UNIVERSIDADE
BEIRA INTERIOR

Analysis and Hardware In the Loop Testing of ADCS Algorithm for the CubeSat 3-AMADEUS

Hugo Brandão Pontes

Dissertação para obtenção do Grau de Mestre em
Engenharia Aeronáutica
(Mestrado Integrado)

Orientador: Anna Guerman Ph.D.
Co-orientador: João Filipe Fortuna Araújo M.Sc.

Novembro de 2020

Acknowledgements

I would like to thank, firstly, all my professors at UBI that have taught me a great deal, not only in theoretical knowledge but in terms of work ethic and encouraged the students to have an attitude of self development. They have accompanied me throughout the whole journey of my university life and to them I am thankful. I would also like to thank Anna Guerman for being my UBI mentor in this project, for I could have not achieved this goal without her expertise and guidance and whose immense knowledge of spacecraft related phenomena I can't help but dream of achieving one day.

My colleagues at UBI are also to be thanked, for I like to believe that we all learned from each other and developed each other in various ways. A special thanks to Jorge Monteiro for believing in UBI's potential for space projects and boosting the space interest and opportunities at UBI for students like me with a passion for space.

I would also like to thank everyone in CEiiA involved in the 3-AMADEUS project for the amazing opportunity of working on this project - not only on this thesis but on the summer internship I did before - for all the help and for giving me such an up close experience of satellite development. It is something I personally gained a lot from. A special thanks to Hélder Covas, who was tireless in trying to provide everything I required for this project as well as always being available for whatever I needed help with. Another special thank you is due to João Araújo - my CEiiA tutor - I appreciate all the knowledge passed on, without which the FPGA implementation would be impossible.

I must also tip my hat in extreme gratitude to Dmitry Roldugin from KIAM who exchanged countless emails with me, without which I couldn't have developed a valid attitude model. I have learnt a great deal from him and also hope that one day I can reach his knowledgeability.

I must also thank all the kind strangers on various internet forums who take their free time to help other's engineering endeavours and have helped me by pointing me in the right direction at the times where I needed it the most.

Last but not least, I would also like to express my gratitude to my friends and family who, with their love and support have probably kept me from descending into madness at the direst times in this journey. Thank you!

Resumo

Um dos grandes entraves dos ADCSs (*Attitude Determination and Control Subsystems*) de *CubeSats* é o elevado peso e o alto consumo dos seus componentes de maior precisão, o que significa que desenvolver opções mais leves e de menor consumo é de extrema importância.

A 3-AMADEUS é uma missão que visa a encontrar uma solução para este mesmo problema. Componentes de ADCS magnéticos estão entre as opções mais leves, de menor consumo energético e mais fiáveis na indústria dos *CubeSats*. No entanto, devido à sua baixa precisão, estes não podem ser utilizados por si só em missões cujos requisitos de precisão de controlo de atitude sejam elevados. Uma das formas de aumentar a precisão deste tipo de componentes é o uso de novos algoritmos que maximizem o desempenho de ADCSs magnéticos, que é a razão pela qual a 3-AMADEUS tem o propósito de desenvolver e testar, em voo, vários destes algoritmos, com a esperança de que um dia a implementação de ADCSs exclusivamente magnéticos seja generalizada em *CubeSats*.

Para que seja possível analisar quais algoritmos devem ser implementados na missão 3-AMADEUS, este trabalho apresenta um modelo de atitude de um satélite que permite uma simulação SIL (*Software In the Loop*). Para além disso, é também feita uma simulação HIL (*Hardware In the Loop*) que procura validar o uso de um FPGA (*Field Programmable Gate Array*) para a implementação deste tipo de algoritmo, já que o uso de FPGAs em CubeSats tem tido um crescimento significativo, e é particularmente interessante num projeto onde a reprogramabilidade é uma característica útil.

Tendo isto em conta, como os algoritmos para esta missão ainda estão em desenvolvimento, um algoritmo puramente magnético desenvolvido noutro contexto é então testado num ambiente SIL, no qual o seu desempenho em termos de precisão e estabilização, assim como a sua viabilidade para a missão 3-AMADEUS, são analisados sob diferentes condições. Por fim, um destes testes é realizado num ambiente de simulação HIL. Os resultados desta simulação, que não têm em conta a determinação da atitude, são comparados com os obtidos no teste em ambiente SIL, fornecendo dados relevantes sobre a viabilidade e desempenho de uma implementação de um algoritmo de ADCS num FPGA na realidade.

Palavras-chave

CubeSat, Controlo de Orientação de Satélite, Modelo de Dinâmica Rotacional de Satélite, *Magnetorquer*, Subsistema de Determinação e Controlo de Atitude, Algoritmo de Controlo de Atitude A Bordo, *Hardware In the Loop*, *Software in the Loop*, Simulação de Atitude de Satélite, *Field Programmable Gate Array*

Abstract

One of the main challenges with Cubesats' ADCSs (Attitude Determination and Control Subsystems) is how heavy and power consuming the most precise systems are. This means that developing lighter, less consuming ones is of the greatest importance.

3-AMADEUS is a mission that aims to find a solution to this exact problem. Magnetic ADCS components are among the lightest, least power consuming and most reliable options in the CubeSat industry. However, due to their low precision, this kind of component can't be used by themselves in missions that require precise attitude control. One of the ways to improve the precision of this kind of component is to use novel ADCS algorithms that maximize system performance for magnetic ADCSs. That is why 3-AMADEUS has the purpose of, not only developing, but also testing multiple of these algorithms in-flight, with hopes that one day the implementation of purely magnetic ADCSs can be generalized in nanosatellites.

In order to possibilitate an analysis of what algorithms are to be implemented in the 3-AMADEUS mission, this work presents a satellite attitude model that allows for a SIL (Software In the Loop) simulation. Furthermore, a HIL (Hardware In the Loop) simulation is made, aiming at validating the usage of an FPGA (Field Programmable Gate Array) for the implementation of this kind of algorithm, since the usage of FPGAs in CubeSats has been rising significantly, and is particularly interesting in a project where reprogrammability is useful.

Having that in mind, since the algorithms for this mission are still under development, a purely magnetic ADCS algorithm that has been developed in another context is then tested in a SIL environment, where its performance in terms of accuracy and stabilization, as well as its suitability for the 3-AMADEUS mission, is analyzed under different conditions. Finally, one of these tests is performed but this time in a HIL Simulation, not considering attitude determination. The results of this simulation are compared to those obtained in the SIL test, providing relevant data on the feasibility and performance of a real life ADCS algorithm implementation in an FPGA.

Keywords

CubeSat, Spacecraft Attitude Control, Satellite Attitude Dynamics Model, Magnetorquer, Attitude Determination and Control Subsystem, On-board Attitude Control Algorithm, Hardware In the Loop, Software in the Loop, Spacecraft Attitude Simulation, Field Programmable Gate Array

Nomenclature

r	Orbit Radius
a	Orbit Semi-Major Axis
e	Orbit Eccentricity
θ	True Anomaly
v	Orbital Velocity
i	Orbit Inclination
Ω	Longitude of the Ascending Node
ω	Argument of Perigee
ω_o	Orbital Angular Velocity
R_A^B	Rotation Matrix from A to B frame
q	Quaternion
ε	Quaternion Sub-vector 1
η	Quaternion Sub-vector 2
a	Euler Axis Vector
ϕ	Angle of Rotation
I	Identity Matrix
ϕ	Roll
θ	Pitch
ψ	Yaw
L	Angular Momentum
v	Tangential Velocity
r	Body Radius
m	Body Mass
ω_o	Orbital Angular Velocity
T_o	Orbital Period
ω_e	Earth's Angular Velocity
T_e	Earth's Rotation Period
I	Inertia Tensor
J	Principal Inertia Tensor
a, b, c	Body Dimensions in the principal axes
ω_{ib}^b	Absolute Angular Velocity
ω_{ob}^b	Relative Angular Velocity
M	Sum of all Torques applied on a body
τ_{gg}	Gravity Gradient Torque
c_3	Rotation Matrix Third Column
u_3	Nadir Unit Vector
μ	Earth's Gravitational Coefficient
D	Aerodynamic Drag Force
ρ	Air Density

C_D	Drag Coefficient
A	Projected Area
u_{aero}	Drag Force Direction Unit Vector
d	Center of Mass Offset
τ_{aero}	Aerodynamic Torque
F_{rad}	Solar Radiation Pressure Force
P_{rad}	Solar Radiation Mean Momentum Flux
C_P	Spacecraft Solar Absorption Characteristic
u_{solar}	Solar Radiation Pressure Force Direction Unit Vector
τ_{rad}	Solar Radiation Pressure Torque
B	Magnetic Field Induction
u_f	Geomagnetic Field Dipole Strength
u	Argument of Latitude
t	Elapsed Time
τ_{mag}	Residual Magnetic Dipole Torque
m_{res}	Residual Magnetic Dipole
τ_{trqr}	Magnetorquer Torque
m_{trqr}	Magnetorquer Magnetic Dipole
h	Simulation Step
I_{max}	Magnetorquer Max Current
m_{max}	Magnetorquer Max Magnetic Dipole
A_{coil}	Magnetorquer Coil Area
n_{coil}	Magnetorquer Coil Turns
k	Control Gain
σ	Standard Deviation

List of Acronyms

ADC	Analog to Digital Converter
ADCS	Attitude Determination and Control Subsystem
ASIC	Application Specific Integrated Circuit
BCR	Battery Charge Regulator
C&DH	Command & Data Handling
CLB	Configurable Logic Block
COTS	Commercial Off The Shelf
CPU	Central Processing Unit
CSU	Colorado State University
DAC	Digital to Analog Converter
DCM	Digital Clock Manager
ECEF	Earth Centered Earth Fixed
ECI	Earth Centered Inertial
EPS	Electrical Power Subsystem
ESA	European Space Agency
FPGA	Field Programmable Gate Array
GNC	Guidance, Navigation and Control
GND	Ground
GNSS	Global Navigation Satellite System
HDL	Hardware Description Language
HIL	Hardware In the Loop
I^2C	Inter Integrated Circuit
ICD	Interface Control Document
IDE	Integrated Development Environment
INTA	Instituto Nacional de Técnica Aeroespacial
IOB	Input/Output Block
KIAM	Keldysh Institute of Applied Mathematics
LEO	Low Earth Orbit
LUT	Look Up Table
MCU	Microcontroller Unit
MISO	Master In Slave Out
MOSI	Master Out Slave In
NASA	National Aeronautics and Space Administration
ODCS	Orbit Determination and Control Subsystem
P-POD	Poly-Picosatellite Orbital Deployer
RAM	Random Access Memory
RMD	Residual Magnetic Dipole
ROM	Read Only Memory
SIL	Software In the Loop

SPI	Serial Peripheral Interface
SSC	Surrey Space Center
SSO	Sun Synchronous Orbit
UART	Universal Asynchronous Receiver/Transmitter
UBI	Universidade da Beira Interior
UCF	User Constraints File
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The first chapter of this thesis introduces not only the project on which the thesis is based, but also its relevance in that context, as well as the personal motivation for the realization of this study. Additionally, the goals that this project aims at achieving are described, and a brief overview of the document's structure is provided.

1.1 3-AMADEUS Mission

The 3-AMADEUS (3 Axis Magnetic Attitude Demonstration Experiment for a Unit Spacecraft) CubeSat is a work in progress 1U CubeSat project that is being developed by UBI (University of Beira Interior) and CEiiA (Centro para Excelência e Inovação para a Indústria Automóvel), with the collaboration of KIAM (Keldysh Institute of Applied Mathematics), that is planned to launch in a near future into a 550 km Sun Synchronous Low Earth Orbit.

Magnetic attitude sensors and actuators are among the cheapest, most lightweight and reliable attitude estimation and control components. They, however, have limited precision. This means that in missions that require great attitude control precision, they must be coupled with heavier ADCS components such as reaction wheels. Additionally, using only magnetic actuators leaves the spacecraft underactuated. With that in mind, the development and in-flight testing of novel ADCS algorithms, that allow for an upgraded performance for this kind of component, is of the uttermost importance.

As a consequence of that, the purpose of this mission is to test and demonstrate the validity and well functioning of a solely magnetic ADCS for providing 3-axis orbital attitude for CubeSats and for nanosatellites in general. This mission objective makes the ADCS effectively the mission payload. One of the key factors for this satellite is the possibility of uploading control algorithms to it during flight, through the usage of an FPGA (Field Programmable Gate Array) that is present in its On Board Computer (OBC), more specifically an ABACUS OBC. This OBC is part of the Command & Data Handling subsystem (C&DH).

1.2 Motivation

Despite having been around for over 60 years, up to very recently, satellites have been financially unreachable for small companies, making space access incredibly difficult for small businesses. This has changed with the rise of CubeSats and Small Sats in general in the early 2000's. This evolution has greatly reduced the costs of space access, due to not only the miniaturization of the available technology but especially due to the standardization of components and interfaces with the launch vehicle. Despite all this progress, efforts must still be made to further reduce satellite costs, whether by using cheaper or lighter technology.

One of the aspects that can be improved in such a way is the ADCS of a satellite. In order to achieve precise attitude control, this subsystem still requires heavy and high power consuming components. One of the solutions to this problem is to use magnetic sensors and actuators, a much lighter and power saving option. Despite this, today, the precision of these components is low, meaning that they must often be combined with heavier and more power consuming ADCS components. This is a problem that must be solved by further developing this technology. One of the ways in which this can be achieved is by developing and testing novel attitude control algorithms that optimize the capabilities of this kind of actuators and sensors. Doing so can ultimately allow for the generalization of solely magnetic ADCSs for a large number of CubeSat missions.

Understanding the functioning, advantages and limitations of this kind of component is then of the utmost importance. Additionally, and with this in mind, it is crucial, not only to understand how these algorithms can be modelled and simulated for the analysis of their performance, but also to do that very analysis. Finally, the goal should be to test these algorithms in-flight and prove their validity, with that being ultimately the purpose of the 3-AMADEUS mission.

The use of FPGAs in nanosatellites has been on the rise as of late so it is also interesting to assess how the aforementioned algorithms can be implemented in them, specially in a mission which benefits immensely from reprogrammability.

1.3 Research Objectives

By creating an attitude simulation that is suited for the analysis of purely magnetic control algorithms, doing such analysis and creating an FPGA design that implements such algorithms, this works aims at not only verifying the validity of a specific algorithm for the 3-AMADEUS mission, but also at creating a basis of models and methodologies that can later be adapted for future testing of novel ADCS algorithms for the 3-AMADEUS mission.

The generated attitude dynamics model must provide essential and realistic attitude data during all times of simulation, that allow one to analyze the system performance, as well as the viability of the usage of the tested ADCS algorithms for the 3-AMADEUS mission.

In this model, a purely magnetic control algorithm, developed by KIAM that should stabilize the satellite in the orbital reference frame [1], shall be implemented and its performance considering a 1U satellite is to be analyzed, in order to assess its viability for the 3-AMADEUS mission.

Finally, the control law from the algorithm that is implemented in the mathematical attitude model ought to be implemented in an actual FPGA, present on 3-AMADEUS's

ABACUS OBC. This real-life implementation must be connected to the machine running the simulation, creating a Hardware In the Loop (HIL) simulation. The HIL Simulation shall provide data regarding not only the feasibility of an FPGA implementation for the 3-AMADEUS mission but how the performance of the algorithm in theory compares with the FPGA implementation, keeping in mind that the HIL simulation featured concerns the attitude control part of the algorithm, but not attitude determination.

1.4 Thesis Outline

The present chapter, Chapter 1, introduces the project and mission that this thesis regards. Additionally, it presents the motivation behind this project, namely why it is relevant within the current context of the space and nanosatellite industry. Besides that, the research objectives are presented as well, to assert what relevant data and information is expected from the completion of this thesis project. Finally, a brief overview of the structure of the document is given.

Chapter 2, State of the Art, describes what the context for this project is in terms of what is being done nowadays and what has been done in the past regarding Small Sats. The usual subsystems found in CubeSats are looked into. A further look is taken at what the ADCS and C&DH designs usually are for CubeSats missions, including what components exist and how they function and including missions with features similar to that of 3-AMADEUS.

The third chapter, Literature Review, reviews some of the theoretical aspects that one must appreciate in order to understand the work that has been undergone for this project. This chapter covers the topics relevant to the ADCS part of this thesis. These topics are attitude and orbital mechanics, attitude modelling, and the numerical analysis method that is used. Regarding the C&DH part of this work, basic concepts such as binary numbers, their arithmetic and communication protocols are covered, as well as important concepts for understanding the programming and design of FPGAs, such as some of their fundamental functional elements and the FPGA design flow.

Mathematical Model For Attitude Dynamics, the fourth chapter of this document, describes the model parameters set for the satellite attitude simulation that has been created in order to analyze a solely magnetic attitude control algorithm. Additionally, that very model is put to test in this chapter in order to assess its validity for analysis in the chapters that follow.

The fifth chapter, Uncontrolled Satellite Dynamics, simply provides an analysis of the behaviour of the satellite when no controlling torques are being exerted on it and all environmental torques are present. The presented results provide data with which attitude control data can be compared, facilitating the assessment of the effect that the attitude control algorithm has on the satellite's behaviour. Additionally, this chapter presents an analysis of the LEO environment that the satellite is expected to be in.

The sixth chapter presents and analyzes the results of the implementation of the purely magnetic control algorithm that has been used for this study [1] on the 3-AMADEUS

CubeSat. This chapter presents not only attitude data, but also an analysis of system performance in parameters such as angular rate stabilization and attitude control accuracy regarding the desired attitude. In this case, that is alignment with the orbital reference frame. Implementation in different scenarios is presented, providing information on the strong points and limitations of this algorithm and how these can be exploited/mitigated, as well as on the viability of the implementation of this algorithm for the 3-AMADEUS mission.

Attitude Control HIL Simulation is the seventh chapter of this document. It describes the tools and components necessary for an FPGA HIL Simulation, as well as the assumptions necessary in the case of this thesis. Additionally, the final FPGA design is presented and examined. Lastly, the results of running the HIL simulation in a specific scenario discussed in Ch. 6 are presented and analyzed, with the analysis consisting of comparing the obtained results with those obtained also in Ch. 6 and assessing the validity of the FPGA implementation.

The final chapter, Conclusions, delivers the conjectures that can be drawn from the analysis of the obtained research data, regarding the appropriateness for the 3-AMADEUS mission, of not only the control algorithm under study and this kind of control algorithm in general, but also of the implementation of these algorithms into an FPGA. Furthermore, this chapter presents the issues and challenges that have been faced in the development of this project, as well as the solutions for them and the methodology that has been used for their obtaining. Finally, suggestions for future work are presented.

Chapter 2

State of the Art

This section aims at reviewing what the latest and most relevant projects and advancements are in the area of, not only Attitude Control and Command & Data Handling for CubeSats, but in Nanosatellites in general. Additionally, it should provide a better understanding of the context in which this thesis has been produced and show the relevance of the results that have been obtained.

For both Command & Data Handling as well as Attitude Determination and Control subsystems, the most commonly found components are presented, along with a brief explanation of their functioning. On top of that, a few nanosatellite missions that are similar in some way to 3-AMADEUS, are presented with the purpose of showing what is being done today in terms of these subsystems.

2.1 The CubeSat paradigm

This section focuses on how Small Sats and CubeSat came to be and how and why they have become such a prominent subsector of the space industry as well as why they have revolutionised the sector, having triggered what some call the *New Space Age*.

2.1.1 CubeSats

Ever since the dawn of humankind, humans have looked up at the sky in awe and wonder about what lies beneath the stars. The first major step in achieving that knowledge was the launch of the Sputnik 1 satellite in 1957, the first of his kind, as part of the Soviet Space Program. The launch of Sputnik is considered by many to be the beginning of the space age and in the sixty-odd years that followed, the space industry has provided humanity with a plethora of applications in fields ranging from biology, military, Earth observation or communications, with its contribution to communications in particular having revolutionised the modern world. At the same rate that the technology available has evolved, electronics have decreased in size at an astonishingly fast rate, leading to miniaturization in countless technological fields, from cellphones to medical cameras. The global trend towards miniaturization has meant that the massive satellites of the past, that could only be developed by space agencies funded by governments, have been replaced by the so called Small Sats: lightweight, small satellites, with short design time and small design teams. In a field where sending a 1 kg object to space costs about 50,000 $ [2] on top of labor expenses, this makes all the difference. Small satellites can be categorized in several categories, as described by table 2.1.

As Small Sats grew in predominance, some professors at Cal Poly [3], pioneered, in 1999, the concept of a *CubeSat*. CubeSats are cube shaped nanosatellites, created with the

Table 2.1: Classification of Small Sats.

Class	Mass [kg]
Minisatellite	100-500
Microsatellite	10-100
Nanosatellite	1-10
Picosatellite	0.1-1
Femtosatellite	0.01-0.1

purpose of standardizing space access and therefore reducing costs, allowing aerospace engineering students to gain full hands-on experience in satellite development, from the mission requirements definition to the testing phase. With that in mind, Cubesats' most important characteristic is how they have drastically reduced costs. Standardizing the actual CubeSat bus - the assembly of all the components that support the payload - has allowed designers to be left with (practically) only the payload design to worry about. The standardization of the deployer, in turn, has increased launch availability and allowed for fast development cycles, as well as low-cost launches.

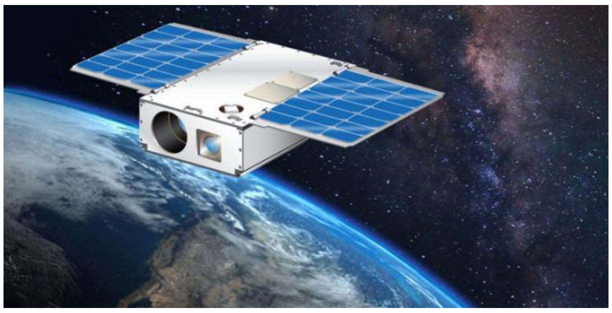


Figure 2.1: Asteria, a NASA microsatellite [4].

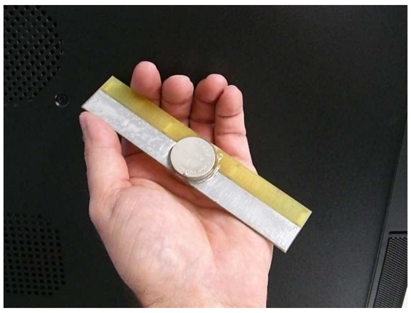


Figure 2.2: WikiSat, a femtosatellite [5].

The standardization of the bus means that all CubeSats must share the same shape. The engineers at CalPoly decided that the most appropriate shape would be a 10x10x10 cm cube, called a Unit (U). This has led to the appearance of Commercial Off The Shelf (COTS) components, specialized for CubeSats, that can be used in different missions. This is attractive not only because there doesn't need to be any adaptation of these components for the specific mission, but also due to the flight heritage (proved to have worked in space) this kind of component has. A unit should also not exceed a mass of 1.33 kg and it is also possible to have combinations of units, the most common being 3U, though there are several other possibilities, as can be seen in Fig. 2.3.

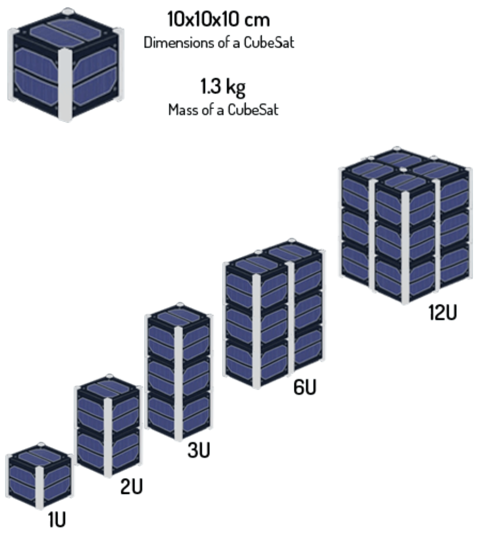


Figure 2.3: Typical CubeSat configurations [6].

What really makes the CubeSat revolutionary is the standardization of the way they can be transported into orbit. Typically, CubeSats are too small of a mission to require a launch vehicle for themselves, meaning most CubeSats ride as secondary payloads (piggybacks) in launch vehicles designated for other missions. By standardizing the CubeSat deployer, one can create an interface that is compatible both with all CubeSats and all launch vehicles interested in piggybacks, minimizing the cost of the launch for the launching agency, by making the most out of the free room in the launch vehicle, as well as the cost for the CubeSat developers, in comparison with the cost of a dedicated launch. On top of this, this interface maximizes launch availability since CubeSats can ride in any launch vehicle prepared for CubeSats. This uniformized interface has been achieved with the creation of the P-POD (Poly-Picosatellite Orbital Deployer).

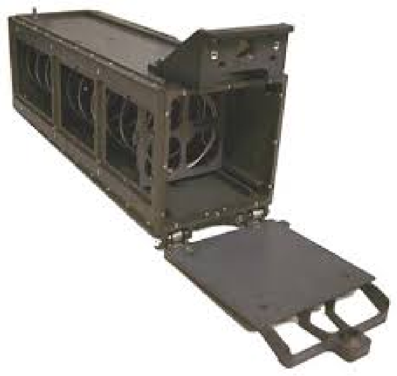


Figure 2.4: P-POD (Poly-Picosatellite Orbital Deployer) [7].

A common P-POD can house up to three units, either as a monolith (3U) or in separate (3 × 1U), though there are variations for other sizes. P-PODs are located in launch vehicles, and through a mechanical spring mechanism deploy the CubeSats onto their desired orbits.

CubeSats must comply with a document called ICD (Interface Control Document) [8, 9], a document that makes sure the CubeSat fits in standard deployers and interfaces with them correctly as well as making sure it poses no threat to the launch vehicle they are in.

By adopting this new standard, the costs associated with a full CubeSat mission can be reduced to around 100,000 $, a value well within the reach of many technological companies. This explains why the number of CubeSat mission has been growing exponentially, with over a thousand CubeSat having been launched at the time of writing. CubeSats have grown out of their initial academic purpose and are now seen as a serious and commercially viable option for space access, allowing many companies to reach space without external funding, and thrusting the space industry even further. Nowadays, CubeSats are applicable in a wide range of missions, that can be classified in different categories, according to their field of application [10], including Communications, Education, Remote Sensing, Science, and Technology Development, with 3-AMADEUS being one of the latter. Figure 2.5 shows, on one hand, how the number of CubeSat launches has been growing steadily, as well as its move away from mostly educational objectives to a broader range of applications. Figure 2.6 shows the bright future that is expected for the CubeSat and Small Sat industry [10], as future advancements in nano technology will cause CubeSats to be used increasingly and for even more purposes, probably for an even smaller price.

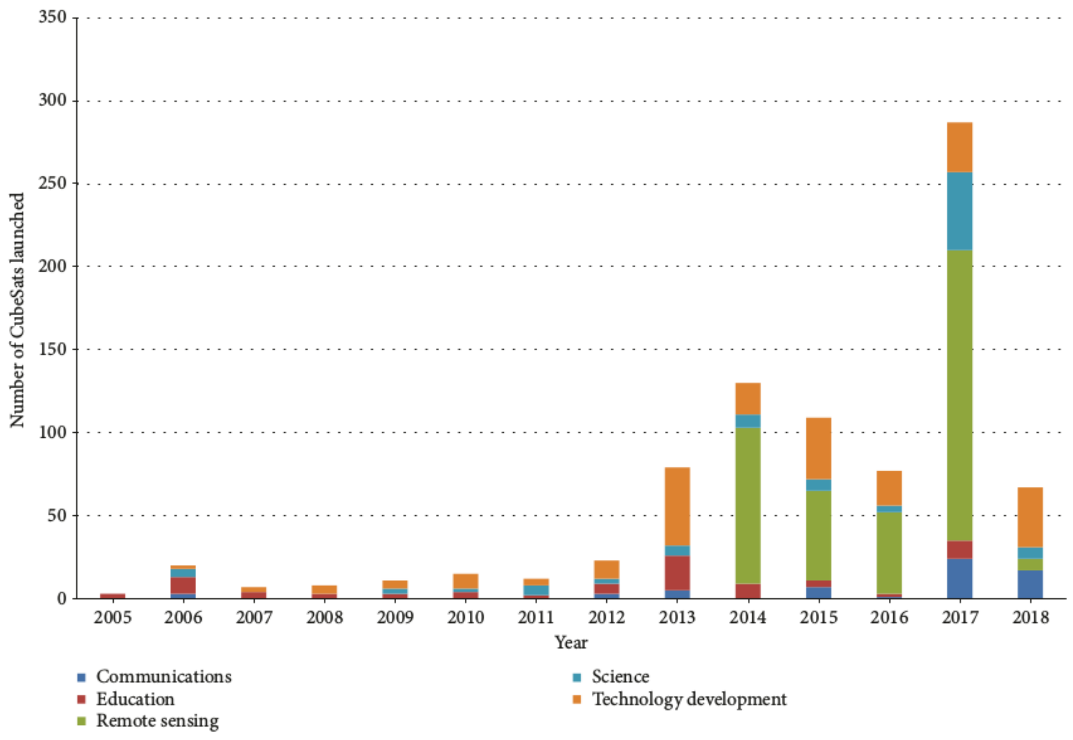


Figure 2.5: Evolution of CubeSat launches and its categories [10].

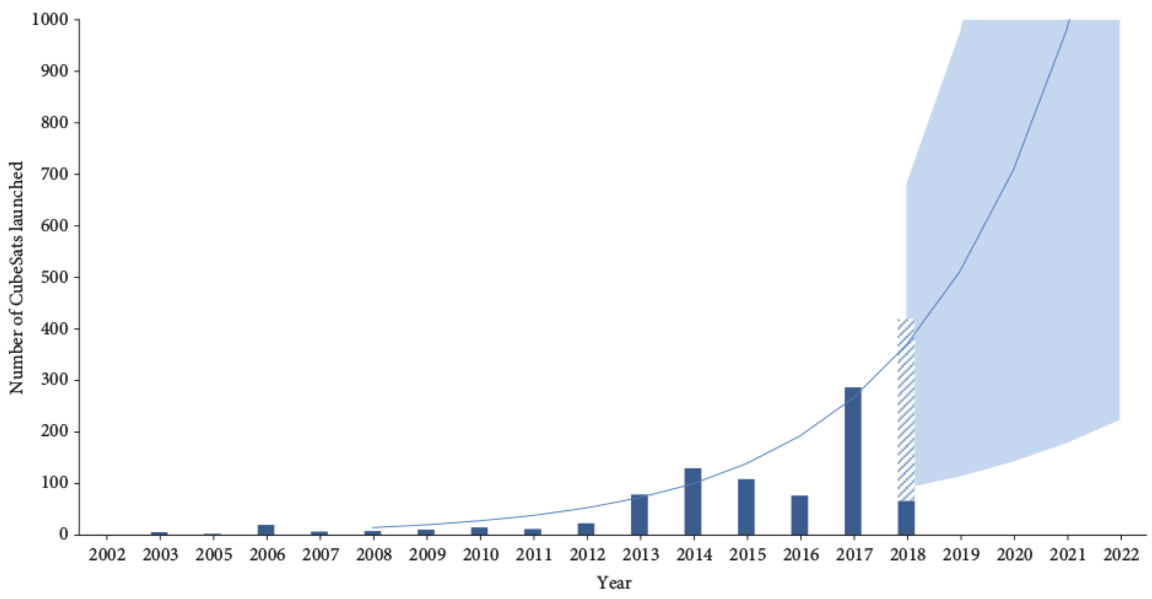


Figure 2.6: Projected evolution of CubeSat launches [10].

Although it is not the case for the mission that is the object of this thesis, CubeSats also allow for satellites to be produced in series [8], almost on a trial and error basis, since the risk and consequences of losing a CubeSat are minimal when compared to the benefits of the data it gathers, that can be used to improve future designs.

2.1.2 CubeSat Subsystems

All spacecraft are built around their main purpose: to transport a *Payload*. The payload is the part of a spacecraft which gives it a reason to exist and makes investors want to pay for its launch and design, while all other components are there solely to make sure the payload performs its mission according to plan. If, for example, a satellite is launched with the purpose of capturing pictures of the Earth, the camera would be the payload while other components would be the so called *Bus*. The bus in this example would be, among others, the system that points the camera to the target that is to be photographed, and the system that sends the pictures to the Earth. In the case of Sputnik [11], for example, the payload was a radio sending a simple signal to the Earth, while the bus was the outside shell that kept the radio at the appropriate conditions as well as the battery that powered it.

In the case of CubeSats, buses are composed of subsystems, that are, like everything in a CubeSat, standardized, meaning that almost all CubeSats should feature the same subsystems, although the subsystems themselves can differ. These subsystems can be divided into several categories. These categories are [12, 13]: Structure; Electrical Power; Propulsion; Guidance, Navigation and Control; Communications; Command & Data Handling and Thermal Control.

