POLITECNICO DI TORINO

I Facoltà di Ingegneria

Corso di Laurea in Ingegneria Aerospaziale

Tesi di Laurea Specialistica

Development of the Attitude Determination and Control System for a nanosatellite

Relatori:
Ing. Sabrina Corpino
Ing. Fabrizio Stesina

Candidato:
Raffaele Mozzillo

Luglio 2012

Ai miei genitori e a zio Salvatore

Earth is the cradle of humanity but
one cannot live in the cradle forever.

From a letter, 1911
Konstantin Tsiolkovsky

Sommario

Il lavoro presentato in questa tesi è relativo al progetto e allo sviluppo di un sistema attivo per la determinazione e il controllo dell'assetto di un nanosatellite (in particolare della categoria Cubesat) ed ai test effettuati sui diversi componenti per la futura integrazione a bordo del satellite *3STAR* sviluppato da un team di studenti del Politecnico di Torino.

Questo lavoro è una parte del progetto del CubeSat *3STAR* coordinato dal gruppo ASSET del Dipartimento di Ingegneria Meccanica e Aerospaziale su iniziativa dell'Educational Office dell'ESA all'ESTEC. L'obiettivo è la realizzazione di una missione spaziale completa, con grandissima importanza riservata al valore formativo dell'intero progetto: la missione consiste nella realizzazione di un satellite della categoria Cubesat che sarà parte di una costellazione di satelliti in orbita bassa terrestre; esso dovrà essere compatibile con la rete GENSO e dovrà avere a bordo un payload dedicato alla missione HUMSAT che prevede l'utilizzo dei satelliti per migliorare le capacità di comunicazione nei Paesi in via di sviluppo e in quelle zone colpite da calamità naturali o da eventi catastrofici che si trovino a dover gestire delle emergenze contingenti. Inoltre essi saranno utilizzati per monitorare sensori installati in posti remoti della Terra e per questo poco accessibili. A bordo è previsto anche un altro payload, P-GRESSION, sviluppato dal Dipartimento di Elettronica sempre del Politecnico di Torino.

Prima di analizzare nello specifico i punti trattati nella tesi, è importante sottolineare alcuni aspetti del programma, anche non prettamente ingegneristici ma al tempo stesso fondamentali:

- il progetto permette agli studenti di lavorare su qualcosa di pratico, dando così la possibilità di testare in prima persona le metodologie di lavoro utilizzate nelle aziende per i grandi satelliti, in quanto *3STAR*, pur essendo un progetto a basso costo, prevede fasi del tutto simili a quelle relative ai grandi progetti;

- il progetto da la possibilità agli studenti di lavorare in team, quindi di sperimentare per la prima volta tutti gli aspetti del lavoro di gruppo, quali per esempio la coordinazione dei diversi elementi del gruppo, l'aggiornamento a cadenza fissa sullo status della propria parte del progetto, la necessità del rispetto di

scadenze in quanto, essendo tutte le varie parti strettamente interconnesse, l'avanzamento deve essere cooordinato;

- grazie alla partecipazione a diverse conferenze e workshop si entra in contatto con altri team studenteschi che stanno lavorando su analoghi progetti per altre università sia europee che mondiali, quindi questo permette di instaurare nuovi rapporti e di entrare in contatto con altre "scuole di pensiero".

Per quanto riguarda il lavoro discusso in questa tesi, in primis sono analizzati lo standard CubeSat e la missione *3STAR* (con uno sguardo a quello che è il background sviluppato al Politecnico di Torino), in modo tale da poter portare a termine la prima parte fondamentale del progetto, cioè la definizione dei requisiti e dei vincoli per la missione sulla base degli obiettivi ed analizzando in dettaglio i documenti forniti.

Nel capitolo successivo sono introdotti tutti gli strumenti matematici utilizzati, cioè i sistemi di riferimento, le tipologie di rappresentazione dell'assetto, le orbite e il modo con cui determinarle, il campo magnetico terrestre ed infine un rapido sguardo alle diverse metodologie di controllo utilizzabili (tra le quali PD, PID e LQR).

Segue la definizione dei requisiti del sottosistema ADCS ed i risultati dell'analisi funzionale condotta su esso, che grazie alla simulazione su MATLAB®-Simulink®del sistema quasi ideale, permette di determinare quale tra le varie configurazioni ipotizzate, sulla base di opportuni parametri (peso, affidabilità, dimensioni, richiesta di potenza elettrica, costo, accuratezza e tempo di stabilizzazione) deve essere adottata per *3STAR* .

Nel capitolo successivo, il modello matematico sviluppato in precedenza viene reso molto più complesso con l'aggiunta dei vari disturbi simulati prima come rumori bianchi e poi con modelli numerici sviluppati appositamente per IMU e magnetometro. Inoltre sono aggiunti il filtro di Kalman ed una diversa tipologia di controllo, LQR, per arrivare così alla simulazione definitiva utilizzando il modello completo. I risultati mostrano una buona risposta del sistema, sia per quanto riguarda l'assetto raggiunto che per il consumo di potenza elettrica.

Segue una fase di test durante la quale sono analizzati in laboratorio, realizzando opportuni banchi prova (alcuni dei quali realizzati ex novo), i comportamenti dei singoli componenti che saranno poi effettivamente utilizzati a bordo, per poi giungere alla realizzazione di un codice in C da implementare sulla CPU di bordo.

