui presenati di più veloce esecuzione rispetto alle classiche simulazioni Matlab, e direttamente utilizzabili nei computer di bordo di veri satelliti.

Chapter 1

Introduction

In this chapter, the Fault Detection and isolation problem is laid down and the reason for its existence is explained. A literature review is carried out and, given the size of the argument, only the main methods used in the space sector are reported. The second section describes how the present work stands with respect to the literature, explaining its strengths and the contribution to the FDI subject.

1.1 Fault Detection and Isolation Problem

During the design phase of engineering projects, two conditions are taken into account: *nominal behavior* and *off-nominal behavior*. The nominal behavior is the desired one and is expected (or better, advised) to be exhibited by the system at all times. However, the real world application poses an unavoidable threat, by introducing a theoretically infinite amount of variables in the system, in contrast with the relatively few considered in the design. The relative weight of those new variables with respect to the design ones can largely vary, but in general they do have an effect on the system. From that comes the highly threatening off-nominal behavior. It is impossible (and often useless) to take into account every possible "secondary" variable so, usually, a survey of the most important ones is carried out beforehand. This means that the designers should have a thorough knowledge of the operating conditions *and* of the possible events that can perturb them. In some (but not all) applications where humans are an active part of the system, this pre-survey can just serve as a manual for the operators, that will be able to tell when and where an off-nominal behavior took place (and eventually correct it). However, in a large number of cases, humans cannot be present, and the need for autonomous actions becomes mandatory.
On this premise, the Fault Detection and Isolation (FDI) problem is introduced, with its purpose of answering:

- **Fault detection:** Is there a fault in the system?
- **Fault isolation:** Which component is faulty?

Those questions are tackled by the majority of industrial engineering fields, from aerospace to automotive, naval, chemical, etc. The common situation is a system composed of multiple sensors and actuators, that might become faulty during the

mission's lifespan. An immediate solution is the introduction of "per component" monitoring, causing redundancy, at the expense of a weight and complexity increase, which might not be that stringent of a trade-off in terrestrial applications. To avoid that, however, multiple FDI techniques are investigated with the purpose of offering the so called *"Analytical Redundancy"*.
It is clear that the lower is the capability to locally act/monitor the system, the more stress should be put on FDI. In the case of space applications this becomes very clear, given the impossibility to either "see" or "touch" a spacecraft. Everything is automated hence also the FDI has to act autonomously. Besides, the space industry is pushing towards more compact spacecraft solutions (e.g. Cubesats) that have the advantage of being relatively low cost and easily configurable, posing however new constraints in terms of mass and power available onboard. The FDI problem is directly influenced by these new trends, that limit the number and kinds of available sensors, forcing the engineering community to come up with innovative ways to guarantee the fulfillment of the FDI requirements.

1.2 Literature Review

The literature of FDI is vast, given the large number of ways to tackle this problem. In [34], [15] and [17] a survey of methods and philosophies is carried out, and is summarized in figure 1.1. Two major families can be identified: *data-based* and *model-based*; the former relying on pattern recognition, the latter on mathematical/physical models. Among those, further distinctions are to be made, but this goes beyond the purpose of this thesis, in which an overview of the most important characteristics is enough to give an idea of where the presented method stands. As expected, the focus will be on space-related applications, and in particular on **actuator faults**.

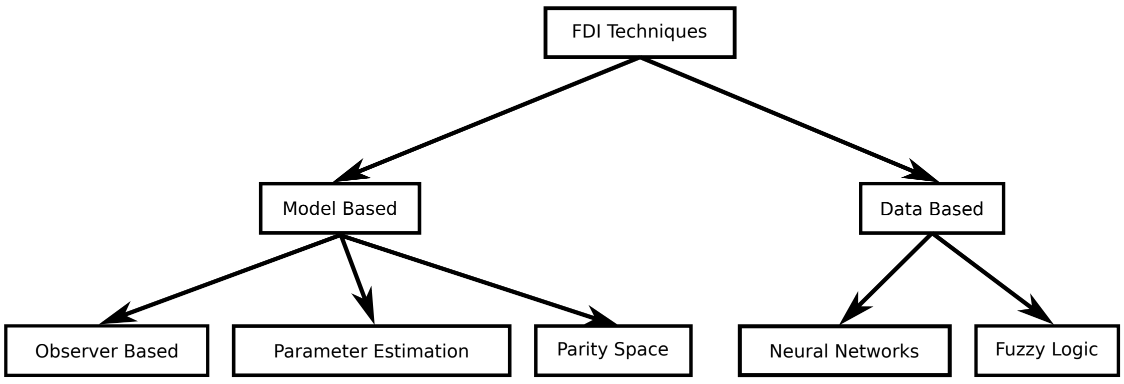

Figure 1.1: Classification of FDI techniques

1.2.1 Data Based

The latest developments are being made in the data-based field, showing how "model–free" (i.e. no need for mathematical modeling) approaches can be especially

advantageous in cases of FDI applied to complex systems. Neural-network (NN) based methods [31], [33] belong to the aforementioned group, and have received much attention because of their outstanding performance in learning nonlinear behaviors and their ability for pattern recognition and association [21]. Neural networks consist of processing elements called neurons, that get activated when the input exceeds a threshold value. Each neuron is connected with others by connection link. Each link has weights that carry the information about the input signal. A Neural network has to be trained for detecting and isolating the faults, and the training can be either offline or online.
Another similar technique employs the Fuzzy Logic[22]. Here a subset of the system in nominal conditions is used for off-line training, in order to create the base for the consequent fuzzy-based if-then conditions. Those are particularly useful in cases when humans' knowledge of the "faulty behavior" is of help, since it can be used in the training, giving a de facto mathematical model for human experience.

1.2.2 Model Based

The most classical approach is the model-based one, which has been thoroughly analyzed and employed in space missions for a long time. Among this group, the most used methods are:

- **Parameter estimation**
- **Parity space**
- **Observer based**

The general working principle of those methods relies on the creation and analysis of a signal, based on the system's measured input-output. The detection methods generate residuals (Observers, Parity equations) or parameter estimates. By comparison with the nominal values, deviations are detected and fault is declared and (when containing enough information) characterized for isolation. High fidelity models and possible noise reduction techniques are the keys to an effective FDI model-based tool, and for cases in which the investigated phenomenon's governing equations are simple enough, those requirements can be easily met, resulting in a fast and easy implementation.

Parameter Estimation

When the process parameters are partially not known or not known at all, *Parameter estimation* methods come to help. Such technique is used for example in [18] for the satellite's orbit and attitude FDIR (the R stands for recovery from the fault). System anomalies caused by faults and/or malfunctions result in changes in the physical parameters in the system, which will be estimated using the least-squares algorithm. Once a deviation (larger than a threshold) in those parameters is detected, the occurrence of a fault is declared and the subsequent analysis of the deviation allows for identification.

Parity Equations

A review of *Parity space* approaches is done in [23]. The underlining idea there is to generate auxiliary signals (residuals or parity vectors) that are independent of system operating conditions and system inputs under nominal operating conditions while carrying fault "information". This property is considered of fundamental importance in FDI, cause it allows for a relatively low probability of "false positives" and a simple (fixed) threshold check for fault declaring. The method compares the process behavior with a process model describing nominal behavior. The key idea is to check the parity (consistency), of the mathematical equations of the system by using the actual measurements. The difference of signals between the process and model is expressed by residuals. Following these guidelines, the residual signals are nominally near zero, and deviate from zero in characteristic ways when particular faults occur. Then they are examined for the likelihood of faults using proper decision functions and decision rules. A decision process may consist of a simple threshold test on the instantaneous values or moving averages of residuals, or it may be used directly on methods of statistical decision theory, e.g., sequential probability ratio testing.

Observer Based

Observer based methods are the last of the aforementioned, and are historically among the first studied for space FDI applications. The basic idea behind the observer or filter-based techniques is to estimate the outputs of the system from the measurements by using either Luenberger observers in a deterministic setting or Kalman filters in a noisy environment [28]. The output estimation error (or its weighted value) is therefore used as residual. When an observer is exploited for FDI purposes, the estimation of the outputs is necessary, whilst the estimation of the state vector is not necessarily needed (depending on the chosen approach). An advantage of using observers is the flexibility in the selection of its gains which leads to a rich variety of FDI schemes, each one with specific applications and some overlapping capabilities.
In [25] an Extended Kalman Filter is used to estimate the torque bias acting on the satellite due to faults in the thruster system using the data from a gyro. The filter innovation is then monitored using a generalized likelihood ratio (GLR) jump detection algorithm. This approach results to be very simple and convenient in terms of computational cost.
In [8] the identification of time-varying thruster faults is done by an iterative learning observer (ILO), designed to achieve estimation of time-varying faults. The proposed ILO-based fault-identification strategy uses a learning mechanism (i.e. previous information is employed to update the fault estimates law) to perform fault estimation instead of using integrators.
The authors of [24] present a practical solution to the problem of robust fault detection and isolation (FDI) for faults affecting the thrusters of Mars Express (MEX). The approach taken is based on both state estimation of an accurate linear model for the satellite system and unknown input de-coupling to achieve robust

FDI in the presence of severe dynamic uncertainty during main engine deployment. In [12] and [13] a multiple observer based scheme is proposed jointly with an online constrained allocation algorithm to detect, isolate and accommodate a single thruster fault affecting the propulsion system of an autonomous spacecraft during a rendez-vous maneuver. The fault detection is achieved through a EKF-based torque bias direction estimator which exploits information coming from the LIDAR relative position sensor. The approach to fault isolation consists of a bank of NUIOs able to isolate the fault to specific groups of thrusters, and ends with a torque bias evaluation of the EKF.

1.3 Presented Solution

A way of performing Fault Detection and Identification is presented in this thesis, which takes its motivation directly from the latest space trends, in terms of size, cost and complexity reduction. The target is a thruster propelled micro-spacecraft, equipped with eight ACS thrusters, capable of producing pure torque. Single thruster faults can be detected, and the two canonical kinds of malfunctions are contemplated: stuck on/off and efficiency loss. The example scenario taken into consideration is that of a de-tumbling maneuver in deep space. The FDI is performed in three stages:

- **Fault Detection (FD)** Fault detection is achieved by monitoring the space-craft's acceleration, and confronting it with a threshold. This triggering threshold is a design parameter.

- **First Fault Isolation (FFI)** As soon as the fault is detected, four NUIOs are activated, each one being sensitive to a fault in one of two opposing firing thrusters. The isolation is now restricted to two suspects.

- **Second Fault Isolation (SFI)** The two isolated thrusters will produce a net force/acceleration on the spacecraft, and by confronting it with their nominal force direction, the faulty one is isolated.

The presented tool is going to used in a specific thruster configuration. This however doesn't limit its applicability to other missions, given that the following conditions are met:

- The thruster configuration is such that the resulting design matrices for the NUIOs respect some characteristics.

- The thruster configuration and the employed control law produce only pure torques in nominal conditions, and the fault can be restricted to suspects with distinct force directions.

Those are going to be explained and understood later in the thesis.

1.3.1 Comparison with Other Methods

The presented method falls into the category of model-based, from which inherits the populated literature and well tested behavior. This nature is due to the usage of techniques such as Signal Monitoring (i.e. putting a limit to the difference between system states to their nominal values, the exceeding of which can indicate a fault situation) but most importantly Nonlinear Unknown Input Observers.

Comparison with Data Based

One of the main advantages of employing data-based is their non-reliance on the mathematical modeling of the system they are considering. This comes in handy with complex systems, with difficult or impossible accurate modeling, and that can present a whole variety of behaviors and responses. This is not the case for the target application, which is governed by simple known equations (attitude dynamics equations for angular velocity), and presents net distinctive and predictable behaviors in case of the considered faults.
Data-based techniques performances are heavily dependent on the database (it must be as extensive and detailed as possible) and/or training process, that would be unnecessary for the "simple" application considered here. Besides, even the most thorough training process cannot completely cancel out unexpected behaviors that especially in the space sector, cannot be tolerated. Or at least not if there are more easy and secure solutions.

Comparison with Other Model Based

Cause of the reasons explained above, the choice has fallen on a model based technique. In *parity based* residual generation, the residuals are formed directly without depending on state estimation. This method cannot be applied to non-linear systems since it is suitable for only linear system [30]. This excludes them for the considered application, that presents non-linearity in its attitude dynamics equations' representation of the angular velocity. Together with the acceleration, this last quantity is clearly affected in case of faults, being also the state of the system of (three) equations. The ability to easily describe and estimate the spacecraft's response in terms of those quantities, makes the choice go towards state estimation techniques, given that those are the parameters that change the most in faulty situations. This is why a *parameter estimation* approach like in [18] (that was applied to a far wider attitude and orbit control system) is not taken into account. The approach used by [25] is of easy and effective implementation, making use of only gyroscope information, and is used as well for fault detection and isolation in a thruster-based ACS. This makes it a good candidate, since it exploits the simple geometry of the problem and its response to faults. However, it is not possible to distinguish between actuators with identical torque vector direction when these are activated collectively (like in the considered application), and a subsequent manual testing strategy is foreseen. This can result to be intrusive, since it requires the active modification of the spacecraft's control law/subsystem.
The work of [8] answers to the problem of fault detection and estimation in thruster

controlled spacecraft, being able to accurately detect and estimate a time varying fault. However it does not address the problem of isolation, of vital importance to a possible decision of shutting off the faulty thruster.
To address this problem, the concept of *bank of observers* is introduced. The idea is to design more than one observer, each one sensitive to faults of specific thrusters. In this direction, Patton et al. [24] solution for the Mars Express missions' thruster system exploits a bank of UIOs, capable of isolation in case of several noise sources and disturbances. The usage of UIOs however implies a linear state system for the otherwise complex spacecraft. The linearization is achieved by using an identification process given the inputs and outputs of the nonlinear system. This assumption is dropped in the scenario considered in this paper, which uses the full nonlinear attitude dynamics equations, applied to a simple subsystem with respect to the whole Mars Express thruster propulsion subsystem.
Also the tool developed in this thesis lies its foundation in the aforementioned principle of banks of observers. Thanks to the work of Weitian Chen and Saif [32], the mathematical modeling basis for the usage of actuator fault isolation NUIOs is carried out, laying down the path for their usage in space. Both the presented work and Fonod et al. [12] exploit this useful technique, and apply it for thruster fault isolation. The detection and final isolation stages of the latter work is however based of an EKF, that exploits information coming from the LIDAR device mounted on-board, differently to the accelerometer based one presented here.

1.3.2 Main Characteristics

Now that the stance of the presented solution with respect to other works is clear, a roundup of the main characteristics is carried out, outlining its main features.

One sensor The only sensor used for FDI is an Inertial Measurement Unit (IMU), which gives information in terms of angular velocity and linear acceleration of the spacecraft. Those measurements are usually available for all sorts of usages, so it is impossible not to find them already onboard. Higher precision/more specific sensors (e.g. star trackers, "per-actuator" sensors or LIDARs for position measuring) might not be present, so the possibility of using just IMU is highly advantageous.

Scalable The weight and volume of an IMU is respectively in the order of tens of grams and cubic centimeters, making it usable by vehicles from the nano/microsat range to bigger targets.

Ready to use During the early phases of the mission (e.g. after release into orbit) or when the spacecraft is in safe mode, multiple subsystems are not in use or can't be used cause of the s/c's initial tumbling. However the IMU is usually fully operative, being the simplest source of information available on the spacecraft's state. Hence FDI can be performed also during those phases.

Control system independence The control system is not touched by the FDI subsystem and the issue of mid-isolation command changing is considered and solved. This means that the tool can be inserted in an already existing spacecraft, simply by feeding it sensors readings and nominal input commands.

Multi Purpose Even if the example scenario is of de-tumbling, the tool can be used also for other thruster-propelled ACS maneuvers (e.g. rendez-vous). A robustness analysis with respect to external disturbances is performed, giving insights on other possible employments of the tool. The only limits are the thruster configuration (that could make the NUIOs not able to isolate) and applicability on spacecraft/missions with external force/torque disturbances higher than some limit thresholds.

Flight proven technology With respect to newer tools (e.g. data-based, model-free), the usage of relatively "simple" observers has been already thoroughly studied and used in space applications, as seen in the references before.

Robustness and sensitivity The tool's performance has been tested in the presence of unmodeled disturbances, giving promising results also in case of small faults. Furthermore, a trustworthy FDI is achieved even if the fault doesn't compromise the maneuver.

Low computational burden The algorithms developed here do not require high computational capabilities. In a real life scenario, for the majority of time, only the simple monitoring of the spacecraft acceleration will be running in background. Only when a fault is detected the algorithms for fault isolation go online, and they usually run for a very short period of time.

C implementation The algorithms of the discussed method have been coded in the low level programming language "C" which is currently the most popular language for onboard satellites' computers (together with C++). In addition to the faster speed of execution, this allows for the presented work to divert from just being a simulation (like MATLAB/Simulink implementations that would require steps like auto-coding to be used in real applications, often with interfacing problems to overcome) and be closer to a real world application.

Basilisk The environment of development and testing is Basilisk, an astrodynamics simulation software with Python/C/C++ scripting. The advantages that it brings to the work are in terms of 1) ease of reconfiguration for creating and testing different scenarios, thanks to its Python interface;2) library of pre-built and flight ready modules regarding different subsystems;3) the aforementioned advantages of the C language.

Chapter 2

Background Knowledge

In order to develop the FDI scheme, a proper background knowledge has to be laid down, in terms of mathematical models and simulation framework in which the tool will be inserted. First Basilisk is introduced, explaining its features. Then the discussion changes its focus on the modeling needed for the making of the FDIS. The spacecraft's attitude is modeled, and the NUIO is explained with its purpose and mathematical modeling
The following mathematical notations will be used here and throughout the thesis:

x = one dimensional variable or constant

$\mathbf{x}$ = vector

$\mathbf{X}$ = matrix

$\dot{x}$ = derivative in time of x

$\mathbf{X}^{\mathbf{T}}$ = transpose of matrix $\mathbf{X}$

$\mathbf{X}^{+}$ = Moore-Penrose inverse of matrix $\mathbf{X}$

$\mathbf{X} > 0$ = matrix $\mathbf{X}$ is positive definite

$\hat{\mathbf{x}}$ = estimate of vector $\mathbf{x}$

2.1 Basilisk Simulation Framework

The choice of using Basilisk as the simulation framework has been already mentioned in 1. Basilisk astrodynamics framework is a spacecraft simulation tool developed with an aim of strict modular separation and modeling decoupling, in regards to coupled spacecraft dynamics, environment interactions and flight software (FSW) algorithms [20]. The modularity comes from the fact that every element of the simulation, whether it is part of a dynamic/physical entity (the spacecraft, its solar panels, actuators, gravity bodies, disturbances, etc.) or of the flight software (algorithms for attitude determination, guidance, optical navigation, etc.) is represented by a module written in the C or C++ programming language. The simulation (*scenario* in Basilisk terms) is created in a Python environment, in which all the required modules are linked together and each one's characteristics (initial values, gains, required parameters) are set. In order to simulate properly a real life mission, different time steps are allowed to be set for different sections of the simulation. This, for example, can allow for a more accurate simulation of the orbital and attitude dynamics, while having the FSW part running at a slower frequency. The spacecraft dynamics are modeled as fully coupled multi-body dynamics with the generalized EOMs being applicable to a wide range of spacecraft configurations. The implementation uses a back-substitution method to modularize the EOMs and leverages the resulting structure of the modularized equations to allow the arbitrary addition of both coupled and uncoupled forces and torques to a central spacecraft hub ([4], [1], [7], [3], [5]). The effects on the spacecraft can be due to:

- **State effectors**. They derive from elements whose dynamics has to be integrated with the states of the spacecraft. Examples are reaction wheels, control moment gyroscopes, fuel slosh.

- **Dynamic effectors**. Those are external forces, or phenomena that can be condensed in external forces. Examples are thrusters, gravity, drag.

The communication between modules is done through a messaging system. Basilisk's messaging system manages the trafficking of inter modules messages and employs a publisher-subscriber message passing nomenclature. A single module may read and write any number of messages, which can be s/c states, forces, torques, flags, module-specific variables, etc. A module that writes output data, registers the 'publication' of that message by creating a new message entry with the message exchange. Conversely, a module that requires data output by another module subscribes to the message published by this last one. The advantages of using Basilisk as the simulation environment for the algorithm developed here are multiple [9] [2], and can be summarized in:

Speed The underlying simulation executes entirely in C/C++ which allows for maximum execution speed, given the low level of the two languages.

Reconfiguration The user interface executes in Python which allows the user to change integration rates, model/algorithm parameters, and output options dynamically on the fly.

Pre-existing modules An already existing wide set of modules, both for high fidelity dynamics and FSW, is present in Basilisk.

OBC implementation The ability to run on the spacecraft OBC the same exact hand-written, optimized, algorithms developed in the Basilisk desktop environment is a substantial advantage in sight of a possible real mission usage.