Structure

The structure of a CubeSat can be seen as its skeleton, in the sense that it mechanically keeps everything in place and also protects the sensitive CubeSat components safe from physical harm, mainly during launch [14]. CubeSat structures are usually made of aluminum alloy, more specifically Al 7075, though Al 6061-T6 is also often found in this kind of component.

Electrical Power Subsystem

The Electrical Power Subsystem (EPS) is the subsystem in charge of collecting, managing and supplying electrical energy to a CubeSat [15]. The EPS is generally composed of a battery, a number of solar panels, and a Battery Charge Regulator (BCR). In spacecraft, the most used components for energy gathering are, by a large margin, solar panels. State of the art solar panels can collect energy with about 30% efficiency, meaning that a CubeSat face with an area of 100 cm^2 can generate around 2W [16] if hit directly by sunlight. This gathered energy is directed, through a BCR, either directly to the various subsystems, powering them, or to the battery, where it is stored for later usage. Energy storage is extremely important in spacecraft especially if they go into eclipse situations quite often. Being in eclipse means that no sunlight is present, which usually happens when the Earth's shadow is cast upon the satellite. No sunlight, in turn, means that no energy is being gathered by the spacecraft, creating a need for the usage of stored energy. On top of that, batteries can also be required if there is a large discrepancy between the components nominal power consumption and their peak power consumption.

Propulsion

Spacecraft need propulsion systems, either to change orbits, whether to a higher or lower orbit, or to maintain an orbit that decays due to dissipative forces. Due to their small size, most CubeSats do not possess a propulsion system and are not designed for orbital maintenance. However, some missions do require such a system to be installed. Furthermore, for spacecraft that do operate at a greater distance from the Earth, due to the strict ESA space debris mitigation policies stating that satellite re-entry must be done within 25 years [17], some engineers are starting to think about some propulsion systems that guarantee such condition.

The vast majority of the propulsion systems used nowadays works with the same principle: by ejecting mass with a certain momentum, the same momentum is exerted in the spacecraft, creating a thrusting force. The best way to assess the efficiency of a propulsion system is to measure the change in momentum in relation to the mass of propellant used. This measurement is called the *Specific Impulse*, usually represented by I_{sp} and its units are seconds s. For those CubeSats that do operate outside of LEO there are three main options for propulsion subsystems (though these options can be divided into subgroups themselves)[18, 14]. The first propulsion option is the most simple, cold gas propulsion. In this type of system, cold gas is simply expelled out of the CubeSat, thrusting it in the opposite direction. This kind of propulsion system usually has a specific impulse of about 30-70 s. Chemical Propulsion systems work similarly but what is expelled are the gases resulting from the combustion of propellant, that can be solid or liquid. This option is much more efficient than cold gas (280-320 s) but needs a more complex and heavier propulsion subsystem. Lastly, propulsion can be achieved electrically [19]. There are several ways to achieve this but the basic principle is that electricity gathered by the solar panels is used to charge the particles of an on-board propellant, that, when exposed to an electric field, accelerate and are expelled out of the spacecraft. This kind of propulsion is by far the most efficient (300-3000 s), but produces a very small amount of thrust, making orbit evolution relatively slow.

Guidance, Navigation and Control

The Guidance, Navigation and Control (GNC) subsystem can be divided into two separate sections [12]: Orbit Determination and Control Subsystem (ODCS) and Attitude Determination and Control Subsystem (ADCS). Orbit determination in CubeSat is often done using a GPS signal from the Global Navigation Satellite System (GNSS), while orbit control is done by applying thrust on the spacecraft, through one of the methods described in Sec. 2.1.2. The ADCS is, in fact, the object of this thesis. It is responsible for, firstly, determining where the satellite is facing and then pointing (controlling) it in the right direction when required. Attitude determination gathers attitude data from *attitude sensors*. This can be done in various ways, combining many different kinds of readings from various sensors, depending on the mission the CubeSat is supposed to perform. Attitude control, points the satellite in the desired orientation by means of *attitude actuators*. This can also be done in various ways, depending primarily on the mission requirements, and secondly, on how tight the mass budget of the satellite is and what kind of orbit it is on. Usually two or more kinds of actuators are found on satellites, giving it more controlability and pointing accuracy. Since this topic is crucial to this thesis, it is covered in greater detail in Sec. 2.2.

Communications

The Communications subsystem [15] consists of the components that create a link between the satellite and the Earth (or rather a ground station on it), and is one of the most essential subsystems in any spacecraft and CubeSats are no exception. The main reason why this subsystem is so important is because even if everything else is fully functional and the satellite is gathering immensely important data, if it can't send it to the Earth, engineers can't see it. Consequently, all payload and telemetry data is rendered useless, jeopardizing the mission itself. To operate a communicating satellite successfully, it is necessary to have a ground station. A ground station is the assembly of the equipment necessary to communicate with a satellite, with most satellites operating using professional ground stations. In the case of Europe based satellites, the ground stations in ESA's Ground Station Network, ESTRACK [20], are the most commonly used. Besides the obvious payload data, it is often also required that spacecraft transmit housekeeping data that show engineers on Earth if everything is going according to plan [21]. It is also interesting to note that the communications subsystem is one of the most underdeveloped parts of the CubeSat industry, in that it is very difficult to achieve data rates high enough to send all the essential data gathered in the CubeSat mission to the Earth [12].

Command & Data Handling

The Command & Data Handling (C&DH) Subsystem is composed of an On-Board Computer (OBC) and it is the part of the spacecraft that is responsible for decoding signals sent and received by other components. It performs the necessary computations that provide the operation sequences to be performed by the different subsystems. It also has the important mission of storing payload and housekeeping data, before encoding it and sending it to the Communications subsystems that can send it to the Earth. OBCs can have many configurations of data handling systems, altering the architecture of how they process data, and that means there are several options for these configurations, depending on the mission requirements in question. This subsystem is also an integral part of this thesis and is discussed further in Sec. 2.3.

Thermal Control

Due to the dire and often unforgiving characteristics of space, spacecraft can experience intense shifts in temperature [12, 15] with temperature changes ranging from about -100°C to 100°C, depending mainly on sunlight conditions. This is extremely prejudicial to the well being of the CubeSat since the kind of micro components it uses are often very sensitive to both high and low temperatures. This means that a satellite must be able to cope with intense heat and intense cold using the same subsystem. This can be done actively, using electrically powered heaters and coolers, although that is not often chosen by CubeSat developers, due to the added power consumption in such a small spacecraft. The most common type of thermal control used in CubeSats is passive thermal control. Passive thermal control is achieved by insulating components, using reflective surfaces, sun shields, and thermal coatings, among other options. Additionally, missions are often designed in a way that exploits the sunlight conditions of the spacecraft's orbit to their benefit, offering a thermal control system that is, in fact, free.

Thermal control for the 3-AMADEUS project is discussed in [22].

2.1.3 State of the Art CubeSat Missions

In this section, two missions are analyzed. The first one is ITU pSAT I, a mission that, like 3-AMADEUS, is a technology demonstration mission for ADCS components.

Despite it being a relatively old mission (launched in 2009), it is quite similar to the 3-AMADEUS mission, and gives a good insight at how these kinds of missions operate to this day, despite the technology demonstrated no longer being of great relevance. The second mission that is discussed is MarCO, a landmark CubeSat mission, that shows the true potential of this kind of spacecraft.

ITU pSAT I

ITU pSAT I [23] is a satellite that has been designed by students at the Istanbul Technical University (ITU). It is Turkey's first student designed picosatellite. The CubeSat has been launched on September 23rd 2009, aboard a PSLV C14 launch vehicle, from Satish Dhawan FLP in India. It has been placed in an almost circular orbit with an altitude of 715 km and an inclination of 98.31°, meaning that it is sun-synchronous.

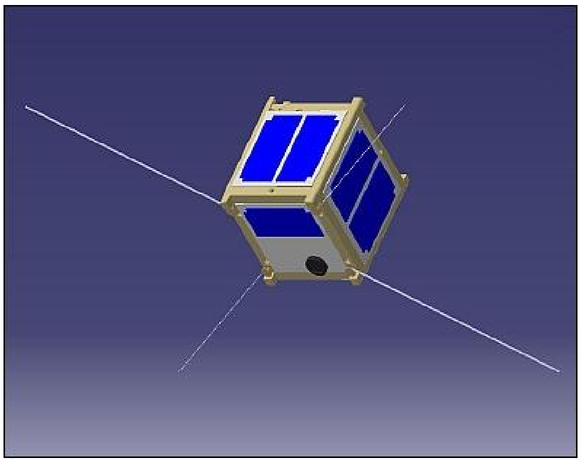


Figure 2.7: 3D render of ITU pSAT I [24].

ITU pSAT I is a technology demonstrator satellite, featuring two payloads: a low resolution camera and a two-axis passive stabilization experiment. ITU pSAT I's ADCS features a magnetorquer rod as an actuator and multiple sensors: three gyros, three accelerometers and a three-axis magnetometer. The OBC is a FM430 flight module, featuring a MSP430 microcontroller, also present on 3-AMADEUS' OBC, although the Abacus OBC also has a FPGA module. FM430's microcontroller has several buses (in this context, a bus is a system that transfers data between electronic components), including I^{2}C, SPI and UART, with most subsystems being controlled by the I^{2}C bus. Regarding the payload, it is interesting to note that the magnetometer data is used to correct the inertial drift as well as the inherent bias. The most important part of this mission is that that the attitude data is sent back to the Earth. This is done by grouping it into packets and sending it to the OBC via the I^{2}C bus. The OBC, in its turn formats the signal for downlink by the Communications subsystems. The satellite is still operational to this day.

MarCO

MarCO (Mars Cube One) [25] was a deep space NASA CubeSat mission that operated alongside *Insight*, a Mars lander mission, providing operational support. The MarCO mission, launched on December 29th 2018, consisted of two 6U CubeSats (MarCO-A and MarCO-B) that flew to Mars with Insight. The CubeSats stayed in Mars' orbit while Insight prepared for landing. The MarCO CubeSats served as information relays, providing Insight with useful telemetry information while it landed. While having the information relayed to Insight was of great utility, the true importance of the MarCO mission was to test and show the potential of using CubeSats on deep space missions, proving not only that they can work there on their own, but also provide important assistance to other larger deep space missions.

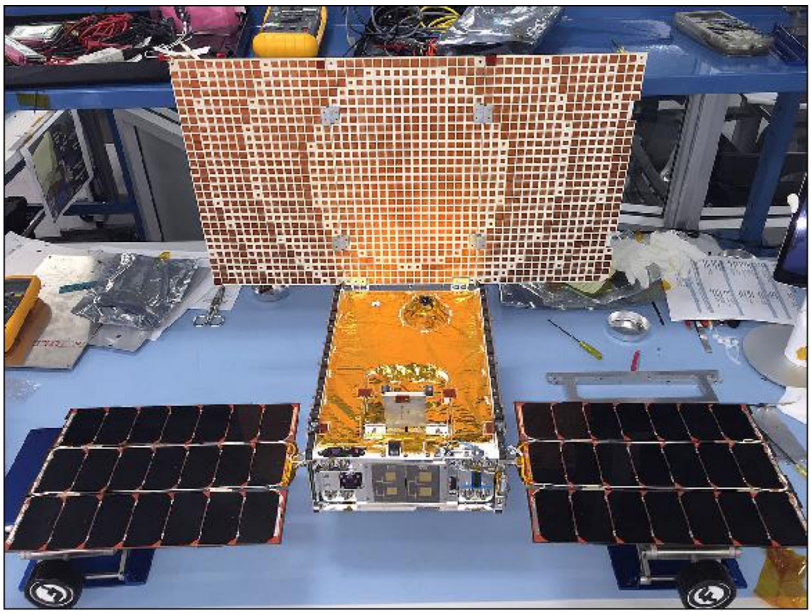


Figure 2.8: One of the MarCO CubeSats during production [26].

The payload of the MarCO CubeSats was a radio, more concretely the Iris v2 radio, an X-band transceiver that includes a UHF receiver [27] that was used to receive telemetry data sent from Insight during its landing. The ADCS for the MarCO mission was composed of a star tracker, a gyro, and coarse sun sensors for sensors and three-axis reaction wheels for actuators. In terms of C&DH, the MarCO CubeSats used an OBC based on the one used in the INSPIRE mission, featuring an MSP430 microcontroller. The software implemented in this OBC had a key feature in the fact that it allowed for fault detection and response to those faults within the satellite subsystems.

2.2 Attitude Determination and Control Subsystem

The *Attitude Determination and Control Subsystem* is, as previously mentioned, the subsystem responsible for satellite orientation identification and control. This subsystem can be divided into sensors and actuators. In this section, the most common types of components for this subsystem are analyzed in greater detail, as well as the combinations of those components that are often found on satellites nowadays.

2.2.1 Sensors

The ”Determination” part of ADCS refers to the ascertaining of the orientation of the satellite, or rather, its attitude. Attitude determination is done with the aid of *Attitude Sensors*. In space, some physical parameters are directly dependent on the attitude of the satellite, meaning that, by getting a reading of one of those parameters, it is possible to compare them with known specific values for specific attitudes and infer the current attitude. Most sensors work in this fashion, and the most common examples are described next [28, 29].

Sun Sensors

Although there are multiple types of *Sun Sensors* the working principle is valid across all types [30]. In typical sun sensors, an array of photosensitive cells detects the sunlight intensity in each cell, yielding a signal that is later translated into a sun vector indicating the sun’s direction. If one knows in what direction the sun is and has access to additional attitude data, it is possible to determine the satellite’s attitude with precision.

Figure 2.9: NSS Fine Sun Sensor, a CubeSat optimized sun sensor [31].

Sun sensors can be a great choice, depending on the mission they are implemented in. While they are extremely light and low power consuming, their biggest flaw is that, quite obviously, they don’t work in eclipse conditions. If a satellite is to go on a sun synchronous orbit that is constantly lit by sunlight, sun sensors are a great option. On the other hand, a mission with a large eclipse period needs to extrapolate its attitude in eclipse conditions, which may be dangerous and require an extra set of sensors.

Star Trackers

Star Trackers are attitude sensors that consist of a camera or photocell that image the sky, as seen by the spacecraft's perspective. Using a star catalog along with data about the spacecraft's spatial position, the star tracker can identify which stars are which. Considering a data base that is uploaded to the satellite, the star tracker relates the stars' sizes and positions in relation to one another to ascertain where the spacecraft is facing, thus determining its attitude.

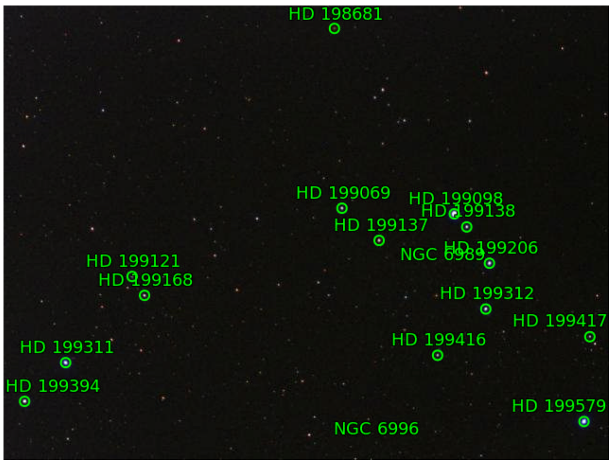


Figure 2.10: A star tracker's "view" [32].

One of star trackers' [30] biggest advantage is that they can determine a spacecraft's attitude with a fair degree of reliability, even in eclipse conditions. On the other hand, they usually weigh up to 300g and consume up to 1W, a lot more than, say, a three-axis magnetometer. Despite being one of the most reliable options for attitude sensing, star trackers also have some risk of failure since they can give bad readings when light is reflected on the satellite itself, as well as having a multitude of optical errors associated with their cameras.

Horizon Sensors

Horizon Sensors, or when Earth based, Earth Sensors, are attitude sensors often found in spacecraft. Their working principle is that by using infrared sensors (some Earth sensors use sensors in the visible spectrum but that is not optimal since it doesn't work in eclipse conditions), they can detect where the Earth's horizon is, in relation to the satellite, giving its attitude [13, 33].

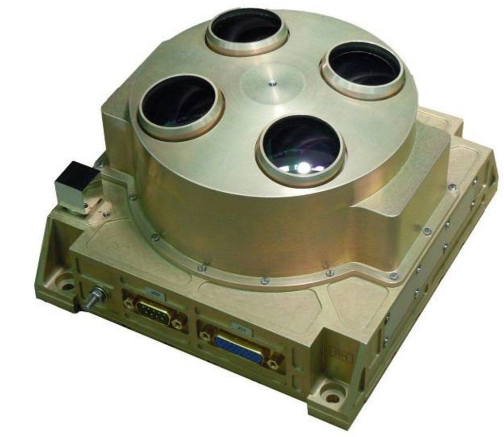


Figure 2.11: An IRES-C infrared Sun Sensor [34].

Much like star trackers, horizon sensors give a fair estimate of the satellite's attitude. In general they share the same advantages and disadvantages, save for the fact that star trackers are more accurate and more expensive, while consuming more power and weighing more. When accuracy is not of the uttermost importance, horizon sensors are a very valid option. Their obvious constraint is that they can't be used when not orbiting a planet, and don't work as well in elliptic orbits.

Gyroscopes

Gyroscopes are a particularly interesting attitude sensor, in that they sense the angular rate in each of satellite's axis, rather than the attitude itself. Gyros are seldom, if not never, used by themselves since they don't provide attitude data. If coupled with another sensor that does yield attitude data, gyros can be a very powerful component to increase accuracy [13, 30].

Magnetometers

The use of *Magnetometers* is central to this thesis, and consequently, the operation and characteristics of this components are explored more thoroughly. As it is widely known, planet Earth has a magnetic field around it. This field can be schematically represented by Fig. 2.12. It is worth noticing that due to the effect of the Sun on this field, Fig. 2.12 depicts the shape of the geomagnetic only in the close vicinity of the planet Earth, since outside of it the effect of the Sun's pull disrupts its symmetry.

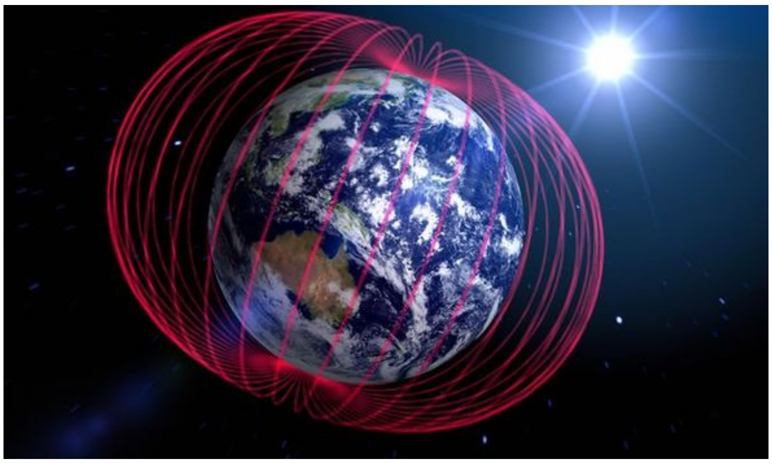


Figure 2.12: The Earth's Magnetic Field [35].

The interesting aspect of this field is that, from years of studying it, scientists on Earth have developed good models for the geomagnetic field induction vector. Since that vector can be calculated a priori, when a satellite is in a known position, the offset between the reading in a magnetometer and the expected reading at that point (in an Earth-bound reference frame) gives relevant data on the attitude of the satellite that can be coupled with other sensor data to yield a precise estimation of the satellite's attitude [13, 30, 36]. This process can also be reversed to give a satellite's position, considering a known attitude. Considering that, a three-axis magnetometer is simply a sensor of the magnetic field induction in each axis.

Due to their exceptionally low mass and power consumption, magnetometers are the most used sensors in the CubeSat industry. Despite all the advantages that make magnetometers such a popular choice, they still have their share of constraints. The most obvious one being that they can only operate in geocentric orbits, rendering them unusable for deep space missions. Another issue with magnetometers is how easy it is to disturb the readings with the dipole moment of the magnetorquer (a common actuator that is discussed in Sec. 2.2.2), as well as with any ferromagnetic materials found within the satellite. A typical magnetometer shouldn't weigh over 0.1 kg and should consume around 0.75W.

2.2.2 Actuators

Actuators are related to the "Control" part of ADCS since they control the satellite attitude. Often, a control law is implemented into the spacecraft's OBC. In this case, the OBC, considering inputs for the attitude sensors, outputs the actuation necessary to achieve the desired attitude. The way in which that actuation can be achieved is, quite intuitively, through actuators. The kind of actuators that is to be used depends on various mission details, with many combinations of these factors leading to a different kind of actuator or even to different kinds of actuators implemented simultaneously. A description of the most common attitude actuators as well as their functioning follows.

Reaction Wheels

Reaction Wheels are one of the most common actuators found in CubeSats. They are accurate and easy to implement, with the working principle being quite simple: by spinning a physical wheels of a certain mass and radius, through conservation of angular momentum, the satellite responds by increasing its angular velocity in the opposite direction [15]. To achieve control on all three axis, at least three wheels are needed, however, configurations of four wheels are often used to achieve some fault tolerance in the satellite's design [29]. One of the biggest issues with reaction wheels is that they may become saturated: if one of the wheels reaches its maximum angular velocity, it can no longer produce torque on that axis, which is problematic. To overcome this problem other actuators are often used to desaturate the reaction wheel. The other important issue with reaction wheels is how heavy they are, considering CubeSats' mass limitations. The weight of reaction wheels can vary depending on the torque output that is required but their performance/weight ratio is low when compared to other actuators. Control algorithms using solely magnetic actuators such as the one present in this thesis intend to solve this issue.

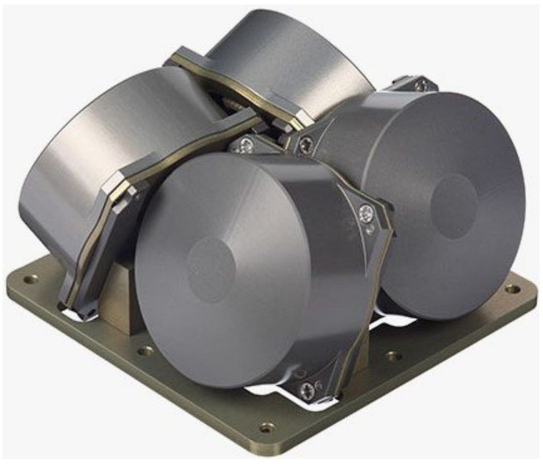


Figure 2.13: A tetrahedral configuration for four reaction wheels. [37]

Momentum Wheels

Momentum wheels are very much like reaction wheels, in that they are spinning fly wheels that control a spacecraft's attitude. The main difference is in how they are operated: In the case of momentum wheels, the wheel is spinning constantly, leading to a much more stable situation. Using momentum wheels, disturbance torques on axes perpendicular to the momentum wheel's spin axis cause the satellite's wheel induced spin axis to tilt slightly, rather than causing an unwanted rotation.

Thrusters

Thrusters are the simplest, most intuitive way to achieve attitude control on a satellite. The system however, is by far the heaviest, as it requires propellant as well as the thrusters themselves. A possible more efficient implementation of thrusters would be on a spacecraft that already features a propulsion system to begin with. One factor in favour of thrusters is the fact that they should work under any circumstances, should there be sufficient propellant [29, 15].

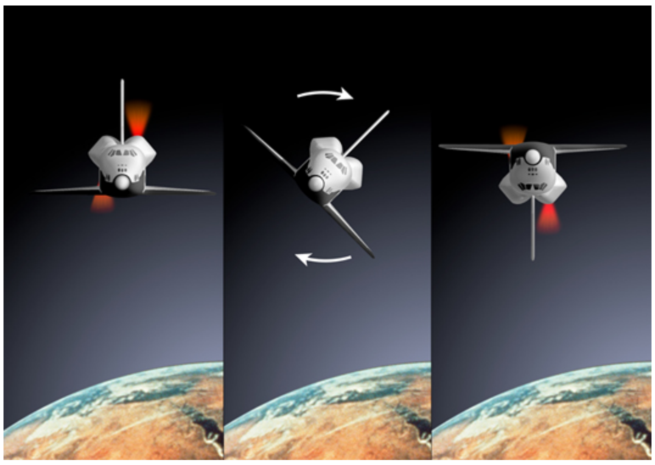


Figure 2.14: Depiction of thrusters working in the Space Shuttle [38].

Thrusters simply push gas out of the spacecraft, generating as opposing force. Thrusters should be placed near the edges of a spacecraft, so that the force is exerted as far away from the spacecraft's center of mass as possible, maximizing the produced torque.

Magnetorquers

Like magnetometers, magnetorquers too are key to this work, so a more in-depth analysis is made. Magnetorquers work in a simple fashion (simplicity is common in CubeSat subsystems). They are merely a metallic coil, through which a current flows, generating a magnetic dipole and respective torque that forces the magnetorquer rod to align itself with the magnetic field vector. Magnetorquers are a lightweight, low power consuming component, that is widely used in the Small Sat sector, often along one or more reaction wheels [15].

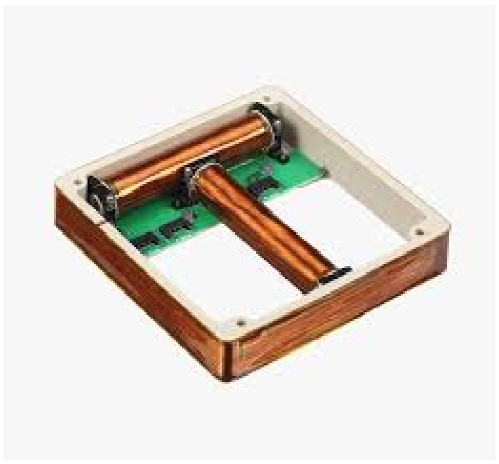


Figure 2.15: A three axis Magnetorquer CubeSat board [39].

This need for a reaction wheel comes from the biggest issue related to magnetorquers: underactuation [40]. Considering a reference frame with one axis aligned with the geomagnetic field vector and the other axes having no magnetic induction, any magnetic dipole vector generated by a set of magnetorquers will result in a torque vector that is always zero on the plane of the geomagnetic field. This means there are three degrees of freedom for the satellite, but only two that can be actuated upon, meaning that this system is *underactuated*. The other obvious disadvantage of this kind of actuator is that, like magnetometers, they are of no use outside beyond the Earth's vicinity. The torque they produce is also small when compared to other actuators.

2.2.3 State of the Art ADCS Designs

There are several options, both for attitude actuators and sensors alike, but what really makes an ADCS work is how different types of sensors and actuators can be put together to achieve a specific goal, while maintaining the power consumption and mass at a minimum. Besides the consumption and mass design criteria, when designing an ADCS, one can set a third indicator. Usually, this third indicator is either to maximize precision or to maximize long term performance [41].

One option for selecting ADCS configurations is the usage of integrated units. Integrated units are pre-assembled ADCS modules that spacecraft designers can acquire directly from the manufacturer. One of the biggest advantages of using this kind of module is that their performance is already validated, with many of them having considerable flight heritage, a crucial factor when selecting CubeSat components. According to [41], from fifteen selected CubeSats missions, seven of them have used integrated units. In this section, a few ADCS state of the art ADCS configurations, including integrated units, are described.

OPTOS

The OPTOS mission [42] is a 3U CubeSat project by the Spanish Space Agency (INTA) that launched in 2013. Its purpose is to test the in-flight capabilities of optical fiber communications as well as novel OBC designs. These designs are, like the one in the center of this thesis, FPGA based. Despite not being directly related to the payload, an ADCS is still required, although its accuracy and precision is not paramount to the success of the OPTOS mission.

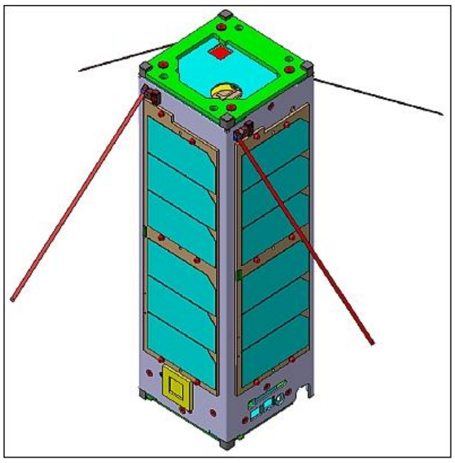


Figure 2.16: A 3D depiction of a fully deployed OPTOS CubeSat [42].

This mission's ADCS is not an integrated unit, but rather a selection of actuators and sensors. More concretely, it features a reaction wheel and five magnetorquers. The inclusion of several magnetorquers is due to their ability to desaturate the reaction wheel, the primary actuator. Additionally, they are used to point the satellite to the Earth when in observation mode. The sensors used are two sun sensors and a three-axis magnetometer. Additionally, its camera can be used as a sunlight sensor for attitude data.

TEMPEST-D

TEMPEST-D [43] is a project by the Colorado State University (CSU) consisting of a 6U CubeSat whose purpose is to demonstrate the capabilities of nanosatellites to perform weather monitoring.

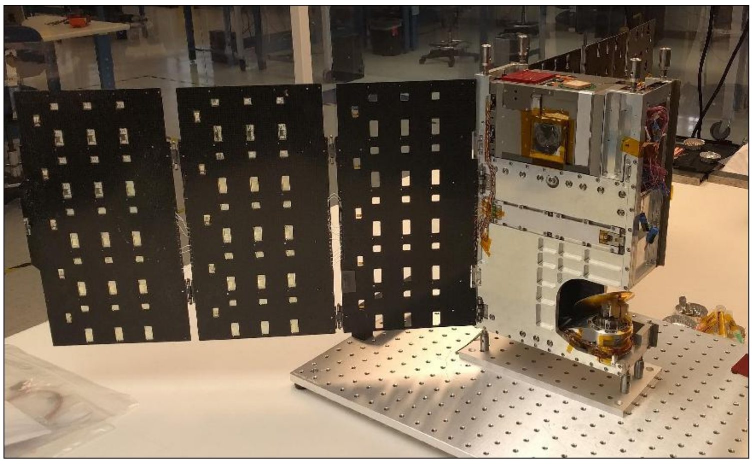


Figure 2.17: TEMPEST-D CubeSat during production [43].

Since this mission requires precise imaging, a precise and reliable pointing system is required, thus, the designers at CSU have opted for an integrated unit for this mission's ADCS. TEMPEST-D uses an XACT-50 ADCS module, produced by Blue Canyon Technology. The XACT-50 module features a star tracker, a gyroscope, a sun sensor, and a magnetometer as sensors, while its actuation is achieved via reaction wheels and magnetorquer rods. This module has an impressive five year life expectancy considering how precise it is.

Figure 2.18: XACT-50 integrated module [44].

DeOrbitSail

The DeOrbitSail CubeSat mission [45] has been developed at the Surrey Space Center (SSC) with the purpose of demonstrating a novel de-orbiting mechanism. This mechanism involves a sail being deployed, maximizing the drag effect of the Earth's atmosphere. In this particular mission, the ADCS is important because in order to maximize this effect, the sail should be perpendicular to the satellite's orbital velocity.

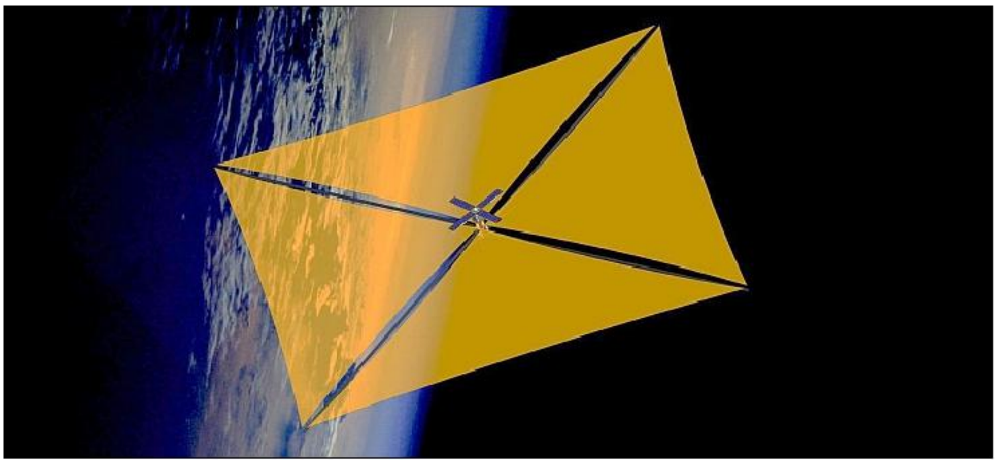


Figure 2.19: Artist's depiction of DeOrbitSail fully extended [45].