Infine si valuta la risposta dell'intero sistema con una simulazione Hardware In the Loop che permette di rappresentare l'ambiente spaziale a terra in modo tale che il sistema possa essere testato con gli input che avrebbe in orbita, fornendo così la possibilità di effettuare poi una ottimizzazione software ed hardware dell'intero sottosistema ADCS. I risultati ottenuti con questa tipologia di simulazione (preceduta da Software In the Loop) mostrano un buono comportamento, abbastanza simile a quello ottenuto con modello MATLAB®-Simulink®.

Abstract

The work presented in this thesis is related to the design and development of an active system for the attitude determination and control of a nanosatellite (in particular the category Cubesat) and to the tests performed on various components for future integration on the *3STAR* satellite developed by a team of students at Politecnico di Torino.

This work is part of the *3STAR* CubeSat project coordinated by ASSET Group of Department of Mechanical and Aerospace Engineering on the initiative of ESTEC ESA Educational Office. The objective is the realization of a complete space mission with a great importance given to the educational value of the entire project: the aim is to build a Cubesat satellite that will be part of a constellation of satellites in low Earth orbit, it has to be fully compatible with the GENSO network and it will have to carry a payload dedicated to the mission HUMSAT which provides the use of satellites to improve communication in the developing countries and in those areas affected by natural disasters or catastrophic events. In addition they will also be used for monitoring sensors installed in remote places of the Earth and therefore not easily accessible. On board there is also another payload, P-GRESSION, developed by the Department of Electronics of Politecnico di Torino.

Before discussing specifically the points addressed in the thesis, it is important to emphasize some aspects of the project, although not strictly engineering yet fundamental:

- the project allows students to work on something practical, giving the chance to see first hand the work methods used in companies for large satellites, because *3STAR* , despite being a low-cost project, provides all phases of similar to those for major projects;

- the project gives the opportunity for students to work in teams, so for the first time to experience all aspects of teamwork, such as the coordination of the various elements of the group, the update on fixed frequency about the status of their part of project, the need for compliance with deadlines because all the various parts are closely interrelated and so progress must be almost uniform;

- through participation in various conferences and workshops, there is the opportunity to get in touch with other student teams who are working on similar projects for other universities in both Europe and the world, so this allows the creation of new relationships and to get in touch with other "schools of thought".

Regarding the work discussed in this thesis, are primarily analyzed the CubeSat standard and the *3STAR* mission (with a look at what is the background developed at the Politecnico di Torino), so it is possible to complete the first important part of project, namely the definition of mission requirements and constraints based on the mission objectives and analyzing in detail documents provided.

In the next chapter are introduced all the mathematical tools used, ie the reference systems, the types of attitude representation, the orbits and the way we define them, the Earth's magnetic field and finally a quick look at different methods of control used (including PD, PID and LQR).

Follows the definition of the requirements of ADCS subsystem and the results of functional analysis conducted on it, that, thanks to the simulation on MATLAB®-Simulink®of almost ideal system, allows to determine which of the various configurations assumed, on the basis of appropriate parameters (weight, reliability, size, power consumption, cost, accuracy and stabilization time) must be adopted for *3STAR* .

In the next chapter, the mathematical model developed earlier is made more complex by adding various disturbances simulated as white noise first and then with numerical models developed specifically for IMU and magnetometer. Moreover are added Kalman filter and a different type of control, LQR, so to get to the final simulation using the complete model. The results show a good response of the system, both as regards the attitude achieved that for the consumption of electric power.

A test phase follows, during which are analyzed in the laboratory, producing appropriate test benches (some of which are made from scratch), the behaviors of the individual components that will be actually used on board, and then there is the creation of a code written in C to implement on the on-board CPU.

Finally, there is the evaluation of the response of the entire system with a Hardware In the Loop simulation which allows to represent the space environment on ground in such a way that the system can be tested with the inputs that it would have in orbit, in order to perform a software and hardware optimization of the entire ADCS subsystem. The results obtained with this type of simulation (preceded by Software In the Loop) show a good behavior, quite similar to that obtained with MATLAB®-Simulink®model.

Contents

List of Tables

List of Figures

List of Listings

Acronyms

ADC: Analog to Digital Converter

ADCS: Attitude Determination & Control System

ASSET: AeroSpace System Engineering Team

C&DH: Command & Data Handling

COMSYS: COMmunication SYStem

COTS: Commercial Off The Shelf

DAC: Digital to Analog Converter

DELEN: Dipartimento di ELEttroNica

DIMEAS: Dipartimento di Ingegneria MEccanica ed AeroSpaziale

ECEF: Earth-Centered Earth-Fixed

EPS: Electrical Power System

ESA: European Space Agency

E-ST@R: Educational - SaTellite @ politecnico di toRino

ESTEC: European Space research and TEchnology Centre

GCS: Ground Control Station

GEO: Geostationary Earth Orbit

GEOID: Genso Experimental Orbital Initial Demonstrator

GSE: Ground Support Equipment

HEO: High Earth Orbit

HUMSAT: HUManitarian SATellite

HIL: Hardware In the Loop

IARU: International Amateur Radio Union

IMU: Inertial Measurement Unit

LQE: Linear-Quadratic Estimator

LEO: Low Earth Orbit

LEOP: Launch & Early Orbit Phase

LQG: Linear-Quadratic-Gaussian control

LQR: Linear-Quadratic Regulator

MEO: Medium Earth Orbit

MGCS: Mobile Ground Control Station

MT: Magnetic Torquer(s)