Analysis Python-standard analysis products like "numpy" and "matplotlib" are actively used to facilitate rapid and complex analysis of data obtained in a simulation run without having to stop and export to an external tool. This capability also applies to the Monte-Carlo engine available natively in the Basilisk framework.

The presence of such a well established framework makes the development of the FDI tool follow those workflow steps:

- Mathematical and physical modeling of the newborn FDI method.
- Creation of the C module containing the algorithm translation of the previous point, and the structure needed to run in Basilisk (e.g. the interfaces with the messaging system).
- Creation of the Python de-tumbling scenario containing the new FDI module and the needed pre-existing ones with which, as in the previous point, a proper interface is needed.
- Creation of the Montecarlo simulation containing the FDI scenario.

It is clear that, given the need to interface with other pre-existing and well tested modules, an understanding of their content is mandatory. The full mathematical modeling for each employed module is already available in Basilisk documentation [35], so it is decided to report only the modeling considered of interest for the current application.

2.2 Attitude Dynamics Equations

Attitude is the three-dimensional orientation of a body with respect to a specified reference frame [29]. The "attitude coordinates" can be expressed in a multitude of ways (DCM, quaternions, MRP, etc.), but all have in common their job of completely describe how a rigid body is placed relative to a reference frame. As seen in figure 2.1 two frames are defined:

- **Inertial Frame (IF)** is a fixed inertial frame taken as reference.
- **Body Frame (BF)** is the frame attached to the spacecraft body. It is important to note that in this digression, the BF is centered in the spacecraft's COM. Its distance from IF is $\mathbf{r}$ (it will always be expressed in BF).

For higher tasks such as attitude determination, guidance, or pointing, the information on the instantaneous attitude is required, cause it will be part of the governing equations of the control subsystem. However, for the FDI purposes of this thesis, the orientation of the spacecraft is not important *per se*, cause it will not enter in any stage of the tool. At the basis of the FDI, lies the idea that a thruster fault will induce an effect on the spacecraft's attitude. This means that instead of the actual "angles" between the IF and BF, it is important to know how those angles evolve in time in terms of *angular velocity*. The fundamental equations of motion for rotational dynamics, governing the angular velocity of a rigid body are the *Attitude Dynamics Equations (ADEs)*. With the assumption that the spacecraft is a rigid body with uniform mass distribution *and* that BF is aligned with the physical principal axes of inertia, the s/c momentum can be expressed in BF coordinates as:

$$\mathbf{h}(t) = \mathbf{J}\boldsymbol{\omega}_{\boldsymbol{BN}}(t) \tag{2.1}$$

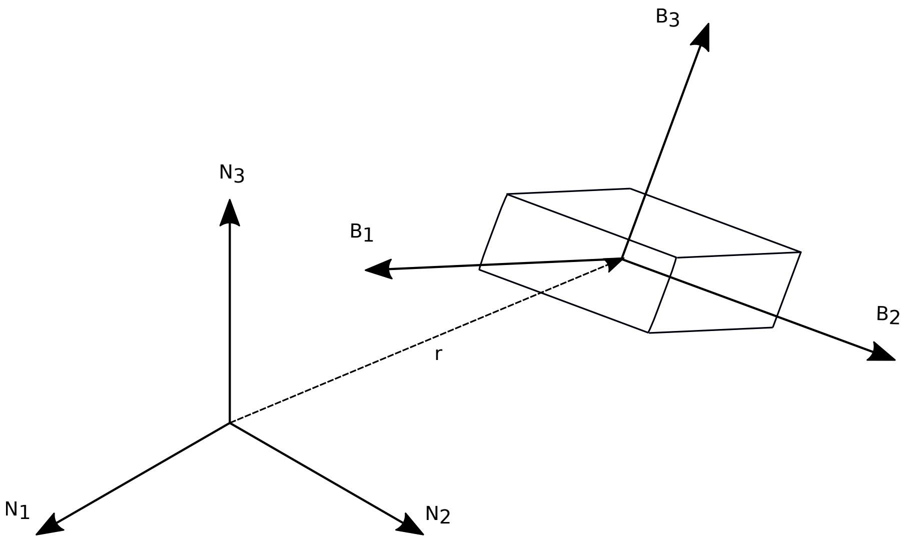

Figure 2.1: Reference frames

Where $\boldsymbol{\omega_{BN}}$ is the angular velocity of the rigid body with respect to IF in BF coordinates (from now on just $\boldsymbol{\omega}$). $\mathbf{J}$ is the s/c's inertia matrix that, with the previous assumptions, reduces to the diagonal one:

$$\mathbf{J} = \begin{bmatrix} J_1 = \int_B (y^2+z^2)dm & 0 & 0 \\ 0 & J_2 = \int_B (x^2+z^2)dm & 0 \\ 0 & 0 & J_3 = \int_B (x^2+y^2)dm \end{bmatrix}$$

The diagonal terms represent the (uniform) mass distribution and are called "Inertia Moments". In order to derive the ADEs for the angular velocity, the time derivative of the angular momentum is taken, in IF coordinates, which can be rewritten thanks to the transport theorem to:

$$\left[\frac{d\mathbf{h}(t)}{dt}\right]_N = \left[\frac{d\mathbf{h}(t)}{dt}\right]_B + \boldsymbol{\omega}(t) \times \mathbf{h}(t) \tag{2.2}$$

In the presence of a generic vector of external moments $\mathbf{m_{ext}}$, the ADEs become (with everything in body coordinates):

$$\mathbf{J}\dot{\boldsymbol{\omega}}(t) + \boldsymbol{\omega}(t) \times \mathbf{J}\boldsymbol{\omega}(t) = \mathbf{m_{ext}}(t) \tag{2.3}$$

By dividing each term by $\mathbf{J}$, and expressing them component-wise:

$$\dot{\omega}_1 = \left(\frac{J_2 - J_3}{J_1}\right)\omega_2\omega_3 + \frac{m_{ext1}}{J_1} \tag{2.4}$$

$$\dot{\omega}_2 = \left(\frac{J_3 - J_1}{J_2}\right)\omega_1\omega_3 + \frac{m_{ext2}}{J_2} \tag{2.5}$$

$$\dot{\omega}_3 = \left(\frac{J_1 - J_2}{J_3}\right)\omega_1\omega_2 + \frac{m_{ext3}}{J_3} \tag{2.6}$$

Those last equations, together with the ones discussed in the next section, will be the basis for the NUIOs, explained in 4. The terms $\mathbf{J}$ and $\mathbf{m_{ext}}$ will be characterized in the next chapter, where all the mission related quantities will be explained.

2.3 Nonlinear Unknown Input Observer

The Nonlinear Unknown Input Observer is the extension of the more common Unknown Input Observer to the nonlinear case. As the name suggests, it deals with systems characterized by one or more unknown inputs, and in particular is able to decouple them from the state estimation process. This powerful property makes them useful in a variety of applications, going from disturbance rejection to both sensors and actuators fault detection.
The usage of NUIOs to assess the FDI problem has been explored by various authors, as seen in the first chapter. The main idea behind their adoption is that, by treating an actuator fault as abnormality with respect to the nominal input, it is possible to create a NUIO whose estimation job is not susceptible to that particular actuator, hence monitoring its fault state.
A mathematical dissertation on NUIOs can be found on [32], [27] and [13], and a review is carried out on this section given the important properties that are going to be derived. While in the first reference the authors lay down a general usage of the NUIO applied to FDI, leading to general conclusions, this dissertation will focus slightly more on the results applicable to the thesis subject.
The following general nonlinear system is considered:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{f}(\mathbf{x}(t)) + \bar{\mathbf{B}}_\mathbf{n}\bar{\mathbf{u}}_\mathbf{n}(t) + \mathbf{B}_\mathbf{n}\mathbf{u}_\mathbf{n}(t) \tag{2.7}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) \tag{2.8}$$

where $\mathbf{x}(t)$ is the state vector of the system, $\mathbf{A}$ the dynamics' matrix, $\mathbf{f}(x)$ the nonlinear part of the dynamics, and $\mathbf{y}(t)$ the output vector. From now on the time dependence is omitted for brevity.
The remaining terms represent the known inputs and the disturbances coming from faulty actuators. They are defined starting from the full input matrix:

$$\mathbf{B}\mathbf{u} = \bar{\mathbf{B}}_\mathbf{n}\bar{\mathbf{u}}_\mathbf{n} + \mathbf{B}_\mathbf{n}\mathbf{u}_\mathbf{n} \tag{2.9}$$

$\mathbf{B}$ and $\mathbf{u}$ are the full input matrix and vector. By removing from $\mathbf{B}$ the column(s) corresponding to the unknown input actuator(s), $\mathbf{B}_\mathbf{n}$ is obtained, with $\mathbf{u}_\mathbf{n}$ its corresponding input vector. The remaining columns of $\mathbf{B}$ and elements of $\mathbf{u}$ are collected into $\bar{\mathbf{B}}_\mathbf{n}$ and $\bar{\mathbf{u}}_\mathbf{n}$.
Two assumptions are made beforehand:

- A1: $\mathbf{A}, \bar{\mathbf{B}}_\mathbf{n}, \mathbf{B}_\mathbf{n}$ are known and $\mathbf{B}_\mathbf{n}$ is of full column rank
- A2: $||\mathbf{f}(\mathbf{x}) - \mathbf{f}(\hat{\mathbf{x}})|| \leq \gamma||\mathbf{x} - \hat{\mathbf{x}}||$

With $\gamma > 0$ being a positive Lipschitz constant, and for all $\hat{\mathbf{x}}$, $\mathbf{x}$ ($\hat{\mathbf{x}}$ being the estimated state).
The inputs coming from the actuators can be either healthy ($\mathbf{u^h}$) or faulty, with $\mathbf{u^h} = \mathbf{u}$ only in case of healthy actuators, and $\mathbf{u^h} = \bar{\mathbf{u}}^\mathbf{h}_\mathbf{n} + \mathbf{u_n}^\mathbf{h}$.
The goal is to design an observer sensitive only to faults coming from actuators represented by $\mathbf{B}_\mathbf{n}$. For that reason a NUIO is proposed, with the following characteristics:

$$\dot{\mathbf{z}}_\mathbf{n} = \mathbf{N}_\mathbf{n}\mathbf{z}_\mathbf{n} + \bar{\mathbf{G}}_\mathbf{n}\bar{\mathbf{u}}^\mathbf{h}_\mathbf{n} + \mathbf{L}_\mathbf{n}\mathbf{y} + \mathbf{M}_\mathbf{n}\mathbf{f}(\hat{\mathbf{x}}_\mathbf{n}) \tag{2.10}$$

$$\hat{\mathbf{x}}_\mathbf{n} = \mathbf{z}_\mathbf{n} - \mathbf{E}_\mathbf{n}\mathbf{y} \tag{2.11}$$

Where the matrices are defined as:

$$\mathbf{N_n} = \mathbf{M_n A} - \mathbf{K_n C} \tag{2.12}$$
$$\mathbf{\bar{G}_n} = \mathbf{M_n \bar{B}_n} \tag{2.13}$$
$$\mathbf{L_n} = \mathbf{K_n(I + CE_n)} - \mathbf{M_n A E_n} \tag{2.14}$$
$$\mathbf{M_n} = \mathbf{I} + \mathbf{E_n C} \tag{2.15}$$

Hence by using 2.11 and 2.8, the estimation error is:

$$\mathbf{e_n} = \mathbf{\hat{x}_n} - \mathbf{x} = \mathbf{z_n} - \mathbf{E_n C x} \tag{2.16}$$

It is useful to also express $\mathbf{\dot{e}_n}$, that will be used later to prove the NUIO's estimation capability. In order to do that, the time derivative of the error is taken, and 2.7, 2.10 are used. After rearranging the terms:

$$\mathbf{\dot{e}_n} = \mathbf{N_n e_n} + \mathbf{M_n(f(\hat{x}_n) - f(x))} + \mathbf{\bar{G}_n(\bar{u}_n^h - \bar{u}_n)} - \mathbf{M_n B_n u_n} \tag{2.17}$$

The sufficient conditions for the NUIO existence are:

$$1)\ \mathbf{M_n B_n} = \mathbf{0} \tag{2.18}$$
$$2)\ \mathbf{N_n}^{\mathbf{T}}\mathbf{P} + \mathbf{P N_n} + \gamma \mathbf{P M_n M_n}^{\mathbf{T}}\mathbf{P} + \gamma \mathbf{I} < \mathbf{0} \tag{2.19}$$

Where $\mathbf{P}$ is symmetric and positive definite, and solves the matrix inequality. Discussions about the existence of those are found in the aforementioned related literature. However, given this paragraph's purpose of designing a NUIO with accurate state estimation properties, only the proof of their ability to bring the error asymptotically to zero is reported.
By recalling 2.18 and noticing that $\mathbf{\bar{u}_n^h} = \mathbf{\bar{u}_n}$ (the bar on top stands for not-unknown actuators, that are healthy by construction of the NUIO) the simplified expression for 2.17 is:

$$\mathbf{\dot{e}_n} = \mathbf{N_n e_n} + \mathbf{M_n(f(\hat{x}_n) - f(x))} \tag{2.20}$$

Moreover, 2.19 can be rewritten introducing the positive definite matrix $\mathbf{Q}$:

$$\mathbf{N_n}^{\mathbf{T}}\mathbf{P} + \mathbf{P N_n} + \gamma \mathbf{P M_n M_n}^{\mathbf{T}}\mathbf{P} + \gamma \mathbf{I} = -\mathbf{Q} \tag{2.21}$$

To show that that $\mathbf{\dot{e}_n} \to \mathbf{0}$ for $t \to \infty$, a Lyapunov function is chosen as:

$$\mathbf{V_L} = \mathbf{e_n}^{\mathbf{T}}\mathbf{P e_n} \tag{2.22}$$

By differentiating in time, and substituting 2.20 in it:

$$\mathbf{\dot{V}_L} = \mathbf{e_n}^{\mathbf{T}}[\mathbf{N_n}^{\mathbf{T}}\mathbf{P} + \mathbf{P N_n}]\mathbf{e_n} + 2\mathbf{e_n}^{\mathbf{T}}\mathbf{P M_n}[\mathbf{f(\hat{x}_n)} - \mathbf{f(x)}] \tag{2.23}$$

At this point the assumption A1 is replaced in the last term of 2.23, leading to a chain of inequalities that make the Lyapunov function more and more positive. By substituting the expression of the Riccati equation in 2.21, the positive definite $\mathbf{Q}$ appears:

$$\mathbf{\dot{V}_L} = -\mathbf{e_n}^{\mathbf{T}}\mathbf{Q e_n} \tag{2.24}$$

Since $\mathbf{Q} > 0$, the second Lyapunov Stability Theorem ([19]) is met, and the error will go indefinitely to zero. Now that this property is verified, and the NUIO's ability to correctly estimate in case of actuator fault is proved, the continuation of its construction is carried on.
Recapping, in order to fully define the NUIO, the matrices defined in 2.12 $\sim$ 2.15 are to be found. In particular:

- $\mathbf{E_n}, \mathbf{K_n}$ are still unknown
- $\mathbf{P} > \mathbf{0}$ such that 2.19 is solved, has to be found

By combining 2.15 and 2.18 it is found:

$$\mathbf{E_n C B_n} = -\mathbf{B_n} \tag{2.25}$$

Since $\mathbf{B_n}$ is full column rank so it is required that $\mathbf{CB_n}$ should also be of full column rank [27]. If $\mathbf{CB_n}$ is full column rank, all possible solutions of $\mathbf{E_n}$ can be expressed in a general way in:

$$\mathbf{E_n} = \mathbf{U} + \mathbf{YV} \tag{2.26}$$

With:

$$\mathbf{U} = -\mathbf{B_n}(\mathbf{CB_n})^+ \tag{2.27}$$

$$\mathbf{V} = \mathbf{I} - ((\mathbf{CB_n})(\mathbf{CB_n})^+) \tag{2.28}$$

The unknowns are now $\mathbf{K}$, $\mathbf{Y}$ and $\mathbf{P}$. It is shown that in order to find those matrices, and at the same time solve 2.19, the following Linear Matrix Inequality system has to be solved:

$$\begin{pmatrix} \mathbf{X_{11}} & \mathbf{X_{12}} \\ \mathbf{X_{12}}^{\mathbf{T}} & -\mathbf{I} \end{pmatrix} < \mathbf{0}$$

With:

$$\begin{aligned} \mathbf{X_{11}} &= [(\mathbf{I} + \mathbf{UC})\mathbf{A}]^T\mathbf{P} + \mathbf{P}(\mathbf{I} + \mathbf{UC})\mathbf{A} + (\mathbf{VCA})^T\bar{\mathbf{Y}}^T + \bar{\mathbf{Y}}(\mathbf{VCA}) \\ &\quad - \mathbf{C}^T\bar{\mathbf{K}}^T - \bar{\mathbf{K}}\mathbf{C} + \gamma\mathbf{I} \end{aligned} \tag{2.29}$$

$$\mathbf{X_{12}} = \sqrt{\gamma}[\mathbf{P}(\mathbf{I} + \mathbf{UC}) + \bar{\mathbf{Y}}(\mathbf{VC})] \tag{2.30}$$

With $\mathbf{Y_n} = \mathbf{P}^{-1}\bar{\mathbf{Y}}$ and $\mathbf{K_n} = \mathbf{P}^{-1}\bar{\mathbf{K}}$. This problem can be solved practically using MATLAB's LMI solver, finding the unknowns $\bar{\mathbf{K}}$, $\bar{\mathbf{Y}}$ and $\mathbf{P}$. Finally all quantities necessary to characterize the NUIO are known, and its estimating properties are met.
The number of faults that can be simultaneously detected is strictly connected to the sufficient conditions in 2.18 $\sim$ 2.19, and a more general analysis is found in [32]. For the particular application of this thesis, those conditions will be verified in 4, where it will be seen how this design can be used for FI in thruster driven ACSs.

Chapter 3

Mission Description

As mentioned before, the Basilisk scenario will be the environment in which the FDI tool will be inserted, so this chapter will cover its surroundings, in terms of modules and mission specific quantities.

3.1 Spacecraft

The choice of the spacecraft has been arbitrary, since the goal of simulating an FDI tool does not depend on the kind of spacecraft. This cannot be said for its thrusters configuration, that will be discussed later. In order to be consistent, however, with the considerations about the usage on smaller s/c (1), the choice has fallen on a *micro-spacecraft* kind. The shape is of a rectangular parallelepiped and can be appreciated in figure 3.1, together with its principal inertia axes alignment with the BF. Given the shape and the uniform mass distribution, it is possible to

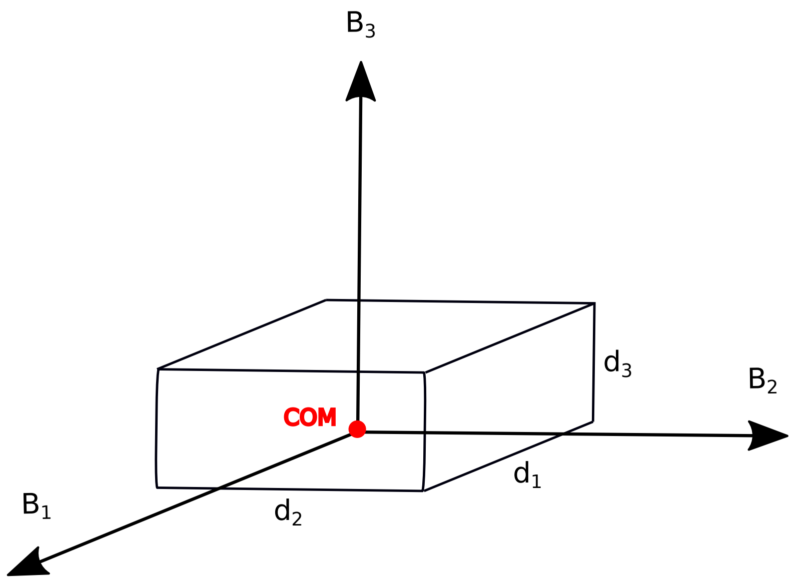

Figure 3.1: Spacecraft shape in BF

Table 3.1: Table of spacecraft characteristics

Spacecraft characteristics	
Mass (m)	100 kg
Dimensions	
d_1	0.8 m
d_2	0.7 m
d_3	0.6 m
Principal inertia moments	
J_1	7.08 kg x m^2
J_2	8.33 kg x m^2
J_3	9.42 kg x m^2

define the principal inertia moments as:

$$J_1 = \frac{1}{12} m d_2^2 d_3^2 \tag{3.1}$$

$$J_2 = \frac{1}{12} m d_1^2 d_3^2 \tag{3.2}$$

$$J_3 = \frac{1}{12} m d_1^2 d_2^2 \tag{3.3}$$

Where d_i is the length of the i-th BF aligned dimension and m is the mass of the spacecraft. Based on that, the numerical values of the aforementioned quantities are reported in table 3.1. The Basilisk module that represents the spacecraft is *spacecraftPlus.cpp*. In this module the equations of motion of the spacecraft are computed, taxing into account the effects of the aforementioned dynamic and state effectors. The states of the spacecraft integrated in this module are (all in BF with respect to IF) the position $\mathbf{r}$, the velocity $\dot{\mathbf{r}}$, the MRP $\boldsymbol{\sigma}$, and the angular velocity $\boldsymbol{\omega}$. The parameters that are being set are the mass defined before, the diagonal inertia matrix containing J_1, J_2, J_3 on the diagonals, and the position vector of the COM in BF, which is $[0, 0, 0]$ in the analyzed case.
SpacecraftPlus can simulate pure translational movement, pure rotational movement, and the coupled problem when both are present. As mentioned in 2.1 the full integration scheme implemented is quite complicated, so it will not be reported here. Besides, as mentioned before, the only equations meaningful for the FDI scheme are the Euler Equations, so the problem could be set as a translational one. However, to give the FDIS more of a real application usage, it is decided to keep both motions, without describing the fully coupled dynamics, that is not object of this thesis. The translational part will require the definition of the orbital motion of the spacecraft, and will be tackled in the next section.