The designers at SSC, aiming for reliable and precise control have chosen to use a integrated ADCS module, CubeADCS. CubeADCS features a three axis magnetometer, a sun sensor yielding data on two axes and a nadir sensor yielding information on two axes as well, a coarse sun sensor and a rate sensor. A momentum wheel is installed as actuator, as well as magnetorquers. Additionally, the sail can extend in a manner that can also control rotation on one axis.

After the satellite's launch in 2015, there has been a failure in the ADCS system, that the developers think was caused by reaction wheel motor cables being disconnected. This failure has led to the inability to detumble (stop the satellite's initial spin rate after deployment) the satellite, making the sail deployment impossible. The expected de-orbit time is of about 20 years.

After looking at these state of the art concepts, one can see how magnetic components are extensively used in ADCS systems and consequently, how improving them is of the uttermost importance.

2.3 Command & Data Handling

Command & Data Handling [15] or *C&DH* is, like mentioned previously, the satellite's subsystem in charge of handling received and sent data, whether that communication is between components or between the satellite and a ground station. This subsystem consists of an On Board Computer, itself consisting of several sub-components.

2.3.1 OBC Overview

An OBC can usually be divided into three main sections [46]: the processing unit, the memory unit and the connecting interfaces. Many factors associated with these parts must be taken into account when designing or selecting an OBC, depending on what is required for the mission's successful completion. Next, the main factors and parameters are discussed.

Processing Capability

An OBC's Processing Capability is one of its most pivotal aspects since it affects how fast and how many processes can be run simultaneously in an OBC and how complex these processes can be. These processes can be of various natures, but in most satellites they are attitude control computations, energy management, coding and decoding radio signals, as well as controlling the payload. These tasks are handled by the OBC's Central Processing Unit (CPU), usually a microprocessor, that can be used standalone or within ASICs (Application-Specific Integrated Circuits) such as Microcontrollers (MCUs) that contain them, or even programmed inside Field Programmable Gate Arrays (FPGAs).

This thesis focuses on the implementation of an attitude control algorithm in the FPGA of an Abacus OBC, produced by Gauss Srl, so it is important to clarify the functioning and why the choice has been to implement the algorithm in the OBC's FPGA rather than on a microcontroller or a standalone microprocessor. This is discussed in Sec. 2.3.2.

Memory and Storage

In any sort of computer, in broad terms, memory means the ability to store data. Typically, memory is divided into Read-Only Memory (ROM) and Random Access Memory

(RAM). The main difference is that while ROM is non-volative, meaning that the data is still kept when the device is turned off (usually ROM is only used to store the program that boots the device), RAM reflects an OBC's ability to keep relevant data while running numerous and complex processes inside the OBC. This is incredibly important when a satellite requires strict management and coordination of all subsystems.

While memory contemplates the computer's capacity to keep track of tasks instantaneously, storage regards its capacity to hold data that it isn't using. This means that storage is pivotal in circumstances such as holding large payload data before sending it to the Earth.

Interfaces

A crucial aspect of any OBC is also its interfaces. Since most components are COTS, they all have different ways of communicating with other devices. Since the OBC is the center of all these communications, it is important that is has as many interfaces as possible. I2C, SPI, CAN, UART and USB are some of the most common interfaces found in this kind of component. The most common interfaces found in OBCs are discussed further in Ch. 3.

Size and Power Consumption

Like any other component, the lighter and least power consuming an OBC is, the better. Despite this, the OBC is central to the well-functioning of a satellite unlike any other component, meaning that it can never really be switched off, only set to power saving modes. Having this in consideration, it is fundamental to assess how these modes in an OBC can be tweaked and managed to prevent unnecessary energy consumption. Despite this energy manageability, depending on the mission, a smaller OBC with less processing power might be a better choice.

2.3.2 Field Programmable Gate Arrays

Field Programmable Gate Arrays or *FPGAs* are a type of integrated circuit that is mostly distinguishable from most other circuits of the sort for its capability to be programmed and reprogrammed after its manufacture [47, 48].

FPGA Overview

Traditionally, only major electronic companies could afford to design circuitry, and as a result, most designs were application specific, and consequently are usually called Application Specific Integrated Circuits (ASICs). Since these integrated circuits weren't customizable at all, for someone to produce a functioning design, large amounts of repetitive and often unused hardware used to be required. The creation of FPGAs - a device whose circuitry can be designed to fit the designer's needs - has come to solve this issue, however, it was not until recently that FPGAs became financially accessible to most companies [49].

What an FPGA is, in practice, is an array of hundreds of thousands or even millions of programmable logic blocks, that can be connected in different ways to produce different designs. More specifically, the fabric of FPGAs are logic gates, registers that hold data, and wires that connect them, with the typical arrangement looking something like Fig. 2.21, even though specific functional elements may be present.

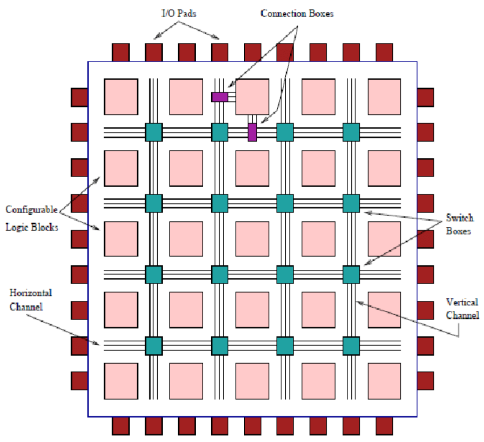


Figure 2.20: Typical FPGA architecture [50].

Usually, to program an FPGA device, one uses a Hardware Description Language (HDL). The most common languages used to program these type of devices are VHDL and Verilog, though some new languages such as OpenCL are also starting to be used.

FPGAs vs. ASICs

Electrical engineers nowadays are often posed with the question [47, 51]: should one use an FPGA or an ASIC for an embedded system design? As a result, it is important to know what exactly is the context for which one is more appropriate than the other, and why an FPGA is the most appropriate choice for this thesis' project.

While ASICs such as microcontrollers/microprocessors have built-in circuitry onto which a program can be loaded in order to perform the required tasks, an FPGA is much more flexible since, when it is programmed, the actual circuitry is changed, so that the necessary task can be performed. This allows for specific capabilities and much more design freedom. In fact, despite having typically lower clock counts (being slower), FPGAs can parallelize hardware, meaning that tasks can be performed concurrently, thus boosting the amount of data that can be treated per second. On top of that, FPGAs are field programmable, allowing for several programs to be uploaded onto the same product at different times. This makes it look like FPGAs would be preferable but what sets FPGAs back is that due to extensive hardware, they are quite power consuming, which is one of the limiting factors in OBC design and management. Apart from that, one must also consider the longer design time for an FPGA. As a result, the management of all the spacecraft subsystems for this mission apart from the ADCS is done by the microcontroller on the ABACUS OBC. For the ADCS, the employment of an FPGA is crucial since this mission is an ADCS technology demonstrator, in which several attitude control algorithms must be tested. Using an FPGA in this particular context is useful since it allows for the satellite to receive a bit-stream from Earth containing a new program that it can load, thus implementing a new control algorithm that can be tested as well.

2.3.3 State of the Art OBC Designs

In this section, a brief look is taken at two CubeSat OBCs so as to better grasp what current OBC technology offers.

IMT CubeSat On-Board Computer

The IMT CubeSat On-Board Computer [52] features a 200 MHz microprocessor (CPU), 16 MB of RAM, two channels for each of the following communication protocols: SPI, CAN Bus, I2C and UART, while it can also be connected by USB and JTAG while on Earth. This OBC weighs 38 grams and should consume about 300 mW of power. It also has a built in interface for a camera, which is a useful feature for Earth observation missions.

ISIS OBC

The ISIS OBC [53] in turn, has a 400 Mhz CPU as well as 64 MB of RAM. It features the same communication ports as the IMT OBC. This component weighs 76 grams and consumes 400 mW of power. An interesting characteristic of this OBC is that it features a three axis magnetometer as well as magnetorquer adapted interfaces to ease the communication with ADCS components.

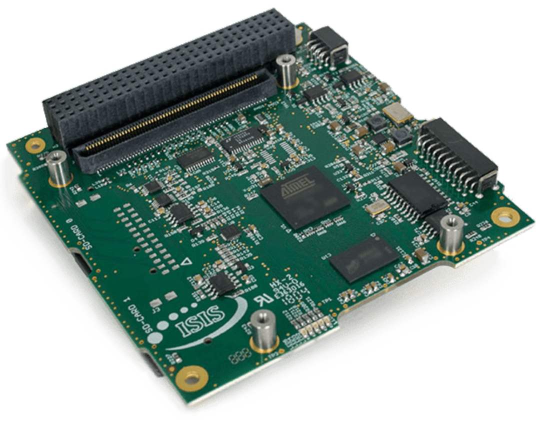


Figure 2.21: ISIS On-Board Computer [54].

It's interesting to see that the ISIS OBC has almost every parameter doubled in relation to the IMT OBC, while the weight and size is also doubled. One can assume that the price also increases significantly. This analysis goes to show that different options are available, and as a consequence it is important to assess what the mission requirements so as not to oversize or undersize such an important subsystem.

2.3.4 FPGAs in Satellite Attitude Control

FPGAs in Space

Before taking a look at specific examples of FPGAs being used in Attitude Control in CubeSats, it's relevant to discuss the many other applications that FPGAs have in the space industry. According to [55], the spaceborne reprogrammability of FPGA is a characteristic that is highly sought after in the space industry due to the hazardous conditions of space that often requires changes to the designer's initial plans.

One example of such applications is discussed in [56], where an FPGAs capabilities are exploited to enable optimized image compression. This is essential in satellites given the limited storage and transmission rates found in spacecraft.

The configurable aspect of FPGAs in this example is relevant in that, when it is compared with a software implementation, for example in a microcontroller, the FPGA implementation is much less complex. This simplicity largely reduces the risk of computational errors due to the effect of radiation found in the Earth's orbit.

The Flying Laptop

An example of an FPGA used in the ADCS of a CubeSat that actually flew can be found in the *Flying Laptop* mission [57, 58]. In this mission, an FPGA has been selected as a processing device since it allows for parallelization, making the required image processing faster. For the ADCS part, the attitude sensors and actuators communicate through RS-422 (UART), digital I/O lines, I^2 C and IBIS buses. The FPGA takes attitude inputs from the sensors via the aforementioned channels and calculates the necessary actuation via a control law, that is then sent to the actuators. This method is somewhat similar to the one used in the current work, as is discussed in greater detail in Ch. 7.

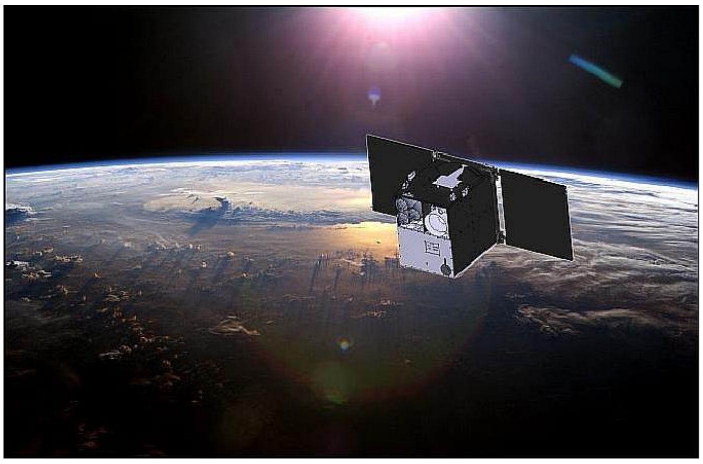


Figure 2.22: Artist's rendition of the Flying Laptop satellite [59].

A FPGA-based Approach to Attitude Determination for Nanosatellites

A final example of an FPGA based attitude control is proposed in [60], where all processing in the OBC is done by the FPGA. The reasoning behind this decision is that the FPGA can run all these processes parallelly, thus maximizing system performance. In this case, the employment of an FPGA regarding attitude control has more to do with the somewhat complex computation done when estimating the current attitude through a Spherical Simplex Unscented Kalman Filter. The computations are perfomed using single floating point precision. An algorithm named CORDIC is used to accurately represent trigonometric functions. The FPGA designers, similarly to the work done on this thesis, have implemented counters in order to employ iterative processes. The authors of this article claim that implementing the OBC's processing solely on the FPGA increases system performance to a level superior to even some desktop computers, due to how they can allocate resources.

Chapter 3

Literature Review

The Literature Review chapter aims at reviewing some of the theoretical concepts that are pivotal to the completion of this thesis' project. Firstly, a few notions of how satellites orbit the Earth are presented, as it is important to understand how some orbital parameters affect a satellite's attitude.

Additionally, the concepts and formulations that allow one to understand and model how a satellite behaves in terms of its attitude, as well as the effect that the space environment and attitude control torques have on such behaviour, are presented.

Finally, the digital systems notions required to understand the real life FPGA implementation of the control algorithm that allows for a HIL simulation are presented, namely how decimal and negative numbers are represented in a way that the FPGA can understand - binary - and how arithmetic can be performed with those numbers. Digital communication between systems such as an FPGA and a personal computer is also covered, while an overview of a typical FPGA's building blocks is also made.

3.1 Orbital Mechanics

3.1.1 Keplerian Orbits

If a body is in the vicinity of a celestial body of great mass, its trajectory about it is a conic section, called an orbit. In his time, Johannes Kepler proposed laws of planetary motion that describe an idealized orbit that are still used to this day in the aerospace sector. Modelling orbits mathematically allows one to parametrize several aspects of a specific orbit and predict orbital motion with a good degree of accuracy, due to the interfering forces existing in space being so small.

These orbits have a focus - the larger body being orbited - while the smaller body orbits around it in a motion that can be described through several types of conic sections, such as ellipses, circles (symmetric ellipses), parabolas, and hyperbolas [61].

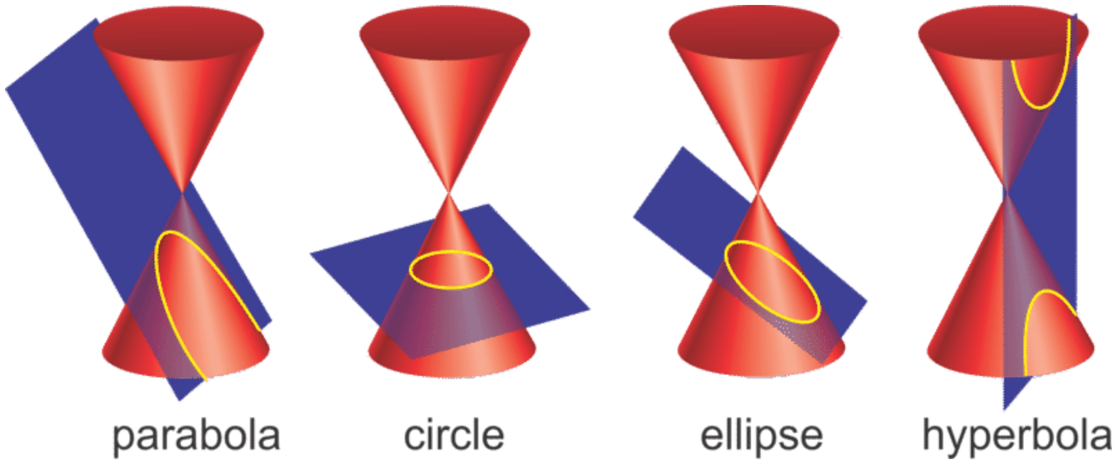


Figure 3.1: Conic Sections [62].

While circles and ellipses are limited by the central body, parabolas and hyperbolas describe the motion of a body that escapes its pull.

The position of an orbiting body in relation to the central one, in polar coordinates is then given by Eq. (3.1) [61], with an elliptical orbit being schematically represented by Fig. 3.2.

$$r = a\frac{1 - e^2}{1 + ecos\theta} \tag{3.1}$$

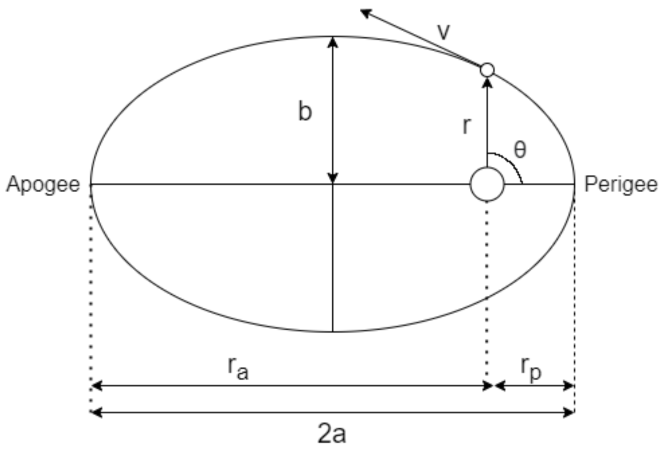


Figure 3.2: Diagram of a 2d elliptical orbit.

In Eq. (3.1), r, the polar radius, is the distance to the central body. e is the orbit eccentricity, and a is the semi major axis. θ is the true anomaly of a given position. It is important to note that the polar radius is the sum of the orbiting body's altitude at a given point and the Earth's radius. These concepts are discussed more thoroughly in Sec. 3.1.2.

It is important to discuss some more orbital concepts relevant in the context of attitude control before moving to three dimensional orbits [61, 63]. With that in mind, the orbital linear velocity for a circular orbit is given by $v = \sqrt{\frac{\mu}{a}}$, where μ is the Earth gravitation parameter discussed further in Sec. 3.3.3. The orbital period, the time it takes for the orbiting body to complete an orbit, is of course inversely proportional to the orbital velocity and is given by $T = 2\pi\sqrt{\frac{a^3}{\mu}}$.

3.1.2 Orbital Elements

The orbital parameters and Eq. (3.1) allow one to describe the motion of the orbiting body in two dimensions, within the plane of an orbit, but if one wants to describe a three dimensional orbit around a three dimensional central body such as the Earth, in total, six keplerian orbital elements are necessary [61, 64]. The first three parameters are used to describe the position of the satellite within the orbit and have been briefly described earlier, while the last three represent the orientation of the orbit plane and the position of the apsides line within that plane. This orientation is described in relation to a reference plane, that for Earth orbiting spacecraft is usually the equatorial plane [61].

Eccentricity

The *eccentricity* is a dimensionless value that represents the shape of the orbit in terms of conic sections: $e = 0$ describes a circular orbit where r is constant and $0 < e < 1$ represents an elliptical orbit. Finally $e = 1$ describes a parabola, while $e > 1$ describes a hyperbola. The remaining orbital parameters are described assuming an elliptical orbit.

Semi-Major Axis

The *semi-major axis*, a, is the descriptor of the orbit size. It is the average between the perigee and apogee radii. For a circular orbit, this parameter becomes simply the orbit radius.

True Anomaly

The *true anomaly*, θ gives the position of the satellite within the orbit. It is the angle between the radius vector at a chosen point and at perigee.

Inclination

The orbital inclination, i, is one of the parameters that describes the orientation of the orbital plane. It is the angle between the normal to the equatorial plane and the normal to the orbit plane. An inclination of 0° means that the orbit is equatorial, while an orbit with an inclination of 90° is called a polar orbit.

Longitude of the Ascending Node

The *longitude of the ascending node*, Ω, is the angle (measured eastwards) between the point of zero longitude and the ascending node. From the two points that are at the ends of the line that intercepts the orbital and reference planes, the ascending node is the one in which the satellite goes from the southern to the northern hemisphere.

Argument of Perigee

The final parameter that describes the orbit orientation is the *argument of the perigee*, ω. It measures a rotation of the orbit about its central body. This rotation is given by the angle between the ascending node and the radius vector at perigee, measured in the direction of motion. For circular orbits this parameter is, of course, meaningless.

Figure 3.3 illustrates the set of orbital parameters for an elliptical orbit in three dimensions.

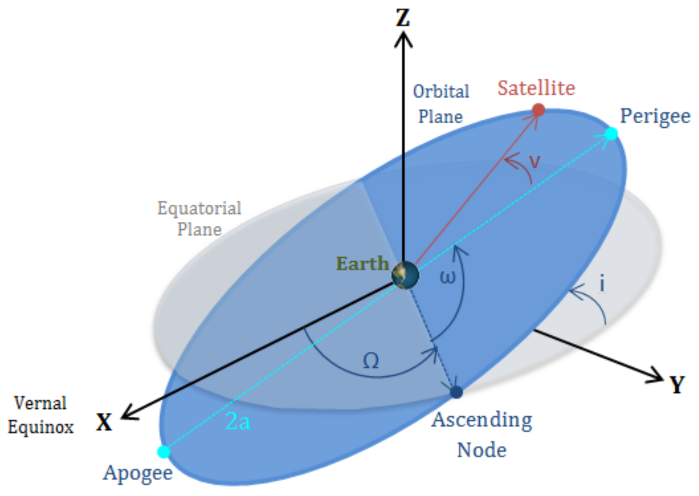


Figure 3.3: Orbital elements depicting an orbit in 3D [65].

3.1.3 Sun-Synchronous Orbits

There are several types of special orbits. One of these types is the geostationary orbit, in which a satellite stays above the same location on the Earth's surface, but one that must be discussed further, since it is the orbit type of the 3-AMADEUS mission, is the *Sun-Synchronous Orbit (SSO)* [66, 67]. An SSO, often called heliosynchronous orbit, is a kind of orbit that, like the name suggests, is synchronized with the sun. What this means is that the angle between the orbital plane and the sun's direction is kept constant. For this to happen, the orbit must precess, i.e., rotate, as the Earth rotates about the Sun, meaning that it needs to rotate about 1 degree per day. The aspect of SSOs that is exploited in this mission is, in fact, the sunlight conditions it allows for.

By placing a spacecraft in a special kind of SSO, the dusk-dawn SSO - that can be achieved by launching it at the right time - the satellite can ”chase” the sunset and sunrise at either side of the globe, making it pass at locations where it is dusk and then where it is dawn. What this means is that the satellite is always sunlit, maximizing the amount of solar energy received by the solar panels. This kind of orbit is ideal for satellites with very small solar array area, as is the case of the 3-AMADEUS CubeSat.

Finally, to compute the necessary orbital inclination leading to SSO orbital precession, given a certain orbit altitude, Eq. (3.2) can be used:

$$\cos(i) = -\left(\frac{a}{12352}\right)^{7/2} \quad (3.2)$$

where a is the orbit semi-major axis and i its inclination.

3.2 Attitude Parameterization

This section presents some of the various ways in which attitude can be represented that allow one to clearly understand an object's orientation. Initially, the references frames that are commonly used are addressed. Additionally, a few ways of describing the current orientation of a body in space, in relation to those reference frames are presented. In this thesis all reference frames used are right hand orthogonal systems.

3.2.1 Reference Frames

When one describes the attitude of an object, its orientation is always expressed in relation to a reference. In light of that, some of the reference frames that are usually used in spacecraft attitude control and are used in this thesis are presented [68]:

Earth-Centered Inertial (ECI) Reference Frame

The first reference frame discussed is the *Earth-Centered Inertial (ECI) Reference Frame*, denoted here with an index *i*. The ECI frame has its origin on the center of the Earth. The frame is fixed in the absolute space, and the Earth rotates about its i_z axis (which points Northwards) with an angular velocity of $\omega_e = \frac{2\pi}{T_e}$, where T_e is the Earth's rotation period. The i_x axis points toward the current epoch vernal equinox and the i_y axis completes a right-hand system.

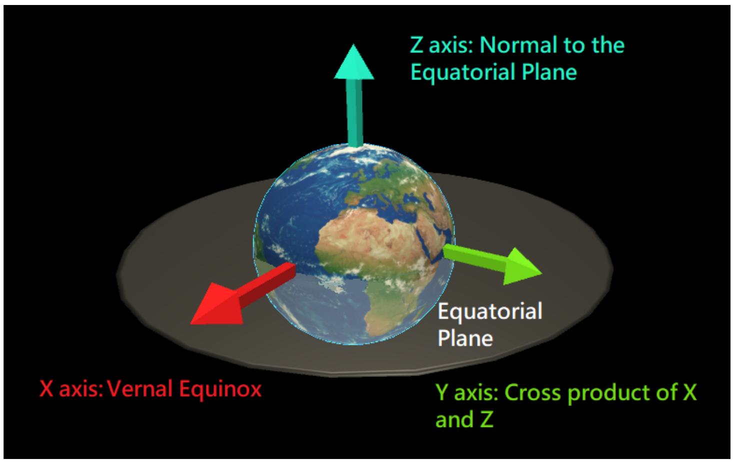

Z axis: Normal to the Equatorial Plane
Equatorial Plane
X axis: Vernal Equinox
Y axis: Cross product of X and Z

Figure 3.4: Representation of the ECI frame.

Orbit Reference Frame

The first of the two satellite-bound reference frames that are discussed is the *Orbit Reference Frame*. This reference frame is denoted with an index o. The chosen orbit frame considers a circular orbit and its center lies on the satellite's center of mass. The o_x axis points towards the satellite's motion (which is tangential to the orbit). The o_z axis points to the local vertical (also called zenith) and the o_y axis is along the orbit normal. This means that the orbit reference frame spins about its o_y axis with an angular velocity of $\omega_o = \frac{2\pi}{T_o}$, with T_o being the orbit period.

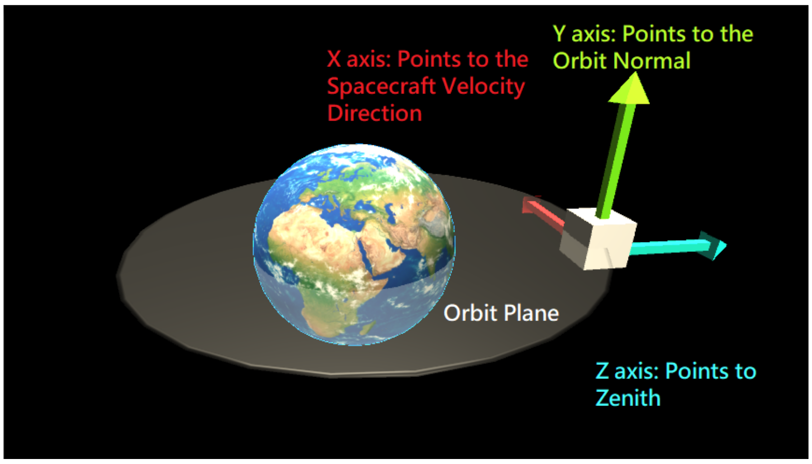

Y axis: Points to the Orbit Normal
X axis: Points to the Spacecraft Velocity Direction
Orbit Plane
Z axis: Points to Zenith

Figure 3.5: Representation of Orbit Reference Frame.

Body Reference Frame

The *Body Reference Frame* - denoted with an index b - shares its origin with the orbit frame and is attached to the satellite's principal inertia axes, that are addressed in Sec. 3.3.1. This means that this frame spins as the satellite spins. In this case it is chosen that the b_z axis is the axis of smallest moment of inertia, while the b_y axis is the axis of largest moment of inertia.

Now that all axes has been defined, it is paramount that one looks back at the definition of attitude and sees that the notion of attitude and the ways to represent it that are discussed in this chapter, in fact describe nothing more than the rotation of the body frame - the object - in relation to other frames (ECI or Orbit) - the reference.

3.2.2 Types of Attitude Parameterization

Some methods to describe attitude in a way that can represent all possible orientations are discussed in this section. While some methods to describe attitude have advantages over others - this is also discussed in this section - the quaternion representation is one of the most commonly used for simulation and mathematical modelling [28]. As a result, quaternions are used to compute this work's attitude model.

Rotation Matrix

Considering an arbitrary vector **v** in an arbitrary frame A so that:

$$\boldsymbol{v}^A = [v_x^A, v_y^A, v_z^A]$$

Consider now the same vector **v** but this time represented in a different reference frame B:

$$\boldsymbol{v}^B = [v_x^B, v_y^B, v_z^B]$$

there is a matrix R_A^B [69, 70] that can transform a vector in frame A into the same vector but with its components in frame B such that:

$$\boldsymbol{v}^B = R_A^B\, \boldsymbol{v}^A \quad (3.3)$$

where R complies with:

$$R \in R^{3x3},\ R^T R = I,\ det(R) = 1$$

also, the matrix that transforms vector **v** from frame B back to frame A is one such that:

$$R_B^A = R_A^{B^{-1}} = R_A^{B^T} \quad (3.4)$$

Such matrix is called a *Rotation Matrix*. Besides this interpretation, in which a frame is rotated to another frame - useful when interpreting vectors in the different frames discussed in Sec. 3.2.1 - a rotation matrix can also represent a rotation of a given vector within a frame. The first interpretation can also be viewed as a representation of the rotation of a frame in relation to another frame. A rotation matrix can also be called a Direction Cosine Matrix, in which its elements represent the cosine of the angle between the axes of two reference frames. If one chooses the body frame as one of those reference frames, the object's attitude/rotation of its body frame is represented.

Quaternions

Quaternions are a parameter that represents attitude/rotation of the body frame through a four dimensional vector. They do so through a three-dimensional sub-vector - an axis about which the reference frame must be rotated in order to get the rotated body frame - and lastly, a one-dimensional vector - the angle of rotation about that axis. The usage of quaternions is recommended since they have no singularities and usually require less computational power, despite the rotation they represent being harder to visualize directly from their parameters [71, 36]. The two parts of a quaternion are then represented as:

$$\varepsilon = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}; \ \eta = q_4 \quad (3.5)$$

so that,

$$\boldsymbol{q} = \begin{bmatrix} \varepsilon \\ \eta \end{bmatrix} = \begin{bmatrix} \boldsymbol{a} \ \sin(\frac{\phi}{2}) \\ \cos(\frac{\phi}{2}) \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \quad (3.6)$$

where $\boldsymbol{a}$ is the Euler axis vector (the one about which a reference frame must be rotated) and ϕ is the angle of rotation (how much it has spun).
To convert from a quaternion rotation notation to the rotation matrix representation one uses [69, 36]:

$$R = I + 2\eta S(\varepsilon) + 2S(\varepsilon)^2 \quad (3.7)$$

where I is the identity matrix and S is the skew-symmetric operator for a three dimensional vector. This means that $S(\varepsilon)$ is given by Eq.(3.8).

$$S(\varepsilon) = S([q_1, q_2, q_3,]^T) = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \quad (3.8)$$

If the quaternions represent a rotation of a frame B in relation to a frame A, the rotation matrix that Eq. (3.7) yields is the rotation matrix that relates the components of a vector in frame B to the components of the same vector in frame A, R^A_B.

Euler Angles

The final way of representing rotations that is discussed here are Euler Angles [71, 72]. To represent rotations (either of reference frames or of vectors), Euler angles interpret them as a sequence of three different sub-rotations about each axis. Three angles represent the sequence of rotations necessary to achieve the complete rotation. The angles naturally depend on the rotation sequence that is chosen.

Considering that consecutive rotations about the same axis and negative rotations about an axis are meaningless then there are only twelve different valid rotation sequences: ZXZ ; XYX ; YZY ; ZYZ ; XZX ; YXY and XYZ ; YZX ; ZXY ; XZY ; ZYX ; YXZ

The angle of rotation about the X axis is commonly referred to as *roll*, ϕ. About the Y axis it is usually called *pitch*, θ, while about the Z axis it is *yaw*, ψ.

The corresponding rotation matrix can be obtained by multiplying matrices of each one of the single rotations. If one assumes the representation of the rotation of frame B in relation to frame A by a (ψ, θ, ϕ) set of Euler Angles, in the ZYX rotation sequence (for example) with the rotation matrix for each of the rotations being:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (3.9)$$

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

then the corresponding rotation matrix representation is:

$$R_B^A = R_z(\psi)R_y(\theta)R_x(\phi) \quad (3.10)$$

$$R_B^A = \begin{bmatrix} \mathrm{c}(\theta)\mathrm{c}(\theta) & \mathrm{c}(\theta)\mathrm{s}(\theta)\mathrm{s}(\phi) - \mathrm{s}(\theta)\mathrm{c}(\phi) & \mathrm{c}(\theta)\mathrm{c}(\phi)\mathrm{s}(\theta) + \mathrm{s}(\theta)\mathrm{s}(\phi) \\ \mathrm{s}(\theta)\mathrm{c}(\theta) & \mathrm{s}(\theta)\mathrm{s}(\theta)\mathrm{s}(\phi) + \mathrm{c}(\theta)\mathrm{c}(\phi) & \mathrm{s}(\theta)\mathrm{s}(\theta)\mathrm{c}(\phi) - \mathrm{c}(\theta)\mathrm{s}(\phi) \\ -\mathrm{s}(\theta) & \mathrm{c}(\theta)\mathrm{s}(\phi) & \mathrm{c}(\theta)\mathrm{c}(\phi) \end{bmatrix} \quad (3.11)$$

or in quaternions [73]:

$$\boldsymbol{q} = \begin{bmatrix} \mathrm{s}(\frac{\phi}{2})\mathrm{c}(\frac{\theta}{2})\mathrm{c}(\frac{\psi}{2}) - \mathrm{c}(\frac{\phi}{2})\mathrm{s}(\frac{\theta}{2})\mathrm{s}(\frac{\psi}{2}) \\ \mathrm{c}(\frac{\phi}{2})\mathrm{s}(\frac{\theta}{2})\mathrm{c}(\frac{\psi}{2}) + \mathrm{s}(\frac{\phi}{2})\mathrm{c}(\frac{\theta}{2})\mathrm{s}(\frac{\psi}{2}) \\ \mathrm{c}(\frac{\phi}{2})\mathrm{c}(\frac{\theta}{2})\mathrm{s}(\frac{\psi}{2}) - \mathrm{s}(\frac{\phi}{2})\mathrm{s}(\frac{\theta}{2})\mathrm{c}(\frac{\psi}{2}) \\ \mathrm{c}(\frac{\phi}{2})\mathrm{c}(\frac{\theta}{2})\mathrm{c}(\frac{\psi}{2}) + \mathrm{s}(\frac{\phi}{2})\mathrm{s}(\frac{\theta}{2})\mathrm{s}(\frac{\psi}{2}) \end{bmatrix} \quad (3.12)$$

where s and c represent the sine and cosine trigonometric functions.