NASA: National Aeronautics & Space Administration

NED: North East Down

NOAA: National Oceanic and Atmospheric Administration

NORAD: North American Aerospace Defense Command

OBC: On-Board Computer

P-GRESSION: Payload for Gnss REmote Sensing and SIgnal detectiON

PD: Proportional-Derivative control

PiCPoT: PIccolo Cubo del POlitecnico di Torino

PID: Proportional-Integral-Derivative control

PIL: Processor In the Loop

POLITO: POLItecnico di TOrino

PWM: Pulse-Width Modulation

P-POD: Poly-Picosatellite Orbital Deployer

QA: Quality Assurance

RAAN: Right Ascension of the Ascending node

RW: Reaction Wheel(s)

SD: Secure Digital

SI: Système International d'unitès

SIL: Software In the Loop

SP: Solar Panel(s)

TBC: To Be Confirmed

TBD: To Be Defined

VHF: Very High Frequency

WMM: World Magnetic Model

Chapter 1

Introduction

In the following chapters, the design and development of an Attitude Determination and Control System for nano satellite applications will be treated. First the CubeSat standard and the *3STAR* program are introduced, then the useful coordinate systems and reference model are analyzed. After the analysis of the orbit and the space environment, the mathematical model and the control laws are developed and tested through simulation with MATLAB®-Simulink®. After this, next phase about testing hardware is descripted.

1.1 The CubeSat project

A CubeSat is a type of miniaturized satellite for space research that usually has a volume of exactly one liter (10 cm cube), it weighs no more than one kilogram and typically it uses commercial off-the-shelf electronics components. Beginning of 1999, California Polytechnic State University (Cal Poly) and Stanford University developed CubeSat specifications to help universities around the world to perform space science and exploration.

The price tag, far lower than most satellite launches, has made CubeSat a viable option for schools and universities across the world. Because of this, a large number of universities and some companies and government organizations are developing CubeSats. Some applications of these satellites are:

- Earth remote sensing;
- materials degradation monitoring;
- electronics degradation check;
- biological studies.

The standard 10×10×10 cm basic CubeSat is often called "1U" CubeSat (meaning one unit) but CubeSats are scalable along one axis, so by 1U increments: CubeSats such as a "2U" CubeSat (20×10×10 cm) and a "3U" CubeSat (30×10×10 cm) have been both built and launched. Since CubeSats are all 10×10 cm (regardless of length) they can all be launched and deployed using a common deployment system, called Poly-PicoSatellite Orbital Deployer (P-POD), also developed and built by Cal Poly.

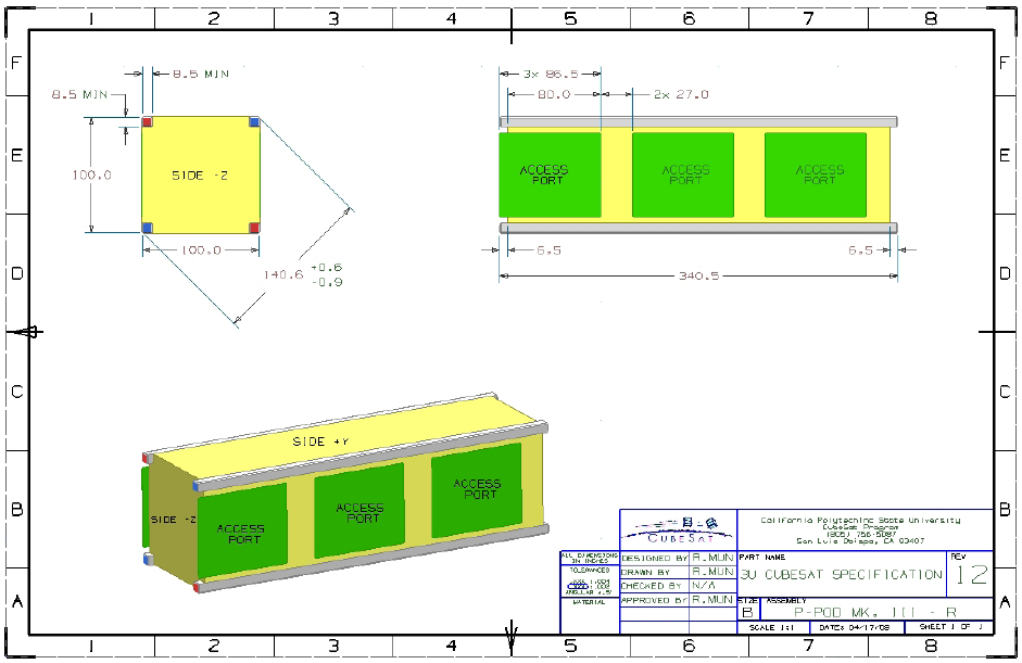


Figure 1.1 – 3U CubeSat specification drawing [1].

1.2 *3STAR* project

3STAR is the new cubesat educational project being carried out at Politecnico di Torino, and comes in response to the European Space Agency call for proposals for the GENSO Experimental Orbital Initial Demonstrator (GEOID) mission, held by its Education Office. GEOID is a mission that consists in launching and operating several satellites in order to test the Global Educational Network for Satellite Operations (GENSO) system and to serve as test-bed for the HUMSAT (HUManitarian SATellite) international satellites constellation, that will act as communication support for areas without infrastructures or for developing countries.

3STAR will be one of the first nine satellites of the GEOID constellation. From the technical point of view *3STAR* mission consists of a 3U cubesat orbiting the

Earth and acting as a data-relay platform and a space-based test bed for an Earth remote sensing experiment. The payloads of this satellite are two: HUMSAT payload, communication equipment (basically a UHF transceiver, an antenna and one data storage device) that meets HUMSAT requirements, nonetheless remaining extremely simple and reliable, and P-GRESSION (Payload for GNSS REmote Sensing and SIgnal detectiON) experiment, whose main goal is to achieve measurements by means of radio occultation techniques and scattering theory, using GNSS signals.