3.2 Environment and Disturbances

In order to test the FDI tool, also an environment had to be set. The choice has fallen on a deep space environment, in which the de-tumbling maneuver takes place. This environment acts as a general test-bed, and is representative of several real life missions. Even though the disturbances in this particular environment are expected to be negligible with respect to an actuator fault, it is decided to make a robustness analysis of the method in presence of those. This means that their magnitude will not be realistically modeling the deep space environment, and are better seen as unknown external forces/torques that will try to corrupt the FDI performance. For this reason the Basilisk module *extForceTorque.cpp* is added to the scenario and will act as a disturbances-creator. This module allows a general force and/or torque to be applied onto a rigid body. The force is the net external force acting through the center of mass, and will be specified in BF coordinates. The torque is taken about the BF origin, and the vector components are given in BF. Their impact on the FDI will be evaluated within the Montecarlo simulations.

3.3 Sensors

The only sensor needed for the FDI tool to work is an Inertial Measurement Unit. This kind of sensors are capable of outputting different kinds of measurements, from linear accelerations to angular rates to accumulated changes in velocity. They usually consist in several different sensors (accelerometers, gyroscopes, inclinometers, etc.) and their usage is widespread in a all kinds of applications, including space. Dimensions can widely differ among different models, but the new electronic miniaturization processes have allowed for drastic reduction in dimensions, leading to the advent of Micro Electro-Mechanical Systems (MEMS). An example of space IMU is the Sensonor STIM300 [16] (in figure 3.2). Its weight of $55g$ and dimensions

Figure 3.2: Sensonor STIM300 IMU

of approximately $45 \times 39 \times 22\ mm^3$ allow for its implementation on very small payloads, making it a strong candidate for the micro-spacecraft analyzed in this mission.
In Basilisk an IMU sensor can be modeled with the module *imu_sensor.cpp*. Among all the possible measurements that can be set, the choice has been a 3-axial gyroscope

for angular velocity and 3-axial accelerometer for linear acceleration. Those are the two measurements needed by the FDIS, and a sensor like the STIM300 is capable of providing these information. The first important choice when adding an IMU is its placement on the spacecraft. By looking, for example, at the accelerometers reading, a placement different from the COM would allow for an additional acceleration due to the eventual spacecraft's rotation. Even though Basilisk is able to model any arbitrary IMU placement, it is chosen here a COM mount, with the sensor axes aligned with the BF. This is not an uncommon choice in real life applications.
With those assumptions the IMU reference frame will be the same as the BF, and all the equations to come will then be in BF with respect to IF. The two measurements will be modeled starting from the information produced by *spacecraftPlus.cpp*. The sensed angular velocity $\boldsymbol{\omega_{sensed}}$ will be simply taken from the aforementioned module, being one of the states of the spacecraft. As for the accelerometer, the general equation for the acceleration felt by the IMU is:

$$\ddot{\mathbf{r}}_{\mathbf{sensed}} = (\ddot{\mathbf{r}} - \mathbf{a}_{\mathbf{grav}}) + \dot{\boldsymbol{\omega}} \times \mathbf{r}_{\mathbf{IMU}} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}_{\mathbf{IMU}}) \tag{3.4}$$

Thank to the placement, the distance of the IMU from the BF ($\mathbf{r}_{\mathbf{IMU}}$) is zero, so also the third and fourth terms (representing the apparent acceleration due to the rotation) are zero. Furthermore, in the deep space environment considered there is no gravity acting, so also the acceleration due to gravity $\mathbf{a}_{\mathbf{grav}}$ will be zero. All that's left is the linear acceleration of the spacecraft, that will be due to external forces, including the thrusters. However this quantity is not part of the states, so it has to be extrapolated. The chosen method is a simple Euler integration scheme across one integration step of the velocity $\dot{\mathbf{r}}$ (which, conversely, is one of the states). In this way:

$$\ddot{\mathbf{r}}_{\mathbf{sensed}} = \ddot{\mathbf{r}} = \frac{\dot{\mathbf{r}}_{\mathbf{post\_integ}} - \dot{\mathbf{r}}_{\mathbf{pre\_integ}}}{\Delta t} \tag{3.5}$$

3.4 Thrusters

In order to perform attitude control, the spacecraft taken into consideration is equipped with eight cold gas thrusters (CGT). Of the chemical propulsion systems frequently used in spacecraft, cold gas propulsion systems have the lowest complexity and cost. They can provide highly repeatable, extremely small impulse bits for accurate orbit maintenance and attitude control, at the expense of the specific impulse. For minor primary propulsion functions and ACS tasks with a relatively short mission duration and a low overall impulse, cold gas systems may work well. For these applications, the simplicity and low dry mass are a benefit, despite the low Isp [11]. They consist of a simple system composed by a tank containing the pressurized gaseous propellant, and a valve that when actuated opens the passage for the gas that gets expelled through a nozzle. Its simple scheme is reported in 3.3. All these characteristics tie well with the small nature of the satellite, and represent a plausible mission test bed for the FDI tool.
The thruster configuration is of fundamental importance for any thruster FDI scheme and the particular method presented here is no exception. In order to produce *pure torque* the choice has fallen onto an eight thruster configuration

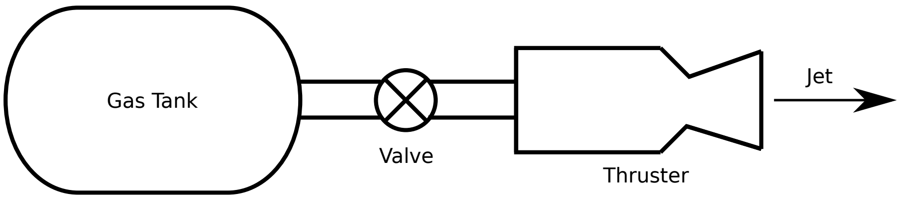

Figure 3.3: Cold gas thrusters scheme

Table 3.2: Table of thruster locations in body frame

	x [m]	y [m]	z [m]
Thruster 1	0.4	0	0.3
Thruster 2	-0.4	0	0.3
Thruster 3	-0.4	0	0.3
Thruster 4	0.4	0	0.3
Thruster 5	0.4	0	-0.3
Thruster 6	-0.4	0	-0.3
Thruster 7	-0.4	0	-0.3
Thruster 8	0.4	0	-0.3

($n_{th} = 8$), analyzed in Basilisk documentation, capable of generating torque in all axes, thus controlling the angular velocity of the spacecraft in the 3D space. Its graphic representation can be seen in figure 3.4. Here the thrusters are represented in terms of firing directions (the vectors originating from and the spacecraft sides, no to be confused with force directions directed opposite) and torque generated (the vectors originating from the origin). The color code is made such that same colored vectors generate same colored torques. The numerical values of the locations and force directions are reported in tables 3.2 and 3.3.
The conditions for which any perturbation torque can be compensated with a positive force exerted by the thrusters using a particular configuration are given in [26] and can be summarized in:

- The mapping matrix (defined later as $\mathbf{D_t}$) is of full rank.
- The mapping matrix null-space has sign definite vectors.

Both these conditions have been verified in Matlab for the analyzed configuration, with the commands *rank* and *null*.

The control unit (discussed later) will handle the firing of those thrusters, but already from here it's clear that a symmetry can be exploited to create pure torque action on the spacecraft. In fact, by always firing together the thruster couples $\mathbf{th_{couples_1}} = [1, 7], \mathbf{th_{couples_2}} = [2, 8], \mathbf{th_{couples_3}} = [3, 5]$ and $\mathbf{th_{couples_4}} = [4, 6]$ (i.e. same colored thrusters in 3.4), the opposing forces will always cancel each other, and no net force is induced on the spacecraft. This property, not mandatory but neither uncommon in ACS thrusters, will be exploited by the FDI subsystem.
Each i-th thruster is capable of generating a torque:

$$\mathbf{t_i} = \mathbf{r_i} \times f_{max_i} \mathbf{\hat{g}_{f_i}} \tag{3.6}$$

Table 3.3: Table of thruster force directions in body frame

	x	y	z
Thruster 1	-0.7071	-0.7071	0
Thruster 2	0.7071	-0.7071	0
Thruster 3	0.7071	0.7071	0
Thruster 4	-0.7071	0.7071	0
Thruster 5	-0.7071	-0.7071	0
Thruster 6	0.7071	-0.7071	0
Thruster 7	0.7071	0.7071	0
Thruster 8	-0.7071	0.7071	0

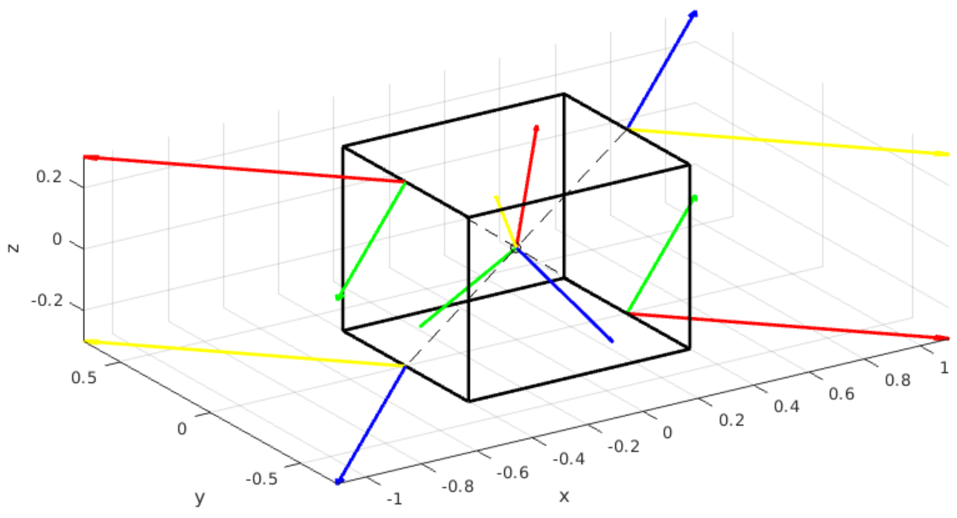

Figure 3.4: Thrusters configuration

Where $\mathbf{r_i}$ and $\mathbf{\hat{g}_{f_i}}$ are the distance from the center of mass and force direction (in BF, reported on the previous tables). f_{max_i} is the nominal force produced by each thruster.
It is possible then, to build a 3×8 torque matrix $\mathbf{B_t}$ whose columns are the torque contributions of each thruster:

$$\mathbf{B_t} = [\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}, \mathbf{t_4}, \mathbf{t_5}, \mathbf{t_6}, \mathbf{t_7}, \mathbf{t_8}] \tag{3.7}$$

Table 3.4: Table of thrusters properties. The location and direction will vary, but all the other properties are considered the same for each thruster

Thrusters properties	
Location	$\mathbf{r_i}$ m
Force direction	$\hat{\mathbf{g}}_{\mathbf{f_i}}$
Max thrust	0.1 N
I_{sp}	65 s
t_{minOn}	0.002 s

Note that cause of the symmetry:

$$\mathbf{t_1} = \mathbf{t_7}; \quad \mathbf{t_2} = \mathbf{t_8}; \quad \mathbf{t_3} = \mathbf{t_5}; \quad \mathbf{t_4} = \mathbf{t_6} \tag{3.8}$$

By multiplying $\mathbf{B_t}$ with a boolean command vector $\mathbf{u_c}$, representing the on-off (1 or 0) state of each thruster, the commanded torque is produced, that will be equal to the contribution of the external forces in the ADEs in 2.2:

$$\mathbf{m_{ext}} = \mathbf{B_t}\mathbf{u_c} \tag{3.9}$$

3.4.1 Thrusters Module

Since they generate an external thrust acting on the spacecraft, the thrusters are seen as a dynamic effector, thus the module *thrusterDynamicEffector.cpp*. This model is used in the Basilisk simulation to emulate the effect of a vehicle's thrusters on the overall system. Its primary use is to generate realistic forces/torques on the vehicle structure and body. This is accomplished by applying a force at a specified location/direction in the body and using the current vehicle center of mass to calculate the resultant torque. A on/off ramp behavior can be enabled, but it won't be done here. The thruster properties to be set are summarized in table 3.4. These values are chosen considering the performances of typical Moog CGTs for micro propulsion, reported in [6], being a reasonable choice for the mission prototype analyzed here. t_{minOn} is the smallest time a thruster can stay open, and is related to its Minimum Impulse Bit. The reason why also a value for I_{sp} is provided, is because of Basilisk's ability to compute the mass flow rate of the propellant, thus its effect of a weight loss on the dynamics, through the formula:

$$\dot{m}_{exp} = \frac{f_{max_i}}{g I_{sp}} \tag{3.10}$$

With g being the Earth's gravity on surface.
The input for this module will come directly from the control subsystem, that will tell for how long and which thrusters have to be firing at each update, in a 8×1 vector called $\mathbf{u_{onTime}}$. At each integration step this vector will be updated with the elapsed time, taking into account the minimum on times, and the boolean command vector $\mathbf{u_c}$ (defined earlier) is created.

3.4.2 Types of Faults

By looking at the working principle of CGTs, and at the available literature, there are two main fault kinds that can happen, and both are contemplated in this thesis:

- Loss of efficiency with reduction of max thrust available.
- Thruster no more responsive to commands.

The first one corresponds to a change in the f_{max_i} parameter, that will become just a fraction of the original value. This situation can be due to leakage problems in the system, and is a well documented problem in literature [6].
The second, instead, contemplates the situation of a "stuck thruster". This means that for some reason there's no more the possibility to command the thruster valve, that will be then stuck in a specific position. Other than the simple stuck on/off situations, also the "stuck on with efficiency loss" is considered, where the CGT is providing a fraction of f_{max_i} (just like in the other case), but without the possibility of actuating the valve. Single thruster failures are explored and, depending on the kind-of-fault/firing-command combination, the faulty thruster might or might not have an effect on the spacecraft. Details on this topic are in the next chapter. What is worth noticing here is that in a non-faulty situation the opposite firing thrusters are always in the same state (from control law) and exert the same force, so only the presence of a fault might induce a net force.
In Basilisk there is no built in way to induce thrusters faults, so it has been decided to inject them at fault time in the following ways:

- The loss of efficiency fault is induced by setting a new value of max force in the faulty thruster physical properties. Since this change is related to the dynamics part of the simulation, the FSW will not be aware of his change, and continue to work with the nominal thrusters values.
- The stuck thruster fault is induced by, for each time step, overriding the on-time command **after** it is produced by the control system and fed to the FDI module. This means that also this time the control and FDIS will not be aware of the modification.

3.5 Control

Even though the FDI scheme does not intervene in the control system, this last does play a significant role in the depicted scenario. The chosen mission is of a de-tumbling maneuver, so it is necessary to have a control law capable of achieving such goal. In this analysis two methods are represented, one already available with Basilisk's current modules, and one created ad-hoc. They will be explored in the next subsections and referred as "Basilisk De-Tumbling Control" (BDT) and "Simple De-Tumbling Control" (SDT). It is worth noticing that the knowledge of the spacecraft's attitude is required for the fist kind of control, so it is necessary to use *simpleNav.cpp*. This module doesn't simulate a specific sensor, but rather a

series of measurements that can come from different sources. The spacecraft MRP and $\boldsymbol{\omega}$ are the information to be retrieved for control purposes. It is clear that in this case the information will not come from the IMU but from some other source. This, however, does not defeat the goal of the thesis to use just an IMU sensor for FDI, cause independently from the control, the FDI module will always just have the IMU information coming from the corresponding IMU module.
Usually a de-tumbling mission would be designed in such a way that when some pre-established conditions on the angular velocity are met, the control turns off. For the analyzed application however the control subsystem is instructed to stay on for the whole simulation duration. This is done in order to raise the probability of excitement of a faulty thruster. On the contrary, if the commands were to be turned off, and, for example, an efficiency-loss kind of fault is introduced, there will never be a detectable effect on the spacecraft.
The simulation time is chosen by looking at the angular velocity behavior in a 100 runs Montecarlo simulation, with different initial conditions on the angular velocity and MRPs, using the BDT control. In a time span of three minutes, the angular velocities go from a maximum initial value of around 0.3 rad/s, to a final one of around 0.002 rad/s. This is considered a useful test bed for the FDI, which will then be tested in a time span of 3 minutes for all the simulations in this thesis.

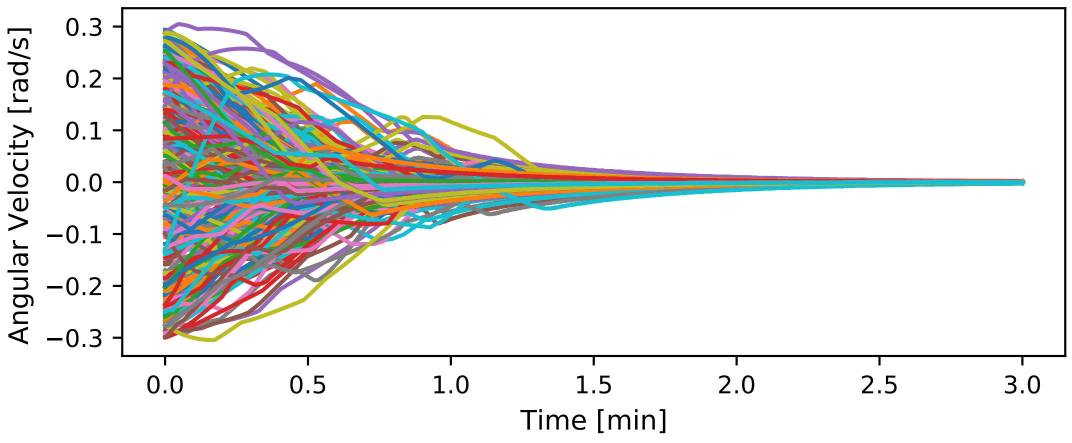

Figure 3.5: De-tumbling control on a 100 runs Montecarlo simulation

3.5.1 Basilisk De-Tumbling Control

The modules handling a de-tumbling control in Basilisk are: *inertial3D.c*, *attTrackingError.c* and *MRP_Feedback.c*. In [10] a full explanation of those modules is carried on, and will be summarized here with the purpose of completely characterize the mission.

Inertial3D

This simple module has the job of defining an inertial reference frame to be followed by the control. The input will be given in terms of MRP of the desired attitude, while the (most important) condition of null angular velocity is set by default in the module. This means that the control will try to align the spacecraft's body frame with this inert frame, that will be the output of this module, with the result of achieving an inertial pointing together with the de-tumbling.

Attitude Tracking Error

The next step is to evaluate the error between the body frame and the reference frame described in the previous module, thus the two inputs will be the current attitude information coming from *simpleNav.cpp*, and the reference frame from *inertial3D.c*. It is contemplated the possibility of aligning another reference frame, different from BF, to the inertial one, but it's not useful for a simple de-tumbling. As expected, since the goal of the control is to perform an inertial pointing more than just de-tumbling, the error is computed both in terms of MRP and angular velocity, with respect to the reference frame, and will be brought to zero by the next modules.

MRP Feedback

The goal of this module is to produce a body-frame control torque vector capable of bringing to zero the attitude tracking error coming from the previous module. The other inputs are the vehicle configuration message containing the inertia matrix, and the gains used in the control law. The full MRP control algorithm can be found in [19], together with the proof of Lyapunov stability, and will not be object of discussion in this thesis. The values of the gains have been tuned for a successful de-tumbling. The output will be a 3D control torque $\mathbf{l_r}$ that the thrusters will have to produce through a proper thrust allocation method.