Despite being a seemingly attractive form of attitude representation, since they are more intuitive than quaternions, Euler angles have a major issues that makes quaternions best for some applications. Using Euler angles, if two axis of rotation align (in the ZYX sequence that happens if $\theta = \pm\frac{\pi}{2}$, as Z and X are aligned), rotating about one of these axis produces the same result as rotating about the other. This means that in such a position it is impossible to unambiguously assert the Euler angles from a given attitude. This phenomenon is a singularity called *Gimbal Lock* that is avoided by applying it in vehicles that mainly spin on two axes like a boat, or by using another attitude representation in the vicinity of the positions that cause this problem [74].

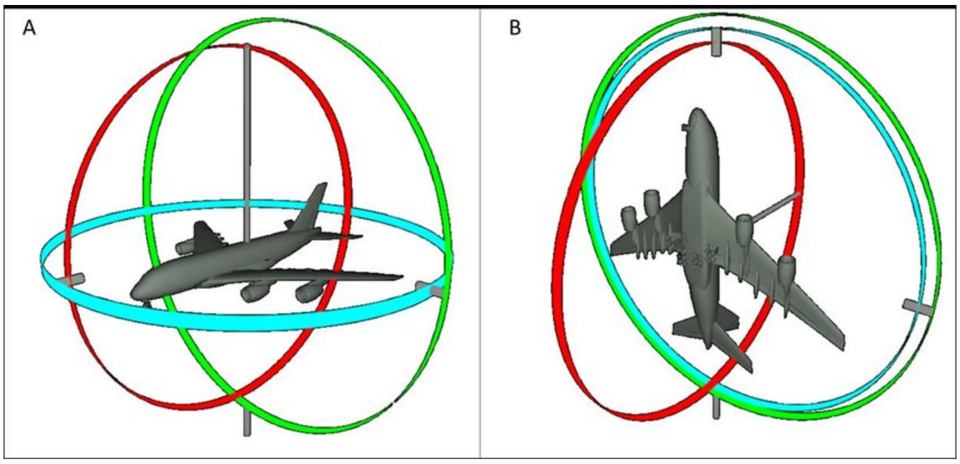


Figure 3.6: Normal Euler angle representation (left) and a Gimbal locked representation (right) [75].

3.3 Attitude Dynamics

Attitude Dynamics refers to how an object rotates and how exactly its attitude changes through time, given how it is spinning. Before discussing those aspects it is important to review what physical characteristics of an object affect the attitude dynamics. Afterwards, the set of equations that rule a satellite's attitude dynamics are presented, in order to establish how different attitude parameters affect each other and how such dynamics can be simulated.

3.3.1 Angular Momentum

A body's *Angular Momentum*, here represented by the letter L, measures a rotating object's tendency to keep rotating. For an object with mass δm, it is given by: [76, 77]:

$$\boldsymbol{L} = \int \boldsymbol{r} \times \boldsymbol{v} \; \delta m \quad (3.13)$$

where m is an object's mass, v its velocity vector and r its position vector in relation to the point used for the evaluation of moments of vector quantities. One also knows that if a body is rotating about its center of mass, the velocity of every mass element can be decomposed into:

$$\boldsymbol{v} = \boldsymbol{\omega} \times \boldsymbol{r} \quad (3.14)$$

where ω is the angular velocity. One can also state that:

$$\boldsymbol{L} = \int \boldsymbol{r} \times (\boldsymbol{\omega} \times \boldsymbol{r}) \;\; \delta m \quad (3.15)$$

if

$$\boldsymbol{r} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \text{ and } \boldsymbol{\omega} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$$

then

$$\boldsymbol{L} = \int \begin{bmatrix} (y^2+z^2)\omega_1 & -(xy)\omega_2 & -(xz)\omega_3 \\ -(xy)\omega_1 & (x^2+z^2)\omega_2 & -(yz)\omega_3 \\ -(xz)\omega_1 & -(yz)\omega_2 & (x^2+y^2)\omega_1 \end{bmatrix} \delta m \quad (3.16)$$

or, since the angular velocities are independent of the position vector,

$$\boldsymbol{L} = \int \begin{bmatrix} (y^2+z^2) & -(xy) & -(xz) \\ -(xy) & (x^2+z^2) & -(yz) \\ -(xz) & -(yz) & (x^2+y^2) \end{bmatrix} \delta m \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \quad (3.17)$$

calculating the integral and replacing the resulting matrix with **I**, which is called *Inertia Matrix* or *Inertia Tensor* from here on out, one gets:

$$\boldsymbol{L} = \boldsymbol{I\omega} \quad (3.18)$$

in which **I** is symmetric:

$$\boldsymbol{I} = \begin{bmatrix} \int(y^2+z^2)\delta m & \int -(xy)\delta m & \int -(xz)\delta m \\ \int -(xy)\delta m & \int(x^2+z^2)\delta m & \int -(yz)\delta m \\ \int -(xz)\delta m & \int -(yz)\delta m & \int(x^2+y^2)\delta m \end{bmatrix} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \quad (3.19)$$

The diagonal elements of the I matrix are called the *moments of inertia* while the off-diagonal elements are called *products of inertia*. Since the matrix $\boldsymbol{I}$ is real and symmetric, its eigenvalues are real and one can always select three eigenvectors that are mutually orthogonal. This premise leads one to conclude that there exists a body reference frame F, in which the inertia tensor becomes diagonal with elements that are the eigenvalues of $\boldsymbol{I}$ [76]. The resulting tensor is often denoted by $\boldsymbol{J}$, as shown in Eq.(3.20).

$$\boldsymbol{J} = \begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix} \quad (3.20)$$

In fact, if frame F has its origin in the body's the center of mass, its axes are called *central principal axes of inertia* and the respective moments of inertia become *central principal moments of inertia*. The satellite body frame used throughout this thesis is the reference frame of central principal axes of inertia.

In this work, it is considered that the CubeSat is a uniform parallelipipede with sides

a, b, and c. Considering that, its products of inertia are 0 and its moments of inertia are:

$$\begin{aligned} I_{xx} &= J_{xx} = \frac{m}{12}(b^2 + c^2) \\ I_{yy} &= J_{yy} = \frac{m}{12}(a^2 + b^2) \\ I_{zz} &= J_{zz} = \frac{m}{12}(a^2 + c^2) \end{aligned} \quad (3.21)$$

3.3.2 Equations of Motion

Now that the physical parameters that affect the attitude dynamics of a body have been described, it is time to introduce firstly the dynamic equations, and later the kinematic equations, the two equations that govern a satellite's attitude behaviour.

Dynamic Equations

Before discussing the dynamic equations, it is important to review how a satellite's angular velocity can be represented. Across this thesis, angular velocities are represented in the body reference frame. They either represent the rotation of the body frame in relation to the ECI frame, the *absolute* angular velocity, or in relation to the orbit frame, the *relative* angular velocity. In the chosen notation ω_{ib}^b means the angular velocity of the body frame in relation to the ECI frame, in the body frame, with the relative velocity being ω_{ob}^b, the angular velocity of the body frame in relation to the orbit frame, in the body frame.

$$\begin{aligned} \boldsymbol{\omega_{ob}^b} &= [\ \omega_{ob_x}^b, \omega_{ob_y}^b, \omega_{ob_z}^b\ ] \\ \boldsymbol{\omega_{ib}^b} &= [\ \omega_{ib_x}^b, \omega_{ib_y}^b, \omega_{ib_z}^b\ ] \end{aligned} \quad (3.22)$$

To get $\boldsymbol{\omega_{ib}^b}$ from $\boldsymbol{\omega_{ob}^b}$, Eq. (3.23) is used [68]:

$$\boldsymbol{\omega_{ib}^b} = \boldsymbol{\omega_{ob}^b} + \boldsymbol{R_O^B}\ \boldsymbol{\omega_o} \quad (3.23)$$

where $\boldsymbol{\omega_o}$ is the orbital angular velocity vector. In the chosen orbital frame this parameter is $[0, \omega_o, 0]$, and $\boldsymbol{R_O^B}$ is the rotation matrix, from orbit to body frame.

The dynamic equations are a set of ordinary differential equations that describe how a body reacts to torques applied to it, i.e., how its angular velocity changes. They were proposed by Leonhard Euler and are also known as *Euler's rotation equations*. They are given by Eq. (3.24) [76, 68]:

$$\boldsymbol{J}\ \dot{\boldsymbol{\omega}}_{\boldsymbol{ib}}^{\boldsymbol{b}} + \boldsymbol{\omega_{ib}^b} \times (\boldsymbol{J}\ \boldsymbol{\omega_{ib}^b}) = \boldsymbol{M} \quad (3.24)$$

where $\boldsymbol{J}$ is the inertia tensor and $\boldsymbol{M}$ is the sum of all torques acting upon the satellite.

From a simulation point of view, it is convenient to put Eq. (3.24) as a function of ω_{ob}^b instead. By inserting Eq. (3.23) into Eq. (3.24) one gets Eq. (3.25).

$$J(\dot{\omega}_{ob}^b + (\dot{R_O^B}\omega_o)) + (\omega_{ob}^b + R_O^B\omega_o) \times J(\omega_{ob}^b + R_O^B\omega_o) = M \quad (3.25)$$

This equation can be decomposed:

$$\begin{aligned}
&J(\dot{\omega}_{ob}^b + (\dot{R_O^B}\omega_o)) + (\omega_{ob}^b + R_O^B\omega_o) \times J(\omega_{ob}^b + R_O^B\omega_o) = M \\
\Leftrightarrow\ &J(\dot{\omega}_{ob}^b + (\dot{R_O^B}\omega_o)) + \omega_{ob}^b \times J\omega_{ob}^b + \omega_{ob}^b \times JR_O^B\omega_o + R_O^B\omega_o \times J(\omega_{ob}^b + R_O^B\omega_o) = M \\
\Leftrightarrow\ &J\dot{\omega}_{ob}^b + J\dot{R}_O^B\omega_o) + \omega_{ob}^b \times J\omega_{ob}^b + \omega_{ob}^b \times JR_O^B\omega_o + R_O^B\omega_o \times J(\omega_{ob}^b + R_O^B\omega_o) = M \\
\Leftrightarrow\ &J\dot{\omega}_{ob}^b + \omega_{ob}^b \times J\omega_{ob}^b = M - J\dot{R}_O^B\omega_o - \omega_{ob}^b \times JR_O^B\omega_o - R_O^B\omega_o \times J(\omega_{ob}^b + R_O^B\omega_o)
\end{aligned}$$

one can transform one step further since [78]:

$$\dot{R}_O^B = W_{ob}^b R_O^B \quad (3.26)$$

where W_{ob}^b is the transpose of the 3×3 skew symmetric operator corresponding to ω_{ob}^b:

$$W_{ob}^b = \begin{bmatrix} 0 & \omega_{ob_z}^b & -\omega_{ob_y}^b \\ -\omega_{ob_z}^b & 0 & \omega_{ob_x}^b \\ \omega_{ob_y}^b & -\omega_{ob_x}^b & 0 \end{bmatrix} \quad (3.27)$$

leading to the final ”new” Euler equation:

$$J\dot{\omega}_{ob}^b + \omega_{ob}^b \times J\omega_{ob}^b = M + Q \quad (3.28)$$

with Q being:

$$Q = -JW_{ob}^b R_O^B\omega_o - \omega_{ob}^b \times JR_O^B\omega_o - R_O^B\omega_o \times J(\omega_{ob}^b + R_O^B\omega_o) \quad (3.29)$$

Kinematic Equations

The dynamic equations tell one how the satellite’s angular velocity reacts to applied torques but how does it influence the actual attitude? Through kinematic equations, that process is described. The kinematic equations that follow use the quaternion representation [28]:

$$\dot{q} = \frac{1}{2}\Omega_{ob}^b q \quad (3.30)$$

where q is the quaternion representation of the rotation of the body frame in relation to the orbit frame and Ω_{ob}^b is the 4×4 skew symmetric operator that corresponds to ω_{ob}^b, as shown by Eq.(3.31).

$$\boldsymbol{\Omega}^{b}_{ob} = \begin{bmatrix} 0 & \omega^{b}_{ob_z} & -\omega^{b}_{ob_y} & \omega^{b}_{ob_x} \\ -\omega^{b}_{ob_z} & 0 & \omega^{b}_{ob_x} & \omega^{b}_{ob_y} \\ \omega^{b}_{ob_y} & -\omega^{b}_{ob_x} & 0 & \omega^{b}_{ob_z} \\ -\omega^{b}_{ob_x} & -\omega^{b}_{ob_y} & -\omega^{b}_{ob_z} & 0 \end{bmatrix} \quad (3.31)$$

If one knows the external torques and the initial conditions, solving Eq. (3.28) and Eq. (3.30) as system of ordinary differential equations yields the full attitude behaviour of the satellite for each instance. The external torques that affect a satellite's attitude, $\boldsymbol{M}$ in Eq. (3.28) [36], are defined in Sec. 3.3.3 and Sec. 3.3.4.

3.3.3 Environmental Torque Models

The first set of torques that interact with the satellite are the *environmental torques*. These are torques present in geocentric orbits. They are of different natures and depend on different parameters, such as the altitude of the satellite or even its magnetic residue.

Gravity Gradient Torque

The gravity gradient torque comes from a physical property of any asymmetric object subject to a gravitational field, as the axis of largest moment of inertia tends to align itself with a plane that is normal to the orbit, while the axis with the smallest moment of inertia aligns itself with the local vertical. This is the same as saying that b_z aligns with o_z and b_y aligns with o_y. The gravity gradient torque, in the orbit frame, is given by:

$$\boldsymbol{\tau}^{o}_{gg} = \frac{3\mu}{r^3}\boldsymbol{u_e} \times \boldsymbol{I}\boldsymbol{u_e} \quad (3.32)$$

where μ is the Earth's gravitational coefficient ($\mu = 3.986 \times 10^{14} m^3 s^{-2}$), r is the orbit radius, $\boldsymbol{I}$ is the inertia tensor and $\boldsymbol{u_e}$ is a unit vector pointing towards zenith in the selected orbit frame [68, 14]. To have this equation yield the torque in the body frame, one must replace $\boldsymbol{u_e}$ with $\boldsymbol{c_3}$, the third column of the rotation matrix, from orbit to body frame. $\boldsymbol{c_3}$ also represents the offset between b_z and o_z, to which the gravity gradient torque is proportional to [36]. By doing such replacement one gets Eq.(3.33), the equation that governs the gravity gradient torque vector in the body frame, in a given instance.

$$\boldsymbol{\tau}^{b}_{gg} = \frac{3\mu}{r_0^3}\boldsymbol{c_3} \times \boldsymbol{I}\boldsymbol{c_3} \quad (3.33)$$

Aerodynamic Torque

In the close vicinity of the Earth, where most satellites operate, some air particles can still be found. That air, when colliding with the satellite's surface at the very high speeds that spacecraft travel at, applies a force on the satellite. This force is called atmospheric drag and can be modelled as having the opposite direction of the satellite's motion, assuming the symmetrical distribution of the satellite's surfaces. As a result, the atmospheric drag is an exerted force D on the satellite's aerodynamic center:

$$\boldsymbol{D^o} = \frac{1}{2}\,\rho\, C_D\, A\, v^2\, \boldsymbol{u_{aero}} \quad (3.34)$$

where ρ is the air density at the orbit's altitude, C_D is the drag coefficient - typically between 2 and 2.5 for CubeSats [14] - A is the effective area orthogonal to the velocity direction, and v is the orbital velocity of the satellite. Lastly, $\boldsymbol{u_{aero}}$ is the vector that describes the direction of the force on the satellite. In the chosen orbital frame this vector is $\boldsymbol{u_{aero}} = [-1, 0, 0]^T$.

If the center of mass is shifted from the aerodynamic center due to an imbalanced mass distribution, the force is applied at a distance that is measured from the center of mass to the aerodynamic center. This distance is represented on all axes by vector $\boldsymbol{d}$, generating a torque of $\boldsymbol{d} \times D^o$, or in its complete form [36]:

$$\boldsymbol{\tau^o_{aero}} = \boldsymbol{d} \times \frac{1}{2}\,\rho\, C_D\, A v^2\, \boldsymbol{u_{aero}} \quad (3.35)$$

which is the aerodynamic torque vector in the orbit frame.

Solar Radiation Pressure Torque

The sun, behaving like a massive fusion reactor, emits all sort of radiation, including electromagnetic radiation. When this radiation hits the spacecraft surface it causes a pressure that acts as a force on the satellite solar radiation pressure center.

As a result, the solar radiation pressure torque behaves analogously to the aerodynamic torque. Much like its counterpart, the solar radiation torque comes from a force being applied at the solar radiation pressure center while the center of mass is at a different position. The solar radiation pressure is then given by:

$$\boldsymbol{F^o_{rad}} = P_{rad}\, A\, C_P\, \boldsymbol{u_{rad}} \quad (3.36)$$

in which P_{rad} is the mean momentum flux of the solar radiation and is of about $P_{rad} = 4.5 \times 10^{-6} kg\ m^{-1} s^{-2}$, A is again the effective area (but this time normal to the solar pressure vector), C_P is the spacecraft surface absorption characteristic, that can be somewhere between 1 and 2 depending on how absorbent the satellite's surface is [14]. Lastly, $\boldsymbol{u_{rad}}$ is the vector defining the direction of the solar pressure vector, that depends on the position of the satellite and on the position of the sun. This force is non-existent in eclipse conditions, but, since the 3-AMADEUS satellite has a dusk-dawn sun-synchronous orbit, there are no eclipse conditions to take into account. In the case of 3-AMADEUS's orbit, because of the sun synchronicity, one can roughly assume that the sun direction is always close to that of o_y, meaning that $\boldsymbol{u_{rad}}$ can be assumed to be $\boldsymbol{u_{rad}} = [0, 1, 0]^T$.

The torque that comes from this force is then given by $\boldsymbol{d} \times F^o_{rad}$, or in its complete form [36]:

$$\boldsymbol{\tau^o_{rad}} = \boldsymbol{d} \times P_{rad}\, A\, C_P\, \boldsymbol{u_{rad}} \quad (3.37)$$

which is the solar radiation pressure torque vector in the orbit frame.

Residual Magnetic Dipole Torque

The gravitational field mentioned earlier isn't the only field around the Earth. While the gravitational field interacts with all bodies that have mass, there is another field, the magnetic field, that interacts with magnetic dipoles. These dipoles can come from ferromagnetic materials, i.e., certain metals that have an uneven distribution of electrons [15, 79], from electric components or even from artificial magnetic dipoles created by magnetorquers. Frequently, CubeSats and spacecraft in general have such circuitry and ferromagnetic materials, leading to the generation of an undesired magnetic dipole, often called residual magnetic dipole (RMD). When the Earth's magnetic field interacts with a magnetic dipole, a magnetic dipole moment is generated. In the case of a spacecraft, it acts as torque that tends to align the spacecraft's RMD with the magnetic field, which can be prejudicial. Since this kind of torque is deeply reliant on the magnetic field ,the first step in modelling it is to choose a model for the magnetic field.

In this work, instead of using the computationally heavy IGRF model [80], a simplified model has been chosen. A significant amount of controllers these days use simplified magnetic field models due to their robustness [36]. The model used is the one described in [81]. This model, considering the orbit frame chosen, suggests that the Earth's magnetic field induction vector, in the orbit frame, should be modelled as:

$$\boldsymbol{B^o} = \frac{u_f}{a^3} \begin{bmatrix} \cos(u)\sin(i) \\ \cos(i) \\ -2\sin(u)\sin(i) \end{bmatrix} \quad (3.38)$$

in which $u_f = 7.9 \times 10^{15}\ Wbm$ is the dipole strength of the Earth's magnetic field [36] while a is the orbit semi-major axis and i the orbit inclination. u is the argument of latitude, which is given by:

$$u = \phi_0 + \omega_o t \quad (3.39)$$

in which ϕ_0 is the initial latitude for the simulation, and t is the time elapsed from its start. Plotting the magnetic field for first 60,000 seconds of simulation for the orbit of this mission yields Fig. 3.7.

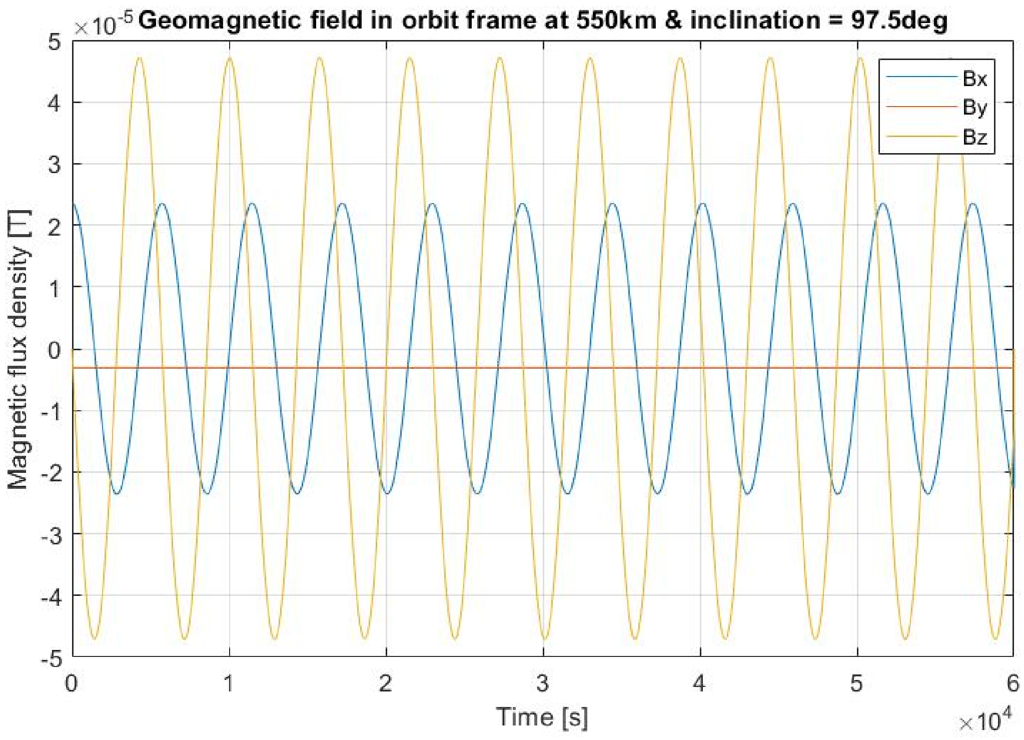


Figure 3.7: Modelled geomagnetic field for the 3-AMADEUS's orbit.

With the magnetic field modelled, one can now model the residual magnetic dipole torque in the body frame, given by:

$$\tau_{mag}^{b} = m_{res} \times B_b \quad (3.40)$$

in which m_{res} is the residual magnetic dipole vector discussed earlier, and B_b is the geomagnetic field, in the body frame. It can be obtained from B_o with Eq.(3.41).

$$B_b = R_O^B \, B_o \quad (3.41)$$

3.3.4 Control Torque Model

By countering (or exploiting) the effects that environmental torques have on satellites, ADCS actuators generate torques that stabilize the satellite in the desired attitude. The torque produced must be added to the sum of the environmental torques to represent M in Eq. (3.28). Since the 3-AMADEUS mission features only magnetic attitude actuators, only magnetorquers are covered in this section.

Magnetorquer

Magnetorquers are attitude actuators that exploit the interaction with the geomagnetic field. By generating a magnetic dipole on specific axes, magnetorquers can be used to orientate the satellite as intended. The equations for the magnetorquer torque are the same as for the residual magnetic dipole torque, except that for the actuating torque, the magnetic dipole is generated by the magnetorquer instead of being residual, as shown by Eq.(3.42).

$$\tau^b_{trqr} = m_{trqr} \times B_b \quad (3.42)$$

The underactuation issue discussed in Sec. 2.2.2 is also visible from the cross product: magnetorquers cannot produce torque on the plane of the geomagnetic field.

All equations that govern the satellite attitude dynamics have been discussed. A model running a solution for the equations of motion along with the equations for the environmental torques is discussed in Ch. 4.

3.4 Numerical Analysis Method

3.4.1 Method of Runge-Kutta of the 4^{th} Order

The method used in this thesis for the numerical analysis of the satellite's attitude is the 4th Order Runge-Kutta Method. In this work, this method is used to solve a system of ordinary differential equations, Eq. (3.28) and Eq. (3.30). In this section these are respectively denoted as $\dot{\boldsymbol{\omega}} = f_\omega(\boldsymbol{\omega})$ and $\dot{\boldsymbol{q}} = f_q(\boldsymbol{\omega}, \boldsymbol{q})$. The RK4 method describes an iterative method for integrating such a system of ordinary differential equations. It proposes that several k coefficients are calculated for each equation for each integration step. These coefficients are then added to calculate the solution of the system of differential equations for the next step.

With the equations of motion, the RK4 method works as described by Eq. (3.43) [28].

$$\begin{aligned}
k_{1_\omega}(i) &= h\ f_\omega(\omega(i)) \\
k_{1_q}(i) &= h\ f_q(\omega(i), q(i)) \\
\\
k_{2_\omega}(i) &= h\ f_\omega(\omega(i) + \frac{k_{1_\omega}(i)}{2}) \\
k_{2_q}(i) &= h\ f_q(\omega(i) + \frac{k_{1_\omega}}{2}, q(i) + \frac{k_{1_q}(i)}{2}) \\
\\
k_{3_\omega}(i) &= h\ f_\omega(\omega(i) + \frac{k_{2_\omega}(i)}{2}) \\
k_{3_q}(i) &= h\ f_q(\omega(i) + \frac{k_{2_\omega}}{2}, q(i) + \frac{k_{2_q}(i)}{2}) \\
\\
k_{4_\omega}(i) &= h\ f_\omega(\omega(i) + k_{3_\omega}(i) \\
k_{4_q}(i) &= h\ f_q(\omega(i) + k_{3_\omega}, q(i) + k_{3_q}(i)) \\
\\
\omega(i+1) &= \omega(i) + \frac{1}{6}(k_{1_\omega}(i) + 2k_{2_\omega}(i) + 2k_{3_\omega}(i) + k_{4_\omega}(i)) \\
q(i+1) &= q(i) + \frac{1}{6}(k_{1_q}(i) + 2k_{2_q}(i) + 2k_{3_q}(i) + k_{4_q}(i))
\end{aligned} \tag{3.43}$$

Here h is the simulation step and the above equations yield the values of the angular velocity and the quaternion representation in each step.

In the chapters to come, in order to implement the attitude control algorithm in the FPGA to perform an HIL simulation, it is necessary not only to represent data in a way that a computer or an FPGA can handle, but also to transmit it using a communication protocol. With that in mind, the next sections show how data can be represented digitally and how that data can be transferred between digital systems.

3.5 Binary Number Representation

When computers were originally designed, their purpose was to automatically perform tasks that humans had to do. One of those tasks and the main task of a computer is, intuitively, to *compute*. For a computation to be performed, numbers must be handled by the computer. One way to represent these numbers is to sequentially set the voltage in an electrical wire to either the value defined as low voltage or the value defined as high voltage. The resulting sequence of high and low voltages (a bit sequence) is then interpreted as a number in base 2 [82], where only 1 - represents high voltage - and 0 - a low one - are the available digits, as opposed to the widely used base 10 representation. In this representation, frequently called *binary*, the rightmost digit represents 2^0 and the n^{th} digit represents 2^{n-1}.

With that in mind, if one considers, for example, the number 12 in the decimal base:

$$12_{(10)} = 1 \times 10^1 + 2 \times 10^0 \quad (3.44)$$

its binary representation is 1100, as shown by Eq.(3.45).

$$1100_{(2)} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 12_{(10)} \quad (3.45)$$

It is important to note that for a computer to interpret these numbers correctly, the number of bits of each number must be specified a priori.

3.5.1 Signed Numbers

In light of this, it is clear how one must represent positive numbers but how can negative numbers be represented? If one adds an extra bit to the left of the number that would indicate whether the number is positive or negative, a problem arises: 0 can either be represented by 1000 or 0000 (assuming a 4-bit word), which is not optimal. Additionally, simply using a sign bit makes binary arithmetic yield wrong results. A better way to represent negative numbers is to use the *2's complement* [82]. The 2's complement representation of signed numbers works as follows: To represent a negative number in binary, for example, -5 one takes the binary representation of 5 (0101), afterwards, all the bits are inverted, yielding 1010. Lastly, a 1 is added to this number, resulting in the final representation of 1011. Using the 2's complement representation, one can consider that the leftmost (sign) bit represents a negative power of 2.

$$1011_{(2)} = -1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = -5_{(10)} \quad (3.46)$$

The ordinary binary algebra operations such as addition and multiplication, that are valid for regular binary numbers are still valid using this representation, as is discussed in Sec. 3.5.3.

3.5.2 Decimal Numbers

With negative number representation described, how is one to represent numbers that aren't integers but rather decimals using a system that merely reads high and low voltages? The two most common and established methods are *fixed point* representation and *floating point* representation.

Floating Point Representation

One way to represent decimal numbers in binary is the floating point representation. This representation works analogously to common scientific notation, but using base 2 instead of the more simply understood, base 10. Using this method, given a certain length for the binary word (usually it's either 32 or 64 bits), a certain part of it represents the sign of the number, another the exponent and lastly, another part represents the mantissa [82].

For a 32-bit word, one has 1 sign bit that determines the sign of the number, 8 bits for the exponent, and 23 bits for the mantissa:

$$N = seeeeeeeemmmmmmmmmmmmmmmmmmmmmmm$$

this binary number, in the IEEE-754 format [83] (the most commonly used), would represent a decimal number such that:

$$N_{(10)} = (-1)^{sign} \times 2^{(e-127)} \times (1 + \sum_{i=1}^{23} n_{23-i}\, 2^{-i}) \quad (3.47)$$

where n_j is the j^{th} bit of N. In this format, the exponent is an 8-bit unsigned integer in biased form. Additionally, the mantissa has 23 fraction bits while one assumes implicitly that there is a leftmost leading integer bit that is always 1. With that in mind, let's look at an example:

$$N_{(2)} = 01000000010010010000111001010110_{(2)}$$
$$0_{(2)} = 0_{(10)}$$
$$10000000_{(2)} = 128_{(10)}$$

$$N_{(2)} = -1^{0} \times 2^{(128-127)} \times (1 \times 2^{0} + 1 \times 2^{-1} + 0 \times 2^{-2} + \ldots + 1 \times 2^{-22} + 0 \times 2^{-23}$$
$$01000000010010010000111001010110_{(2)} = 3.1414999961853027 34375_{(10)}$$

the resulting value happens to be the closest representation of 3.1415 in this format. As a result, one can see that this representation isn't completely accurate. If it is considered that the achieved accuracy is enough for a specific design then this is fine, however, if more accuracy is needed, more bits are required.

Fixed Point Representation

The other approach that can be taken to represent decimal numbers is via fixed point representation. This method assumes that a given binary number with n bits represents a decimal number that is scaled by 2^f. The interpretation must then be that the leftmost $n - f$ bits represent the integer part of a number and that the rightmost f bits are the fractional bits in the number [82]. For example, normally, 11001010 must be interpreted as described by Eq. (3.48).

$$11001010_{(2)} = -54_{(10)} \quad (3.48)$$

Despite this, interpreting this number as being scaled by 2^4 for example, means that there are 4 integer bits and 4 fractional bits.