Figure 1.2 – *3STAR* logo.

1.2.1 Introduction and background

3STAR is an educational project which is developed by a multidisciplinary team of students from several engineering departments of Politecnico di Torino. In particular the project will be developed at the Department of Mechanical and Aerospace Engineering (DIMEAS) of Politecnico di Torino by the students of the AeroSpace Systems Engineering Team (ASSET), in collaboration with students from the Electronics Department (DELEN). The final goal is to test a network of ground stations, and to provide data relay communication services for areas with scarce infrastructures and/or affected by calamities.

At the Politecnico di Torino, several teams are involved in designing space missions and systems. Among these, the AeroSpace Systems Engineering Team (ASSET) of the DIMEAS, has been carrying out programs on small space platforms

for many years. In the last decade, the team has focused the attention on the development of small satellites for educational and research purposes. The first program was the PiCPoT nano-satellite, which has been completed in 2006. The PiCPoT satellite was developed and launched, therefore representing a good success for the developers' team. Unfortunately, it never reached its intended orbit due to a failure in the launch vehicle occurred a few seconds after liftoff. Despite the unsuccessful launch, the project represents an important stepping stone in terms of knowledge, experience and educational relevance.

The heritage of PiCPoT has been reaped by the e-st@r program, which has been successfully launched on February 13, 2012. The e-st@r program, mainly educational, was selected by the ESA Education Office as one of the nine university Cubesats on the Vega maiden flight.

1.2.2 Mission statement

The mission objectives for the 3STAR project have been derived by means of the typical system engineering process, which starts with the definition of the mission statement. The mission statement for the 3STAR project can be summarized as follows:

> "*The project aims at educating and inspiring space engineering students on complex systems development and operations, international cooperation and team work. The mission wants to contribute to the humanitarian exploitation of Space, by supporting communications capability in developing countries and/or allowing areas without infrastructure to access space-based services, and to enhance the knowledge on remote sensing applications for future small space missions.*"

1.2.3 Mission objectives

The following objectives can be derived from the mission statement:

- the program shall have educational relevance: hands-on practice education and training of students on a real spacecraft project;
- the mission shall carry one or more payload related to the peaceful and humanitarian exploitation of space;
- the mission shall demonstrate one or more remote sensing applications based on non-space qualified systems.

The *3STAR* program is a project developed at university level, so the main objectives are both the scientific and the educational relevance of the activity. The main constraint is represented by the limited available budget for the program development. Figure 1.3 illustrates the guidelines which are assumed as high level objectives and constraints for the program. Figure 1.4 shows the logical process implemented to

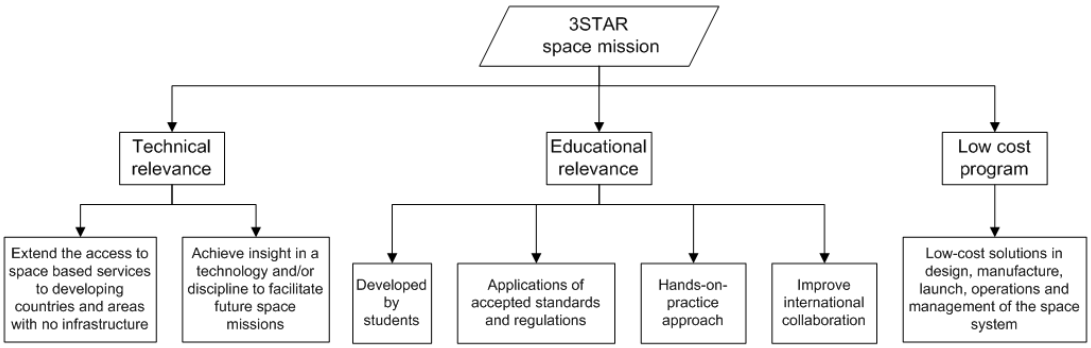


Figure 1.3 – *3STAR* project drivers.

obtain the scientific objectives of the mission. Taking into account these assumptions the mission and system requirements can be established, and the technical specifications can be derived for both the space and the ground segments. The primary

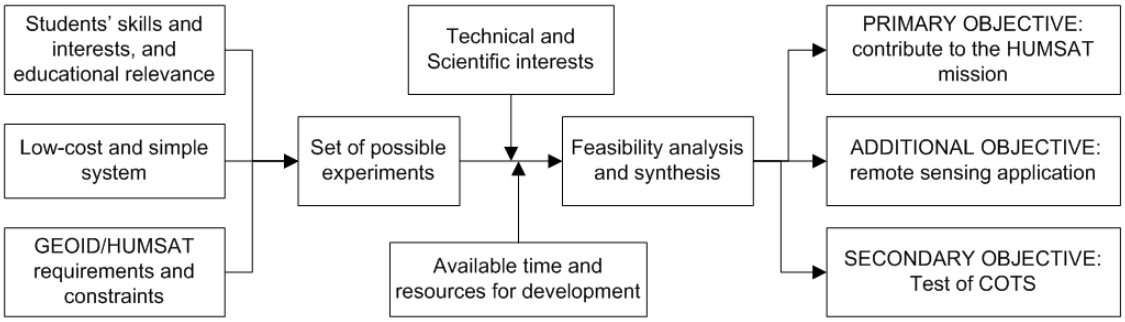


Figure 1.4 – Technical objectives definition.

objective for *3STAR* program is to support and contribute to the HUMSAT mission. In particular, several primary program sub-objectives can be defined:

- to provide telecommunications services in order to support to humanitarian and emergency applications;
- to monitor parameters related to climate change;
- to settle international collaboration among universities and research centres from all over the world.