3.5.2 Simple De-Tumbling Control

In order to have a scenario containing just the IMU as sensor, also for the control, an ad-hoc simple c module has been created, that does not take into account the current attitude (MRPs) of the spacecraft. The module is called *deTumb.c* and takes as input a gain k_{det} and the current $\boldsymbol{\omega}$ sensed by the IMU. The control law employed is the simple but effective:

$$\mathbf{l_r} = -k_{det}\boldsymbol{\omega} \tag{3.11}$$

The Lyapunov stability of this law is well known in literature and can be simply proven by considering the kinetic energy as Lyapunov function:

$$V(\boldsymbol{\omega}) = \boldsymbol{\omega} \cdot \mathbf{J}\boldsymbol{\omega} \tag{3.12}$$

By deriving it and substituting the expression of the ADEs with $\mathbf{m_{ext}} = -k_{det}\boldsymbol{\omega}$, the negativity of $\dot{V}$ is ensured, being:

$$\dot{V}(\boldsymbol{\omega}) = -\boldsymbol{\omega}^{\boldsymbol{T}} k_{det}\boldsymbol{\omega} \tag{3.13}$$

Hence the stability of the chosen control law.
The biggest difference between the two laws is their final objective. Two goals are

being chased by BDT: bringing the angular velocity to zero and aligning the MRPs with an inertial frame. SDT, instead, only brings the angular velocity to zero. This translates to a wider variety of commands instructed by BDT, that will create more chance of exciting a faulty thruster, thus provoking the fault effects. Since the whole point of the dissertation is to verify the FDI tool performances, the BDT results to be more useful, and will be the only one used in the simulations. In this sense the design of SDT has been just a proof of concept for an IMU-based control.

3.5.3 Generation of the On-Time Input Vector

The $\mathbf{l_r}$ computed by both the two laws is a 3D vector, representing the torque to be generated by the thrusters. The final goal of the control subsystem is to create the aforementioned $\mathbf{u_{onTime}}$ vector and, to do so, two steps are required:

- $\mathbf{l_r}$ has to be transformed in an 8×1 vector of forces ($\mathbf{u_{forces}}$).
- $\mathbf{u_{forces}}$ has to be transformed in the 8×1 $\mathbf{u_{onTime}}$, that is an input for *thrusterDynamicEffector.cpp*.

The two modules responsible for that are *thrForceMapping.c* and *thrFiringSchmitt.c*, and will be analyzed below.

Thruster Force Mapping

The goal of the thruster mapping strategy is to find a set of thruster forces that yield $\mathbf{l_r}$. The number of controlled axes can be tuned, and will be maximum for the chosen scenario (3D control). This module can handle both ACS and DV thrusters, so the first operational mode is chosen. The assumption of the module is that the ACS configuration is capable of producing pure torques, and is coherent with the chosen one.
Starting from 3.6, and with everything in BF, it is possible to extract the mapping matrix $\mathbf{D_t}$, that maps the generic 8×1 thruster forces vector $\mathbf{f}$ to their produced 3D torque $\boldsymbol{\tau}$:

$$\boldsymbol{\tau} = \mathbf{D_t}\mathbf{f} \tag{3.14}$$

If $\boldsymbol{\tau}$ is set to be the requested torque $\mathbf{l_r}$, it is possible to invert the previous expression (i.e. performing the minimum norm inverse), and find the forces that can produce such torque:

$$\mathbf{f} = \mathbf{D_t}^{\mathbf{T}}(\mathbf{D_t}\mathbf{D_t}^{\mathbf{T}})^{-1}\mathbf{l_r} \tag{3.15}$$

The capability of 3D controllability mentioned in 3.4 has already ensured that $\mathbf{D_t}$ is of full rank. For this reason the inversion present in the previous equation is possible, being also $\mathbf{D_t}\mathbf{D_t}^{\mathbf{T}}$ of full rank. The $\mathbf{f}$ vector, however, will contain also negative values, so a smart way to eliminate them is subtracting from it another 8×1 vector whose elements are the minimum value of $\mathbf{f}$:

$$\mathbf{f_{pos}} = \mathbf{f} - \mathbf{min}(\mathbf{f}) \tag{3.16}$$

When all the thrusters are on, there is no torque applied to the spacecraft. This means that the addition of all the elements for each $\mathbf{D_t}$ row brings to a 3×3 zeros vector. The same happens if the rows are post-multiplied by a 3×3 vector of all equal values. Then, the operation $\mathbf{D_t min(f)}$ (i.e. $\mathbf{min(f)}$ belongs to the null-space of $\mathbf{D_t}$) will output a zero vector. This property is exploitable to show that also $\mathbf{f_{pos}}$ generates $\mathbf{l_r}$. If this is true then:

$$\mathbf{f_{pos}} = \mathbf{f} - \mathbf{min(f)} = \mathbf{D_t}^{\mathbf{T}}(\mathbf{D_t}\mathbf{D_t}^{\mathbf{T}})^{-1}\mathbf{l_r} \tag{3.17}$$

By pre-multiplying both sides by $\mathbf{D_t}$:

$$\mathbf{D_t}(\mathbf{f} - \mathbf{min(f)}) = \mathbf{l_r} \tag{3.18}$$

By expanding the LHS and using 3.14:

$$\mathbf{D_t}\mathbf{min(f)} = \mathbf{0} \tag{3.19}$$

Since $\mathbf{min(f)}$ does belong to the null-space of $\mathbf{D_t}$, this condition is verified, thus also $\mathbf{f_{pos}}$ generates $\mathbf{l_r}$.
Just like $\mathbf{B_t}$, $\mathbf{D_t}$ yields the 3.8 property. This means that when performing the minimum norm inverse in 3.15, the positions on the $\mathbf{f}$ vector corresponding to indices of equal columns in $\mathbf{D_t}$ will be occupied by equal values [36]. In the current studied configuration this translates to having $\mathbf{f_1} = \mathbf{f_7}; \quad \mathbf{f_2} = \mathbf{f_8}; \quad \mathbf{f_3} = \mathbf{f_5}; \quad \mathbf{f_4} = \mathbf{f_6}$. The implication of that is that equal torque-producing thrusters will always be fired together, hence no command generating net forces is created. A simple way of deriving this property its found by looking at its physical meaning. The minimum norm inverse solution operation corresponds to finding the minimum force vector that can generate $\mathbf{l_r}$. One can think of taking each same-torque thrusters couple and condense it in a single imaginary thruster with double the force and same "arm". $\mathbf{D_t}$ is then reduced to $\mathbf{D_{t_{red}}}$, a 3×4 matrix with the same columns of $\mathbf{D_t}$, but without repetitions. $\mathbf{l_r}$ will then be generated by a 4×1 force vector $\mathbf{f_{red}}$. Cause of the uniqueness of the norm inverse solution, and given the fact that firing one imaginary thruster produces the same effect as firing both the corresponding real thrusters with half the force:

$$\mathbf{f_{th_{couples_{i_1}}}} = \mathbf{f_{th_{couples_{i_2}}}} = \frac{\mathbf{f_{red_i}}}{2} \tag{3.20}$$

Hence, they will always be fired together and no net force is produced in nominal conditions.

Thruster Firing Schmitt

As mentioned before *thrusterDynamicEffector.cpp* needs to know the opening times for each thruster ($\mathbf{u_{onTime}}$) in order to produce the command $\mathbf{u_c}$. Then $\mathbf{f_{pos}}$ (from now on just $\mathbf{f}$) be properly translated through a firing law. Basilisk is provided with a module containing the well known "Firing Schmitt Law". At each control update, for each thruster, the nominal decision on-time is computed:

$$t_{on_n} = \frac{f}{f_{max}}\Delta t \tag{3.21}$$

With Δt being the time step of the module. Based on the relation between t_{on_n}, Δt and $t_{minOn_{sc}}$ (i.e. how much force will be produced across a time step, with respect to the force required), it is possible to build a logic that avoids the chattering problem. If a parameter $l = t_{on_n}/t_{minOn}$ is introduced, the logic will have this form:

$$If \quad t_{on_n} < t_{minOn_{sc}} : \quad \begin{cases} if \quad l > l_{on} \implies & t_{on} = t_{minOn_{sc}} \\ if \quad l < l_{off} \implies & t_{on} = 0 \end{cases} \tag{3.22}$$

In which t_{on} is the value that will populate the $\mathbf{u_{onTime}}$ vector. l_{off} and l_{on} are two threshold values. The first case of 3.22 will remain true as long as the second is false. The other cases are not touched by the Schmitt logic, since they are far from the shutting off condition:

$$If \quad \Delta t > t_{on_n} \geq t_{minOn_{sc}} \implies t_{on} = t_{on_n} \tag{3.23}$$

$$If \quad t_{on} > \Delta t \implies t_{on} = 1.1\Delta t \quad \text{(thruster saturation)} \tag{3.24}$$

The thresholds are usually manually tuned in order to avoid chattering, however, since the Montecarlo simulations will present a wide variety of situations, their values might not be optimal for each simulation.
The value of $t_{minOn_{sc}}$ cannot be lower that the physical thruster t_{minOn}, and the higher it is, the lower chattering effect will be present, at the expense of thrust resolution. Considering the simple de-tumbling maneuver a value of $t_{minOn} = 0.02$ is enough to grant the success of the maneuver across all the simulation campaign.

Chapter 4

Proposed Solution

In this chapter will be explained the FDIS design. First an overall look of the whole method is given. Then the FDI interaction with the other subsystems is discussed in terms of frequency interaction. Finally, the detailed design of each FDI stage is reported.

4.1 Overview of the Solution

As briefed in 1.3, the approach presented in this thesis makes use of three stages. Three are also the information needed in terms of input to the FDI subsystem (from now on FDIS):

- Linear acceleration
- Angular velocity
- Command vector

The origin and shape of those information are explained in 3. The fist two come from the IMU unit, and the third from the control subsystem, as reported in figure 4.1. It is clear from the picture the passive interaction of the FDIS with the

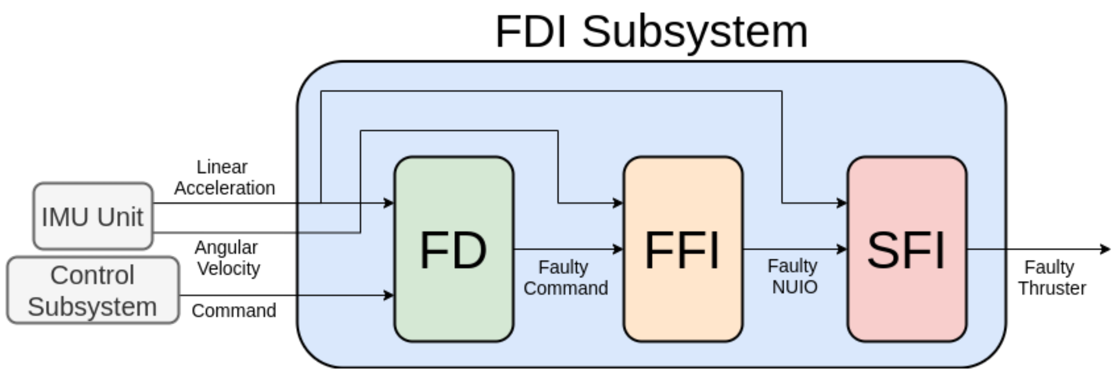

Figure 4.1: Input/output relation for the FDI subsystem

control subsystem. This means that the tool can be inserted as an algorithm in the spacecraft, without the need of changing the control law. The possible feedback

that can be done, is of the faulty thruster information back to the control, in order to either exclude it, or, in general, take decisions. This is part of the broader argument called "Fault Recovery".
With respect to the theory of FDI (1.2) the present method classifies as model-based, and in particular:

FD and SFI make use of signal monitoring for the acceleration of the spacecraft. With respect to classical model based techniques the system modeling is hidden by the fact that in nominal condition this quantity should be zero, given the ACS thruster configuration.

FFI represents a classical model based approach, given the presence of Nonlinear Unknown Input Observers.

The underlining idea is that a fault of one of the thrusters will result in a non-zero accelerometer's reading, cause of the new net force acting on the spacecraft. Only when this happens, the four NUIOs are activated and will start their fault isolation job, while at the same time feeding to SFI a couple of candidate faulty thrusters. Thanks to the design of the NUIOs, those candidates will always have opposing firing direction. This means that the net force can be in one of two directions. It is the SFI job to confront the accelerometer reading with the nominal thrust directions, and finally declare the faulty thruster. All the three stages will be wrapped up in the newborn *thrustersFDI.c* module.

4.2 FDI Subsystem Interfacing

The FDI algorithm has to be inserted in the spacecraft ecosystem, characterized by a variety of components that, in general, work at different frequencies. As stated before, three are the subsystems/components interfacing: IMU (frequency $freq_{IMU}$), FDI ($freq_{FDI}$) and control ($freq_{cont}$). Those interactions are taken into account in the design, achieving two goals:

- Perform FDI within the duration of one command time step.
- Solve the problem of mid-firing command change

First of all, a reasonable assumption to be made is that $freq_{IMU} > freq_{cont}$. Accelerometers can work at very high sampling times, in the order of kHz [16]. Such frequencies of operation are not reachable by cold gas thruster systems like the one considered here, neither are necessary for the analyzed case. In order to avoid the creation of false positives, the FDIS has to have a certain *time window* to declare the faulty thruster. This time window will reflect in a design parameter $t_{FDI} = 1/freq_{FDI}$, representing the FDIS time step duration. The choice is to have $freq_{IMU} > freq_{FDI} > freq_{cont}$. In this way, the fault can be successfully isolated within the time window of one command update. This is reasonable, since $freq_{FDI}$ is part of the flight software algorithms of the OBC, and does not depend on other physical component that could limit this operating frequency.

Another problem, is the "mid-isolation" command change. This can manifest when a command changes during the isolation window, possibly modifying the faulty thruster state. This condition is illustrated in figure 4.2. In the figure the

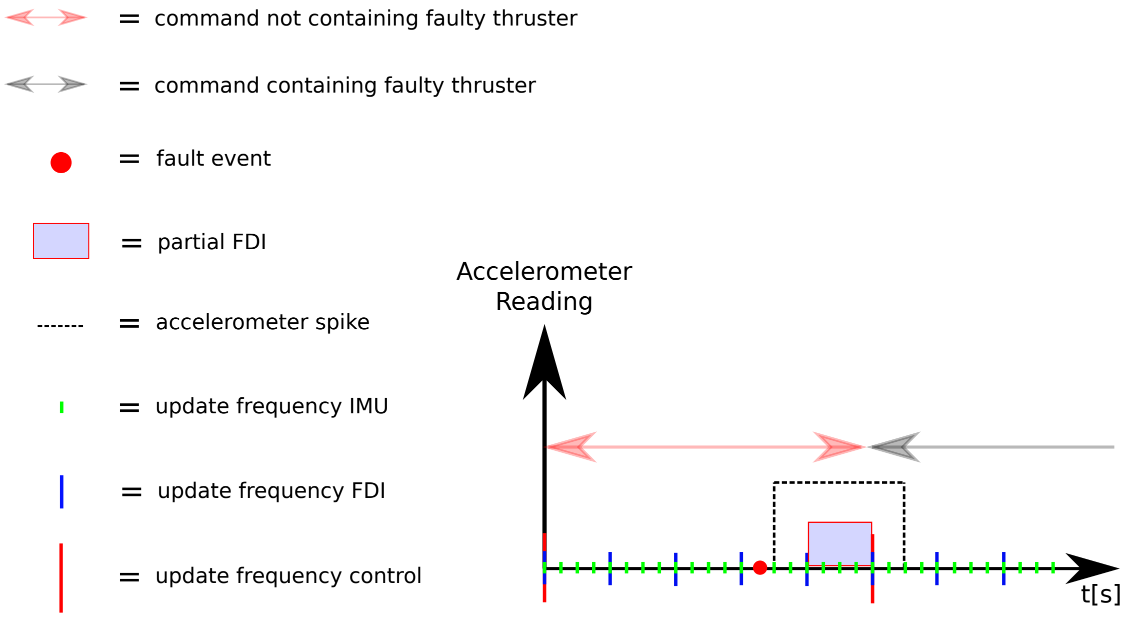

Figure 4.2: Representation of a partial FDI

colored segments on the continuous time line correspond to the update points of the different subsystems, and are spaced of $1/freq_{subsystem}$ seconds. Every command, accelerometer and FDI update will happen respectively at each red, green and blue line, and with the dashed black one representing the accelerometer's reading as it picks up the acceleration coming from a fault. It is hypothesized in these illustrations that a trustworthy FDI time window corresponds to a time $t_{FDI_{window}} = k_{FDI}t_{FDI} = k_{FDI}/freq_{FDI}$ with $k_{FDI} = 3$. The first command generates a faulty behavior, and will thus cause a spike in the accelerometer. However, at a time $t < t_{FDI_{window}}$, the command changes and the faulty thruster is no longer showing its faulty behavior. This means that, depending on the kind of fault, it might no longer induce an effect on the spacecraft to be detected by the IMU. Proceeding with the FDI would result in a wrong isolation, cause a non faulty input would enter FFI and SFI.
It has already been mentioned that different fault-kind/command combinations generate different effects on the s/c. In order to illustrate that, four possible cases are depicted in picture 4.3. It can be noticed how in all cases the effect of a command change gets picked up by the accelerometer after two of its time steps. This is due to the way in which the acceleration signal is generated (3.3).

Case A This condition can only be possible if, despite its fault, F_{TH} is not showing faulty behavior. For this reason this situation can happen:

- In an efficiency loss fault, if the command is not instructing F_{TH} to stay on.
- If F_{TH} is stuck on-at-max-efficiency and commanded to stay on, or stuck off and commanded to stay off.

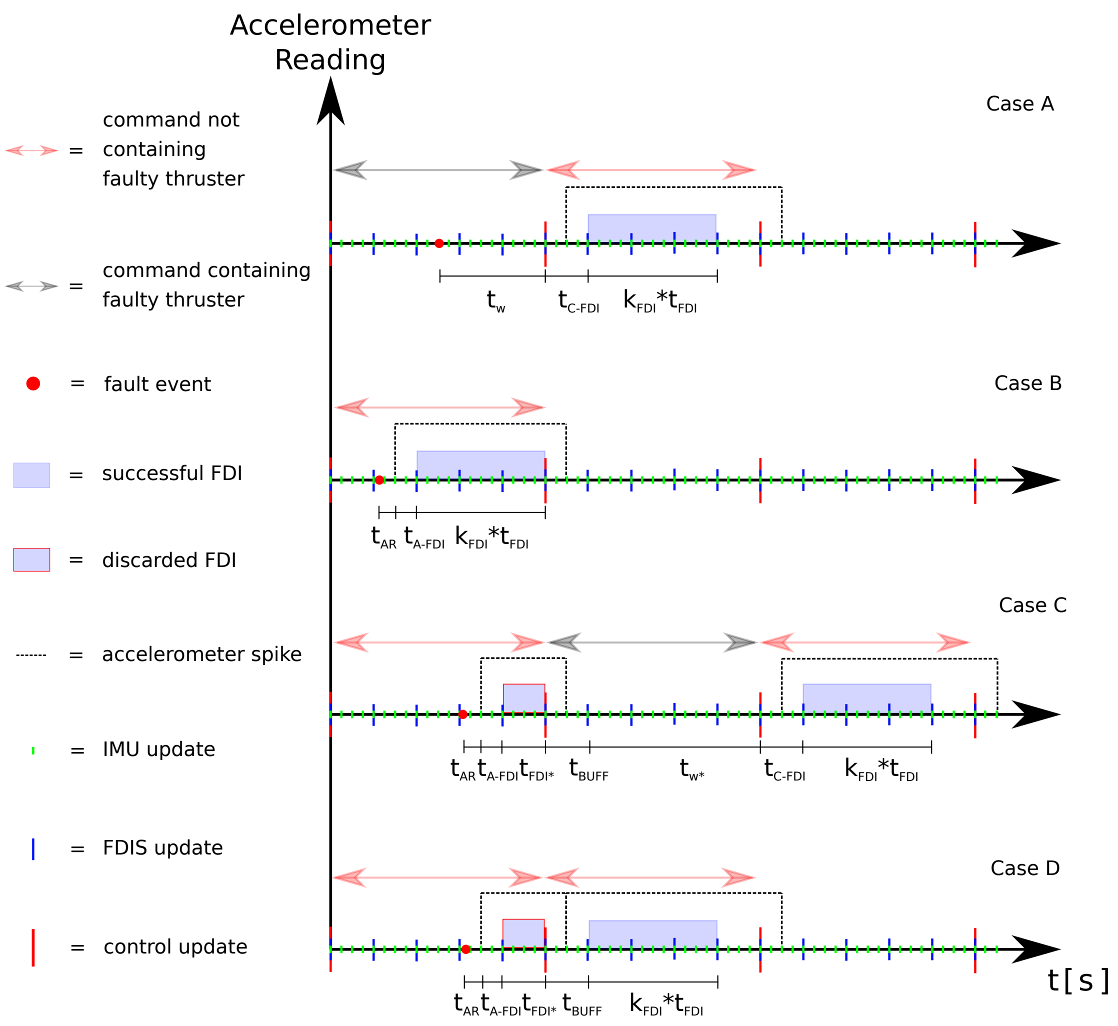
Accelerometer Reading
command not containing faulty thruster
command containing faulty thruster
fault event
successful FDI
discarded FDI
accelerometer spike
IMU update
FDIS update
control update
Case A
t_w $t_{C\text{-}FDI}$ $k_{FDI}*t_{FDI}$
Case B
t_{AR} $t_{A\text{-}FDI}$ $k_{FDI}*t_{FDI}$
Case C
t_{AR} $t_{A\text{-}FDI}$ t_{FDI*} t_{BUFF} t_{w*} $t_{C\text{-}FDI}$ $k_{FDI}*t_{FDI}$
Case D
t_{AR} $t_{A\text{-}FDI}$ t_{FDI*} t_{BUFF} $k_{FDI}*t_{FDI}$
t[s]

Figure 4.3: Representation of the four cases of FDI

After the fault, a time t_w has to pass, that depends on when F_{TH} will be commanded to change its state. Starting from that event, after a time t_{C-FDI} the FDI will pick up the accelerometer reading and start. After $k_{FDI}t_{FDI}$ seconds, the fault is isolated.