Having that in mind, for this particular case, one can say that:

$$1100.1010_{(2)} = -54_{(10)} \times 2^{-4} = -3.375_{(10)} \quad (3.49)$$

or

$$1100.1010_{(2)} = -1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$
$$+1 \times 2^{(-1)} + 0 \times 2^{(-2)} + 1 \times 2^{(-3)} + 0 \times 2^{(-4)} = -3.375_{(10)} \quad (3.50)$$

this, as shown, is still valid for numbers in 2's complement. Since this method is simply scaling up the number one wishes to represent, all binary arithmetic is still valid, since one can just operate with the scaled numbers and scale down the result accordingly.

While most computer systems today use floating point representation since it is best for most general applications, fixed point representation is better at handling operations between incredibly large and incredibly small numbers, assuming that one knows the interval in which these numbers can be [84], as is the case for the attitude control of the 3-AMADEUS satellite. Additionally, floating point logic is much harder to implement on hardware without floating point units - often the case for FPGAs - than fixed point is. On top of that, since an FPGA is used in this project, there are no great limitations in terms of bit length, since the FPGA's design freedom allows one to easily go over the 64 bits used in floating point representation, increasing its accuracy. What this means is that the limitations that fixed point arithmetic may imply on other types of hardware for other types of implementations aren't as relevant in this case. Additionally, fixed point increases processing speed [85]. With that in mind, the FPGA implementation for this thesis is done considering fixed point representation.

3.5.3 Signed Fixed Point Arithmetic

Now that the way to represent numbers has been defined, it is now time to take a look at how to handle their arithmetic. While the general rules for binary arithmetic apply, since fixed point numbers have a fixed number of bits, in which both operands and results must be represented, some extra steps are required in order to obtain the correct result in the correct format.

Addition

Adding numbers in base 2 is exactly the same as adding them in base 10, the only difference is that there are only two digits that one can use, but the same rules of carry still apply. For fixed point, the point is just an abstraction so it doesn't affect the addition in any way - adding 1001 and 1010 is the same as 10.01 and 10.10. One very important thing to take into account when adding binary numbers is that the allocated number for the result must be enough to represent the operands as well as the result. For example, when calculating $-1.75 + (-1.75) = -3.5$, it is impossible to represent the result with 2 bits for the integer and 2 fractional bits since even though -1.75 is representable with this bit length, -3.5 is not. For this to be possible one would need 3 bits for integer and 2 for the fractional part. Having the same example in mind, the actual addition would go as follows:

	110.01	(-1.75)
+	110.01	(-1.75)
1	100.10	(-3.5)

notice how the leftmost 1, which has been carried, is separated from the resulting number. This happens because in fixed point representation the leftmost carry bit must be discarded, as the actual result in the selected representation are the remaining rightmost bits, 100.10. In fact, for addition, the resulting number is always 1 bit longer than the operands before it is truncated. In terms of hardware an adder like this is usually implemented as a *ripple carry adder* [48].

Subtraction

Subtracting is merely adding negative numbers, meaning that:

	111.01	(-0.75)
-	010.10	(2.5)

is the same as:

	111.01	(-0.75)
+	101.10	(-2.5)
1	100.11	(-3.25)

this means that if one wishes to subtract a number from another all that is necessary is to logically implement the conversion from the number that is to be subtracted to the 2's complement representation of its negative and add it to the other number.

Multiplication

Multiplication in binary also works analogously to that of regular base 10 numbers [86, 87]. Multiplication is interpreted as a series of sums for each order of magnitude of one of the operands, but for the case of signed numbers, since abstractions such as the 2's complement are necessary, a few extra steps are required. The fact that a fixed point representation is being used affects only how one should treat the result but not the multiplication itself, as it is done as if two scaled numbers are being multiplied.

The first nuance of multiplying signed numbers is that the partial products (the numbers that must be added) must be sign-extended to match the size of the result of the product between the two operands - the sum of their sizes. Additionally, to represent the negative weight of the most significant bit of the second operand, one must calculate the 2's complement of the first operand and sign-extend it, resulting in what is the last partial product. To make this clear, if one were to multiply $11.001_{(2)} = -0.875_{(10)}$ (the first operand) by $10.010_{(2)} = -1.75_{(10)}$ (the second operand), the first step would be to calculate the last partial product, the sign-extended 2's complement of the first operand: 000111. When that is done, the computation can be done, keeping in mind that all partial products must also be sign-extended, and then added to obtain the final product:

$$\begin{array}{ccccccccccc}
 & & & & & & 1 & 1 & 0 & 0 & 1 \\
 & & & & & \times & 1 & 0 & 0 & 1 & 0 \\
\hline
 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 & 0 & 0 \\
 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 & 1 & 0 & 0 & 1 & \\
 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & & \\
 & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & & & \\
+ & \mathbf{0} & 0 & 0 & 1 & 1 & 1 & & & & \\
\hline
 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0
\end{array}$$

the result from this operation is simply the result from multiplying $11001_{(2)}$ by $10010_{(2)}$, but it still doesn't represent the product of $11.001_{(2)}$ and $10.010_{(2)}$ as is desired. To get to this result, one must truncate the result of the product, by the number of integer bits (2) at the left and by the number of fractional bits (3) at the right, yielding the final product of $11.001_{(2)}$ and $10.010_{(2)}$ in the desired format, $01.100_{(2)} = 1.5_{(10)}$. The actual product of -0.875 and -1.75 should in fact yield 1.53125 but with only 3 fractional bits, this is the best approximation one can get. In hardware this multiplication is often done with the use of shift registers [48].

Division

The division of binary numbers can be described as an algorithmic implementation of successive shifts and subtractions. To divide 2 binary numbers, one can use the basic long division that is valid for base 10 numbers. In this case, one must analyze whether the divisor is small enough so that if fits within the dividend and fill the quotient accordingly. To account for negative numbers, one can change the sign of the divisor and/or the dividend to their positive counterpart and afterwards adjust the sign of the quotient accordingly. Looking at the example where $110.100_{(2)} = -1.5$ must be divided by $000.100_{(2)} = 0.5$, first the negative 110.100 is transformed to its positive version, 001.100, then a normal division can be computed:

$$\begin{array}{r@{}r}
 & \overline{000011} \\
000100) & 001100 \\
 & \underline{100} \\
 & 0100 \\
 & \underline{0100} \\
 & 0
\end{array}$$

the quotient of this division is 000011. Since there is no remainder to this operation, the remainder field is empty, otherwise that would be where one would get the fractional bits. The quotient must be truncated to give the result in the desired format, yielding 011 for the integer bits and 000 (no remainder) for the fractional bits. A sign change is still due, so one takes the result 011.000 and changes its sign, yielding the final result $101.000_{(2)} = -3$. The hardware for this operation is similar to that of multiplications, but the 2's complement must be used in the form of additions to perform the required subtractions.

3.6 Communication Protocols

Now that all numbers and the operations between them can be represented, it is time to take a look at how devices transmit this data between them. This is is pivotal for a work like this, where FPGA data must be received from and sent to a computer running a simulation, and more importantly, in a real life application, where the OBC running the satellite attitude control algorithm must communicate with the other components, such as ADCS actuators and sensors.

3.6.1 UART

The first method to be looked at is *UART*, or *Universal Asynchronous Receiver/-Transceiver*, a serial communication protocol that, being asynchronous, has no clock signal (a signal oscillating between high and low voltage at a given frequency) synchronizing the sending and receiving of bits between two devices. [88, 89]. UART is a very popular choice due to its simplicity and ease of implementation, with it being implemented in all kinds of embedded systems that have low data rate requirements (the maximum data rate for UART nowadays is of about 115200 bps). In fact, low data rates is the greatest limitation of communicating through UART. It consists of two connections between two devices, as depicted in Fig. 3.8.

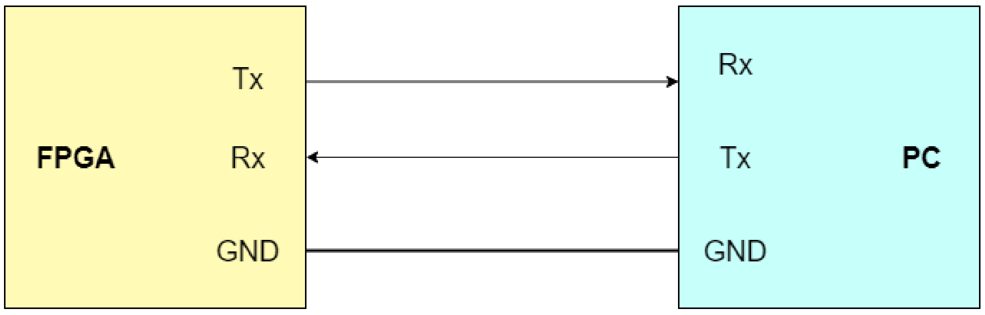


Figure 3.8: Full Duplex UART Communication between an FPGA and a PC.

With two connections, UART can be made to be full-duplex, meaning it can receive and transmit data simultaneously. Since there is no clock involved in UART communications, both receiver and transmitter must agree on how fast the data must be transmitted, i.e., the Baud rate (bits per second) of the communication. Receiver and transmitter must also agree on what composes the transmitted data: the number of data bits (usually data is sent in packets of 8 bits or 1 byte), whether there is a parity bit (used to detected errors) or not, and how many stop bits there are. In light of this, a common UART transmission looks like Fig. 3.9.

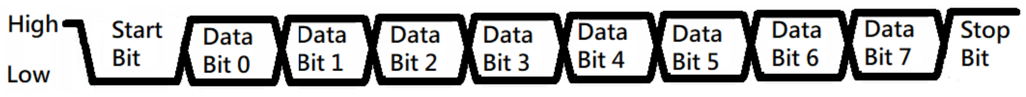


Figure 3.9: Common UART transmission.

A UART transmission is composed of a start bit that triggers the receiving of data, data bits that actually contain the information that is to be transmitted, and the stop bit, ending the transmission. In the depicted transmission, the start bit goes to low level activating the transmission, afterwards, eight data bits are transmitted. At the end, the stop bit sets the voltage to to high again, making the line susceptible to stimulation by the start bit of the next transmission. It's important that the receiving end of the UART communication samples the data at the middle of the data bits (half a bit length away from the rising edge of the data signal) since near the falling and rising edge of a signal the voltage measured can be ambiguous, introducing errors in the data.

Since the communication between magnetometers and an FPGA doesn't require a large data rate and UART is overwhelmingly simple, the preferred option for this thesis is to use the UART serial communication protocol.

3.6.2 SPI

SPI stands for *Serial Peripheral Interface* [88, 89], and it is a communication protocol that, unlike UART, allows for communication between several components, assuming one is given the role of "master". This protocol is synchronised by a clock signal, allowing for faster communication (about 1 Mbps) than that provided by UART.

Being slightly more complex than UART, SPI has four connections in the master component. In the case of a satellite's OBC, this could be the FPGA, as is depicted in Fig. 3.10.

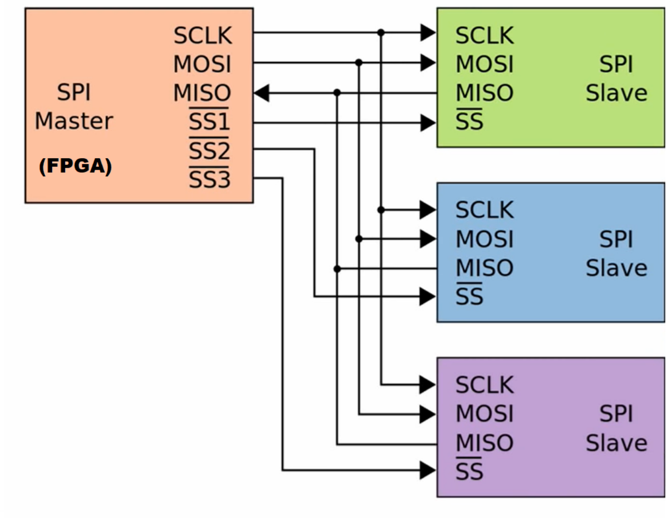


Figure 3.10: Full Duplex SPI communication [90].

The clock signal serves as an input that synchronizes the exchange of data. While MOSI (Master Out Slave In) is the line in which data flows from the master to the slave components, MISO (Master In Slave Out) is the line carrying the data that is sent from the slaves to the master. Finally, the SS (Slave Selector) lines are set to low to select which slave is actively communicating with the master. When the slave is selected the transmission process is similar to that of an UART communication apart from a few details regarding the clock, namely the fact that data is transmitted only when the clock signal allows it. This synchronicity allows for a few parameters to be set, for example the clock polarity (whether the clock is idle in high voltage or low voltage setting) and clock phase setting (what edge - rising or falling - of the clock signal does data get transmitted on).

3.6.3 I^2C

I^2C stands for *Inter-Integrated Circuit* and just like SPI, it is synchronous and connects a master component to several slave components serially, but is an upgrade in terms of pin count, since it requires only two connecting wires between all the components running on this communication protocol, of course at the cost of simplicity, speed (can run at about 400 kbps) and full duplexability [88, 89]. It usually has the layout described by Fig. 3.11, which depicts an I^2C interface between a master microcontroller with a DAC (Digital to Analog Converter), an ADC (Analog to Digital Converter) and another microcontroller as slaves.

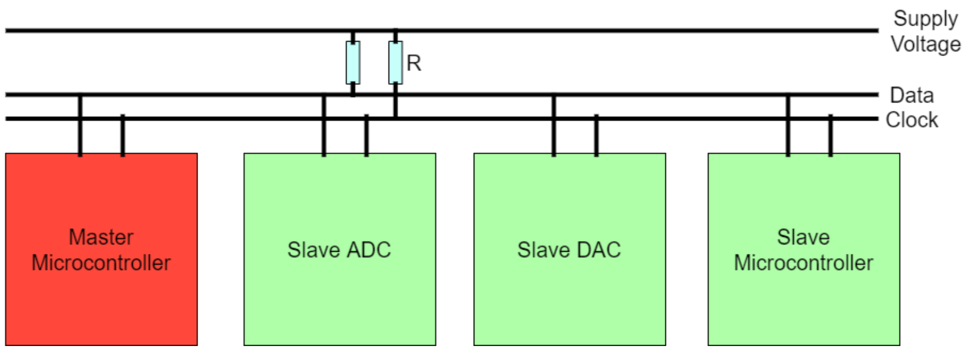


Figure 3.11: I^2C communication.

From Fig. 3.11, the key feature of I^2C is clear: it has only one data line and one clock line, with both of them running from the master through all the slaves. The two pull-up resistors depicted are essential in the implementation of this line that can be driven by several components. That being said, all components read the data from the line simultaneously, but if no problem occurs, only one should drive it at a time. For each component to understand what it is supposed to do with a single line as both input and output, an I^2C message requires a few more informative bits than UART and SPI.

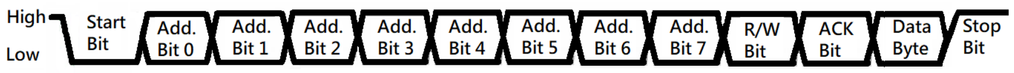


Figure 3.12: An I^2C message.

The transmission starts with the master setting the data line to low, triggering the transmission process as soon as the clock signal falls to low as well. After that, there are

7 bits that describe the I^2C address, containing information about which slave the master is communicating with. Afterwards, a R/W (Read/Write) bit tells the slave if the master wants to write to it or read from it. If the slave recognizes his address and instruction and gets ready to communicate, it transmits an ACK (Acknowledge) bit to the master. After this process, I^2C behaves like a regular serial communication, with data bits being sent until a stop condition is reached.

3.7 FPGA Programming

Now that the main notions of digital systems required for the understanding of the work that has been undergone for the development of this project have been presented, it is time to look at how all these concepts should be programmed into an FPGA, so that it not only communicates with other devices, namely a personal computer or ADCS components, but also handles the data received correctly.

3.7.1 FPGA Design Flow

Before looking into any details, it is paramount that one understands all the steps that must be taken in designing an FPGA, or what is usually called the *FPGA design flow*, depicted in Fig. 3.13 [91].

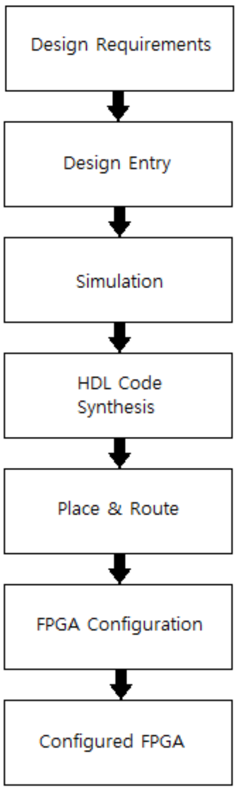


Figure 3.13: FPGA Design Flow.

Design Requirements

The first step is to analyze the requirements of the design. Some of the most important parameters that must be set are the accuracy of the required computations and the processing speed requirements, that define, for example, what communication protocol the design should use. All the structure of the design should be defined before moving on to the next step, and if any fundamental alterations are required, the design flow must restart at this step.

Design Entry

The first step in the actual design of an FPGA is the *Design Entry*. In this step, the FPGA design is specified. For that, the designer must use the IDE (Integrated Development Environment) of their choice to add a multitude of files to a project, including library files, the actual source file in which the behaviour of the FPGA is described and even an User Constraint File (UCF) that assigns the signals defined in the previously mentioned files to actual pins on the FPGA device.

Simulation

The simulation step has no practical effect on the design process, but is extremely important. Since programming an FPGA is very low level, it is usually impossible to correctly assess what is wrong with a design just by looking at its actual behaviour. To solve this problem, designers, using a simulator tool that allows them to look at each signal in every instance, simulate the behaviour of the FPGA and find any bugs present in the design, reverting the design process back to the design entry step, where the design can be corrected. To simulate an HDL design, a testbench file, with code that stimulates the source code, must be created.

HDL Code Synthesis

The *HDL Code Synthesis* [92], is a step in which the synthesis tool, often provided by the device vendor, creates a netlist of all the connections between the logic blocks that a design requires.

Place & Route

The *Place & Route* step is one that can take several hours if the target FPGA is a large one. In this step, the IDE used, having the target FPGA in consideration, takes the synthesized netlist and associatse those connections with physical resources in the FPGA, creating a map that is used to create the programming file.

FPGA Configuration

Once all the previous steps have been completed successfully, the designer uses the IDE's feature to generate a binary bitstream programming file that contains all the information required to implement the design in the FPGA. After this file is generated, a tool with FPGA programming capabilities must be used to download the program to the FPGA, usually via a programming cable. Additionally, the same tool must generate and implement a PROM (Programmable Read-Only Memory) file so that the program the code describes stays programmed in the FPGA even when it is turned off.

Configured FPGA

Once all these steps are completed, the FPGA is ready to use, and the implementation must be tested. Due to the simulations performed, there shouldn't be design flaws. Should there be any, further simulations must be made until the issue is identified.

3.7.2 Tools

Xilinx (the main FPGA manufacturer) offers a very complete IDE, which nowadays is *Vivado*. This IDE features not only a text editor where code can be inserted but also tools to automatically generate the UCF as well as a simulator tool, among others. Additionally it has synthesizing capabilities and can even generate the bitstream programming file and download it onto the target device. In fact, the whole design of a Xilinx FPGA can be done solely using their IDE. For FPGAs that aren't supported by the most recent IDE, an outdated version must be used.

Despite this, there are other alternatives for specific steps in the design flow mentioned earlier. As a result, some parts of the design can be done in the Xilinx IDE, while others can be done elsewhere, as is the case for this thesis.

3.7.3 Programming Language

To program an FPGA, one must use an HDL (Hardware Description Language). HDLs were created with the intent of replacing the digital systems design methods of old, in which engineers had to use schematics and design the system at gate level, with a method that is at a higher level of abstraction that would simplify that process. HDLs are a unique type of programming language that notably differ from software programming languages as they allow for a description of the physical signal, and, perhaps more significantly, its commands are not executed sequentially, but rather concurrently, meaning that time delays and propagation must be described via hardware as well. The two most commonly used languages are Verilog and VHDL (Very High Speed Integrated Circuit Hardware Description Language), with both of them working at a Register-Transfer-Level abstraction level. Among other things, while VHDL [93] is not as close to C based programming languages as Verilog [94] is, it is a simpler, more verbose language that is more easily understood by itself. Additionally, synthesis tools often find errors more frequently on VHDL than they do on Verilog codes. Ultimately, the usage of one or the other is a matter of preference, and due to the availability of documentation, the FPGA programming done for this thesis is done in VHDL.

As mentioned above, these kinds of language describes hardware, meaning that even if a design is functionally correct and works in simulation, one must always keep in mind what hardware the tool that synthesizes the code is inferring. By knowing the available resources, the designer must always consider what resources the code is using and manage that usage. For example, if one wishes to multiply five 18-bit numbers, it is important to write code that the synthesizing tool identifies as one multiplier used five times rather than five multipliers used one time. This is important since inferring excessive hardware might lead to a design that doesn't fit within the available resources.

3.7.4 Architectural Overview

This section overviews the usual resources that physically exist on an FPGA and are available when designing it, with focus on the Spartan 3E FPGA, the one used for this project [95].

Configurable Logic Blocks (CLBs)

The CLBs are the main building block of an FPGA. They contain Look Up Tables (LUTs) that can be set to specific outputs, implementing logical processes or storage elements such as flip flops and latches.

Input/Output Blocks (IOBs)

IOBs control the flow of data between the FPGA input and output pins and the logic inside it, supporting several signal standards.

Block RAM

Block RAM supplies the device with data storage. They usually come in the sizes of 4,8,16 or even 32 kilobits.

Multiplier Blocks

Multiplier blocks take two binary numbers of a predefined length as its input and outputs its product. Usually, a number of them exist inside an FPGA and they can be coupled with more multipliers to provide multiplying of words of greater lengths.

Digital Clock Manager Blocks (DCMs)

DCMs provide support for clock signals, namely solutions for distributing, delaying, multiplying, dividing, and phase-shifting clock signals.

Chapter 4

Mathematical Model For Attitude Dynamics

In this chapter, the concepts that are introduced in Ch. 3 are put to use by simulating the attitude dynamics of the satellite. This model serves as base not only for the chapters to come, in which a control algorithm is tested in different conditions, but also to future algorithm testing for the 3-AMADEUS CubeSat. The model is ran using MATLAB software. In this work, the arc cosines of the satellite's diagonal direction cosines, which represent the offset angles between the axes of the body and orbit frames, are referred to as either offset angles or orientation angles.

To fully assess the validity of this model, firstly, a simple attitude control test simulation is proposed: a gravitational stabilization test simulation in which there are no attitude perturbances besides the gravity gradient torque, and the magnetorquer magnetic dipole moment abides by the control law described by Eq. (4.1) - a simplified version of the one present in [1].

$$\boldsymbol{m_{trqr}} = -k_\omega \boldsymbol{B_b} \times \boldsymbol{\omega_{ob}^b} \quad (4.1)$$

In these conditions, if the model is correct, then the satellite's orientation should tend to orbitally stable attitudes. These attitudes should be such that the axis of smallest moment of inertia, b_z aligns with the nadir/zenith direction, and the axis with the largest moment of inertia, b_y aligns with the orbit normal. If one of these attitudes is achieved, the orientation angles should converge to either 0 or 180 degrees of offset between the axes of the body and orbit frames. Consequently, the angular rate should converge to 0. If this happens, the model should be validated.

In order to magnify the effect of the gravity gradient torque, for this section, a test satellite model is used. The test satellite has considerably different moment of inertia on its axes so that the attitude control is more effective, contrary to the 3-AMADEUS satellite, which is cube-like.

Afterwards, in an environmental perturbance test simulation, the same test satellite is simulated in a series of environments where the perturbance torques discussed in Sec. 3.3.3 are present and there is no control torque. By looking at the spacecraft behaviour it should be possible to assess the correct implementation of these torques.

All the simulation parameters set in this chapter are the ones used in the actual control simulations in Ch. 6, except when explicitly stated otherwise.

4.1 Simulation Parameters

4.1.1 Numerical Parameters

For the first test, the simulation time, t, is of 45,000 seconds or 12.5 hours, the expected time for the test satellite to completely gravitationally stabilize in these conditions. In the second test, the same simulation time is used. The simulation step is of 1 second for both simulations. On one hand this step size is small enough so that decreasing it further would only negligibly affect the simulation results, and on the other, is large enough, so that the computation time is reasonably low. This combination of course leads to $\frac{t}{h} = 45,000$ iterations for both tests.

4.1.2 Control Parameters

The control gain k_ω used for the implementation of control law of Eq. (4.1) is of 6,500 $\frac{Nm}{T^2}$.

4.1.3 Physical Parameters

As mentioned earlier in Sec. 3.2.1, the axis of smallest moment of inertia is the b_z, while the axis of largest moment of inertia is b_y. These settings lead to the intuitive conclusion that b_z is along the longest dimension of the satellite and the b_y axis on the shortest. The dimensions along b_x, b_y and b_z are denoted as a, b, and c, respectively.

3-AMADEUS

It should be stated that the real dimensions of the 3-AMADEUS satellite are still to be defined, meaning that the ones discussed in this section and used in the chapters to come are mere estimates of 3-AMADEUS' final dimensions.

Despite having considered a perfectly uniform mass distribution when calculating the inertia tensor, in order to increase the realism of the external torques, it is assumed that the center of mass is located 2 mm away from its geometrical center in the b_z direction. This assumption is only used for computing the aerodynamic and solar radiation pressure torques. The solar radiation pressure center and the aerodynamic pressure center are both assumed to lie on the geometrical center of the satellite. The dimensions and center of mass offset vector used for the 3-AMADEUS satellite are then:

$$\begin{aligned} a &= 0.100\ [m] \\ b &= 0.090\ [m] \\ c &= 0.110\ [m] \\ \boldsymbol{d} &= [0, 0, 0.002][m] \end{aligned} \tag{4.2}$$

the mass of the satellite is the expected mass of a 1U CubeSat, $m = 1.33\ kg$.

This leads, according to Eq. (3.21) and assuming a uniform mass distribution, to the principal inertia tensor:

$$\boldsymbol{J} = \begin{bmatrix} 0.0022 & 0 & 0 \\ 0 & 0.0024 & 0 \\ 0 & 0 & 0.0020 \end{bmatrix} \tag{4.3}$$

fully describing the 3-AMADEUS satellite for the simulations in this thesis, physically.

Test satellite

The test satellite physical parameters that are used only for the present chapter are described by Eq. (4.4).

$$\begin{aligned} a_{test} &= 0.100\ [m] \\ b_{test} &= 0.050\ [m] \\ c_{test} &= 0.200\ [m] \\ \boldsymbol{d} &= [0, 0, 0.002][m] \end{aligned} \tag{4.4}$$

Considering the same mass as the 3-AMADEUS model, that leads to the principal inertia tensor described by Eq.(4.5).

$$\boldsymbol{J_{test}} = \begin{bmatrix} 0.0047 & 0 & 0 \\ 0 & 0.0055 & 0 \\ 0 & 0 & 0.0014 \end{bmatrix} \tag{4.5}$$

4.1.4 Orbital Parameters

The orbital parameters considered for this simulation are those of a circular 550 km SSO that allows for optimal sunlight conditions. For there to be such conditions, an orbital inclination of approximately 97.5^{o} is required.

This means that the orbital parameters relevant for this simulation are:

$$\begin{aligned} r_{earth} &= 6378\ [km] \\ h &= 550\ [km] \\ a = r_{earth} + h &= 6928\ [km] \\ e &= 0 \\ \mu &= 3.986 \times 10^5\ [km^3/s^2] \\ T = 2\pi\sqrt{\frac{a^3}{\mu}} &= 5739\ [s] \\ \omega_o = \frac{2\pi}{T} &= 0.0627\ [deg/s] \\ i &= 97.5\ [deg] \\ v = \sqrt{\frac{\mu}{a}} &= 7.58\ [km/s] \end{aligned} \tag{4.6}$$

where elements such as the longitude of the ascending node and the argument of perigee aren't defined. Since a circular orbit is being considered, the argument of perigee is meaningless, and as a result it is not necessary to define it. In the case of the longitude of the ascending node, it does not affect the attitude dynamics, but is important for the 3-AMADEUS mission since the way it changes over time defines the orbit as a dusk-dawn SSO.

4.1.5 Aerodynamic Parameters

The drag coefficient used for this simulation is that of a regular CubeSat [14]. The exposed area is ever-changing, but for this simulation, it is considered that it is the area of a regular 1U CubeSat face while the air density is the average air density found at an altitude of 550 km [96].

$$\begin{aligned} C_D &= 2 \\ A &= 0.01\ [m^2] \\ \rho_{550km} &= 1.25 \times 10^{-12}\ [kg/m^3] \end{aligned} \tag{4.7}$$

4.1.6 Solar Pressure Parameters

The solar parameters that affect the satellite's behaviour are the mean momentum flux of the solar radiation, as discussed previously and C_P, for which a worst case scenario is considered. The exposed area considered is the same as for the aerodynamic pressure torque.

$$\begin{aligned} C_P &= 2 \\ A &= 0.01\ [m^2] \\ P_{rad} &= 4.5 \times 10^{-6} [kg\ m^{-1} s^{-2}] \end{aligned} \tag{4.8}$$

4.1.7 Magnetic Parameters

The magnetic parameters refer to the residual magnetic dipole discussed in 3.3.3 as well as the magnetorquer characteristics that produce magnetic moment. The residual magnetic dipole is the typical value for CubeSats [97, 98] on all axes, while the magnetorquer data (maximum magnetic dipole moment and maximum current for each torquer coil) considers a system based on the MOVE-II CubeSat mission's magnetorquer board [99].

$$\begin{aligned} I_{max} &= 0.34\ [A] \\ m_{max} &= 0.10\ [Am^2] \\ m_{res} &= 0.01\ [Am^2] \end{aligned} \tag{4.9}$$

4.1.8 Initial Conditions

The initial conditions for this simulation, i.e., the initial orientation and angular velocities, are given respectively by Eq.(4.10) in Euler Angles (sequence ZYX), and by Eq.(4.11).

$$\begin{aligned} \phi &= 50\ [deg] \\ \theta &= 50\ [deg] \\ \psi &= 50\ [deg] \end{aligned} \tag{4.10}$$

$$\begin{aligned} \omega^b_{ob_x} &= 0.01\ [rad/s] \\ \omega^b_{ob_y} &= -0.015\ [rad/s] \\ \omega^b_{ob_z} &= 0.005\ [rad/s] \end{aligned} \tag{4.11}$$

The initial Euler angles must be converted to quaternions for computations. Additionally, it is important to note that the initial angular velocities are the usual ones found after satellite deployment [1]. With all those parameters discussed, some results are presented next so that the model can be validated.

4.2 Gravitational Stabilization Test Simulation

It is worth reminding that in this simulation all perturbance torques are turned off except for the gravity gradient torque. It is expected that the satellite converges to a gravitationally stable position, with b_z facing the Earth or away from it, meaning it should be aligned with the direction of o_z (so the orientation angles should produce 0 or 180 degrees of offset between the two axes). Additionally, b_y should align with the direction normal to the orbit, meaning that in the orbit frame selected it should have a 0 or 180 degree offset from o_y. The same goes for b_x since it is bound to the two other axes. The results, in orientation angles, are described by Fig. 4.1.

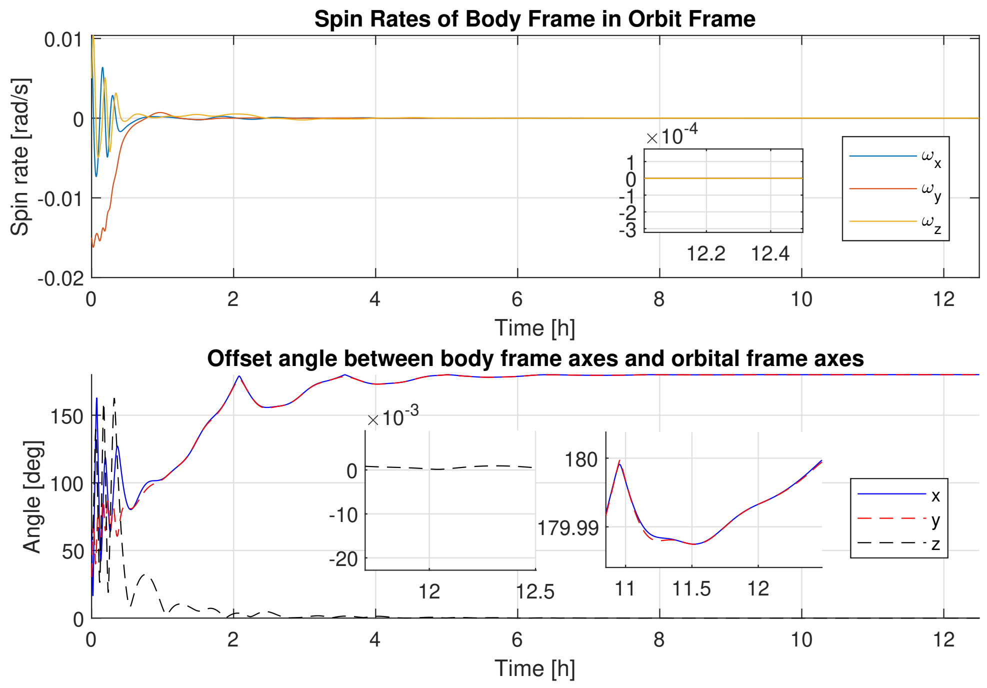


Figure 4.1: Orientation angles and spin rate - Gravitational Stabilization Test Simulation.