Moreover, GEOID, strongly linked with HumSat program, adds the following objectives:

- to validate the GENSO network on a large-scale basis;
- to promote high-level education on space systems.

An additional objective is to perform on orbit remote sensing measurements, employing different remote sensing techniques for Earth observation, atmosphere profiling for climate studies and eventually warning services. Secondary objectives are the set up of permanent space education project based on small-missions development and the test of low cost technologies in orbit to facilitate future small space missions.

1.2.4 Mission scenario

A cubesat shall be inserted into a LEO by the beginning of 2013. The orbit is a SSO at an altitude in the range 500-750 km. The launcher is still undefined, but typical launch vehicles may be assumed for early design phases. Mission duration shall be longer than 12 months. The Cubesat shall be operated from ground in a simply and cheap way. The full compatibility with the GENSO network shall be demonstrated. High grade of operations autonomy is desirable. Students shall be designers, developers, manufacturers, operators and managers of the entire mission. The mission shall demonstrate some kind of non-space technology and try to space-qualify them by means of a relevant application. The primary payload shall be a HumSat compatible equipment while, as additional payload, is considerd a remote sensing experiment. The mission data shall be available to the Cubesat community, to the end users of the HumSat project and to radio-amateurs union. No commercial purposes shall be pursued. In Figure 1.5, it is shown the mission scenario involving satellites in the GEOID constellation (amongst which the *3STAR* cubesat) with some of the GENSO ground stations on the surface of the Earth. As can be observed there will be three orbital planes, with three satellites each, formed by three sun-synchronous orbits. This allows a good compromise between coverage and repeat-time.

1.2.5 Mission architecture

Figure 1.6 shows the *3STAR* mission architecture and its elements:

Space segment: it is composed by a 3U Cubesat encompassing 3-star bus and two payloads (see Figure 1.7);

Ground segment: it is composed by a main ground station, a mobile and transportable backup station and the GENSO stations network. Radio-amateurs can receive Cubesat signal but they can't command it;

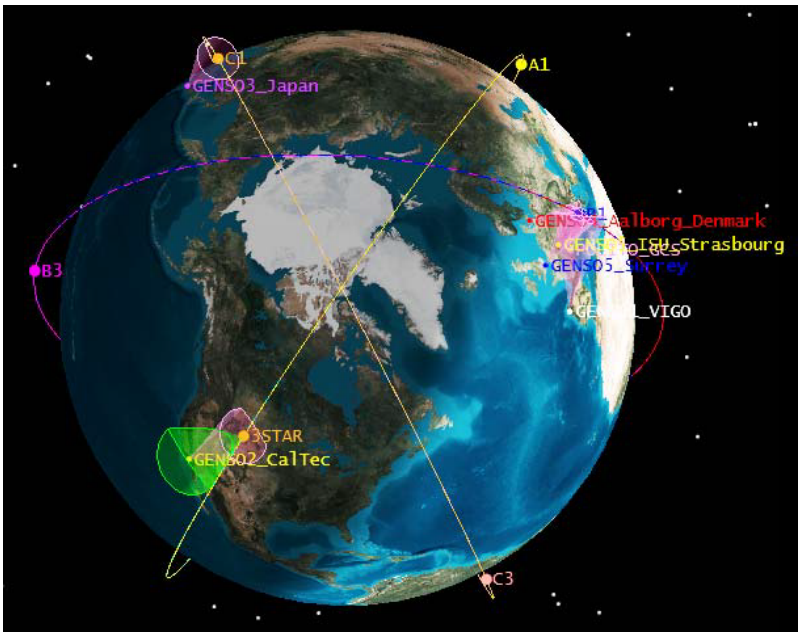


Figure 1.5 – STK simulation of *3STAR* in the GEOID constellation.

Subjects: they are the Earth surface and the Earth atmosphere, and sensor over the surface;

Launch vehicle and launch site: they are not now defined as also the parameters of LEO orbit;

Communications: they are maintained and managed according to the HUMSAT requirements and IARU regulations;

Operations: they will be managed by operators at GENSO stations and by the student at the main and backup stations.

1.2.6 Requirements and functional analysis

The mission requirements come from different sources, in particular from the HumSat Mission Requirements Document [2] while the system requirements come from the HumSat System Requirements Document [3]. To this set of requirements, other requirements and constraints have been settled by the team in order to address specific needs and interests. The system requirements and the low levels requirements are derived by means of typical system engineering tools, such as the functional analysis. It has been used to derive the lower level requirements for the system and to determine which subsystems are needed to carry out the mission. The second