Case B This is the best case scenario, in which the fault instigates the faulty behavior and there is enough time for the FDI to end its task before an eventual command change. This conditions can happen:

- In an efficiency loss fault, if the command instructs F_{TH} to fire.
- If F_{TH} is stuck on-at-max-efficiency but commanded to shut off, or stuck off but commanded to stay on.
- Whenever the thruster is stuck on with a sub-nominal max force.

Differently from Case A, the time before FDI start is going to depend on t_{AR} which is the "raising time" of the accelerometer (i.e. when it picks up the fault disturbance), and t_{A-FDI}, at which the FDI starts. This waiting is going to be dependent on $freq_{IMU}$ and $freq_{FDI}$ but will always be less than $2t_{FDI}$.

Case C As mentioned before, it might happen that during $t_{FDI_{window}}$ the control subsystem instructs F_{TH} to change its state. The contemplated fault-kinds/command combinations are:

- Efficiency loss fault, if the command instructs F_{TH} to fire and then shut off.
- Thruster stuck on-at-max-efficiency but commanded to shut off, or stuck off but commanded to stay off. At the next command step, however the commanded F_{TH} changes.

To avoid false positives, it is decided to discard every FDI during which a command change happens while $t_{FDI_{window}}$ is not yet fulfilled. The discarded time is going to depend on how many FDIS updates were done before the cutting (t_{FDI*}). After t_{FDI*}, a time t_{BUFF} is waited, consisting of one FDI time step. In addition, a time t_{w*} will be spent before a new faulty command will be introduced.

Case D This is similar to Case C, however the command changes to another one inducing F_{TH}'s faulty behavior. The contemplated fault-kinds/command combinations are:

- Efficiency loss fault, if F_{TH} is instructed to fire for two commands time steps.
- Thruster stuck on-at-max-efficiency but commanded to shut off, or stuck off but commanded to stay on. At the next command step the commanded F_{TH} state is the same.
- Whenever the thruster is stuck on, with a sub-nominal max force.

It is important to notice that the fault is considered unknown until FDI is reached, so the cut is done whenever the command change is detected, that would mean a change in the NUIOs input. The cut will lead to a buffer time t_{buff} equal to one FDIS update ($freq_{FDI}$), before starting another FDI.

It is clear that a smaller k_{FDI} will lead to a lesser probability of FDI cutoff.
By analyzing the four cases, the following conclusions can be drawn:

- With efficiency-loss, stuck-on-at-maximum-efficiency and stuck-off kinds of faults, all the previous cases are possible, leading to the presence of waiting times.

- If the thruster is stuck *on* at a fraction of the max thrust only cases B and D are possible, cause no matter the command, F_{TH} is going to fire differently than expected. An eventual mid-isolation command change will indeed interrupt the FDI, that will however always restart with the new command, without waiting times. Cause of that the overall isolation time for a thruster stuck fault is statistically going to be smaller than the one for the efficiency loss case. This will be confirmed in simulations.

4.3 Fault Detection

Fault detection is the first of the three steps, and serves as a triggering point for the FDI chain of events. The underlining idea is to confront the nominal value of the linear acceleration with the one coming from the IMU, and declaring a fault when their difference becomes greater than a threshold value FD_{trs}. When generically using the term "acceleration", it is intended the vector norm considering its three components in BF coordinates.
In 3.4, the configuration of the ACS thruster system is discussed, laying down the concept of "pure torque", that is the basis of the employed fault detection algorithm. This means that there is no need for a mathematical model of the *nominal acceleration*, that will always be zero. FD_{trs} will then simply correspond to the maximum allowed acceleration before declaring fault. The off-nominal conditions are then:

- A fault in one of the thrusters.
- An unmodeled external force acting on the spacecraft.

By a sensed-acceleration stand point, both cases generate a reading of the accelerometer. In the first case, despite the ACS trying to create pure torques, a force unbalance is created by the difference between the healthy and faulty thruster in one of the $\mathbf{th_{couples}}$. An example is reported in figure 4.4. The thruster 1 is stuck off at 60s, while it was firing. The first two figures are the acceleration readings, in terms of components and vector norm, while the third represents the fault detection. The effect of the fault is clearly visible, and the magnitude of the acceleration is correctly displayed at 0.001 m/s. After the fault is introduced, in fact, only thruster 7 out of $\mathbf{th_{couples_1}}$ will be firing, causing an acceleration of $acc = f_{max}/m = 0.1N/100Kg = 0.001m/s$.
The introduction of unmodeled forces on the spacecraft will inevitably have an effect on the reading, potentially causing problems for the FD. In this sense the tuning of FD_{trs} will result in a trade-off between detection sensitivity to small faults and robustness to external forces. This topic is addressed in 5.3. An FD fault declaration will trigger the FFI.

4.4 First Fault Isolation

Once a fault has been declared by FD, the FFI starts. The theory discussed in 2.3 will be applied and tailored to this instance of fault isolation. The general nonlinear system in 2.7 will then be based on the ADEs(time dependencies have been omitted for brevity):

$$\dot{\boldsymbol{\omega}} = -\mathbf{J^{-1}}\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \mathbf{J^{-1}}\mathbf{m_{ext}} \tag{4.1}$$

$$\mathbf{y} = \boldsymbol{\omega} \tag{4.2}$$

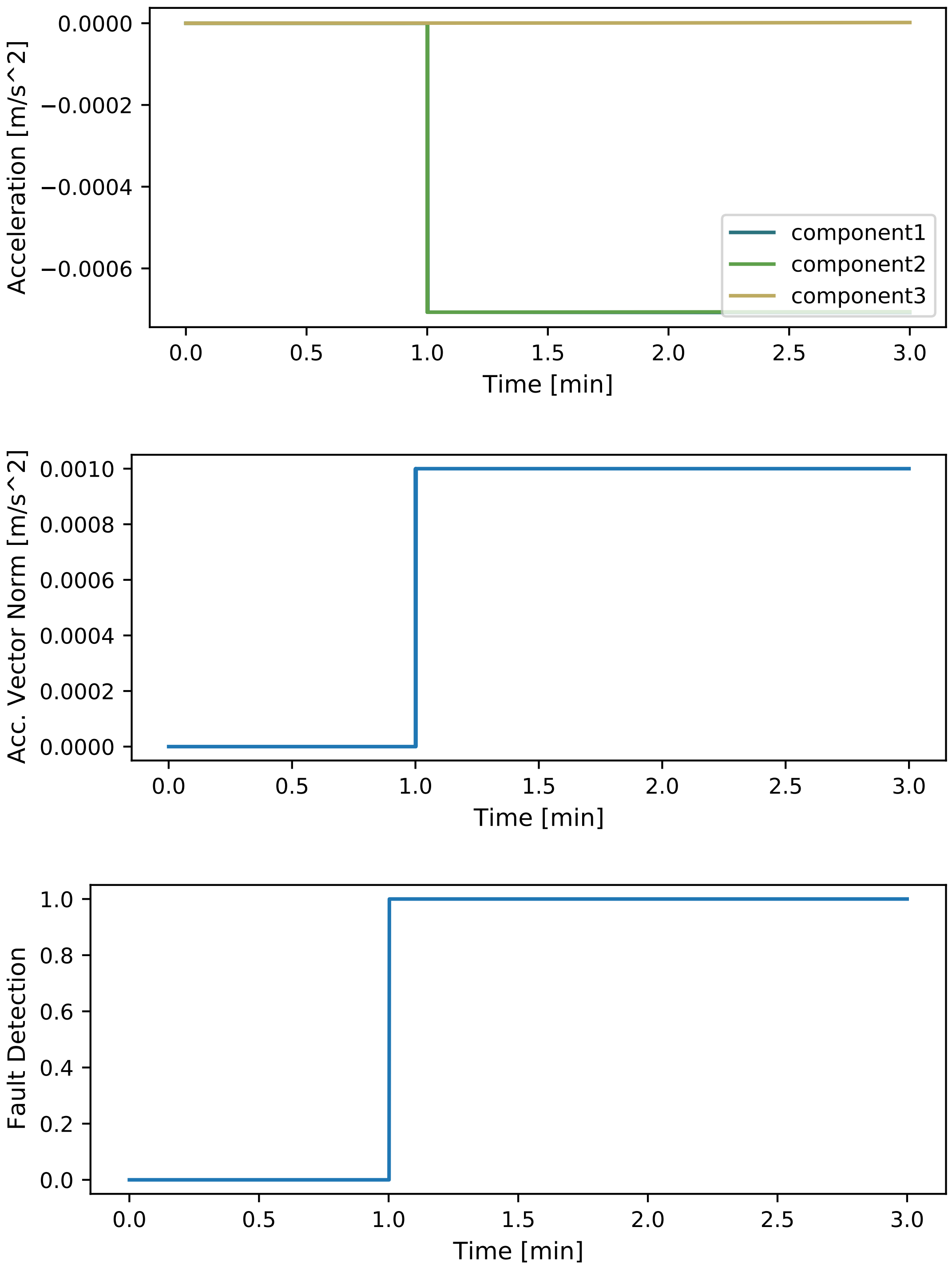

Figure 4.4: Acceleration readings and consequent fault detection

With respect to 2.7 the following substitutions have been made:

$$\mathbf{x} = \boldsymbol{\omega} \tag{4.3}$$
$$\mathbf{A} = \mathbf{0} \tag{4.4}$$
$$\mathbf{f}(\mathbf{x}) = -\mathbf{J}^{-1}\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} \tag{4.5}$$
$$\mathbf{C} = \mathbf{I} \tag{4.6}$$

The most important decision now comes for the choice of the known and unknown inputs. In 3.5 it has been established that $\mathbf{m_{ext}} = \mathbf{B_t u_c}$. This means that in order to exploit the isolation capability of the NUIOs, each one of them will treat different parts of $\mathbf{B_t u_c}$ as unknown inputs ($\mathbf{B_{n_i} u_{n_i}}$). Theoretically there is no unique way to choose the $\mathbf{B_{n_i}}$, but it is mandatory for the assumptions 2.3 to be fulfilled, in order create functioning fault isolating observers. Before checking those, a consideration is done.
The $\mathbf{B_t u_c}$ product is a torque, so it means that ADEs are susceptible to faults in terms of torques. Consequently, a fault produced by thrusters belonging to the same $\mathbf{f_{th_{couples}}}$ will have the same effect torque-wise. Since the NUIOs are based on ADEs, once one of the thrusters is faulty, it will impossible to isolate it from the other belonging to the same $\mathbf{f_{th_{couples_i}}}$. Spreading them as unknown inputs on different NUIOs is not a good idea cause when a fault happens, two of them will always be triggered, creating ambiguity. A reasonable choice is then to have thrusters belonging to the same $\mathbf{f_{th_{couples}}}$ as unknown inputs of the same NUIO. This solves the problem of multiple triggering and four NUIOs, each one handling a $\mathbf{f_{th_{couples}}}$, can be designed. A fault would trigger just the NUIO containing the faulty thruster as UI. However, it is still unable (and will never be able, cause of the previous considerations) to isolate between the two thrusters it is handling. The second fault isolation step is then justified, that will handle this last problem.
Based on the previous considerations, and without any loss of performance, it is decided to use a reduced torque matrix $\mathbf{B_{t_{red}}}$, just like in 3.5.3. Using the same analogy, four pure torque sources (PTS) are hypothesized, capable of producing double the torque of a single thruster. By using the real numerical values of the thrusters:

$$\mathbf{B_{t_{red}}} = \begin{bmatrix} 0.0424 & 0.0424 & -0.0424 & -0.0424 \\ -0.0424 & 0.0424 & 0.0424 & -0.0424 \\ -0.0566 & 0.0566 & -0.0566 & 0.0566 \end{bmatrix}$$

So the remaining matrices to define for each NUIO are $\mathbf{B_{n_i}}$ and $\bar{\mathbf{B}}_{\mathbf{n_i}}$. In order to ensure that each NUIO is sensitive just to a fault in one of the PTS, the $\mathbf{B_{n_i}}$ matrices (remembering that those are related with the unknown inputs) correspond, for each NUIO, to a column of $\mathbf{B_{t_{red}}}$, and with $\bar{\mathbf{B}}_{\mathbf{n_i}}$ being the remaining columns. These design choices result in:

- NUIO 1 will be sensitive to faults in $th_{couples_1}$.
- NUIO 2 will be sensitive to faults in $th_{couples_2}$.
- NUIO 3 will be sensitive to faults in $th_{couples_3}$.
- NUIO 4 will be sensitive to faults in $th_{couples_4}$.

Going back to the assumptions 2.3, it can be seen that while the first is easily verified by looking at the ranks of the $\mathbf{B_n}$ matrices, the second one requires another assumption. The condition was:

$$||\mathbf{f}(\mathbf{x}) - \mathbf{f}(\hat{\mathbf{x}})|| \leq \gamma||\mathbf{x} - \hat{\mathbf{x}}|| \tag{4.7}$$

By substituting the values for this specific case:

$$|| - \mathbf{J}^{-\mathbf{1}}\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \mathbf{J}^{-\mathbf{1}}\hat{\boldsymbol{\omega}} \times \mathbf{J}\hat{\boldsymbol{\omega}}|| \leq \gamma||\boldsymbol{\omega} - \hat{\boldsymbol{\omega}}|| \tag{4.8}$$

Even if the Jacobian of $\mathbf{f}(\mathbf{x}) = -\mathbf{J}^{-\mathbf{1}}\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}$ is not globally bounded, it continuously differentiable in the 3D space. This means that $\mathbf{f}(\mathbf{x})$ can be locally Lipschitz, given that the $\mathbf{x}$ (i.e. the angular velocity) is bounded in magnitude. In literature, the commonly used procedure for finding γ is referred in [12] and consist in a constraint optimization. In this thesis a large number of Montecarlo simulations will be performed, and the values of the angular velocity will vary significantly among different runs. This means that the approach of constraint optimization over the maximum $\boldsymbol{\omega}$ used in [12] would require an optimal evaluation for each run, considering each time the different initial conditions. Furthermore, because in a de-tumbling maneuver the maximum angular velocity is reached just at the first instants and tends to be near zero afterwards, the aforementioned method would result in an overly conservative value for γ. Thanks to the Montecarlo simulations, however, it has been possible to find a value that guarantees the correct NUIOs behavior in all cases even if it's not an optimal one. The value used throughout all the simulations will be $\gamma = 2 \cdot 10^{-6}$.
Now that all the matrices of the nonlinear system are characterized, the design of the observer can continue. The matrices $\mathbf{U}$ and $\mathbf{V}$ in 2.27 can be defined and will be part of the LMI problem. Its shape will be the same as 2.3. With the new definition of the system matrices, $\mathbf{X_{11}}$ and $\mathbf{X_{12}}$ will simplify to:

$$\mathbf{X_{11}} = -\bar{\mathbf{K}}^T - \bar{\mathbf{K}} + \gamma\mathbf{I} \tag{4.9}$$
$$\mathbf{X_{12}} = \sqrt{\gamma}[\mathbf{P}(\mathbf{I} + \mathbf{U}) + \bar{\mathbf{Y}}\mathbf{V}] \tag{4.10}$$

As mentioned above, in order to solve the LMI problem, the Matlab LMI toolbox is used [14]. This represents the standard way to solve it cause of its ease of usage with respect to an otherwise difficult problem. Its employment is widespread in literature. For this particular application a Matlab script has been created, that first creates $\mathbf{B_{t_{red}}}$ as described before. Than, for each NUIO the steps are:

1) $\bar{\mathbf{B}}_\mathbf{n}$ and $\mathbf{B_n}$ are defined.

2) $\mathbf{U}$ and $\mathbf{V}$ are found.

3) $\mathbf{P}$, $\bar{\mathbf{Y}}$ and $\mathbf{K_n}$ are set as variables with the *lmivar* command. $\mathbf{P}$ is set to be a symmetric matrix 2.18 while the other two are generic matrices.

4) The system is going to be composed of two LMIs, whose components are set with the *lmiterm* command. The first inequality is the positive definiteness of $\mathbf{P}$ ($\mathbf{P} > 0$). The second is the one with 4.9 and 4.10.

5) With the command *getlmis* and *feasp* the LMIs are solved.

6) The values for the solution matrices $\mathbf{P}$, $\bar{\mathbf{Y}}$ and $\mathbf{K_n}$ are recovered from the solution made of decision variables, using the command *dec2mat*.

7) $\mathbf{Y_n}$ and $\mathbf{K_n}$ are computed using the solution matrices (2.3, 2.3).

8) The matrices $\mathbf{E_n}$, $\mathbf{N_n}$, $\bar{\mathbf{G}}_\mathbf{n}$, $\mathbf{L_n}$ and $\mathbf{M_n}$ are computed using 2.26, 2.12~2.15.

The last computed matrices will be part of the inputs of *thrustersFDI.c* and, being only dependent from the thrusters and s/c characteristics, they are defined just once, offline.
Once designed the NUIOs matrices, the 2.10 system is defined for each of the four. In the practical application two things have to be noticed:

1) The vector $\mathbf{y}$, equal for each NUIO, is none other than the angular velocity coming from the IMU.

2) The vector $\bar{\mathbf{u}}_\mathbf{n}^\mathbf{h}$ is the "healthy" input. Given the reduced four imaginary thruster assumption, also the command vector coming from the control will have to be reduced. By looking at 3.5 it is clear that coming out from *thrFiringSchmitt.c* will be a vector of on-times. At each step, it will be transformed by the FDIS in the boolean command vector $\mathbf{u_c}$ by simply checking which entries are different than zero. Those will be the *1* values, since they represent a "on" command. Thanks to *thrForceMapping.c*, thrusters belonging to the same $\mathbf{th_{couples}}$ will always have the same command. In this way, for each NUIO, $\bar{\mathbf{u}}_\mathbf{n}^\mathbf{h}$ will be a 3×1 vector, whose components are the three commands targeting the PTSs considered as input, hence discarding the one referring to the PTS treated as unknown.

All the terms are defined and the integration can be performed. An Euler integration scheme is used, so for each step k the new states:

$$\mathbf{z_{n_{k+1}}} = \Delta t \mathbf{z_{n_k}} \tag{4.11}$$

The estimated angular velocity will be:

$$\hat{\mathbf{x}}_\mathbf{n_k} = \mathbf{z_{n_k}} - \mathbf{E_n y_k} \tag{4.12}$$

By taking the difference between the estimated and IMU-sensed angular velocity, the residuals can be found and their Euclidean norm is evaluated:

$$\mathbf{res_{n_k}} = \hat{\mathbf{x}}_\mathbf{n_k} - \mathbf{y_k} \tag{4.13}$$
$$resNorm_{n_k} = norm(\mathbf{res_{n_k}}) \tag{4.14}$$

At each isolation step, the smallest $resNorm_{n_k}$ is found. Thanks to the properties of the NUIOs, the observer producing the smallest $resNorm_{n_k}$ will be the one having a fault in the PST considered unknown input. In the analyzed configuration:

- If $min(resNorm_{n_k}) = resNorm_{n1_k} \implies$ fault in $\mathbf{th_{couples_1}} = 1, 7$.

- If $min(resNorm_{n_k}) = resNorm_{n2_k} \implies$ fault in $\mathbf{th_{couples_2}} = 2, 8$.
- If $min(resNorm_{n_k}) = resNorm_{n3_k} \implies$ fault in $\mathbf{th_{couples_3}} = 3, 5$.
- If $min(resNorm_{n_k}) = resNorm_{n4_k} \implies$ fault in $\mathbf{th_{couples_4}} = 4, 6$.