The spin rate rapidly converges to 0, which is ideal. The b_z axis is 0 degrees away from the zenith direction, as expected. Additionally, b_y aligns with the direction opposite of o_y axis, while b_x completes the right-hand system by aligning with the direction opposite of the linear velocity. These are the expected results for a well implemented model.

4.3 Environmental Perturbance Test Simulation

In this simulation run, the environmental perturbance torques are activated individually at a time, while all other torques are absent. Analyzing the attitude plots considering the environmental perturbance torques should give enough information about the correctness of, not only their implementation, but of how the satellite reacts to these torques. To stimulate a clear analysis of these torques, the center of mass offset vector is switched from its regular setting of $\boldsymbol{d} = [0\ 0\ 0.002]^T$ to the same distance, but in other axes, causing different torques on the satellite. Additionally, both the initial conditions for the orientation angles and for the angular velocity are set to 0 on all axes. Since what is being tested is how external torques individually affect the offset between the orbit and body reference frames, the angular velocity of the orbital frame, $\boldsymbol{\omega_o}$, is set to 0 on all axes, making it immobile. This setting for $\boldsymbol{\omega_o}$ is valid for this test only and is not considered anywhere else on this work.

4.3.1 Gravity Gradient Torque

As shown in Sec. 4.2 the gravity gradient torque helps stabilize the spacecraft in the expected position, leading to the conclusion that it is well implemented.

4.3.2 Residual Magnetic Dipole Torque

Since the equation for the residual magnetic dipole torque is analogous to that of a magnetorquer and such torque has been successfully implemented in Sec. 4.2, one can infer that it is likely working correctly.

4.3.3 Aerodynamic and Solar Radiation Torque

The aerodynamic torque is resultant of a force applied on the direction opposite to that of o_x, meaning that given an offset in the b_z axis $\boldsymbol{d} = [0, 0, 0.002]^T$, the torque generated should make the satellite spin clockwise about b_y. If the offset vector is set to the b_y axis $\boldsymbol{d} = [0, 0.002, 0]^T$, a counter clockwise rotation about the b_z axis should occur. An offset on the b_x axis obviously produces no torque.

In the case of the solar radiation torque, resultant of a force being applied on the direction of o_y, an offset in the b_x axis $\boldsymbol{d} = [0.002, 0, 0]^T$ should cause a counter clockwise rotation about b_z. An offset in the b_z axis $\boldsymbol{d} = [0, 0, 0.002]^T$ should cause a clockwise rotation about b_x.

Considering these conditions, the satellite behaves as depicted by Fig. 4.2.

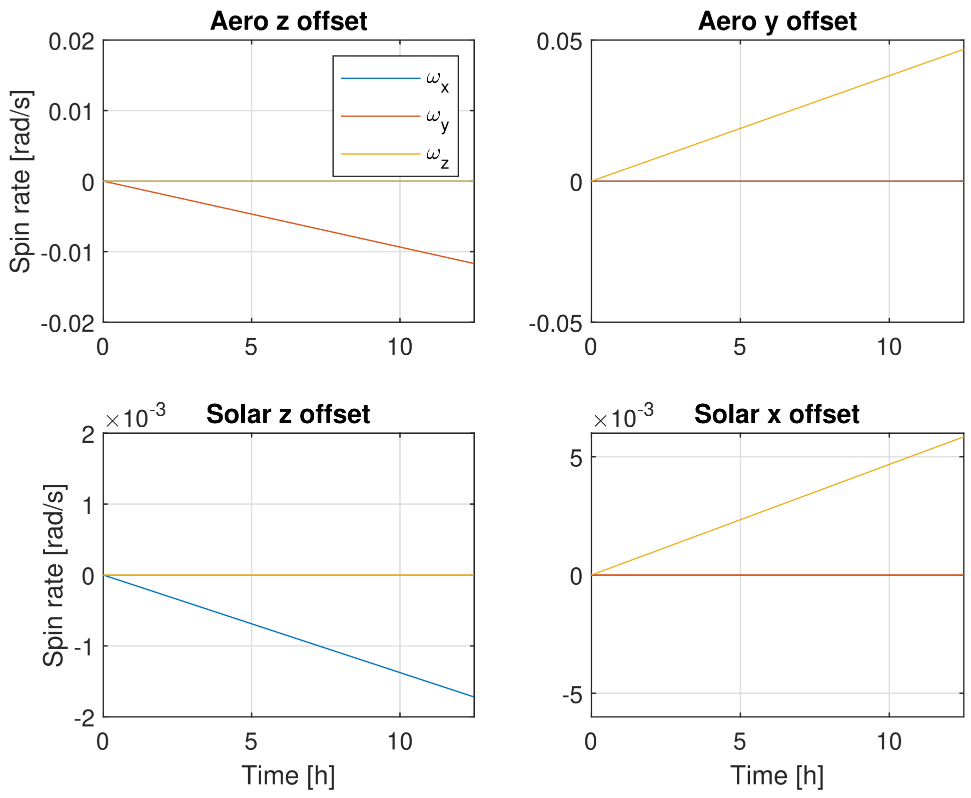


Figure 4.2: Spin rates of the body frame in relation to the orbit frame when different torques and offsets are tested.

All the torques produce the expected results. One can conclude that the environmental torques are well implemented, as the satellite attitude model is. With that in mind, the LEO environment and the uncontrolled behaviour of the satellite are examined in Ch. 5.

Chapter 5

Uncontrolled Satellite Dynamics

Before implementing an attitude control law, the Uncontrolled Satellite Dynamics chapter analyzes the behaviour of the 3-AMADEUS satellite when no controlling torque is applied to it, but all perturbance torques that are present in LEO are. Looking at what happens to the satellite's attitude in these circumstances should give a better understanding of exactly what effects control torques have. Additionally, the LEO environment in which the spacecraft lies is examined, including the magnitude of all the perturbances.

5.1 LEO Environment

Considering the orbit and the parameters specified in Ch. 4, the perturbance torques to be found in LEO are described by Fig. 5.1, in the orbit frame.

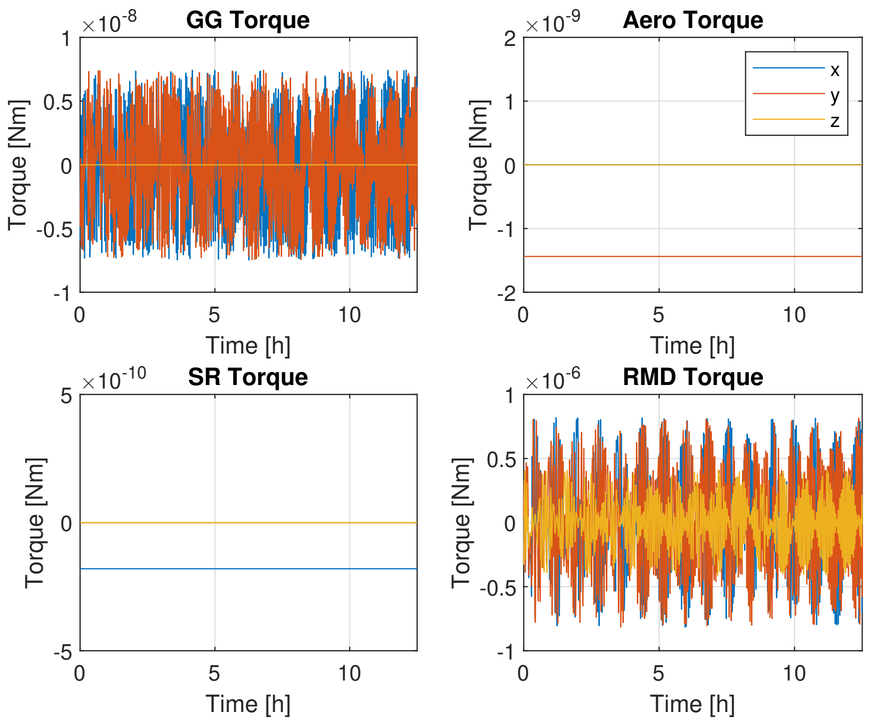


Figure 5.1: Perturbance torques in the LEO environment of the 3-AMADEUS mission in the orbit frame.

Both the aerodynamic drag torque and the solar radiation torque are constant. This makes sense considering how they are implemented. Additionally, they both provide a negative torque, causing the satellite to spin in a clockwise direction. Looking at the gravity gradient, one can see that it is very chaotic for the o_x and o_y axes, but is null in the o_z axis. This also makes sense since it causes the b_z axis to align with the zenith direction, not to spin about it. The residual magnetic dipole also seems somewhat chaotic but one can actually see the influence of the magnetic field given by the dipole model used, as visualized in Fig. 3.7.

In terms of magnitude, at $1\times10^{-6}Nm$, the residual magnetic dipole surpasses the other three torques, and so it is the one that the satellite will have the most trouble countering, followed by the aerodynamic torque; the gravity gradient torque, at $1 \times 10^{-8}Nm$, is helpful in providing orbital attitude. In light of that, the solar radiation torque is somewhat neglectable.

In the body frame, the torques are represented in the way that most accurately represents how they affect the body frame's spin, but they are too chaotic to be analyzed as can be seen from Fig. 5.2.

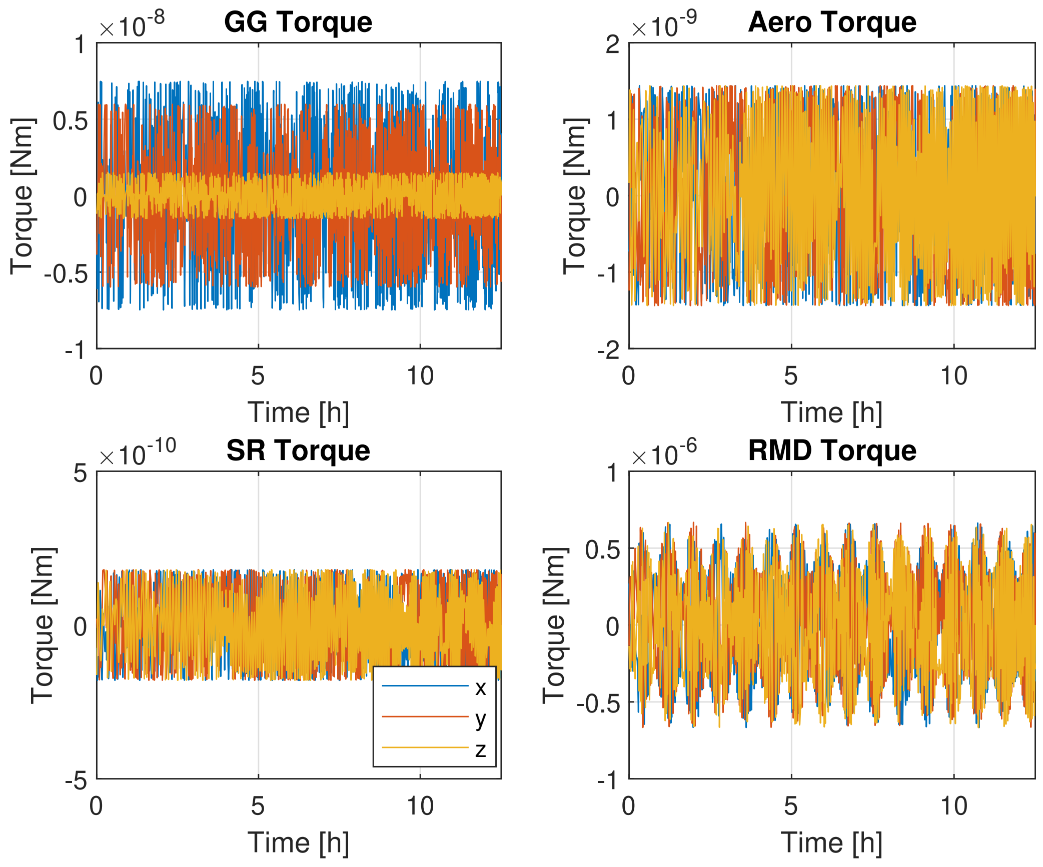


Figure 5.2: Perturbance torques in the LEO environment of the 3-AMADEUS mission in the body frame.

5.2 Uncontrolled Behaviour

Considering that the 3-AMADEUS is subject to all these torques, and that this simulation considers the starting angular rates described in Eq.(4.11), the orientation angles and the relative angular rate evolve as described by Fig. 5.3.

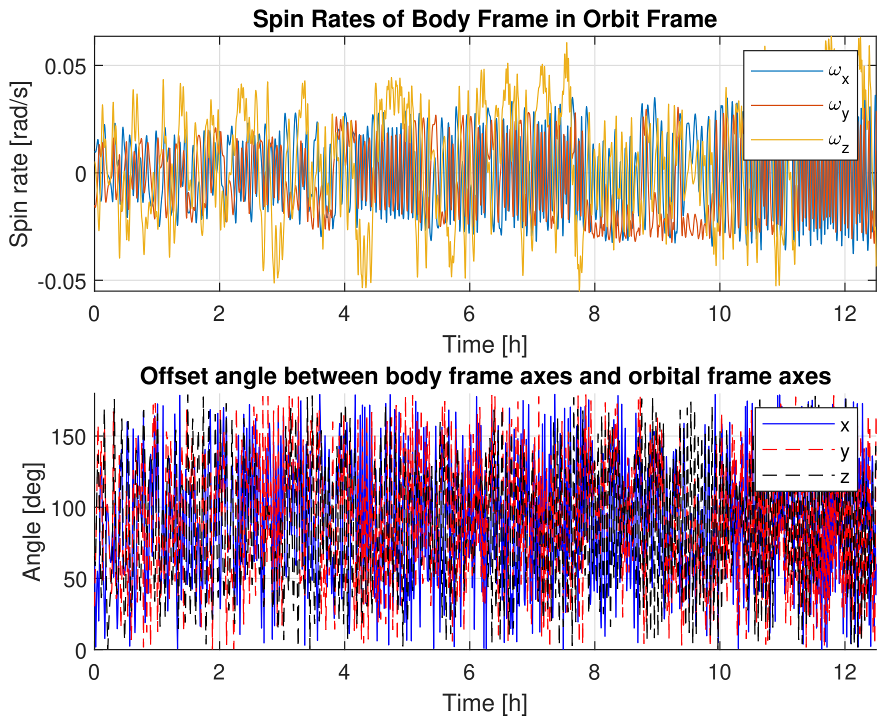


Figure 5.3: Evolution of the angular rate and orientation angles of the uncontrolled satellite.

This shows that the evolution of the orientation angles as well as the spin rate is completely erratic. In fact, if left uncontrolled, the spin rate of the satellite tends to increase over time, indicating that control torque is crucial for the 3-AMADEUS satellite. In Ch. 6 this effect is avoided by using the ADCS algorithm from [1].

Chapter 6

Mathematical Model For Attitude Control

This chapter aims at testing the ADCS algorithm from [1] by implementing the respective control law in the 3-AMADEUS CubeSat, in order to verify the viability of its implementation in this mission, as well as to demonstrate the methodology for this kind of study. This chapter's simulations run in the same LEO environment that has been analyzed in Ch. 5, a quite realistic model proposed for this mission. Initially, the control law of this algorithm is presented and afterwards five tests are made. The first one is performed assuming no torques besides gravity gradient torque; the second one encompasses all perturbance torques and assumes that there is no knowledge of the residual magnetic dipole, while the third assumes that there is such knowledge. The fourth test assumes all that has been tested previously but includes the presence of uncertainties on the attitude data. Lastly, the same as fourth test is performed but this time doubling the center of mass offset.

6.1 The Control Law

The control law in study [1], is given by:

$$\boldsymbol{m_{trqr}} = -k_{\omega}\boldsymbol{B_b} \times \boldsymbol{\omega_{ob}^{b}} - k_a\boldsymbol{B_b} \times \boldsymbol{S} \quad (6.1)$$

where **S** is given by Eq.(6.2).

$$\boldsymbol{S} = [a_{23} - a_{32}, a_{31} - a_{13}, a_{12} - a_{21}]^T \quad (6.2)$$

In the above equation a_{ij} are the elements of R_O^B, the rotation matrix from orbit to body frame.

The goal of this algorithm is to provide orbital attitude control to the satellite, that is, to align its body axes with the axes of the orbit frame. More specifically, the objective is to stabilize the satellite in a nominal position, where the orientation angles are all 0. The first term of this control law is responsible for reducing the angular rate while the second term is responsible for driving the satellite into the desired attitude. This control law is deeply reliant on the gravity gradient torque, as this torque drives the satellite to the desired attitude when in the vicinity of the nominal position. In addition to the gravity gradient torque, the magnetorquer control torque acts to diminish the offset between the body and orbit frame axes.

6.2 Attitude Control Tests

6.2.1 Algorithm Verification

Firstly, to verify the algorithm, it is tested by putting the 3-AMADEUS CubeSat in an environment where no perturbance torques exist, beside the gravity gradient torque. The initial conditions are those defined in Sec. 4.2. Using $k_\omega = 8,000\ \frac{Nm}{T^2}$ and $k_a = 2\ \frac{Nm}{T^2}$ the satellite behaves as shown in Fig. 6.1.

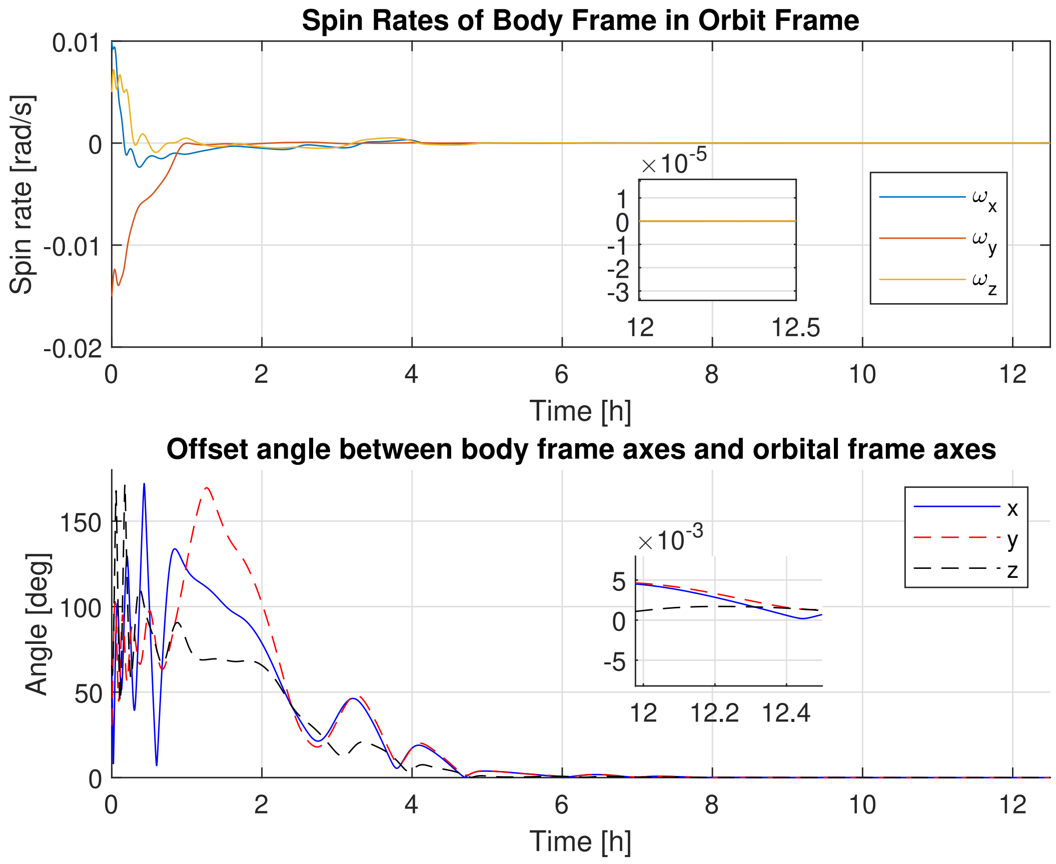


Figure 6.1: Orientation angles and spin rate - Ideal Environment.

The satellite behaves exactly as expected, as all orientation angles tend to 0 and stabilize there. The algorithm works, now it is crucial to understand how its performance changes in the presence of the perturbance torques in a LEO environment.

6.2.2 Attitude Control with no RMD estimation

Using the exact same model but with $k_\omega = 20,000\ \frac{Nm}{T^2}$ and $k_a = 2\ \frac{Nm}{T^2}$ and all disturbance torques active one gets the behaviour described by Fig. 6.2.

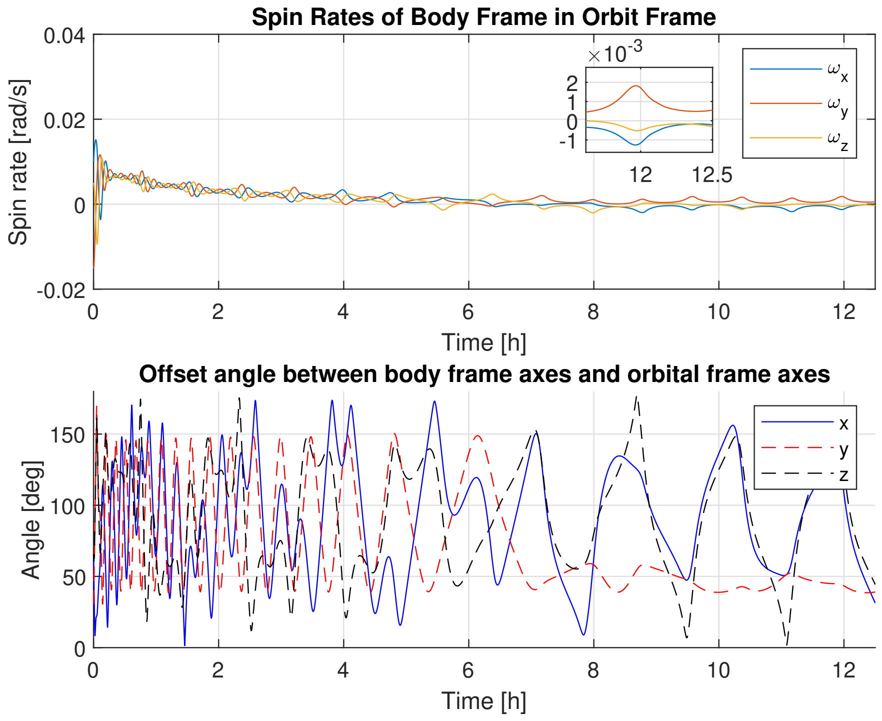


Figure 6.2: Orientation angles and spin rate - LEO Environment, no RMD estimation.

Even after several adjustment of the control gains in order to optimize the attitude control, the results are clear: the satellite can't avoid oscillating around the desired attitude with quite a large amplitude. Despite this, in terms of spin rates, this algorithm is somewhat viable since it manages to decrease the spacecraft's spin rate to about $2 \times 10^{-3}\ rad/s$.

The reasoning behind this high amplitude oscillation is due to the satellite's low mass and size and subsequent inertia tensor. Since this algorithm is so dependent on the gravity gradient torque for attitude control, it requires that, in order to achieve high accuracy, the gravity gradient torque - whose magnitude is proportional to the satellite's mass and size - is of at least the same magnitude as the disturbing torque of largest magnitude - the RMD torque. This isn't the case as can be seen in Sec. 5.1. In fact, if one increases the simulation time, uses $k_\omega = 5,000,000\ \frac{Nm}{T^2}$ and $k_a = 5,000\ \frac{Nm}{T^2}$ and uses the inertia tensor provided by [1] $J = diag(3.2, 5.2, 2.2)[kg\ m^2]$, the results are much better, with an accuracy of about 4°, as depicted by Fig. 6.3.

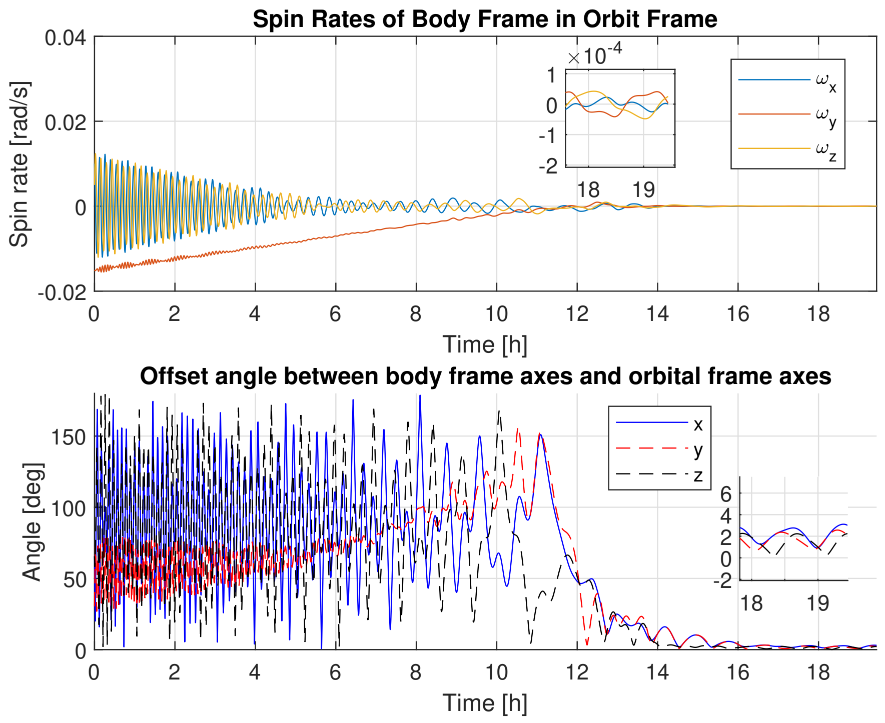


Figure 6.3: Orientation angles and spin rate - LEO Environment, large satellite.

Considering these results, it is clear that this algorithm, as it is, is suited for larger spacecraft but not for a very small and light 1U CubeSat, as is the case of 3-AMADEUS. But if one could estimate the residual magnetic dipole, and compensate for it in real time, could this algorithm work? Sec. 6.2.3 analyzes this possibility.

6.2.3 Attitude Control with RMD Estimation

If one knows at all times what the residual magnetic dipole is in all axes, then the magnetorquer can generate a control magnetic dipole of equal magnitude but in the opposite direction, cancelling the torque of the residual magnetic dipole in real time. If that is the case, then the gravity gradient torque can better stabilize the satellite since it is of a larger magnitude than the other torques, as has been seen in Sec. 5.1. With this assumption, one can create an alternate version of the control law:

$$\boldsymbol{m_{trqr}} = -k_\omega \boldsymbol{B_b} \times \boldsymbol{\omega_{ob}^b} - k_a \boldsymbol{B_b} \times \boldsymbol{S} - \boldsymbol{m_{res}} \quad (6.3)$$

it is now considered that there is RMD knowledge. To implement such knowledge in the attitude model, the model considers a constant RMD of $m_{res} = 0.01$ (in reality it has to be estimated in real time since it is not constant).

With this new control law, and using the control gains $k_\omega = 2,600 \ \frac{Nm}{T^2}$ and $k_a = 2 \ \frac{Nm}{T^2}$, the 3-AMADEUS satellite behaves as described by Fig. 6.4.

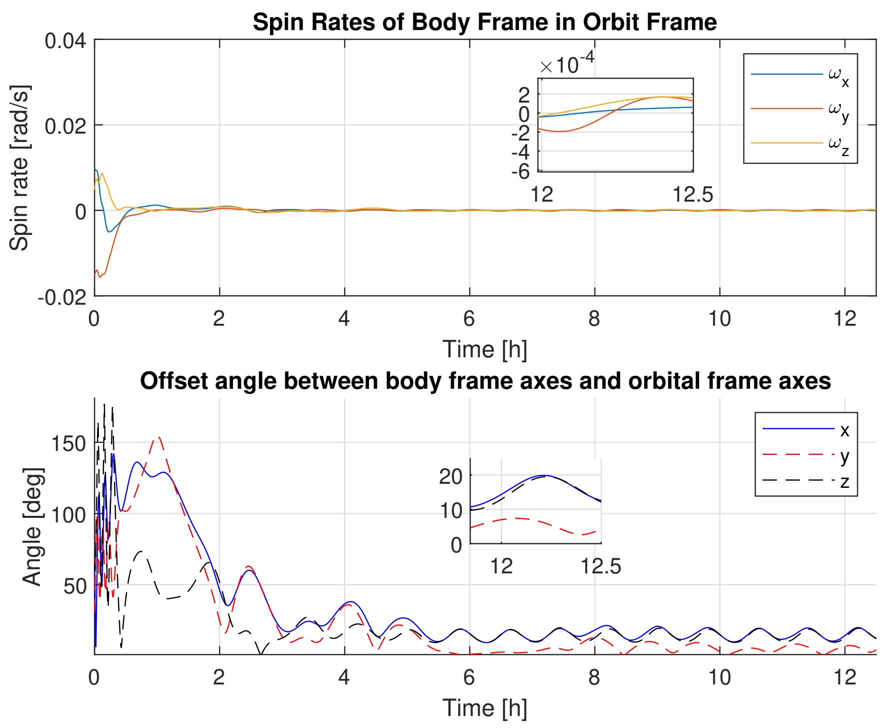


Figure 6.4: Orientation angles and spin rate - LEO Environment, with RMD estimation.

The control law does provide orbital attitude with an accuracy of 20° for the b_x and b_z axis and of 9° for the b_y axis. Depending on the mission requirements, this control algorithm can be applicable, despite its accuracy being relatively low. The satellite also stabilizes its spin rate with a 2×10^{-4} rad/s accuracy, which is acceptable.

This study has shown the performance of this algorithm but it has assumed that the inputs to it are completely correct, but errors in attitude determination must be accounted for, if these results are to be of any relevance to the actual mission.

6.2.4 Attitude Control with RMD Estimation and Uncertainties

The same test as the previous one is performed, but this time assuming a 20% error on the reading of the relative angular velocity, of the magnetic field and of **S**. For the RMD, an error of 0.1% is considered. Through a series of tests, these errors have been found to be the maximum the control algorithm can endure before losing the ability to stabilize the satellite within the vicinity of the nominal attitude.

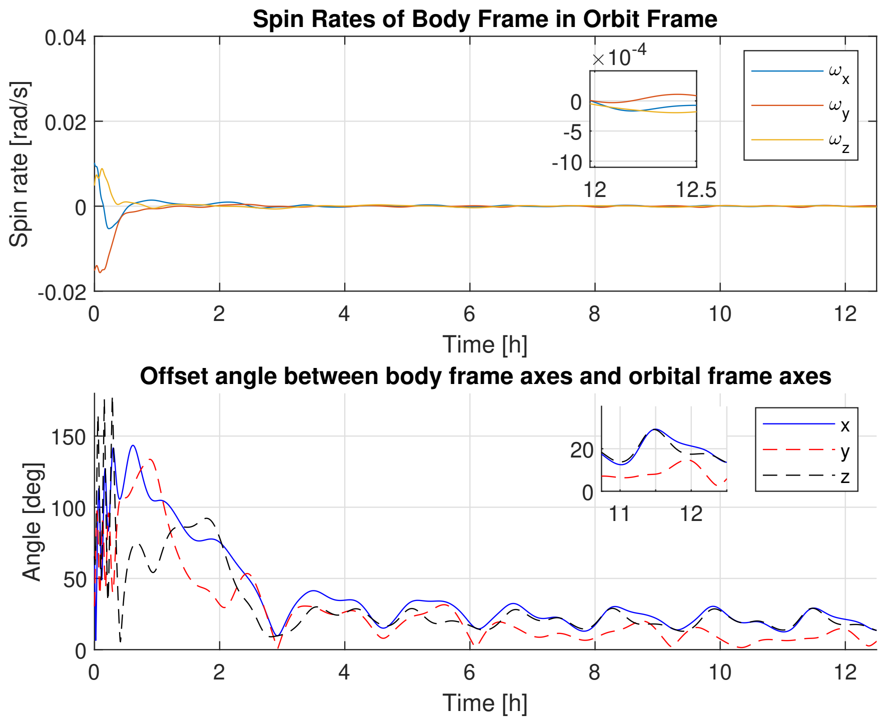


Figure 6.5: Orientation angles and spin rate - LEO Environment, with RMD estimation and uncertainties.

As can be seen from Fig. 6.5, with all these limitations, the accuracy drops even further to 31° on the b_x and b_z axes and 15° on the b_y axes. This, again, is far from ideal but can be used in some scenarios.