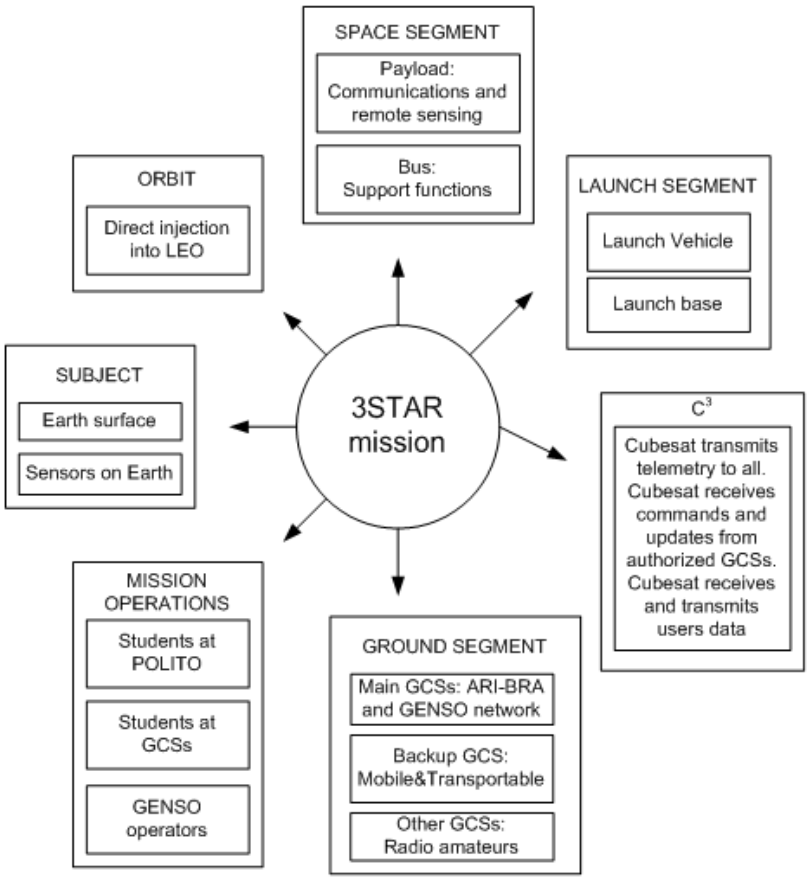


Figure 1.6 – *3STAR* mission architecture.

iteration of the functional analysis allows to derive next lower level requirements for equipment and components. Requirements for CubeSat (space segment) are divided into:

General requirements: they apply to the CubeSat as a whole. In case the CubeSat incorporates any deviations from these requirements, a DAR shall be submitted and the waiver process carried out;

Mechanical requirements: this class of requirements applies to the CubeSat's geometry (dimensions, interfaces), its mass and inertia properties (mass, moments of inertia, COM location), and to the materials used to manufacture the CubeSat;

Environmental requirements: this class of requirements derives from the environmental conditions at launch and during operations in orbit. It includes the

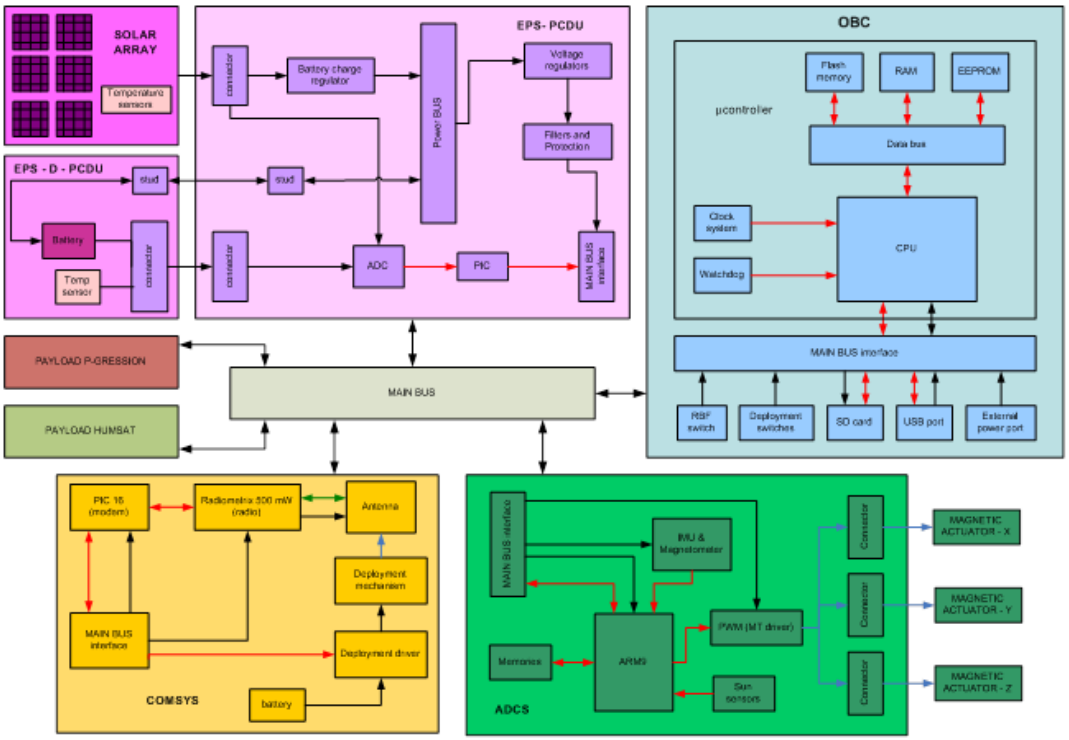


Figure 1.7 – *3STAR* space segment preliminary architecture.

vibration and acoustic environment encountered during the launch phase, and the thermal environment during lifetime in orbit;

Interface requirements: this class of requirements applies to the CubeSat's interfaces with the P-POD, to other spacecraft and to the launch vehicle. It includes physical, electrical, thermal and communications interfaces;

Functional requirements: they stem directly from the functional analysis. This class of requirements establishes which functions the CubeSat and its subsystems shall carry out and how they shall be performed to accomplish the intended mission. Performance requirements are included;

Operational requirements: this class of requirements applies to the operation phase and includes operative modes implementation from ground, on orbit operations in general, communication between the CubeSat and the GCS and related aspects (frequencies licences and coordination, etc), and debris issues;

Configuration requirements: this class of requirements applies to the configuration of the CubeSat. It includes requirements such as the components list and for the assembly/disassembly of the CubeSat;

Maintenance: integration and logistic requirements. This class of requirements applies to the operations to be accomplished in order to maintain the CubeSat in the specified flight-ready-condition during storage before launch.