An example of NUIOs isolation, with the $resNorm_n$ behaviors, can be seen in pictures 4.5 and 4.6. In both the situations it is introduced a stuck-off fault in thruster 2. The first one represent the output of the FFI during its time window. The estimation error arising from NUIOs 1,2,4 means that a fault happened in either thruster 2 or 8. This is confirmed by their numerical values at the end of the isolation: $resNorm_{n1} = 9.51e-5$, $resNorm_{n2} = 2.73e-9$, $resNorm_{n3} = 8.71e-5$, $resNorm_{n4} = 9.52e-5$.
In 4.6 the command that caused the fault has been blocked, in order to show the behavior of the NUIOs not limited to the FDI time window. By looking at the values of the residuals it is clear that in a time span of one FDI window, the transitory behavior is not finished. However, since the goal is just to find which residual grows the least after a fault, this doesn't pose a problem for FDI. The designed NUIOs are then responsive enough to accomplish this goal. Further comments on the FFI behavior is reported in the simulation campaign.
It can be noticed the absence of transitory in the faulty NUIO, whose signal remains very close to zero even during the first instants after the fault injection. This result is achieved by initiating the FFI with the real angular velocity values, coming from the sensor. This means that at time zero:

$$\mathbf{\hat{x}_{n_0}} = \mathbf{y_0} \tag{4.15}$$

$$\mathbf{z_{n_0}} = \mathbf{E_n y_0} + \mathbf{\hat{x}_{n_0}} = (\mathbf{E_n} + \mathbf{I})\mathbf{y_0} \tag{4.16}$$

The information on the two suspect faulty thrusters is now passed to the SFI.

4.5 Second Fault Isolation

Now that the fault is isolated to one of the two thrusters in $\mathbf{th_{couples}}$, the final step can take place. For this job, the accelerometer information is retrieve again and used. Thanks to the subdivision of the NUIOs, every PST is made of opposing firing thrusters, so once one PST is isolated, the two suspect faulty thrusters will have opposing force directions. This means that by analyzing the direction of the induced net force, and confronting it with the nominal thrusters force, the final isolation can be achieved. In particular two different cases can happen, depending on the command that is being instructed by the control subsystem to the two suspect thrusters (from now on th_{susp_1} and th_{susp_2}). They are depicted in figure 4.7 and correspond to:

Case a) The thrusters are instructed to stay on. In this case the healthy one will generate the nominal f_{max} while the faulty one will not. The net force then will be in the direction of the healthy, declaring then the other one faulty.

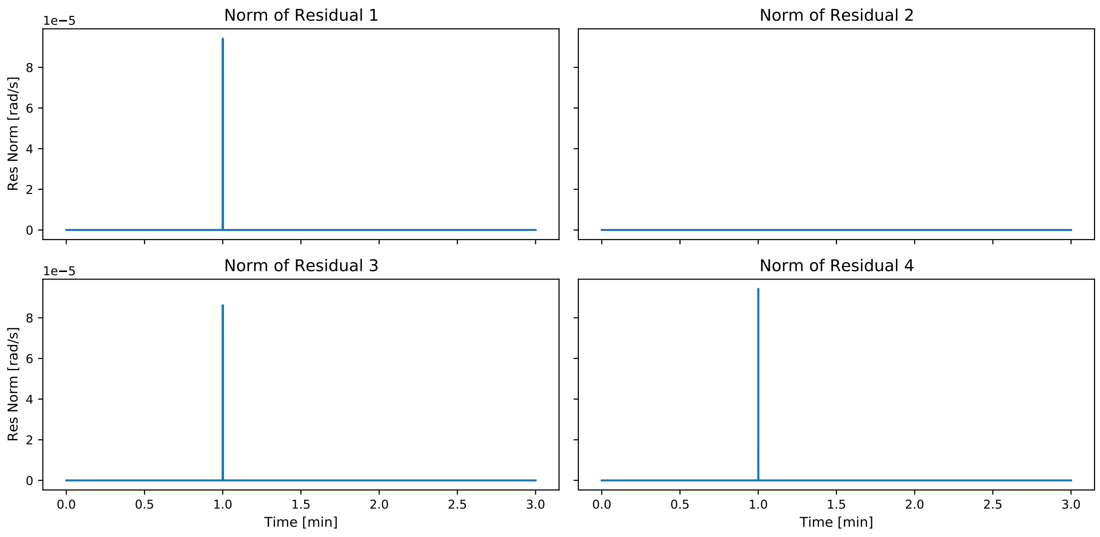

Figure 4.5: Norm of the NUIOs residuals

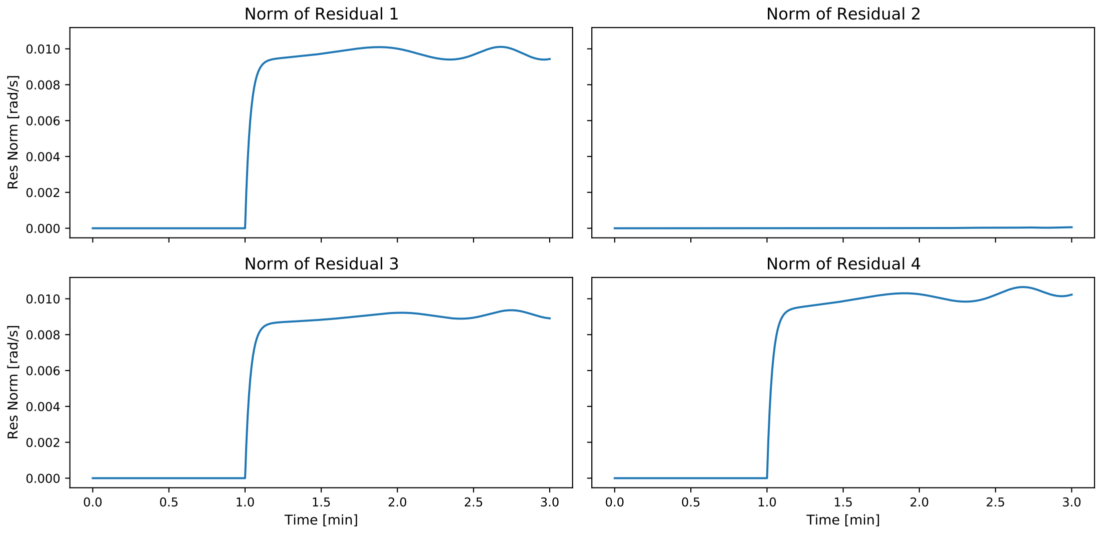

Figure 4.6: Norm of the NUIOs residuals when the faulty command is blocked

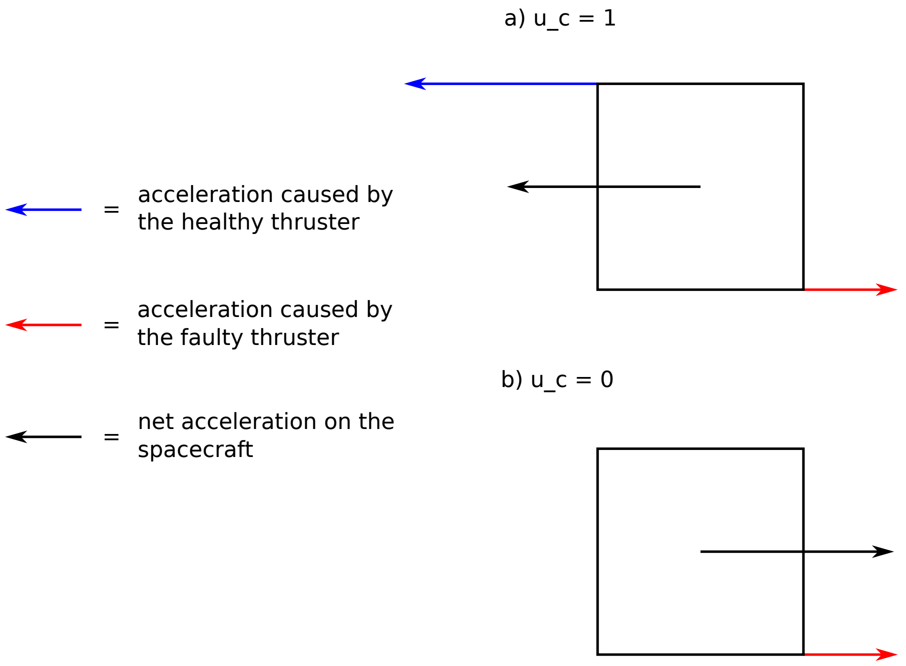
a) u_c = 1
= acceleration caused by the healthy thruster
= acceleration caused by the faulty thruster
= net acceleration on the spacecraft
b) u_c = 0

Figure 4.7: Visualization of the SFI principle

Case b) If the the instructed command is off, however, the situation is the opposite. Since the healthy one will not be firing, the generated net force will be in the direction of the faulty.

Both the thruster forces directions and accelerometer measurements are in BF, thanks to the placement of the IMU, so no coordinate change is needed to confront those two quantities. The angle between those two vectors, at each isolation step, for both the isolated thrusters, is:

$$\alpha_{th_{susp_1}} = \cos\left(\frac{\mathbf{\hat{g}_{f_{th_{susp_1}}}} \cdot \mathbf{\ddot{r}_{sensed}}}{||\mathbf{\hat{g}_{f_{th_{susp_1}}}}|| \cdot ||\mathbf{\ddot{r}_{sensed}}||}\right)^{-1} \tag{4.17}$$

$$\alpha_{th_{susp_2}} = \cos\left(\frac{\mathbf{\hat{g}_{f_{th_{susp_2}}}} \cdot \mathbf{\ddot{r}_{sensed}}}{||\mathbf{\hat{g}_{f_{th_{susp_2}}}}|| \cdot ||\mathbf{\ddot{r}_{sensed}}||}\right)^{-1} \tag{4.18}$$

So, with the considerations done before, the following conditions are found:

$$If \quad u_{c_{th_{susp_{1,2}}}} = 1: \quad \begin{cases} if \quad \alpha_{th_{susp_1}} > \alpha_{th_{susp_2}} & \Longrightarrow \quad th_{susp_1} \text{ is faulty} \\ if \quad \alpha_{th_{susp_1}} < \alpha_{th_{susp_2}} & \Longrightarrow \quad th_{susp_2} \text{ is faulty} \end{cases} \tag{4.19}$$

$$If \quad u_{c_{th_{susp_{1,2}}}} = 0: \quad \begin{cases} if \quad \alpha_{th_{susp_1}} > \alpha_{th_{susp_2}} & \Longrightarrow \quad th_{susp_2} \text{ is faulty} \\ if \quad \alpha_{th_{susp_1}} < \alpha_{th_{susp_2}} & \Longrightarrow \quad th_{susp_1} \text{ is faulty} \end{cases} \tag{4.20}$$

The final isolation is achieved, and the predicted faulty thruster $FT_{thr_{pred}}$ has been isolated.

4.6 Implementation

The coding side of this work has been performed in three main languages: Matlab, Python and C.
Matlab has been used to solve the LMI problem, and it has been explained how in 4.5. Its usage its limited in one script that has to be run in order to find the NUIOs design matrices.
The scenario for the de-tumbling has been created in the Basilisk's Python interface, and consists in tying together all the modules involved, defining their input values based on the mission specifications, and connect them with the messaging interface. The NUIO design matrices are imported from Matlab, and become part of the FDI module input. A visual representation of the scenario and its modules' interfacing is provided in 4.9, where the modules with different colors are set to have different time steps.
The FDI tool has been coded in C. Most of Basilisk FSW modules are programmed in this language, although lately more C++ based are being added. The interfacing with the scenario is done by Basilisk via SWIG. The flowchart in 4.8 shows the operations and decisions coded in the module.

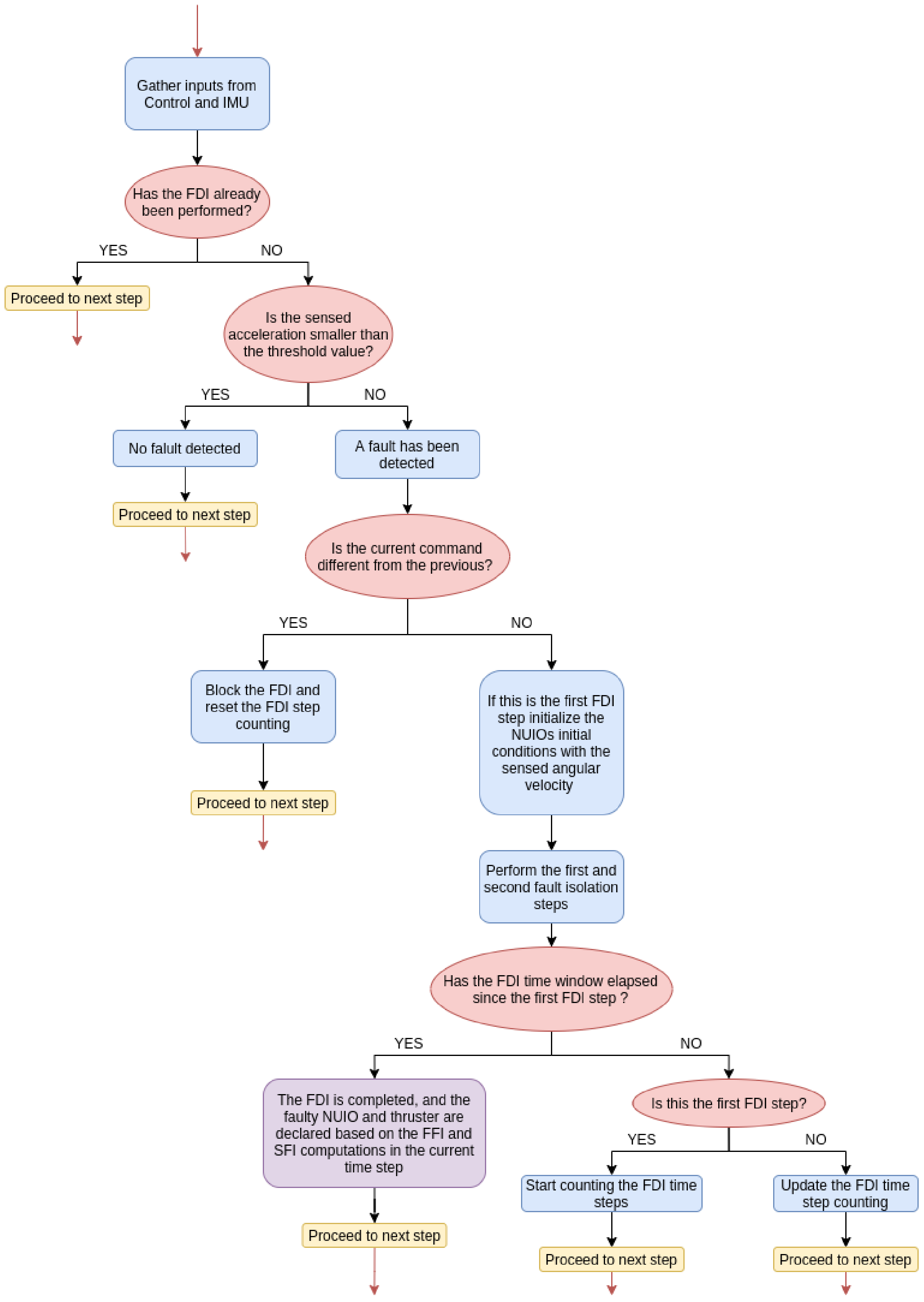
Gather inputs from Control and IMU
Has the FDI already been performed?
YES
NO
Proceed to next step
Is the sensed acceleration smaller than the threshold value?
YES
NO
No falult detected
A fault has been detected
Proceed to next step
Is the current command different from the previous?
YES
NO
Block the FDI and reset the FDI step counting
If this is the first FDI step initialize the NUIOs initial conditions with the sensed angular velocity
Proceed to next step
Perform the first and second fault isolation steps
Has the FDI time window elapsed since the first FDI step ?
YES
NO
The FDI is completed, and the faulty NUIO and thruster are declared based on the FFI and SFI computations in the current time step
Is this the first FDI step?
YES
NO
Start counting the FDI time steps
Update the FDI time step counting
Proceed to next step
Proceed to next step
Proceed to next step

Figure 4.8: Visualization of the thrustersFDI module flowchart

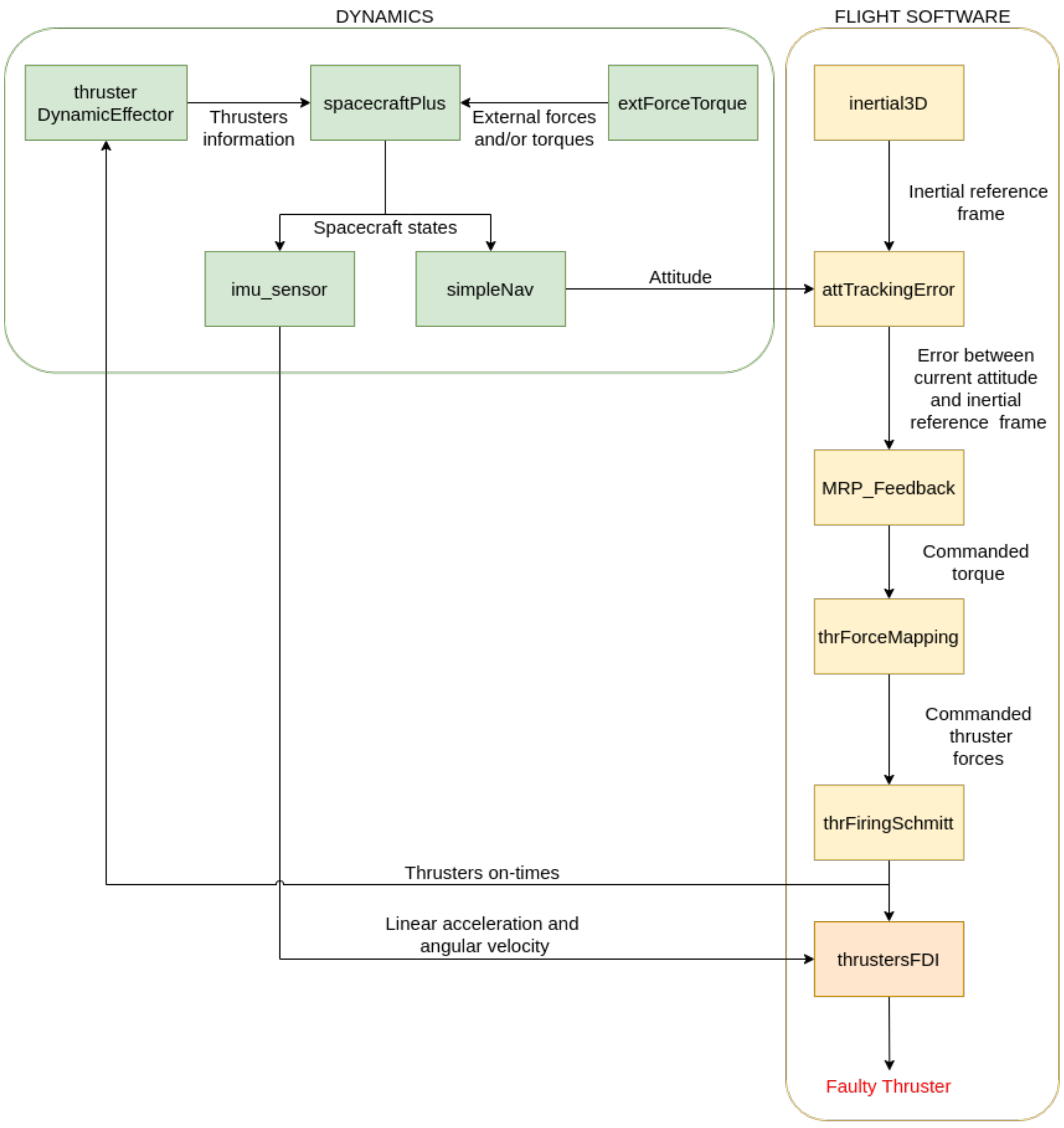

Figure 4.9: Visualization of the Basilisk simulation scenario

Chapter 5

Simulation and Results

In this chapter the validity and performances of the FDI have been assessed via a series of Montecarlo simulations (MCS), capable of accounting for all sorts of parameters variation. This tool is particularly useful when there is a large number of variables combination to be explored. This is the case for the current application, that is required to work in case of such unexpected events like faults. The simulation variables are explained and the results are reported and commented.

5.1 Simulation Variables

For the process of "fault injection" four are the identified variables and must be explored thoroughly:

1) The faulty thruster FT_{thr}.

2) The fault kind FT_{kind}.

3) The fault magnitude FT_{mag}.

4) The fault time FT_{time}.