6.2.5 Attitude Control with RMD Estimation, Uncertainties and Larger Center of Mass Offset

The offset of 2 mm that has been considered is not a worse case scenario. It has been found that the algorithm still provides attitude control with an offset increased to 4 mm. However, in that case the disturbance torques double and the accuracy decreases even further, as depicted by Fig. 6.6.

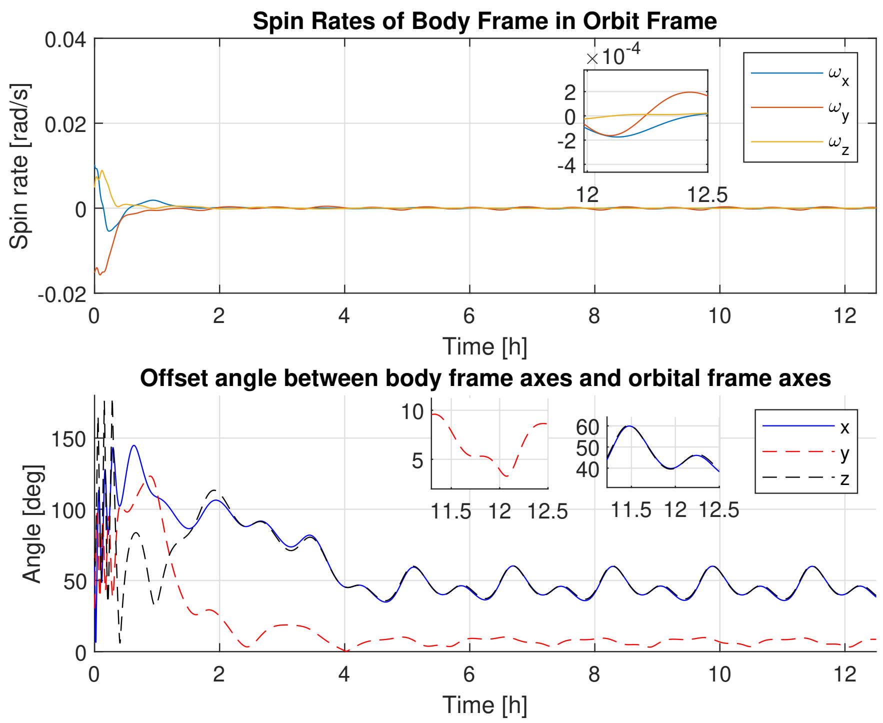


Figure 6.6: Orientation angles and spin rate - LEO Environment, with RMD estimation, uncertainties, and 4 mm offset.

The main issue with implementing this algorithm in the 3-AMADEUS CubeSat is how small and light it is, since the algorithm is best suited for a different class of satellite. Despite this, the algorithm works to an extent in detumbling the spacecraft after deployment. Considering that in the context of the 3-AMADEUS mission, the satellite's low size and mass can't be altered, the main issue is the RMD estimation, which must be done with a maximum error of 0.1%. Nevertheless, the algorithm seems very tolerant of bad readings for the other parameters, which is a good sign.

In Ch. 7, a HIL simulation is made considering the alternate control law, RMD knowledge and uncertainties, with an offset of 2 mm. Consequently, the results of this test should match the results of Fig. 6.5.

Chapter 7

Attitude Control HIL Simulation

In this chapter, the results that have been obtained in the model with RMD estimation, uncertainties and an offset of 2 millimeters are replicated. This time, the test is performed using the FPGA present in an ABACUS OBC as a processing unit for the attitude control algorithm, instead of the personal computer in which the simulation is ran, with this process being a HIL simulation. As for the rest of this work, attitude determination is not considered. If the results match the one that have been discussed in Ch. 6, with the algorithm being processed by the simulating environment, then the FPGA implementation is validated, not only for this algorithm but for ADCS algorithms for the 3-AMADEUS mission in general.

Firstly, the software tools and hardware components used to make this HIL simulation possible are presented. Afterwards, the assumptions and simplifications that have been made are discussed. Additionally, the FPGA design that has been used for the ADCS algorithm implementation is presented. Lastly, the HIL Simulation is performed and its results discussed.

7.1 Components and Tools Used

7.1.1 Components

The implementation of the control algorithm is made in a Xilinx Spartan 3E Starter Kit board, that contains the Spartan 3E FPGA. This FPGA is the same as the one present in the ABACUS OBC. The implementation is valid for the FPGA itself, meaning that it works exactly the same if implemented on the ABACUS OBC, with just the User Constraint File (UCF) - the file that associates signals in the code with pins on the actual board - having to be slightly altered.

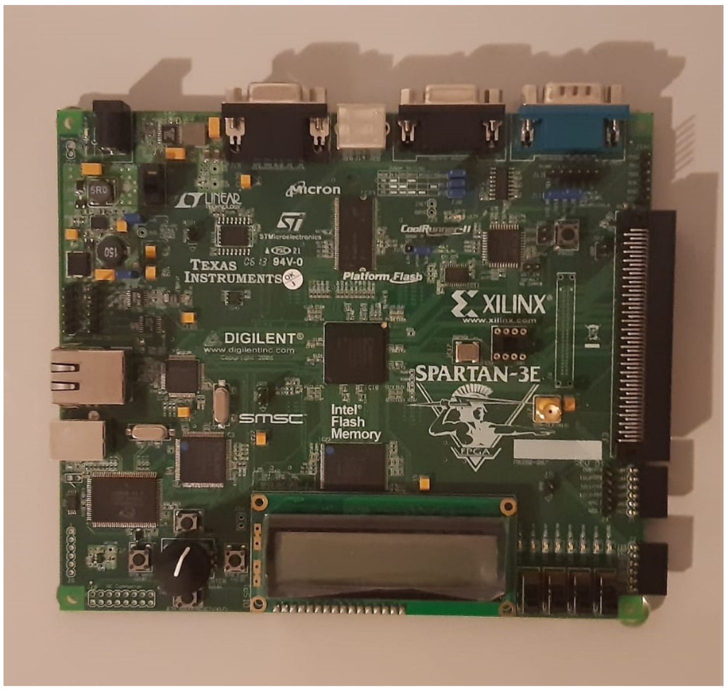


Figure 7.1: The Xilinx Spartan 3E Starter Kit.

To connect the FPGA and the computer running the simulation, an USB to RS-232 cable is used, just like the one depicted in Fig. 7.2, with visible wiring.

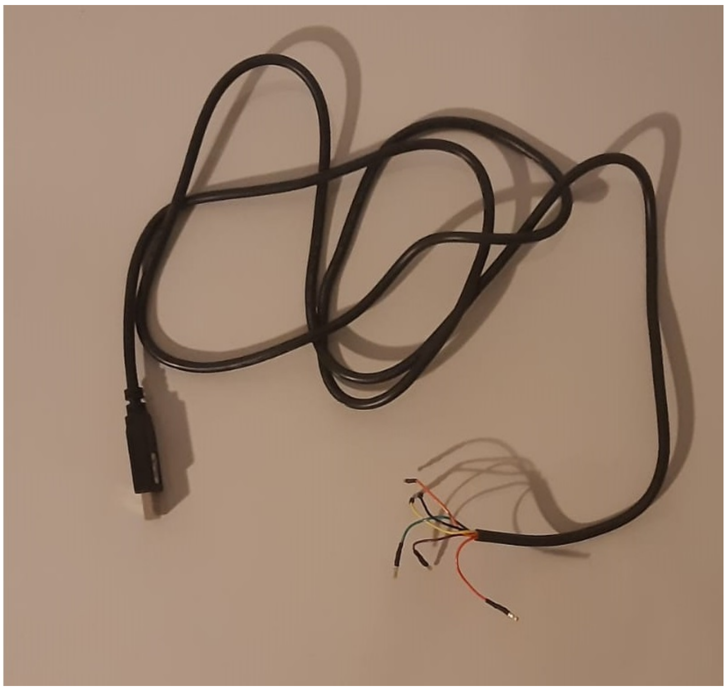


Figure 7.2: An USB to RS-232 cable.

To download the bitstream file onto the FPGA, an additional cable is required. In this case, it is the platform cable available with the Spartan 3E starter kit, an Universal USB Type B cable.

Figure 7.3: Xilinx platform cable.

Finally, the simulation is run on MATLAB software on a Windows 10 PC. Due to platform cable compatibility issues, the FPGA programming is done using a Windows XP virtual machine that itself runs the iMPACT software for bitstream downloading.

7.1.2 Tools

As mentioned in Ch. 3, the programming of Xilinx FPGAs is mostly done using their IDE, Vivado. However, due to the fact that the Spartan 3E is a 15 years old FPGA, the most recent version of the Xilinx IDE doesn't support it. As a result, it is necessary to install and use the Xilinx ISE Design Suite 14.7, an outdated version. The design entry is done on this software, with the VHDL code and the UCF file being defined there. The simulation of the design is made using the ModelSim software, simply for a matter of preference, since a similar tool is available in the IDE. As mentioned earlier, the programming of the FPGA and its PROM is done separately using the iMPACT software on a Windows XP virtual machine.

To communicate with the FPGA, the serial communication features of MATLAB are used.

7.2 Assumptions

Attitude determination is beyond the scope of this thesis, therefore one must assume that the FPGA receives all the required attitude data - the magnetic field, angular velocity, **S**, and RMD readings - directly. The objective of this FPGA implementation is then to directly receive all attitude data from MATLAB, calculate the required magnetorquer current to generate the required dipole specified by the control law in [1] and send it back to MATLAB, so that it can insert it into the next step of the simulation.

If one takes a look at a common magnetometer's datasheet [100], it is possible to see that the magnetometer data is sent as a 16-bit word divided into 2 bytes, meaning that the range of the data is actually limited to 16 bits. Having this in mind, to make this implementation more realistic, it is considered that all attitude data that is sent to the FPGA must be comprised into 16 bits, more specifically as 16-bit signed numbers. This means that the data must be scaled to fit within that range. The output current must also be sent as 16-bit signed numbers.

Scaling by powers of 2 facilitates hardware implementation since only a bit shift is required to obtain the scaled data. For example, if the angular rate is of $-0.0144\ rad/s$, it must be scaled by a factor of 2^{20}. This means that one gets approximately $-0.0144 \times 2^{20} \approx -1511[2^{-20}\ rad/s]$, which is the most accurate representation that one can get with 16 bits. This scaled data is what the FPGA receives. The scale for the magnetic field is of 2^{29}, for $\boldsymbol{S}$ of 2^{13} and for $\boldsymbol{\omega_{ob}^b}$ and for $\boldsymbol{m_{res}}$ it is of 2^{20}. Finally, the current that is passed to the magnetorquers, the output of the FPGA, is in turn scaled by 2^{19}. Let's now review the actual design of the FPGA.

7.3 FPGA Design

7.3.1 Design Requirements

The first step in the design of every FPGA (and system in general) is to determine what the design requirements are, and then devise what needs to be done to achieve said requirements.

For this work, it is intended that the FPGA outputs the required current for the magnetorquer rods in each axis, based on the control law from Eq. (6.3). For that, it is required that the FPGA implementation considers the control law as three separate equations, replacing the cross product with products between scalars. Additionally, the required dipole is converted into required current, by multiplying it by Z:

$$\begin{aligned} I_{req_x} &= Z(\ k_\omega(B_{b_z}\omega^b_{ob_x} - B_{b_x}\omega^b_{ob_z}) - k_a(B_{b_z}S_x - B_{b_x}S_z)\ - m_{res_x}) \\ I_{req_y} &= Z(\ k_\omega(B_{b_x}\omega^b_{ob_y} - B_{b_y}\omega^b_{ob_x}) - k_a(B_{b_x}S_y - B_{b_y}S_x)\ - m_{res_y}) \\ I_{req_z} &= Z(\ k_\omega(B_{b_y}\omega^b_{ob_z} - B_{b_z}\omega^b_{ob_y}) - k_a(B_{b_y}S_z - B_{b_z}S_y)\ - m_{res_z}) \end{aligned} \quad (7.1)$$

where Z is simply the inverse of the total magnetorquer area. Considering the typical magnetorquer data found in [99], with n_{coil} as the number of coil turns and A_{coil} as the area for each coil turn, Z is given by Eq.(7.2).

$$Z = \frac{1}{n_{coil}\ A_{coil}} = \frac{1}{78 \times 0.0039} = 3.2873\ [m^{-2}] \quad (7.2)$$

Now that it is clear what the FPGA needs to do, it is necessary to define how well this task must be performed. The main issues to consider are the precision of the output and the processing time of this algorithm. The processing time is hardly a limitation in this context since when running the simulation with a 1 second step, the results have been fine. This means that any processing time that is below that, works. In terms of precision, the limitations are somewhat greater: it has been found that for the algorithm to be accurate, the current must be precise, with an error below $1 \times 10^{-5} A$. So now the three requirements for this design are set: It must receive 14 (3 for each axis of each parameters plus gains) 16-bit numbers for attitude data and output 3 16-bit numbers for the current (one for each axis), do this process in less than 1 second, and have an error that is below $1 \times 10^{-5} A$, when compared to the real required current.

7.3.2 Proposed Architecture

Overview

Let's start with the definition of the FPGA inputs and outputs. The FPGA features a UART transceiver and has only one input pin, called data_in, through which it receives 16-bit attitude data from MATLAB, and one output pin, called data_out through which it sends the 16-bit required current to MATLAB. Internally, a clock signal synchronizes the processes inside the FPGA.

To achieve a design that allows for the precision requirements to be met, obviously, a word length of 16 bits isn't enough to perform the necessary mathematical operations, especially considering that it is necessary to multiply numbers whose products wouldn't fit in a 16-bit format. The proposed solution is, after receiving the input numbers from the UART transceiver, to convert them into another format, inside the FPGA. More precisely, a 64-bit format using fixed point representation with 16 integer bits and 48 fraction bits is used, allowing for the desired precision. In fact, since the new format allows it, the numbers are not only formatted to 64 bits but also scaled so that their new 64-bit representation now represents their true value and not their scaled value that is received by the FPGA.

After scaling and formatting, the FPGA should now hold 14 64-bit numbers and must now do the necessary computations with them so that one gets 3 64-bit numbers, corresponding to the required current. To achieve this, multiplications, additions and subtractions are done in accordance to Eq. (7.1), using the fixed point arithmetic discussed in Sec. 3.5.3.

Having the 3 desired outputs in a 64-bit format, the inverted process of the scaling and formatting done earlier is performed. The FPGA must now scale the 64-bit numbers so that their integer part becomes an integer as large as possible within the 16-bit interval (-2^{15} to 2^{15}). After that is done, the FPGA truncates the fractional bits, and gets a scaled 16-bit version of the 3 desired currents.

When the output data is available, it must be sent. With the design of a UART transceiver, it is possible to serially send the current data to MATLAB, so that it can insert it into the simulation. By driving the data_out line to low, the FPGA signals MATLAB to receive the incoming data. This data is sent by setting the data_out line to high or low depending on the binary number representing the current. The sending of data to MATLAB is the last step in the FPGA data flow. This implementation is made possible by the FPGA's design freedom, that allows for conversion between format at will, facilitating the meeting of the precision requirements.

All this operation is done using 7 concurrent processes that run continuously and simultaneously within the FPGA. Next, these processes are further looked into.

Processes

There is a main UART process, a state machine that, besides handling the data transmission to and from from MATLAB, drives to and gets data from all other processes. There is also another state machine, the Computations process. This process computes the required current while the UART process is in a waiting state. Additionally there is a process for each arithmetic operation and for each format/scale operation that constantly output the result of whatever their inputs are. Considering the light processing time requirement, this FPGA design is done via concurrent processes that are used at a time. This causes the hardware resources usage - one of the main issues of this design - to drop significantly.

All these processes execute concurrently and continuously, but its outputs and inputs are driven and read by the UART and computations blocks in a timely manner. A block diagram is presented in Fig. 7.4, displaying the FPGA design in a very top level and chronological manner.

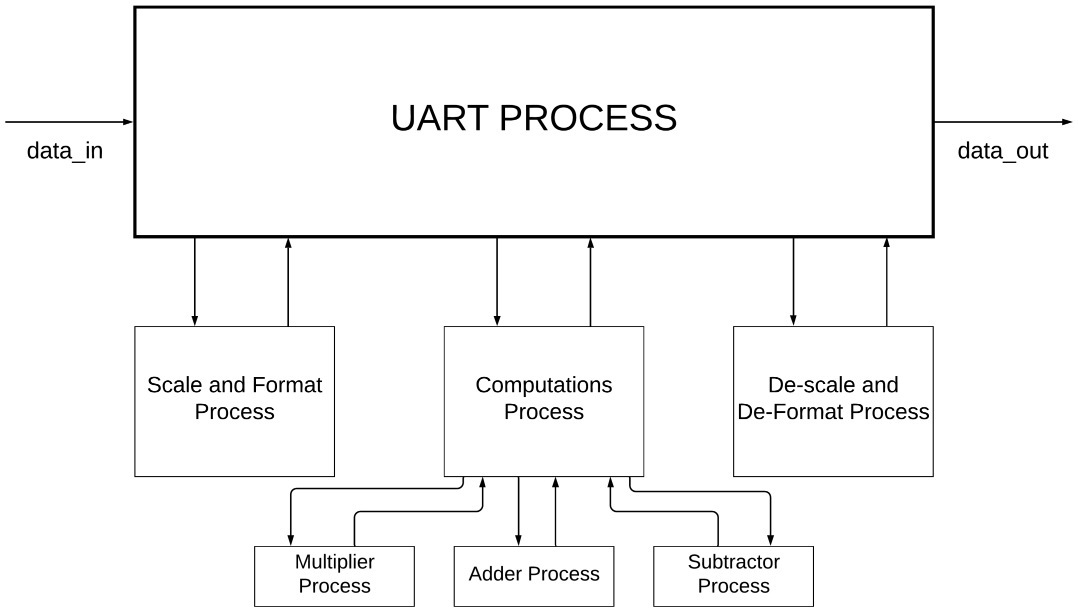


Figure 7.4: Block diagram of the proposed FPGA architecture.

UART Process

The UART process is the main process in this design as it interacts with all others processes and is responsible for receiving and sending data, the most fundamental aspect of the design. It has one main input and one main output, the data_in and data_out lines. Additionally, it has several other inputs and outputs for connecting with the formatting, deformatting, and the computations processes. This process receives data from the data_in line and assigns it to different signals. These signals are sent to other processes that treat the data sequentially. While those processes run, the UART process waits until their outputs are ready. When that happens, it sends the data they output to the next process and waits again, until the 3 16-bit output numbers are available. At that time, the UART process sends them through the data_out line, using the same UART protocol.

Computations Process

The computations process has 14 inputs that are directly connected to the 64-bit formatted data signals from the UART process. When the UART process goes into its waiting mode, the computations process starts computing the required current, one operation at a time until it outputs either the required current, or if that is too large, the maximum current that the magnetorquer rod can handle. When this process is done, the UART process proceeds with its execution. In reality, the computations process merely sends and receives data sequentially and orderly from and to the multiplier, adder and subtractor processes until finally it gets the required current and sends it to the UART process.

Scale and Format Process

The scale and format process has a single input, the 16-bit number that must be converted to the scaled 64-bit format. However, it has 3 outputs, one for each possible scaling factor. It works by appending 48 zeros to the right of the 16-bit number, and, depending on the output, shifting its bits to the right by 13, 20 or 29 bits (the scale factors discussed earlier). The UART process selects which output it requires depending on what data it gives to this process.

De-Scale and De-Format Process

The De-scale factor only has one input, a 64-bit word, and one output, a 16-bit word. Only one output is required since it only has current signals as its inputs, with all of them sharing the same scaling factor. It works reversely to the scale and format process, as firstly it de-scales the current by left shifting by 19 bits and then removes the 48 fractional bits entirely.

Multiplier Process

The multiplier process takes 2 64-bit (16 integer bits, 48 fractional) numbers, and outputs another 64-bit number that corresponds to their product. It works by creating a 128-bit word that holds the product of the 2 64-bit words and then truncating in on both sides, according to the fixed point arithmetic discussed earlier.

Adder/Subtractor Processes

The adder and the subtractor processes take 2 64-bit (16 integer bits, 48 fractional) numbers, and output another 64-bit number that corresponds to their sum/difference. They work by creating a 65-bit word that holds the sum/difference of the 2 64-bit words and then truncating by 1 bit on the left side, also according to fixed point arithmetic.

7.3.3 User Constraints

The User Constraints, the association of input and output signals in the code with actual pins on the board, must be done via a User's Constraints File. In this case there are three signals that must be associated with physical elements: The data_in, data_out and clock signals. Since the data is transmitted serially, it makes sense to use the available RS-232 serial ports available on the starter kit (these ports are also available on the ABACUS OBC). With that in mind, one must look in the board's datasheet [101] and find what locations the RS-232 Tx (associated with the data_out signal) and RS-232 Rx (associated with the data_in signal) are in, and associate them with their respective signals in the UCF. Additionally, since a clock signal is necessary for this design, one can use the built-in 50 MHz clock and associate it with the clock signal in the code.

In the case of this board, the association of the board LEDs with internal signals has been extremely useful when debugging the design.

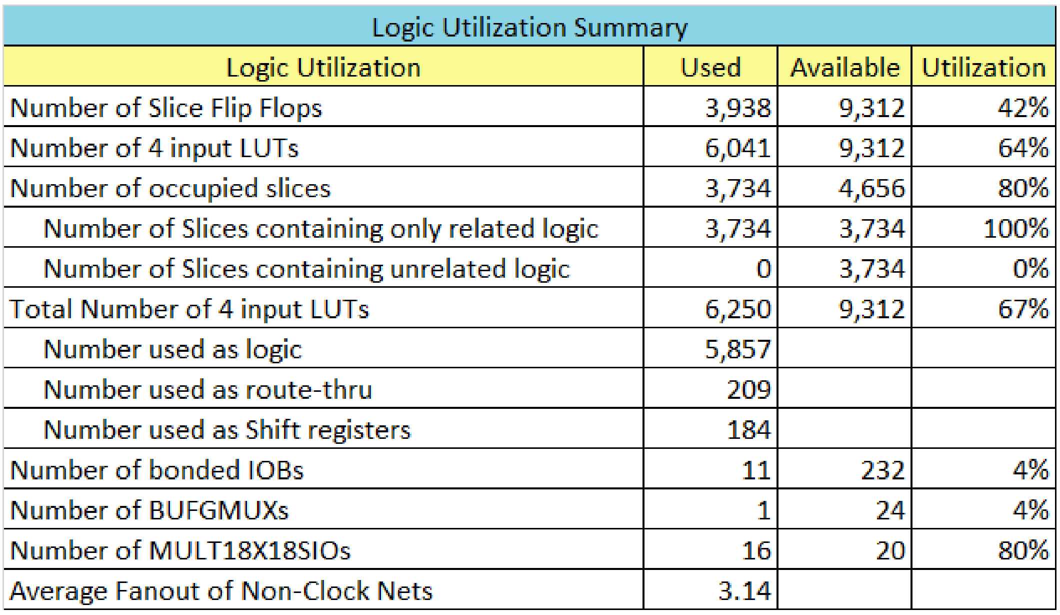
7.3.4 System Performance

Simulations can be used to assess system performance before implementation, although verifying the precision of this system using only one set of inputs is not very reliable. Having that in mind, the real precision test is made after the design is implemented, though the well functioning of the design can be checked using a simulator. Apart from that, what actually is checked using simulation is the processing time. The simulation results are present in Appendix A.

The output values obtained in the simulation correspond to the values expected for a given set of inputs, suggesting a good implementation of the control algorithm. The results also indicate that the processing time for this design is of about 0.0024 seconds, clearly under the limit set earlier.

7.3.5 Resource Usage

The ISE Design Suite produced the logic utilization summary described by Fig. 7.5.



Logic Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	3,938	9,312	42%
Number of 4 input LUTs	6,041	9,312	64%
Number of occupied slices	3,734	4,656	80%
Number of Slices containing only related logic	3,734	3,734	100%
Number of Slices containing unrelated logic	0	3,734	0%
Total Number of 4 input LUTs	6,250	9,312	67%
Number used as logic	5,857		
Number used as route-thru	209		
Number used as Shift registers	184		
Number of bonded IOBs	11	232	4%
Number of BUFGMUXs	1	24	4%
Number of MULT18X18SIOs	16	20	80%
Average Fanout of Non-Clock Nets	3.14		

Figure 7.5: Utilization Summary for the proposed FPGA design.

Despite the usage of concurrent processes having decreased resource usage by a massive amount (it was over 100% before optimization), the utilization percentage is still quite high. The high utilization percentage shown is expected to some extent due to the usage of 14 64-bit words and a multiplier for them. For future applications of more computationally heavy algorithms, this utilization can be further optimized by altering the code to ensure that block RAM is inferred rather than registers, for some of the signals.

7.4 HIL Simulation

7.4.1 Setup

In Ch. 4, 5, and 6, both the kinematic and dynamic equations and the ADCS algorithm have been computed in MATLAB. In the case of a HIL simulation, the dynamic equations are ran inside MATLAB but the ADCS computations are performed by the FPGA, as depicted by Fig. 7.6.

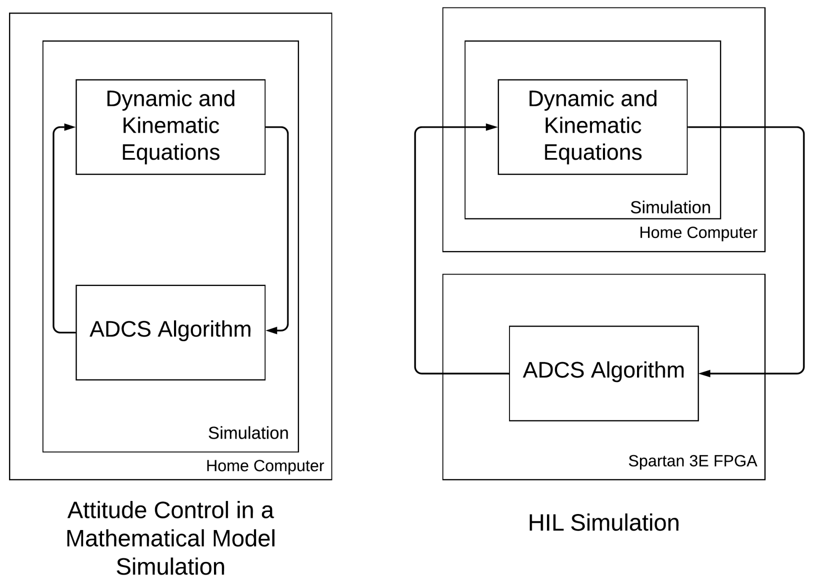

Dynamic and Kinematic Equations
ADCS Algorithm
Simulation
Home Computer
Attitude Control in a Mathematical Model Simulation
Dynamic and Kinematic Equations
Simulation
Home Computer
ADCS Algorithm
Spartan 3E FPGA
HIL Simulation

Figure 7.6: Block diagram comparing the mathematical and HIL simulations.

For MATLAB to exchange data serially with the FPGA, the USB to RS-232 cable must connect the computer and the FPGA, with the cable Tx connecting with the FPGA Rx and the FPGA Tx connecting with the cable Rx. Additionally, the programming cable is used to upload the program to the FPGA before running the simulation. A GND connection must also be made between the board and the RS-232 cable.

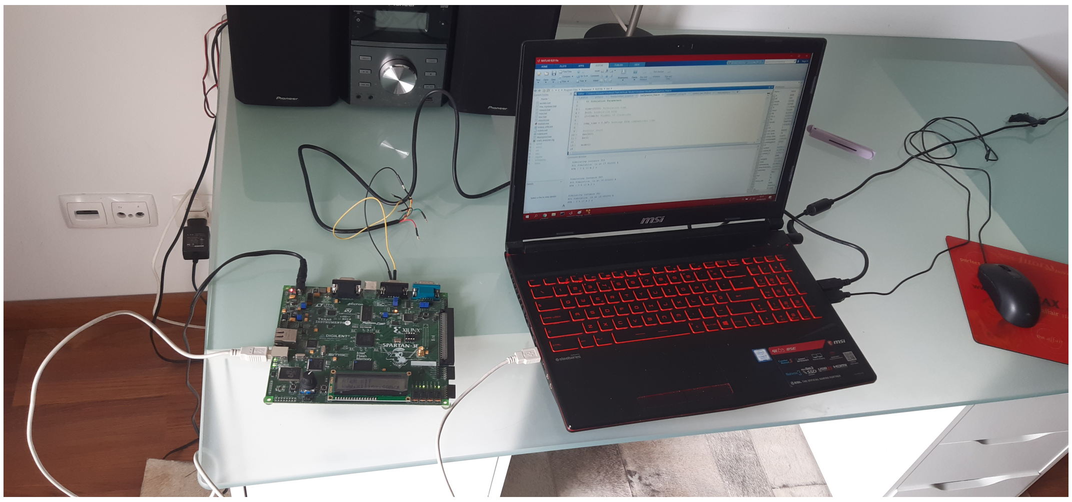


Figure 7.7: Real life setup used for the HIL Simulation

7.4.2 Results

Now that all the steps in designing and setting up the FPGA have been described and examined, the FPGA implementation can be tested by performing a HIL simulation. The objective is to reproduce the results of the test done in Sec. 6.2.4, where RMD estimation, estimating errors, and an offset of 2 mm have been considered. If the results using the FPGA as the ADCS processing unit match the results for that test, the FPGA implementation should be validated for the 3-AMADEUS mission. Note that attitude determination is not considered here.

The results for the HIL simulation using these conditions and the FPGA design presented earlier are described by Fig. 7.8.

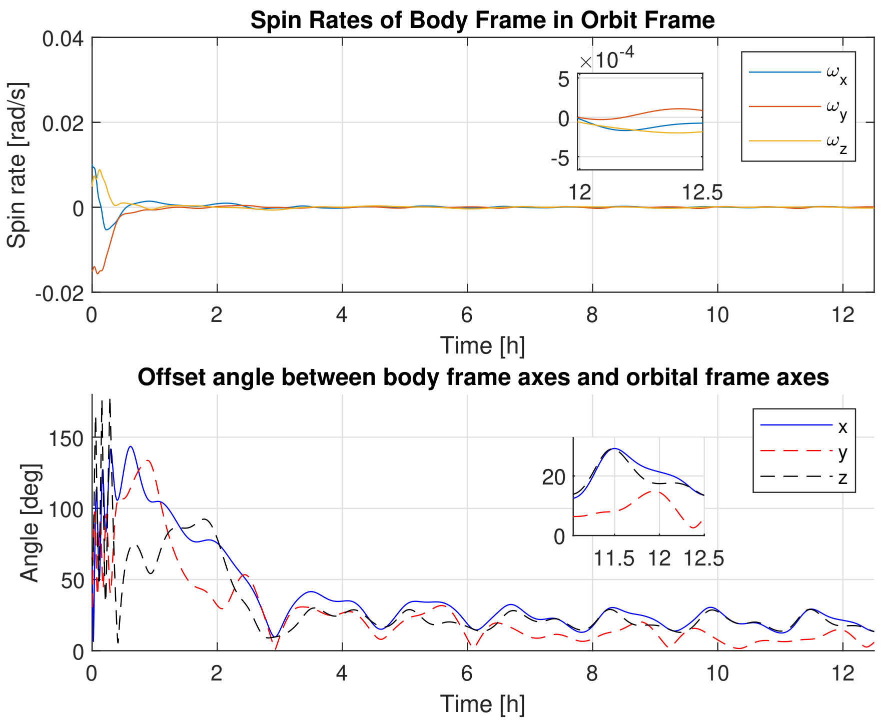


Figure 7.8: Orientation angles and spin rate in a HIL Simulation.

At first sight, it is possible to see that the simulation results match almost exactly the ones of Fig. 6.5. The little difference between the ideal and FPGA implementations indicates that the latter is well implemented. Despite this, it is important to examine more thoroughly the differences and consider their impact, if any. To do that, the σ_3 standard deviation of the offset between the required current value calculated by MATLAB and the one calculated by the FPGA, is examined. By doing this analysis, one can find the 99.7% confidence interval [102] for this offset, assuming of course that the attitude data received by the FPGA is within the foreseen limits.

Additionally, the same analysis is made for the relative angular velocity and for the orientation angles, more specifically the offset between the values of the simulation in Sec. 6.2.4 and the HIL simulation. This comparison should provide data on the performance of this implementation when compared to the earlier one.

Firstly, performing the analysis of the required current, one gets Fig. 7.9.