1.2.7 Mission profile and preliminary operative modes

Preliminary mission simulations (Figure 1.8) have been performed with a reference orbit and a reference ground station (Table 1.1).

Torino ground station		
Latitude	[deg]	45°03'N
Longitude	[deg]	7°40'E
Altitude	[km]	0.3
Orbit parameters (they are indicative)		
Semi-major Axis	[km]	6978.137
Circular Altitude	[km]	600
Inclination	[deg]	97.40
Eccentricity	[-]	0

Table 1.1 – Ground station parameters and indicative orbit parameters.

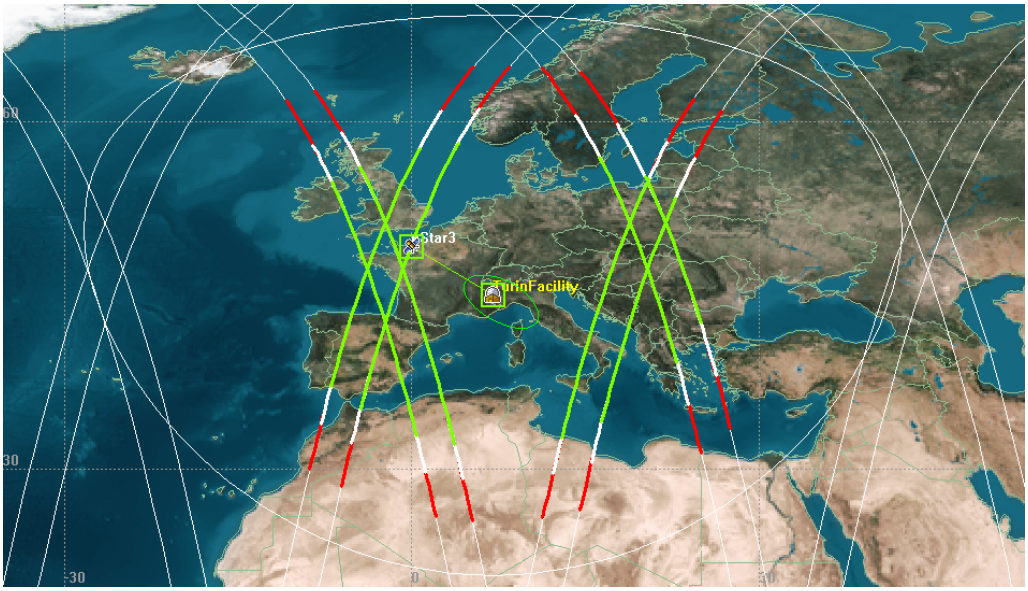


Figure 1.8 – *3STAR* access area over Torino at different elevation angles above the horizon.

The satellite may be operated in different ways, depending on which phase of its mission is exploited. *3STAR* is intended to be part of the HumSat mission, but it also has its own independent remote sensing payload. It is worth to be notice that the additional payload can also support the HumSat mission objectives to some extent. A set of operative modes may be defined and they are given in Table 1.2.

Table 1.2 – *3STAR* operative modes.

Operative mode	***Description***	***Payload***
LEOP	Cubesat right after launch and during commissioning is tested to prepare next mission phase	none
HUMSAT mission	Cubesat used as an element of the HumSat constellation	HUMSAT
P-GRESSION mission	Cubesat used as a remote sensing space platform	P-GRESSION
Basic mission	Cubesat used as space test bed for COTS equipment	3STAR
Safe mission	Off-normal mode, used in case the cubesat presents some failures and needs to be restored	none
Dormant	During launch the Cubesat systems are deactivated. The Cubesat may be turned on the dormant mode also upon request of international authorities or HumSat mission control board	none

Chapter 2

Definition and reference model

The mathematical model of the satellite and its environment can be developed in a number of different reference frames and methods to represent attitude. In this chapter a short introduction to methods and mathematical tools is given.

2.1 Reference frames

It is possible to identify different types of coordinate systems based on the number and type of the coordinates. The most common three-dimensional coordinate systems are:

Cartesian: also called rectangular, it is formed by three perpendicular lines whose intersection identify the origin. Each of the three lines, normally indicated as X, Y and Z, has an associated unit and direction. The generic coordinates of a point in the space are indicated with the letters x , y and z and the three coordinates are written with the symbol *(x,y,z)*.

Cylindrical: in this reference system the coordinates are ρ , ϕ and z. Considering a generic point P, and its projection Q on the X-Y plane, the coordinate z indicates the distance $\overline{PQ}$, ρ is the distance from the origin and the point Q, while ϕ is the angle between the vector $\vec{\rho}$ and the X axis.

Spherical: it is based on the coordinates ρ, θ and ϕ.Considering a generic point P, and its projection Q on the X-Y plane, the coordinate ρ indicates the distance of P from the origin, θ is the angle between the vector $\vec{\rho}$ and the Z axis and, calling $\vec{\rho'}$ the vector that connect the origin and the point Q, ϕ is the angle that this vector form with the X axis.

It is possible to describe the same point in more than on coordinate system, so a set of transformations exist in order to be able to change from a coordinate system to another. Now the different frames used in this report are defined.

2.1.1 Inertial Reference Frame

An Inertial reference frame is a coordinate system in which is verified Newton's first law: with an acceptable approximation the so called *fixed star reference frame* is considered inertial and it includes the Sun, the stars and every other body with a uniform rectilinear motion as regards to it (not accelerating or rotating).