The FT_{thr} variable represents which thruster is going to be faulty, and is going to be a number between 0 and 8. Note that the zero will represent the "no fault" condition, and will override all the other parameters, making them useless.
The fault kind FT_{kind} is representative of the two fault modes explored here:

- $FT_{kind} = 0$ refers to the condition of efficiency loss.
- $FT_{kind} = 1$ simulates a thruster stuck condition.

As mentioned in 3.4 there is also going to be a magnitude of the fault, and is going to be represented by FT_{mag}. It has been decided for FT_{mag} to assume values from 0 to 1, with 0.05 increments, thus representing 5% increments from 0 to max thrust. In relation to FT_{kind}, this translates in:

- If $FT_{kind} = 0$, the FT_{thr} will have a new maximum force $f_{max_{new}} = f_{max}FT_{mag}$.

- If $FT_{kind} = 1$, the FT_{thr} will be stuck and firing with a constant force $f_{max_{new}} = f_{max}FT_{mag}$.

Finally, the fault event will be injected at different times, depending on the FT_{time} parameter. It will be always kept as a random value within the whole simulation time.
In addition to those, four other variables will be included in the simulation:

- The initial MRPs will vary, letting the spacecraft assume any random initial attitude.
- The initial angular velocity is set to vary. Each axis is allowed to have a value between -0.3 and $+0.3$ rad/s.
- The force and toque disturbances, described later.

All the variables will have a uniform distribution dispersion. The FT_{time}, initial angular velocity, and initial MRP will vary in each MCS, thus their inclusion in the simulation inputs will be implied.

5.2 Simulation Fixed Parameters

In addition to the quantities that change between different runs, there are some pre-chosen simulation parameters. Some of them (e.g the spacecraft properties, the thrusters characteristics, etc.) have been already defined, while the other significant ones will be addressed here.
The time steps are chosen according to the frequencies discussion in 4.2:

- The IMU time step tsp_{IMU} is equal to 0.005s.
- The FDI time step tsp_{FDI} is equal to 0.01s.
- The control system time step tsp_{cont} is equal to 0.1s.

The tsp_{IMU} will be also the time step for the dynamics since there no higher frequencies phenomena of interest.
The gains for the control subsystem have been manually tuned in order to grant a more or less efficient de-tumbling, having in mind the wide variety of initial attitude conditions. Their fine tuning is more related to the control than the FDIS analyzed here.
The value for the fault detection threshold FD_{trs} is an important parameter, and its tuning is going to depend on:

- The nominal force of the thruster.
- The minimum detectable fault.
- The external forces.

Its tuning will be discussed in the simulation campaign.
The $t_{FDI_{window}}$ will be composed of 3 FDIS updates, that grant a trustworthy FDI while having a low probability of mid-isolation command change. Even if this last event is contemplated and solved by the FDI, it is still better to reduce its occurrence probability to have an overall faster FDI. The simulated time will be of 3 minutes.

5.3 Simulation Campaign

The simulation campaign will cover variations for all the previously discussed variables. In order to account for the differences in FDI behavior with respect to FT_{kind}, the two fault modes will be investigated separately. Furthermore, a separate analysis will be made for the cases considering external disturbances. Each Montecarlo simulation will produce the superimposed plots of useful variables, as well as log files containing the performances of the FDI in terms of:

- Success percentage FDI_{succ} found by comparing the injected FT_{thr} with the predicted $FT_{thr_{pred}}$.
- Fault times injected FT_{time}, fault times predicted $FT_{time_{pred}}$, and their difference. This last quantity is averaged on the successful runs, giving the mean time for FDI completion. It will be referred as $FT_{meanTime}$.

Throughout the simulation campaign the MCSs will have different numbers of runs depending on the result being analyzed. The more comprehensive ones are composed of 1000 runs, that would result in useless chaotic plots, if shown. Furthermore, the quantities of interest for FDI (e.g. fault detection, faulty observer, faulty thruster, fault time) are all values that don't express a characteristic "trend" to be usefully shown in plots. They can either be zero or assume a certain value. For all these reasons it is chosen to show MCSs plots of 100 runs max, independently of the variable. This means that even when discussing the bigger 1000 runs MCSs, their graphical representation will be referred to a simulation with all the same settings, but the number of runs, set to 100 (this will be highlighted also in the plot description).

5.4 Efficiency Loss Fault

This is the case for which $FT_{kind} = 0$. In this case the thruster will correctly respond to the control, however its reduced max thrust will have an impact on the spacecraft. For the case analyzed here, the complete absence of disturbances allows for a fine tuning of FD_{trs} (discussed in 4.3), set to a value of 0.00001. This means that even the smallest allowed allowed fault, corresponding to FT_{thr} working at 95%, are detected by the FDI. A 1000 runs MCS is performed, with the following settings:

- FT_{thr} can assume all values.
- FT_{mag} can assume values from 0 to 0.95.

It is noted that the condition $FT_{mag} = 1$ is not contemplated in efficiency loss faults, since it would correspond to a non fault condition. **The result is a FDI_{succ} of 100%, so all the faults are correctly detected and identified**. The corresponding 100 runs plot for the angular velocity is shown in 5.1, and serves just as a visualization of the different runs. It can be noticed how, despite the

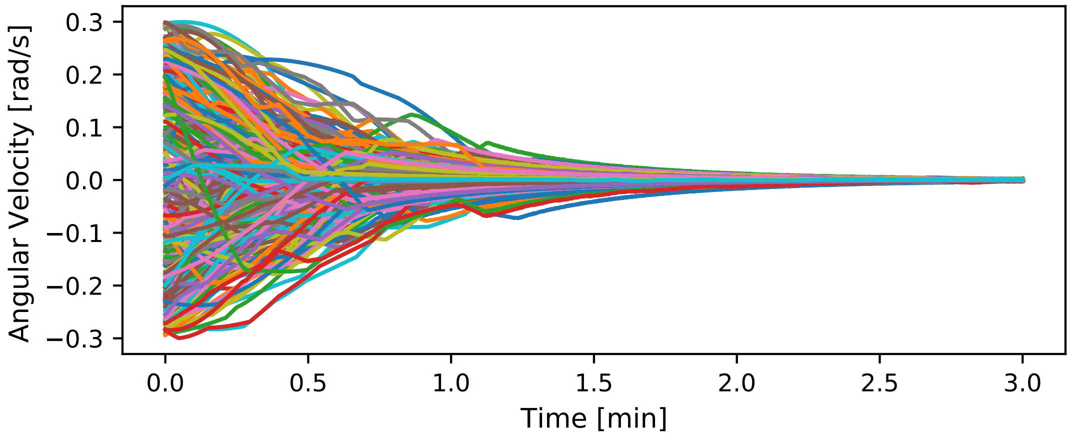

Figure 5.1: Plot of the angular velocity components in the corresponding 100 runs MCS of the efficiency loss case

faults happening, the control system is able to perform the maneuver. **This is a great achievement for the FDIS, since it is able to perform perfectly even when the de-tumbling mission is not compromised.**
Concerning the FDI time, for the depicted MCS it is found that $FT_{meanTime} = 1.272s$. This information, however, is not going to be particularly useful per se, since it will depend on when FT_{thr} will be fired for the first time after FT_{time} (4.2).
While still in the same fault kind situation, a useful analysis can be done by fixing FT_{thr}. A 100 run MCS is performed with the same parameters as before, but for all simulations $FT_{thr} = 1$. As expected the FDI success percentage is the same, but with this new constraint it is possible to better visualize the NUIOs performance. In figure 5.2 are shown the norms of the NUIOs residuals at the time of the fault identification completion. FT_{thr} belongs to the first NUIO, and it can be clearly seen how the corresponding plot is the only one without spikes. The correct NUIO behavior is then confirmed, since it is the only one whose residual doesn't rise in case of fault. By zooming in (5.3), another behavior can be caught. On a much smaller scale with respect to the others, the first NUIO's residual norm spikes have a descending trend. This behavior is to be attributed to the choice of γ that will be more optimized for smaller angular velocity values. For the sake of visualization, the plot for the results of the final isolation is reported in 5.4.

As mentioned before, the fault detection threshold has been tuned to guarantee a maximum success rate. The impact of this parameter on the FDI overall performance can be shown by increasing its value, thus making the fault detection less sensitive. A 100 runs MCS is performed, with the following characteristics:

- FT_{thr} can assume all values.

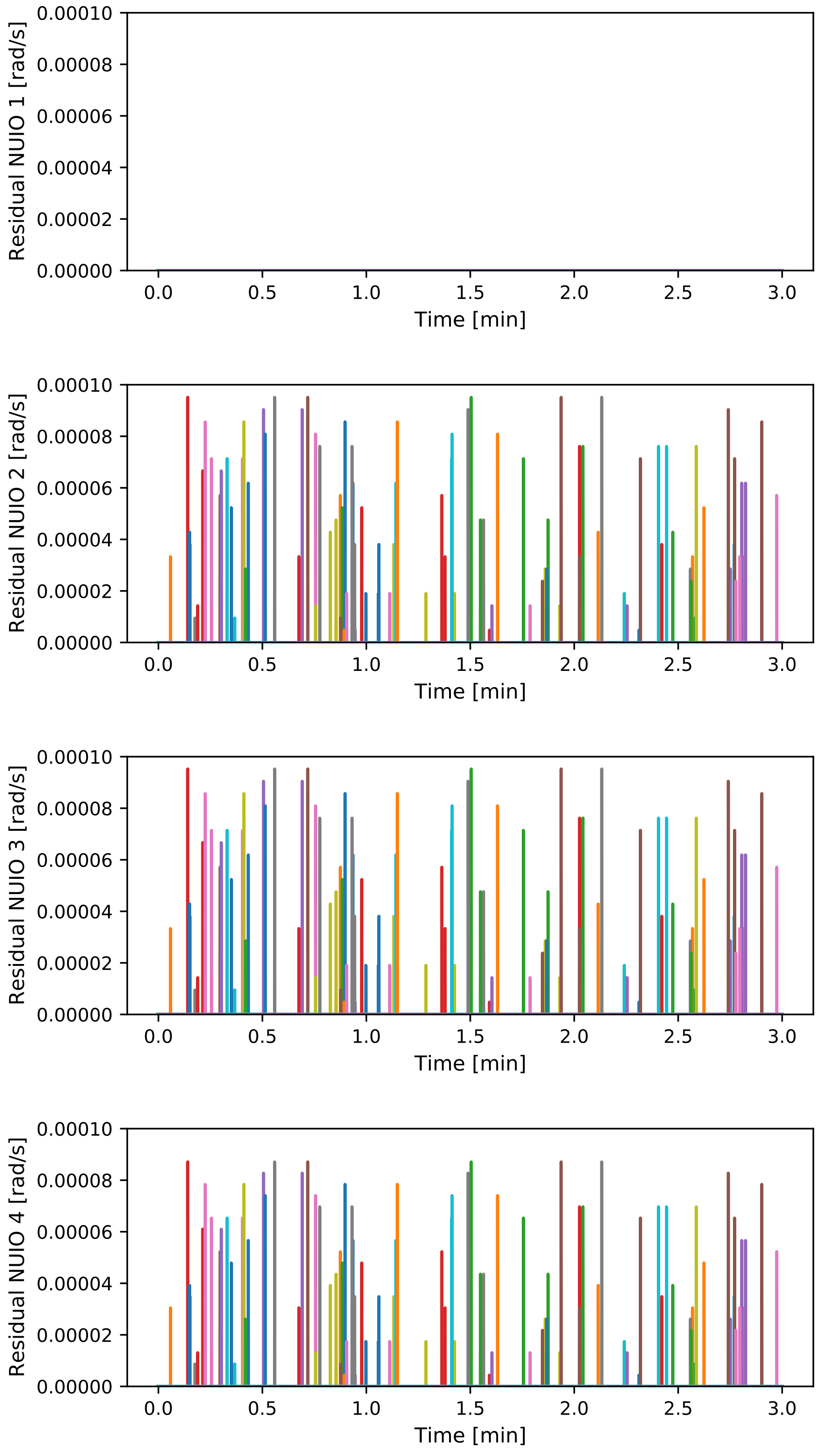

Figure 5.2: Behavior of the four NUIOs in the efficiency loss case with faulty thruster 1

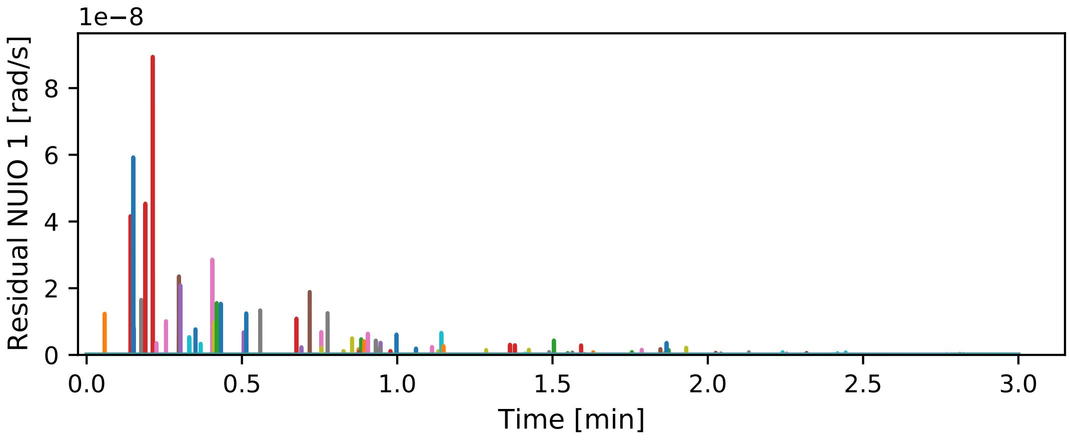

Figure 5.3: Zoom on the first NUIO

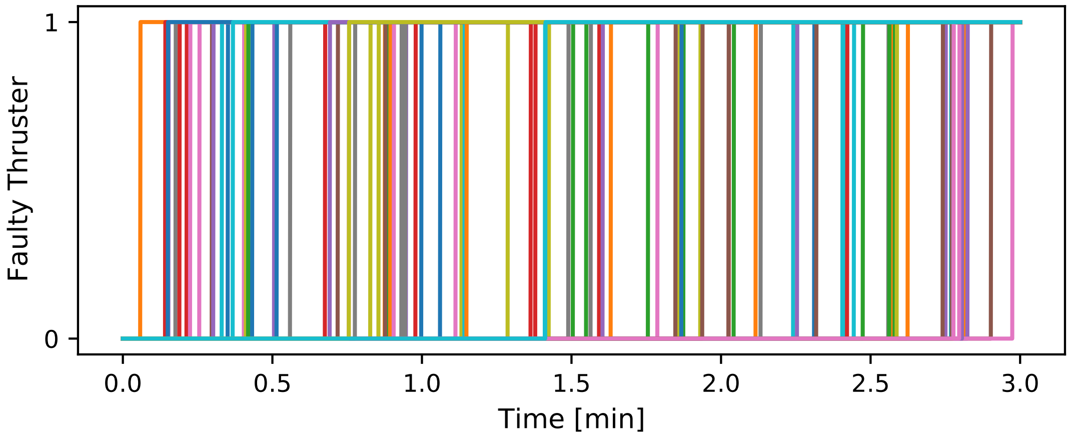

Figure 5.4: Final isolation results. All the lines end up in the top right corner, at a value corresponding to 1, in accordance with the real faulty thruster

- FT_{mag} can assume values from 0 to 0.95.
- FD_{trs} is raised to a value of 0.0001.

The success rate for this case drops to 95%, and from the MCS log file (table 5.1) it can be noticed that in all the failed FDIs the injected fault is very small.

5.5 Stuck Thruster Fault

The case for which $FT_{kind} = 0$ is going to be analyzed here. As before, a 1000 runs MCS is done with following parameters:

- FT_{thr} can assume all values.
- FT_{mag} can assume all values.

Table 5.1: Log file containing the 5 failed FDIs

FT_{thr}	$FT_{thr_{pred}}$	FT_{mag}	FT_{time} [s]
3	0	0.95	95.487
3	0	0.95	56.751
1	0	0.9	154.683
4	0	0.95	138.643
1	0	0.95	53.583

In this case all the FT_{mag} values are contemplated. If a thruster is stuck, and its command changes at least once, the fault will always generate a net force, regardless of FT_{mag} (as seen in 4.2).
The efficiency loss and thruster stuck faults are detected and isolated in the same way, and the plots show the same behavior (e.g. figure 5.5 for angular velocity). The only performance parameter whose difference is worth noticing is the mean

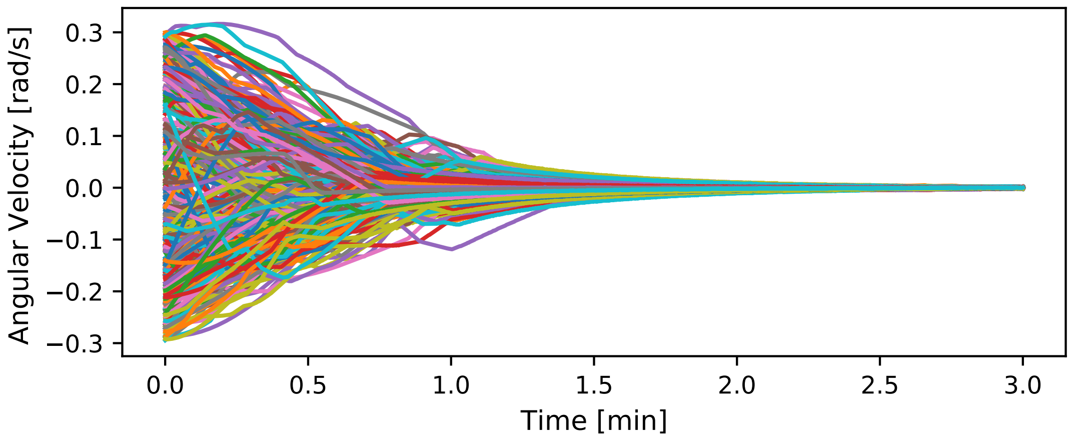

Figure 5.5: Plot of the angular velocity components in the corresponding 100 runs MCS of the thruster stuck case

FDI time. For the cases in which the thruster is stuck on at a sub-nominal value different from zero (remembering the considerations in 4.2), there are no waiting times t_w and t_{w*}. Instead, if the thruster is stuck completely on or off, the situation is the same as a loss of efficiency, time-wise. To prove this last sentence, a 1000 run MCS is done, in which only $FT_{mag} = 0$ and $FT_{mag} = 1$ are injected. The results are in agreement with the theory and are:

- Efficiency loss $\implies$ $FT_{meanTime} = 1.272s$.
- Thruster stuck - all FT_{mag} values $\implies$ $FT_{meanTime} = 0.086s$.
- Thruster stuck - $FT_{mag} \in [0, 1]$ $\implies$ $FT_{meanTime} = 1.340s$

5.6 Disturbances

In order to investigate the robustness of the FDI method in a mission in which disturbances are present, the module discussed in 3.2 is included in the simulation. It generates disturbances in terms of constant external forces $\mathbf{ext_{force}}$ and torques $\mathbf{ext_{torque}}$. The torques are referred to the origin of the BF, that, in this case, corresponds to the COM.
Those two objects will impact differently the FDI, and the reason is to be found in the core idea of the FDIS. The steps that require accelerometer information will be negatively affected by the presence of an external force (mainly FD), while the FFI accuracy will resent from an external torque.
The analysis is done considering the same fault detection threshold as before, that allowed to have a 100% success rate. A tuning of this value is fundamental and strictly related to the expected disturbances. The MCSs performed here provide a useful tuning tool, since it allows to explore the success rate changes depending on the disturbances magnitude. In particular, the disturbances are set up with a uniform dispersion on all components, given their maximum absolute component-wise value $max(\mathbf{ext_{force,torque}})$.
All the upcoming MCSs will allow all possible values for FT_{thr} and FT_{kind}. Without loss of generality the FT_{mag} will assume all values but 1. This is done in order to exclude the injection of $FT_{thr} = 0$-$FT_{mag} = 1$ combinations.

5.6.1 External Forces

When $\mathbf{ext_{force}}$ is introduced, the simple threshold check on the acceleration norm might not be enough to perform a trustworthy fault detection. The composition of the fault-produced net force with the external one can have two effects:

- Mistakenly trigger the FDIS before an actual fault, if the directions are similar, causing *false positives.*
- Mitigate the effect of a fault and consequently miss its effect. This happens when the directions are opposite-like.

The FDI performs well until the max disturbance allowed is in the same order of magnitude as the threshold. A series of 100 runs MCSs are performed with increasing $max(\mathbf{ext_{force}})$ value, and indeed a FDI behavior shift happens between $10^{-4}N < max(\mathbf{ext_{force}}) < 10^{-3}N$. Across this value, the FDI_{succ} drops from 100% to 53%. The resulting trade off is:

- A lower FD_{trs} allows for a more sensitive FDI, however the robustness to disturbances will drop.
- On the contrary, a higher FD_{trs} can be tolerated if not interested in small faults, allowing for a bigger $max(\mathbf{ext_{force}})$.