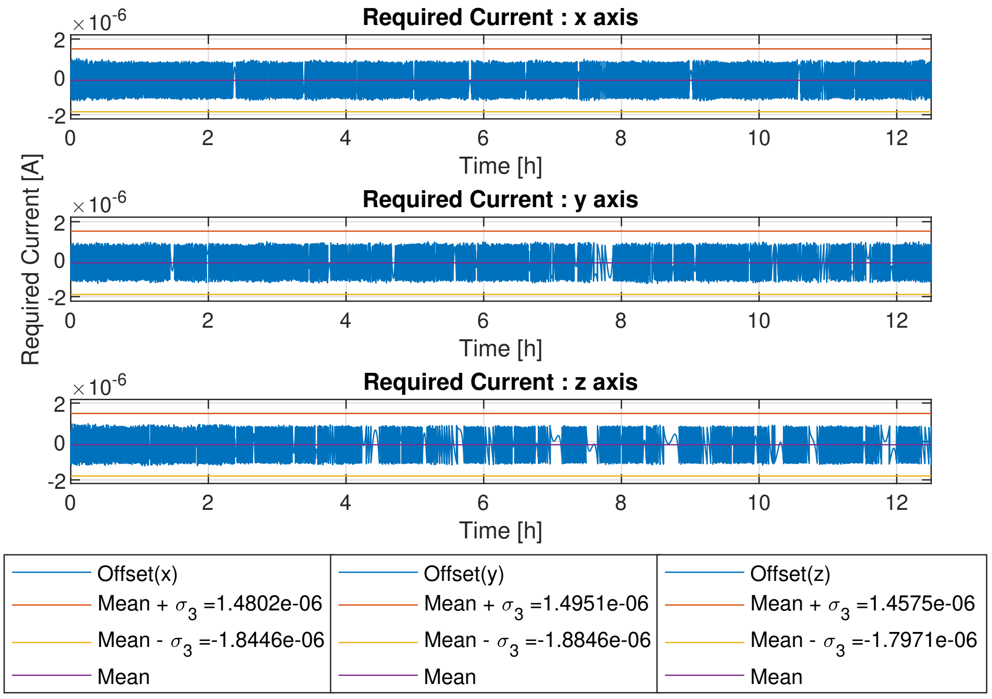


Figure 7.9: σ_3 analysis for the offset of the desired current.

The data show that all offset samples collected are below 1.1×10^{-6}, indicating that the precision requirement of a maximum error of 1×10^{-5} is fulfilled. Additionally, one can be 99.7% sure that the error is smaller than 2×10^{-6}. These findings indicate a good implementation of the ADCS algorithm.

In the case of the relative angular rate and of the orientation angles the results are given by Fig. 7.10 and Fig. 7.11.

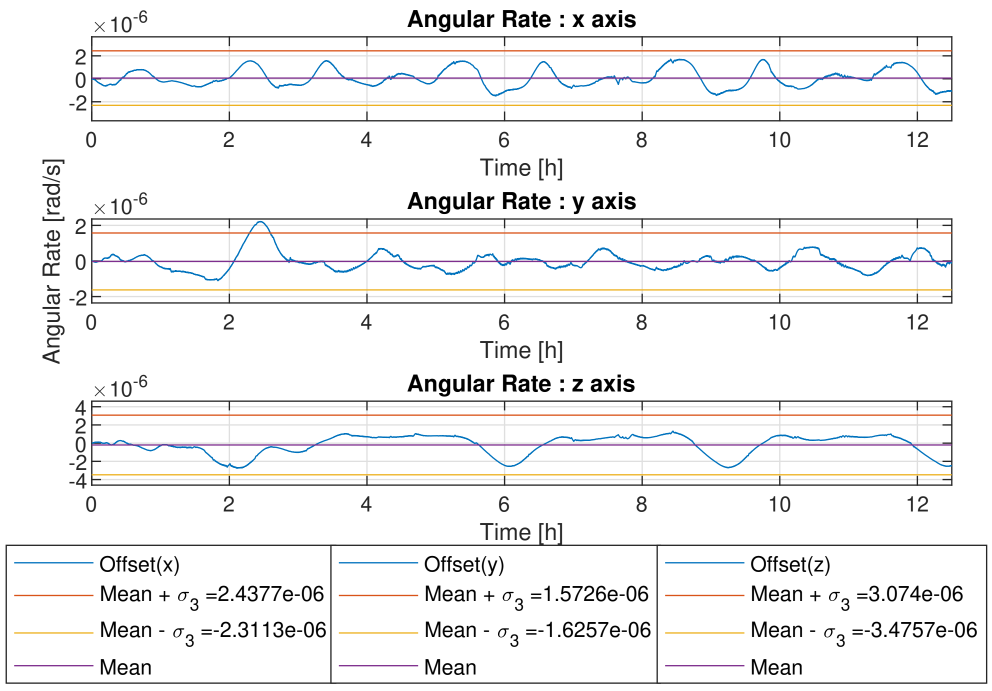


Figure 7.10: σ_3 analysis for the offset of the relative angular rate.

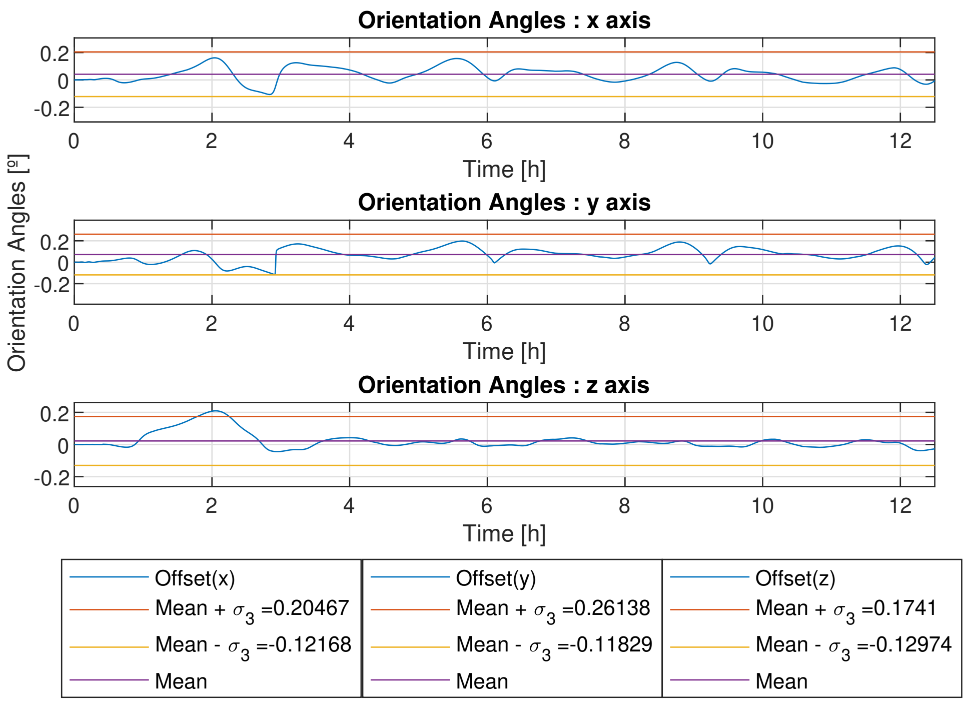


Figure 7.11: σ_3 analysis for the offset of the orientation angles.

As expected, due to the small differences in magnetorquer current, there are also small differences in these parameters.

For the angular rate, an extremely small difference below 4×10^{-6} is expected and its mean is very close to 0 on all axes, therefore this difference is completely negligible. As a result, one can state that the satellite stabilizes as fast with the FPGA implementation as it did earlier.

In terms of the orientation angles, the difference is somewhat more noticeable, despite also being very small. The analysis shows that, on all axes, 99.7% of all instances should have a difference that is below 0.3°. This error is negligible considering the actual precision of the algorithm and of most ADCS systems today. It is also noticeable that the difference is larger during the early stage of simulation, where the spin rate is larger, than on the end of the stabilization process, when the satellite approaches the nominal position.

Based on above results, one can conclude that the HIL Simulation and the associated FPGA design have been successful, with only minimal differences between the ideal and FPGA results. This lets one confirm the suitability of the ABACUS OBC and its Spartan 3E FPGA for the ADCS of the 3-AMADEUS CubeSat.

Chapter 8

Conclusions

8.1 Overview

The reduction of weight and consumption of ADCS components is of the uttermost importance to the further reducing of space access costs for small spacecraft and, consequently, to further develop the Small Sat industry. To alleviate this issue, magnetic ADCS components are often used, reducing costs but decreasing ADCS accuracy. The development of this technology, namely through the in-flight testing of novel ADCS algorithms, can potentially improve the performance of these magnetic components, and consequently lead to the broadening of their applications.

This thesis develops an attitude model in which this sort of algorithm can be tested, and performs said testing to an ADCS algorithm, assessing its validity for the 3-AMADEUS mission. Additionally, an FPGA design that implements this algorithm is proposed and tested in a HIL simulation to assess the viability of using an FPGA, not only for this specific algorithm, but for ADCS control algorithms in general.

The developed model is based on well documented equations and formulations and has produced the expected results for every set of conditions and parameters that it has been submitted to.

In terms of the control algorithm that has been studied, its functioning in principle has been proven in an unperturbed environment. When submitted to more realistic LEO conditions expected for the 3-AMADEUS mission, the algorithm manages to detumble the satellite, but the orientation accuracy is very low, mainly due to the presence of the residual magnetic dipole torque. This result is due to the fact that this algorithm relies immensely on the gravity gradient torque, which depends on the mass distribution and size of the spacecraft. 3-AMADEUS, being small and light, is not adequate for the application of this algorithm in LEO conditions. If, however, one assumes that it is possible to estimate the residual magnetic dipole in real time, it is possible to adapt the control law to compensate for the RMD torque. In this case, the spacecraft improves its orientation accuracy, although the results still aren't great, with a pointing accuracy of about 20°. Additionally, in the presence of a significant error in the attitude data readings, the algorithm still manages to detumble the satellite, keeping it oscillating in the vicinity of the nominal position, although the amplitude of these oscillations in these conditions is quite large.

In terms of the FPGA implementation, performing a HIL simulation using the FPGA as a processing unit for the ADCS algorithm provides virtually the same results as doing it in the numerical simulation environment. Nevertheless, one should keep in mind that for this HIL simulation to be performed, a few assumptions have been done and attitude estimation hasn't been considered.

Concluding, the gathered data lead one to believe that the model is correct and thus, its usage in the study of future ADCS algorithms is possible and recommended. The tested ADCS algorithm in particular succeeds at detumbling the satellite, but its orientation accuracy is quite low, since it has been developed for larger satellites. The analysis performed and the methodology used provide relevant data and can be applied to other algorithms. Taking a look at the HIL simulation results, the FPGA implementation has been successful and the FPGA design basis and methodology appear to be a good option for attitude control, not only for this algorithm, but for ADCS algorithms as a whole.

Finally, and with all this in mind, one can conclude that the objectives for this work have all been accomplished, with the only disappointment being the low accuracy of the control law used. This is a problem that can be fixed only by using a newly developed algorithm that is better suited for the 3-AMADEUS, or by including other types of ADCS actuators.

8.2 Constraints and Challenges

This work has two main constraints in the fact the algorithm doesn't stabilize the spacecraft very accurately, and in the fact that attitude determination isn't considered, which removes realism from this study.

During development of the attitude model, since the used control law isn't the best suit for the 3-AMADEUS CubeSat, it was difficult at times to grasp if that was the case or if the problem was with the model. Upon gaining a better understanding of how the control law worked, it has become clear it was the first option.

The FPGA design has also been quite troublesome, mainly due to the fact that the available documentation is often scarce and vague. One of the main issues to solve was the fact that the data to be exchanged between the computer and the FPGA is of different magnitudes. This has led to significant number of iterations being necessary until a solution was found. Even after solving this particular issue, the earlier designs took up too much resources when the bitstream generation was attempted. This has led to a deeper study regarding how the synthesis tools infer logic, and, consequently, to a complete revamp of the design. The MATLAB/FPGA interface was also quite difficult to grasp at first, and has been solved only when an oscilloscope was used to measure the voltage on the RS-232 lines.

8.3 Open Points and Future Works

In terms of works that can be developed in the future, an obvious choice would be to deepen this study by implementing attitude determination, both to the model and to the FPGA design. This would considerably improve the realism of this thesis and provide a work that is much closer to the final design for the 3-AMADEUS CubeSat.

Considering the low accuracy of the algorithm used for this thesis, it would also be interesting to do the same tests to the novel ADCS algorithms being developed at KIAM and assess their viability for the 3-AMADEUS mission.

It could also be interesting to improve the realism of the attitude model proposed by reducing the number of assumptions made. For example it could be interesting to model the perturbance torques considering a varying exposed area, considering a heterogeneous mass distribution for the computation of the inertia tensor, or even calculating the direction of the solar radiation torque more accurately, based on the positions of the Earth and the sun.

Bibliography

[1] D. Ivanov *et al*, “Advanced numerical study of the three-axis magnetic attitude control and determination with uncertainties,” 2017.

[2] A. Toorian *et al*, “The cubesat approach to space access,” 2008.

[3] H. Heidt *et al*, “CubeSat: A new Generation of Picosatellite for Education and Industry Low-Cost Space Experimentation,” 2000.

[4] ESA Earth Observation Portal, “Asteria (arcsecond space telescope enabling research in astrophysics).” `https://directory.eoportal.org/web/eoportal/satellite-missions/a/asteria`.
Accessed: 09-02-2020.

[5] L. N. Morcillo *et al*, “Use of hardware-on-the-loop to test missions for a low cost mini-launcher,” 2011.

[6] Alén Space, “A Basic Guide to Nanosatellites.” `https://alen.space/basic-guide-nanosatellites/`.
Accessed: 13-02-2020.

[7] R. Nugent *et al*, “CubeSat: The Pico-Satellite Standard for Research and Education,” 2008.

[8] R. P.Welle, “The CubeSat Paradigm: An Evolutionary Approach to Satellite Design,” 2016.

[9] CDS, “CubeSat Design Specification Rev.13,” 2016.

[10] T. Villela *et al*, “Towards the Thousandth CubeSat: A Statistical Overview,” 2019.

[11] M. T. Tikhonravov, “The creation of the first artificial Earth satellite: some historical details. ,” 1994.

[12] A. Poghosyan *et al*, “CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions,” 2016.

[13] NASA, “State of the Art of Small Spacecraft Technology.” `https://www.nasa.gov/smallsat-institute/sst-soa/guidance-navigation-and-control`.
Accessed: 10-02-2020.

[14] NANOSTAR, “NANOSTAR Project Methodology.” `https://nanostar-project.gitlab.io/main/contents.html`.
Accessed: 15-02-2020.

[15] J. Wertz, *Space Mission Analsysis and Design*. 1991.

[16] ESA Earth Observation Portal, “DeOrbitSail (DOS) Nanosatellite Mission.” https://directory.eoportal.org/web/eoportal/satellite-missions/d/deorbitsail.
Accessed: 12-02-2020.

[17] I. S. O. (TEC-QI), “ESA Requirements on EOL De-orbit,” 2015.

[18] K. Lemmer, “Propulsion for CubeSats,” 2017.

[19] A. Bost, “Materials for Small-Scale Space Propulsion Systems,” 2017.

[20] ESA, “ESTRACK ground stations.” https://www.esa.int/Enabling_Support/Operations/Estrack/Estrack_ground_stations.
Accessed: 05-05-2020.

[21] J. Bouwmeester, “Lecture Notes - Spacecraft Technology ,” 2016.

[22] D. C. *et al*, “Thermal Experiments for Validation of 3-AMADEUS Cubesat,” 2017.

[23] C. Kurtuluş *et al*, “ĐTÜ- pSAT I: Istanbul Technical University StudentPico-Satellite Program,” 2007.

[24] ESA Earth Observation Portal, “Itupsat-1 (istanbul technical university picosatellite-1).” https://directory.eoportal.org/web/eoportal/satellite-missions/i/itupsat-1.
Accessed: 11-02-2020.

[25] A. Klesh *et al*, “MarCO: CubeSats to Mars in 2016,” 2014.

[26] ESA Earth Observation Portal, “Marco (mars cube one).” https://directory.eoportal.org/web/eoportal/satellite-missions/content/-/article/marco.
Accessed: 09-02-2020.

[27] JPL, “Iris V2.1 CubeSat Deep Space Transponder Brochure,” 2016.

[28] J. Wertz, *Spacecraft Attitude Determination and Control.* 1978.

[29] F. L. Markley, J.L. Crassidis, *Fundamentals of Spacecraft Attitude Determination and Control.* 2014.

[30] X. Xia *et al*, “NanoSats/CubeSats ADCS Survey,” 2017.

[31] NewSpace Systems, “Sun Sensor Datasheet.” https://www.newspacesystems.com/wp-content/uploads/2020/04/NewSpace-Sun-Sensor_7b.pdf.
Accessed: 14-02-2020.

[32] A. P. E. A. v. T.-J. Chin, S. Bagchi, “Star tracking using an event camera,” in *CVPR Workshop on Event-based Vision and Smart Cameras*, 2019.

[33] S. Zafar *et al*, “Earth horizon sensor for attitude determination of LEO satellites,” 2019.

[34] ESA Artes, “Coarse InfraRed Earth Sensor IRES-C.” `https://artes.esa.int/projects/coarse-infrared-earth-sensor-ires-c`.
Accessed: 13-02-2020.

[35] K. Mizokami, “The Air Force May Ditch GPS for Earth’s Magnetic Field.” `popularmechanics.com/military/research/a33512412/air-force-magnetic-field-gps/`.
Accessed: 13-02-2020.

[36] F. Stray, “Attitude Control of a Nano Satellite,” 2010.

[37] G. Lavezzi *et al*”, “Attitude Control Strategies for an Imaging CubeSat,” 2019.

[38] S. Ulrich, *Aerospace Stream Selection.* 2017.

[39] Nano Avionics, “CubeSat Magnetorquer SatBus MTQ.” `https://nanoavionics.com/cubesat-components/cubesat-magnetorquer-satbus-mtq/`.
Accessed: 13-02-2020.

[40] J. Gießelmann, “Development of an Active Magnetic Attitude Determination and Control System for Picosatelliteson highly inclined circular Low Earth Orbits,” 2006.

[41] J. Guo *et al*, “Where is the limit? The analysis of CubeSat ADCS performance,” 2016.

[42] E. E. O. Portal, “OPTOS (Optical Nanosatellite.” `https://directory.eoportal.org/web/eoportal/satellite-missions/o/optos`.
Accessed: 12-02-2020.

[43] ESA Earth Observation Portal, “TEMPEST-D (Temporal Experiment for Storms and Tropical Systems Technology - Demonstration).” `https://directory.eoportal.org/web/eoportal/satellite-missions/t/tempest-d`.
Accessed: 12-02-2020.

[44] Blue Canyon Technologies, “Our Components.” `https://bluecanyontech.com/components`.
Accessed: 13-02-2020.

[45] EnduroSat, “1U SOLAR PANEL X/Y.” `https://www.endurosat.com/cubesat-store/all-cubesat-modules/1u-solar-panel-x-y/?v=35357b9c8fe4`.
Accessed: 13-10-2020.

[46] N. Prasad, “An overview of on-board computer (OBC) systems available on the global space marketplace.” `https://blog.satsearch.co/2020-03-11-overview-of-on-board-computers-available-on-the-global-space-marketplace`.
Accessed: 19-02-2020.

[47] J. Rajewski, *Learning FPGAs*. 2017.

[48] Charles H. Roth, Jr., L.K. John, *Digital Systems Design Usign VHDL*. 2007.

[49] A. Moore, *FPGA for Dummies*. 2017.

[50] F. Piltan *et al*, "Design FPGA-Based CL-Minimum Control Unit," 2016.

[51] OurPCB, "FPGA Vs Microcontroller-Which Is Better For Your Needs." `https://www.ourpcb.com/fpga-vs-microcontroller.html`.
Accessed: 01-03-2020.

[52] IMT, "IMT Cubesat On-Board Computer." `http://www.imtsrl.it/obc-cubesat.html`.
Accessed: 08-06-2020.

[53] IMT, "ISIS On Board Computer." `https://www.isispace.nl/product/on-board-computer/`.
Accessed: 05-03-2020.

[54] ISIS, "ISIS On Board Computer." `https://www.isispace.nl/product/on-board-computer/`.
Accessed: 12-02-2020.

[55] S. Habinc, "Suitability of Reprogrammable FPGAs in Space Applications," 2002.

[56] A. S. Dawood *et al*, "On-board Satellite Image Compression Using Reconfigurable FPGAs," 2002.

[57] G. Grillmayer *et al*, "FPGA Based Attitude Control System Architecture for Increased Perfomance," 2008.

[58] M. Yasir *et al*, "Development of a Safe Mode Attitude Control for a FPGA based Micro Satellite," 2008.

[59] ESA Earth Observation Portal, "Flying Laptop." `https://directory.eoportal.org/web/eoportal/satellite-missions/f/flying-laptop#:~:text=Flying%20Laptop%20is%20is%20the,OSIRIS%20terminals%20(Figure%2019)`.
Accessed: 16-02-2020.

[60] J. Soh *et al*, "A FPGA-based Approach to Attitude Determination for Nanosatellites," 2011.

[61] H. D. Curtis, *Orbital Mechanics for Engineering Students*. 2004.

[62] C.-. Foundation, "Conic Sections." `https://www.ck12.org/book/ck-12-algebra-ii-with-trigonometry-concepts/section/10.0/`.
Accessed: 24-06-2020.

[63] V. A. Chobotov, *Orbital Mechanics* . 1991.

[64] NASA, "Basics of Space Flight Section." https://solarsystem.nasa.gov/basics/chapter5-1/.
Accessed: 23-07-2020.

[65] E. G. S. Centre, "Orbital and Technical Parameters." https://www.gsc-europa.eu/system-service-status/orbital-and-technical-parameters.
Accessed: 13-10-2020.

[66] ESA, "ESA - Types of orbits." https://www.esa.int/Enabling_Support/Space_Transportation/Types_of_orbits.
Accessed: 23-07-2020.

[67] R. J. Boain, *A-B-Cs of Sun-Synchronous Orbit Mission Design* . 2004.

[68] J. T. Gravdahl *et al*, "Three Axis Attitude Determination and Control System for a Pico-satellite : Design and Implementation," 2003.

[69] K. G. Ganesh, "Controls Algorithm for a Satellite Using Earth's Magnetic Field: Orbit Maneuvers Attitude Positioning," 2005.

[70] M. J. Baker, "Euclidean Space: Maths - Rotation Matrices." https://www.euclideanspace.com/maths/algebra/matrix/orthogonal/rotation/index.htm.
Accessed: 15-05-2020.

[71] J. Diebel, *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*. 2006.

[72] E. W. Weinstein, "Euler Angles." https://mathworld.wolfram.com/EulerAngles.html.
Accessed: 15-07-2020.

[73] D. Henderson, "Euler Angles, Quaternions and Transformation Matrices," 1977.

[74] A. Alaimo *et al*, "Comparison Between Euler and Quaternion Parametrization in UAV Dynamics," 2013.

[75] H. Tanous, "Interactive and connected rehabilitation systems for e-health," 2018.

[76] G. Avanzini, *Spacecraft Attitude Dynamics and Control*. 2008.

[77] M. Sidi, *Spacecraft dynamics and control: a practical engineering approach*. 1997.

[78] A. Kurdila *et al*, *Dynamics and Control of Robotic Systems*. 2019.

[79] A. Aharoni, *Introduction to the theory of ferromagnetism*. 2000.

[80] P. Alken, “International Geomagnetic Reference Field.” https://www.ngdc.noaa.gov/IAGA/vmod/igrf.html.
Accessed: 01-04-2020.

[81] M. Ovchinnikov *et al*, “Geomagnetic field models for satellite angular motion studies,” 2018.

[82] Milos D. Ercegovac, Tomás Lang, Jaime H. Moreno, *Introduction to Digital Systems*. 1998.

[83] W. Kahan, “IEEE Standard 754 for Binary Floating-Point Arithmetic,” 1997.

[84] C. lnacio, “The DSP Decision: Fixed vs Float,” 1996.

[85] D. Silage, *Signal Processing Algorithms into Fixed Point FPGA Hardware*. 2017.

[86] A. Gilli, *Binary Arithmetic and Boolean Algebra*. 1965.

[87] S. Arar, “Multiplication Examples Using the Fixed-Point Representation.” https://www.allaboutcircuits.com/technical-articles/multiplication-examples-using-the-fixed-point-representation/.
Accessed: 02-07-2020.

[88] J. Steinmeyer, “Communication Protocols: Notes and/or Reference, MIT,” 2017.

[89] Jonathan Valvano, Ramesh Yerraballi, *Embedded Systems - Shape The World*. 2019.

[90] D. Inc., “Communications - SPI Serial Protocols.” https://reference.digilentinc.com/learn/courses/unit-4-lab4c/start.
Accessed: 13-10-2020.

[91] Y. S. Ong *et al*, “Plastic Optical Fibre Sensor System Design Using the Field Programmable Gate Array,” 2018.

[92] P. Kurup, Taher Abbasi, *FPGA Synthesis*. 1997.

[93] B. C. Readler, *Vhdl by Example: A Concise Introduction for Fpga Design*. 2014.

[94] S. Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis*. 1996.

[95] Xilinx, “Spartan-3A FPGA Family - Data Sheet.” https://www.xilinx.com/products/silicon-devices/fpga/spartan-3.html.
Accessed: 24-06-2020.

[96] J. R. French, Michael D. Griffin, *Space Vehicle Design*. 1991.

[97] S. A. Rawashdeh, “Passive Attitude Stabilization for Small Satellites,” 2010.

[98] J. M. Mbuthia *et al*, “1KUNS-PF: 1st Kenyan University NanoSatellite-Precursor Flight,” 2016.

[99] J. Kiesbye, “Hardware-in-the-Loop Verification of the Distributed, Magnetorquer-Based Attitude Determination Control System of the CubeSat MOVE-II,” 2017.

[100] FreeScale Xtrinsic, “Xtrinsic MAG3110 Three-Axis Digital Magnetometer: Data Sheet.” https://www.nxp.com/docs/en/data-sheet/MAG3110.pdf. Accessed: 13-10-2020.

[101] Xilinx, “Spartan-3E FPGA Starter Kit Board User Guide.” https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf. Accessed: 24-06-2020.

[102] F.-W. Wellmer, *Statistical Evaluations in Exploration for Mineral Deposits*. 1998.

Appendix A

ModelSim Simulation

This appendix presents the simulation performed on the proposed FPGA design, using ModelSim software.

A.1 3-AMADEUS FPGA Design Simulation

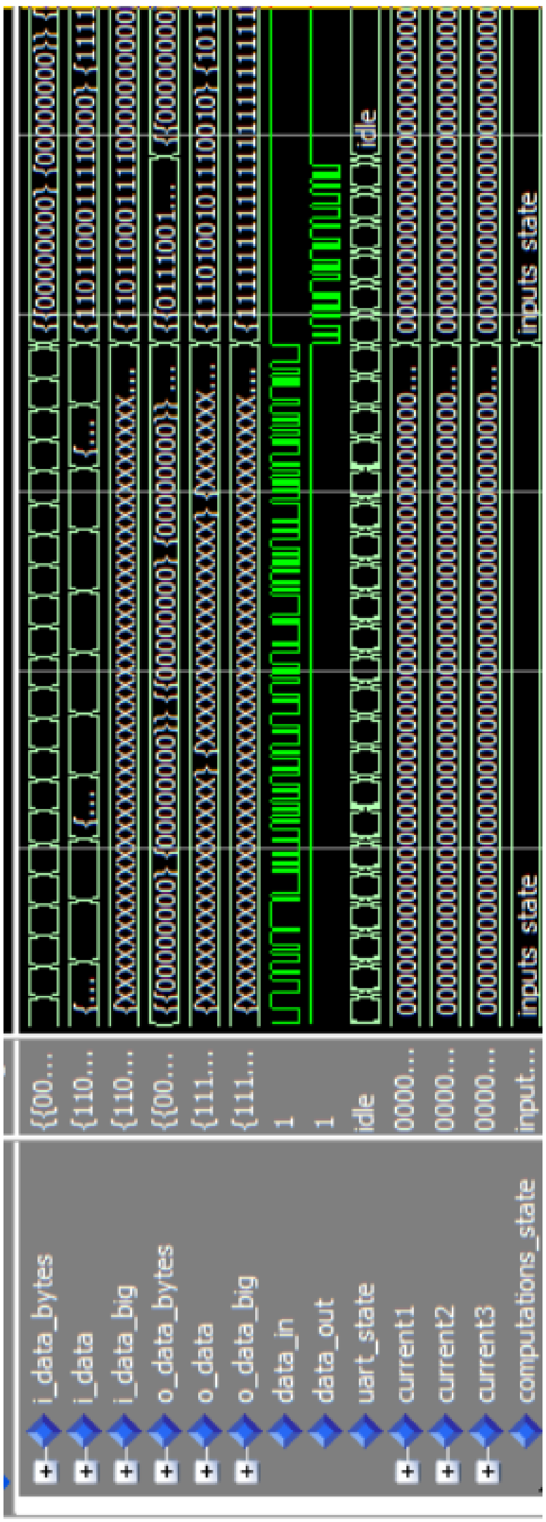
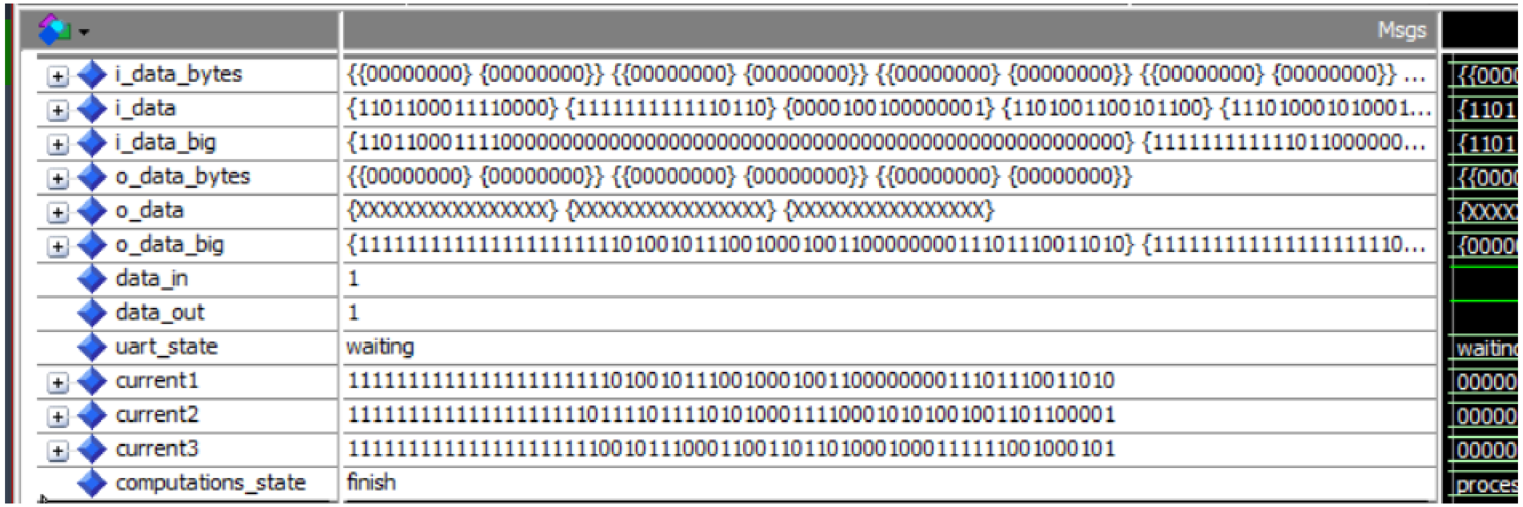


Figure A.1: ModelSim simulation of the proposed FPGA design: Data flow



```
                      Msgs
+ i_data_bytes        {{00000000} {00000000}} {{00000000} {00000000}} {{00000000} {00000000}} {{00000000} {00000000}} ...    {{0000
+ i_data              {1101100011110000} {1111111111110110} {0000100100000001} {1101001100101100} {111010001010001...      {1101
+ i_data_big          {11011000111100000000000000000000000000000000000000000000000000000} {11111111111101100000000...       {1101
+ o_data_bytes        {{00000000} {00000000}} {{00000000} {00000000}} {{00000000} {00000000}}                               {{000
+ o_data              {XXXXXXXXXXXXXXXX} {XXXXXXXXXXXXXXXX} {XXXXXXXXXXXXXXXX}                                              {XXXX
+ o_data_big          {11111111111111111111111010010111001000100110000000011101110011010} {11111111111111111111110...       {0000
  data_in             1
  data_out            1
  uart_state          waiting                                                                                               waiting
+ current1            11111111111111111111111010010111001000100110000000011101110011010                                    00000
+ current2            1111111111111111111111011110111101010001111000101010010011011000001                                  00000
+ current3            11111111111111111111111001011100011001101101000100011111001000101                                    00000
  computations_state  finish                                                                                                proces
```

Figure A.2: ModelSim simulation of the proposed FPGA design: Current values