5.6.2 External Torques

The external torque $\mathbf{ext_{torque}}$ consists in a pure moment applied to the spacecraft. This will become another unknown input for the NUIOs, and potentially compromise their estimation. This disturbance is completely un-modeled in the dynamics, and this has to be with the fact that the nominal mission is in deep space, where the disturbances are negligible with respect to a disruptive thruster fault. With this analysis, however, it is possible to establish *when* a disturbance modeling might be helpful.
In a similar fashion to $\mathbf{ext_{force}}$, for $\mathbf{ext_{torque}}$ a series of 100 runs MCSs are performed, this time with increasing values of max torque. The success rate drops from 100% to 87% in the interval: $10^{-3}N \cdot m < max(\mathbf{ext_{force}}) < 10^{-2}N \cdot m$. This drop is related to a NUIOs loss of state estimation ability. This phenomenon can be visualized by restricting the fault to a single thruster (as in 5.4). By injecting only faults in thruster 1, in a 100 runs MCS, the NUIOs behavior shown in 5.7 is very different from 5.2.

In the new case, the spikes representing the residuals of the first observer are not negligible with respect to the others, and in some runs they are even the highest among the four (i.e. wrong isolation). The failed runs can be visualized in 5.6. In this case, out of 100 runs, 13 result in a predicted faulty thruster not equal to 1.

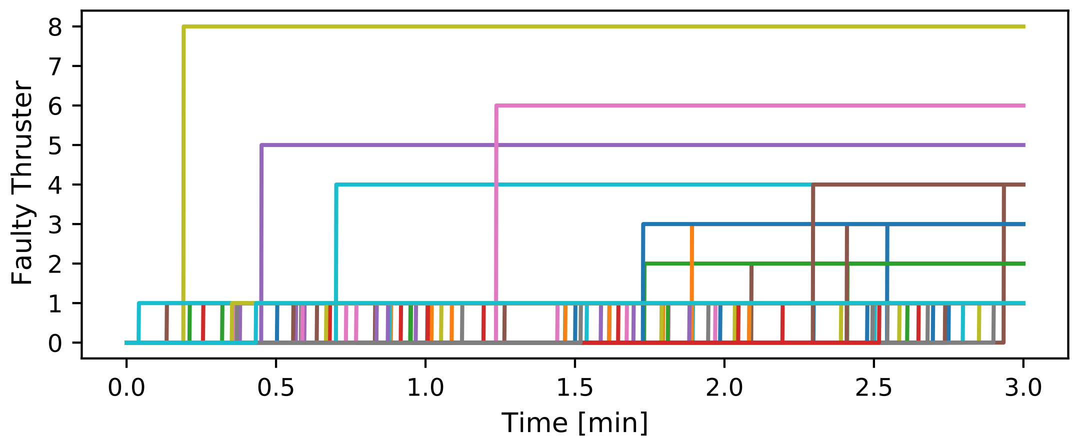

Figure 5.6: Predicted faulty thrusters, overly perturbed case

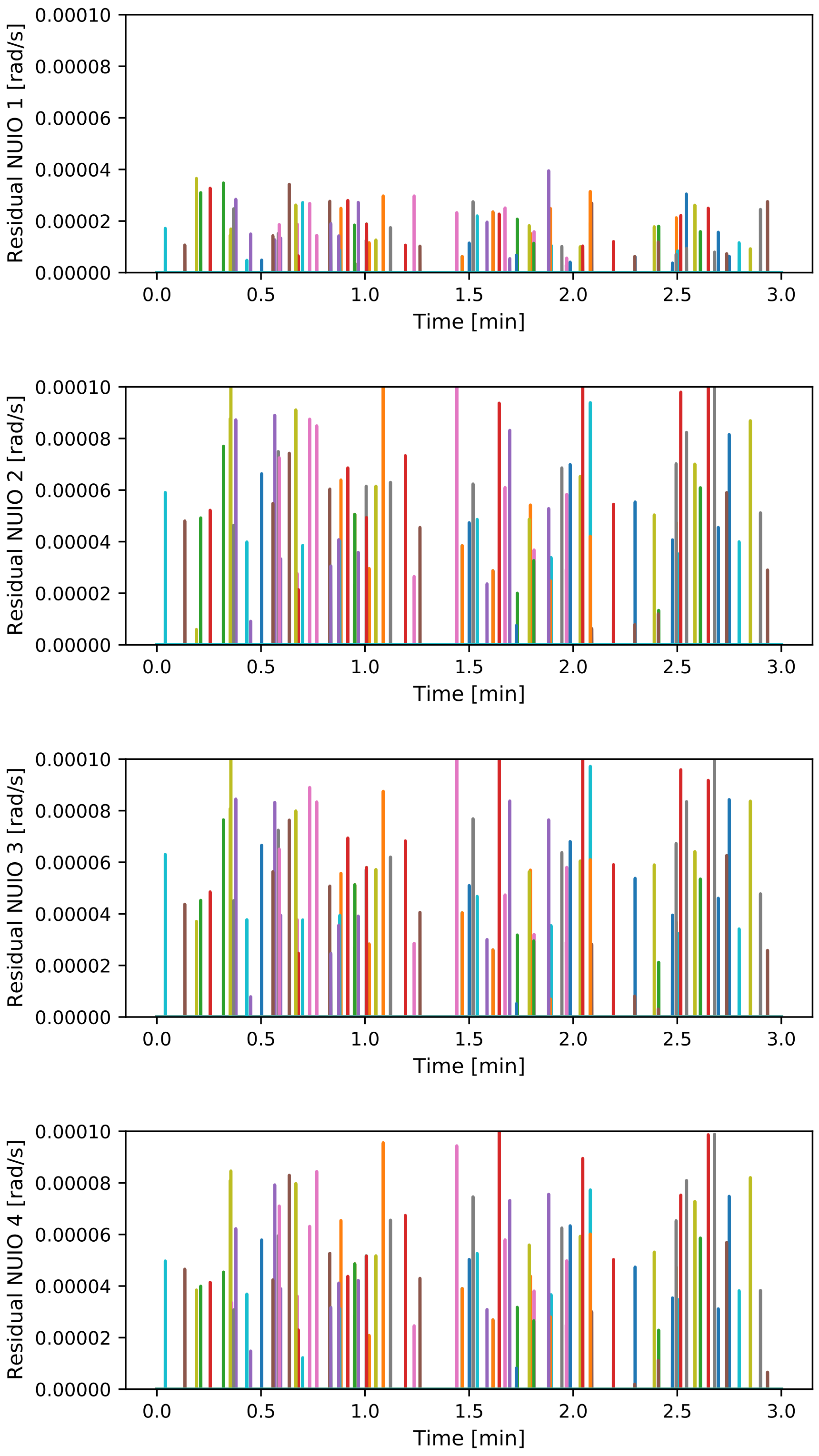

Figure 5.7: Behavior of the four NUIOs in case of disturbances and faulty thruster 1

5.7 Final Simulation

Considering all the previous analysis, a final 1000 runs MCS is performed, with the following parameters:

- $FT_{kind} \in [0, 1]$.
- All FT_{mag} values but 1.
- $FD_{trs} = 0.00001$
- $max(\mathbf{ext_{force}})$ for each BF component: $10^{-4}N$
- $max(\mathbf{ext_{torque}})$ for each BF component $10^{-3}N \cdot m$

The success rate is of 100%. This simulation represent the near worst-case scenario, in terms of disturbances, that still holds the maximum success rate. The mean FDI time $FT_{meanTime}$ is 0.328, which is an intermediate value between the times relative to $FT_{kind} = 0$ and $FT_{kind} = 1$.

Chapter 6

Conclusions and Possible Developments

In this final chapter first an analysis of the results is given with comments about the FDI performance and the consequent limits of application. Finally an insight on possible future developments is discussed.

6.1 Interpretation of the Results

The goal of the thesis was to design, implement and test the newborn FDI tool. Thanks to the Montecarlo simulation campaign its performances have been assessed and explored. The results are very promising, easily achieving a 100% success rate even with faults as small as a thruster firing at 95% of its nominal value, and the presence of unmodeled disturbances. In the simulation campaign the limits for the tolerable magnitude of those disturbances are analyzed. Thanks to its sensitivity, one of the achievements of the tool is the ability to correctly detect and identify faults even if there are no macro-effects on the de-tumbling. The information on the fault, then, might be helpful in case of later usage of the thrusters for other purposes, and actions have to be taken on the faulty thruster to address the problem.
In a possible real life application the developed algorithms would be implemented in the spacecraft OBC (with no need of auto-coding) and left running in the background. This would likely not be a burden for the OBC, since most of the time they will consist in the simple acceleration signal monitoring of the FD. Only when the a fault happens and is detected the NUIOs will go online, together with the simple SFI. Even in this case running these algorithms don't require a large amount of computational power. In fact, thanks to the C programming, all the testing and simulations have been done on an actual OBC-like code, with very good computational performance results.
The chosen mission scenario has been a simple de-tumbling, but this doesn't limit the tool's application to this maneuver only. In this sense, as long as the thrusters are fired, the FDI can be performed. Among all the possible maneuvers, however, the de-tumbling usually implies the usage of simple sensors (like the IMU used here) cause other more complex sources of information like star trackers (if present)

are not compatible with the high initial angular velocities that have to be corrected. The only limits to the applications on different missions are:

- The thruster configuration, for two reasons. First, in order for the FD and SFI to work as they are presented now, the assumption of an ACS with pure torque production is necessary. The SFI, in particular, needs to be able to choose between suspect thrusters with distinguishable nominal force directions. The other condition, still related on the configuration, is imposed by the NUIOs in the FFI. As seen on the theory, in fact, it is necessary to meet the rank requirements on the unknown input matrices. It is difficult to predict a priori which are the all the possible thruster configurations compatible, so a per-case check is advised.

- If the thrusters have a low enough nominal maximum force, it can be possible that, either cause of disturbances or various noises present in the system, their effect on the spacecraft cause of a fault does not get detected.

- If the spacecraft has moving parts, their actuation might perturb the FDIS, potentially making the employed method not ideal for the application.

6.2 Future Work

In terms of possible future work applicable to the method, here are some suggestions.

Fidelity of the simulation In order to test the tool in more realistic situations, noises and delays can be introduced in the system, depending on the particular mission and components specifics. These quantities could worsen the FDI performance and their effect is worth being studied and predicted.

Disturbances The disturbances have been modeled as simple constant external forces and torques. While this assumption was useful for the sample mission analyzed here, for other cases it might be useful to include the disturbances in the tool's modeling.

Fault recovery The natural step after FDI is the "Fault recovery". This is another important topic in the space engineering community and it can be tackled in different ways. Usually, however, it implies either a change of the control law, or the cut off of the faulty thruster for the system. Those actions are rather dependent on the specifics of the mission, and this is why it has not been considered in this thesis, in which the mission serves just as a test-bed for the developed algorithms.

Acronyms

FDI	Fault Detection and Identification
FD	Fault Detection
FI	Fault Identification
NUIO	Nonlinear Unknown Input Observer
ADE	Attitude Dynamics Equations
UI	Unknown Input
NN	Neural Network
EKF	Extended Kalman Filter
FFI	First Fault Isolation
SFI	Second Fault Isolation
FDIS	FDI Subsystem
OBC	On-Board Computer
FSW	Flight Software (algorithm)
s/c	Spacecraft
IF	Inertial Frame
BF	Body Frame
DCM	Direction Cosine Matrix
MRP	Modified Rodrigues Parameters
CGT	Cold Gas Thrusters
PTS	Pure Torque Source
MEMS	Micro Electro-Mechanical Systems
BDT	Basilisk De-Tumbling Control
MCS	Montecarlo Simulation

Bibliography

References

Pubblications and Manuals

[1] John Alcorn, Cody Allard, and Hanspeter Schaub. “Fully coupled reaction wheel static and dynamic imbalance for spacecraft jitter modeling”. In: *Journal of Guidance, Control, and Dynamics* 41.6 (2018), pp. 1380–1388 (cit. on p. 10).

[2] John Alcorn et al. “Simulating attitude actuation options using the basilisk astrodynamics software architecture”. In: *67th International Astronautical Congress, Guadalajara, Mexico*. 2016 (cit. on p. 10).

[3] C Allard, Hanspeter Schaub, and Scott Piggott. “General hinged solar panel dynamics approximating first-order spacecraft flexing”. In: *AAS Guidance and Control Conference, Breckenridge, CO*. 2016 (cit. on p. 10).

[4] Cody Allard, Manuel Diaz Ramos, and Hanspeter Schaub. “Computational Performance of Complex Spacecraft Simulations Using Back-Substitution”. In: *Journal of Aerospace Information Systems* 16.10 (2019), pp. 427–436 (cit. on p. 10).

[5] Cody Allard et al. “Modular Software Architecture for Fully Coupled Spacecraft Simulations”. In: *Journal of Aerospace Information Systems* 15.12 (2018), pp. 670–683 (cit. on p. 10).

[6] Raymond Bzibziak. “Update of cold gas propulsion at Moog”. In: *36th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*. 2000, p. 3718 (cit. on pp. 23, 24).

[7] Paolo Cappuccio, Cody Allard, and Hanspeter Schaub. “Fully-Coupled Spherical Modular Pendulum Model To Simulate Spacecraft Propellant Slosh”. In: *AAS/AIAA Astrodynamics Specialist Conference*. 2018 (cit. on p. 10).

[8] Wen Chen and Mehrdad Saif. “Observer-based fault diagnosis of satellite systems subject to time-varying thruster faults”. In: (2007) (cit. on pp. 4, 6).

[9] M Cols Margenet, Hanspeter Schaub, and Scott Piggott. “Modular Platform for Hardware-in-the-Loop Testing of Autonomous Flight Algorithms”. In: *International Symposium on Space Flight Dynamics, Matsuyama-Ehime, Japan*. 2017 (cit. on p. 10).

[10] Mar Cols-Margenet, Hanspeter Schaub, and Scott Piggott. “Modular attitude guidance development using the basilisk software framework”. In: *AIAA SPACE 2016*. 2016, p. 5538 (cit. on p. 25).

[11] Wim De Groot. "Propulsion options for primary thrust and attitude control of microspacecraft". In: *COSPAR Colloquia Series*. Vol. 10. Elsevier. 1999, pp. 200–209 (cit. on p. 20).

[12] R. Fonod et al. "Thruster Fault Detection, Isolation and Accommodation for an Autonomous Spacecraft". In: *IFAC Proceedings Volumes* 47.3 (2014). 19th IFAC World Congress, pp. 10543–10548. ISSN: 1474-6670. URL: http://www.sciencedirect.com/science/article/pii/S147466701643288X (cit. on pp. 5, 7, 39).

[13] Robert Fonod et al. "Robust FDI for fault-tolerant thrust allocation with application to spacecraft rendezvous". In: *Control Engineering Practice* 42 (2015), pp. 12–27. ISSN: 0967-0661. URL: http://www.sciencedirect.com/science/article/pii/S0967066115000945 (cit. on pp. 5, 13).

[14] P Gahinet et al. "LMI Control Toolbox User's Guide. The Math Works Inc". In: *Natick.—1995.—310 p* (1995) (cit. on p. 39).

[15] Inseok Hwang et al. "A survey of fault detection, isolation, and reconfiguration methods". In: *IEEE transactions on control systems technology* 18.3 (2009), pp. 636–653 (cit. on p. 2).

[16] *Inertia Measurement Unit Datasheet*. STIM300. Rev. 25. Sensonor. Feb. 2020 (cit. on pp. 19, 32).

[17] Rolf Isermann. "Model-based fault-detection and diagnosis–status and applications". In: *Annual Reviews in control* 29.1 (2005), pp. 71–85 (cit. on p. 2).

[18] Tao Jiang, Khashayar Khorasani, and Siamak Tafazoli. "Parameter estimation-based fault detection, isolation and recovery for nonlinear satellite models". In: *IEEE Transactions on control systems technology* 16.4 (2008), pp. 799–808 (cit. on pp. 3, 6).

[19] John L Junkins and Hanspeter Schaub. *Analytical mechanics of space systems*. American Institute of Aeronautics and Astronautics, 2009 (cit. on pp. 15, 26).

[20] Patrick W Kenneally, Scott Piggott, and Hanspeter Schaub. "Basilisk: a flexible, scalable and modular astrodynamics simulation framework". In: *7th International Conference on Astrodynamics Tools and Techniques (ICATT), DLR Oberpfaffenhofen, Germany*. 2018 (cit. on p. 9).

[21] Yunosuke Maki and Kenneth A Loparo. "A neural-network approach to fault detection and diagnosis in industrial processes". In: *IEEE Transactions on Control Systems Technology* 5.6 (1997), pp. 529–541 (cit. on p. 3).

[22] Paolo Massioni, Guido Sangiovanni, and Michèle Lavagna. "Innovative Software for Autonomous Fault Detection and Diagnosis on Space Systems". In: Jan. 2006 (cit. on p. 3).

[23] Ron J Patton and Jie Chen. "Review of parity space approaches to fault diagnosis for aerospace systems". In: *Journal of Guidance, Control, and Dynamics* 17.2 (1994), pp. 278–285 (cit. on p. 4).

[24] Ron J Patton et al. “Robust FDI applied to thruster faults of a satellite system”. In: *Control Engineering Practice* 18.9 (2010), pp. 1093–1109 (cit. on pp. 4, 7).

[25] Andre Posch et al. “Model-based on-board realtime thruster fault monitoring”. In: *IFAC Proceedings Volumes* 46.19 (2013), pp. 553–558 (cit. on pp. 4, 6).

[26] Pablo A Servidia and RS Sanchez Pena. “Thruster design for position/attitude control of spacecraft”. In: *IEEE Transactions on Aerospace and Electronic Systems* 38.4 (2002), pp. 1172–1180 (cit. on p. 21).

[27] V Sharma et al. “Unknown input nonlinear observer design for continuous and discrete time systems with input recovery scheme”. In: *Nonlinear Dynamics* 85.1 (2016), pp. 645–658 (cit. on pp. 13, 15).

[28] Silvio Simani, Cesare Fantuzzi, and Ronald Jon Patton. “Model-based fault diagnosis techniques”. In: *Model-based Fault Diagnosis in Dynamic Systems Using Identification Techniques*. Springer, 2003, pp. 19–60 (cit. on p. 4).

[29] Scott R Starin and John Eterno. “Attitude determination and control systems”. In: (2011) (cit. on p. 11).

[30] M Thirumarimurugan, N Bagyalakshmi, and P Paarkavi. “Comparison of fault detection and isolation methods: A review”. In: *2016 10th International Conference on Intelligent Systems and Control (ISCO)*. IEEE. 2016, pp. 1–6 (cit. on p. 6).

[31] Arturo Valdes, Khashayar Khorasani, and Liying Ma. “Dynamic neural network-based fault detection and isolation for thrusters in formation flying of satellites”. In: *International Symposium on Neural Networks*. Springer. 2009, pp. 780–793 (cit. on p. 3).

[32] Weitian Chen and M. Saif. “Fault detection and isolation based on novel unknown input observer design”. In: *2006 American Control Conference*. 2006, 6 pp.- (cit. on pp. 7, 13, 15).

[33] Edward Wilson and Stephen M Rock. “Reconfigurable control of a free-flying space robot using neural networks”. In: *Proceedings of 1995 American Control Conference-ACC’95*. Vol. 2. IEEE. 1995, pp. 1355–1359 (cit. on p. 3).

[34] Youmin Zhang and Jin Jiang. “Bibliographical review on reconfigurable fault-tolerant control systems”. In: *Annual Reviews in Control* 32.2 (2008), pp. 229–252. ISSN: 1367-5788. URL: http://www.sciencedirect.com/science/article/pii/S1367578808000345 (cit. on p. 2).

Online Material

[35] Hanspeter Schaub. *Basilisk Documentation*. 2020. URL: https://hanspeterschaub.info/basilisk/Documentation/index.html (cit. on p. 11).

[36] Hanspeter Schaub. *Basilisk Force Mapping Algorithm*. 2020. URL: https://hanspeterschaub.info/basilisk/Documentation/fswAlgorithms/effectorInterfaces/thrForceMapping/thrForceMapping.html (cit. on p. 28).

References

Non Cited References

[37] Sara Ghasemi and Khashayar Khorasani. “Fault detection and isolation of the attitude control subsystem of spacecraft formation flying using extended Kalman filters”. In: *International Journal of Control* 88.10 (2015), pp. 2154–2179.

[38] David Henry. “Fault diagnosis of microscope satellite thrusters using H-infinity/H_ filters”. In: *Journal of Guidance, Control, and Dynamics* 31.3 (2008), pp. 699–711.

[39] Bingyong Yan et al. “Fault diagnosis for a class of nonlinear systems via ESO”. In: *ISA transactions* 47.4 (2008), pp. 386–394.