2.1.1.1 Earth-Centered Inertial (ECI) frame

Considering the Earth as third body, the reference frame obtained can not be considered as a real Inertial reference frame because of Earth's revolution and rotation movements. In particular, the rotary motion submit the objects on the surface of the Earth far from the poles to a little centrifugal force. However this acceleration can be neglected in some cases and the Earth considered, with a good approximation, as an Inertial reference frame (called ECI).

The ECI frame is an inertial frame used for navigation. It is fixed in space and its origin is located at the center of the Earth with the Z-axis pointing towards the North Pole. The X-axis points towards vernal equinox, the point where the plane of the Earth's orbit around the Sun crosses the Equator going from south to north, and the Y-axis completes the right hand Cartesian coordinate system. All the different motions of the satellite could be presented in this frame, but only the velocity of the Orbit frame and the motion of the Sun is directly compared to this frame. The frame is denoted I.

The rotary motion, present in reality, brings objects far from the equator to the so called Coriolis force that causes a deviation of the motion of every objects towards right in the north hemisphere and towards left in the south hemisphere, as demonstrated by the well known Foucault pendulum. For every objects orbiting the Earth it is possible to define a Inertial reference frame based on the ECEF reference frame (2.1.2.1) with the following rotation matrix:

$$\boldsymbol{R}_I^E = \begin{bmatrix} \cos(\alpha) & \sin(\alpha)\cos(\beta) & \sin(\alpha)\sin(\beta) \\ -\sin(\alpha) & \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta) \\ 0 & -\sin(\beta) & \cos(\beta) \end{bmatrix} \tag{2.1}$$

where α is the angle given by $\omega_e t$, with ω_e the Earth rotational speed and t the time, and β is the angle given by the following equation:

$$\beta = \left(23.439281083 - \frac{46.815}{6300} Juliancenturies\right)\frac{\pi}{180} \tag{2.2}$$

in which *Juliancenturies* is:

$$Juliancenturies = \frac{day2000 - 2451545}{36525} \tag{2.3}$$

$$\begin{aligned} day2000 =& 2451543.5 + (year - 2000)365 + \\ &+ 1 + floor\Big(\frac{year - 2000}{4}\Big) + day \end{aligned} \tag{2.4}$$

2.1.2 Non-Inertial Reference Frame

A Non-Inertial reference frame is a coordinate system in which the description of the dynamic of objects does not verify the principle of inertia. It is a system in which an object subject to a resultant of forces equal to zero however has a non-uniform motion. All and only the reference frames that move of accelerated motion in reference to the fixed star reference frame have this property and can be defined as Non-Inertial.

2.1.2.1 Earth-Centered Earth-Fixed (ECEF) frame

The ECEF reference frame has its origin located in the center of the Earth but the X and Y axes rotate with the Earth relative to the ECI frame. This rotation is around the Z-axis, both of the ECI and the ECEF frame, and has a rate of

$$\omega_e \approx \frac{1 + 365.25 cycles}{(365.25)(24)h} \frac{2\pi rad/cycle}{3600 s/h} \approx 7.292115 \times 10^{-5} rad/s. \tag{2.5}$$

The Z-axis points towards the North Pole, X-axis points toward the intersection between the Greenwich meridian and the Equator, which is at 0° longitude and 0° latitude, and the Y-axis completes the right handed orthogonal system. The frame is denoted E.

2.1.2.2 North-East-Down (NED) frame

The North-East-Down reference frame is one of the Geodetic coordinate systems. It is a local reference frame and it depends on the position on the Earth. The X-Y plane coincides with the local horizon and it has unit vectors pointing the local North and East; the Z axis completes the right-handed triad pointing toward the center of the Earth (Down). Thanks to this property the NED reference frame is one of the most utilized for Earth surface studies. The frame is denoted N.

It is possible to define a rotation matrix in order to pass from the ECEF to the NED reference frame as:

$$\boldsymbol{R}_E^N = \begin{bmatrix} -\sin(lat)\cos(lon) & -\sin(lat)\sin(lon) & \cos(lat) \\ -\sin(lon) & \cos(lon) & 0 \\ -\cos(lat)\cos(lon) & -\cos(lat)\sin(lon) & -\sin(lat) \end{bmatrix} \tag{2.6}$$

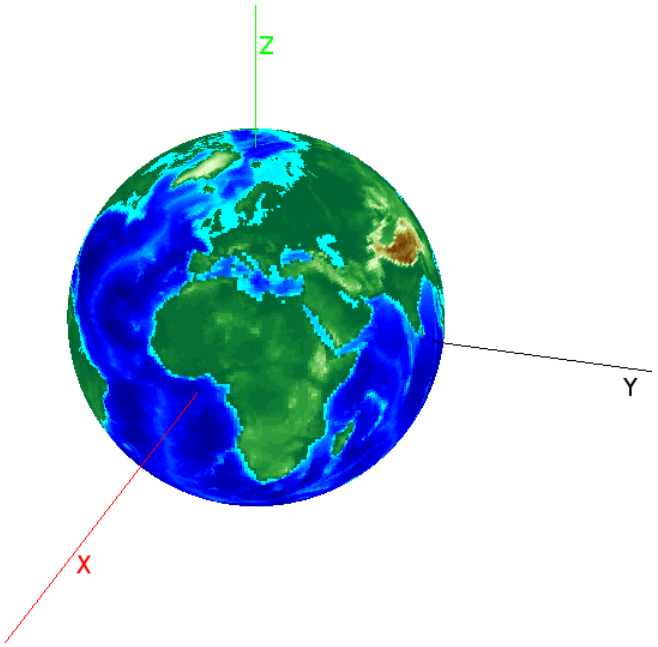


Figure 2.1 – ECEF reference frame.

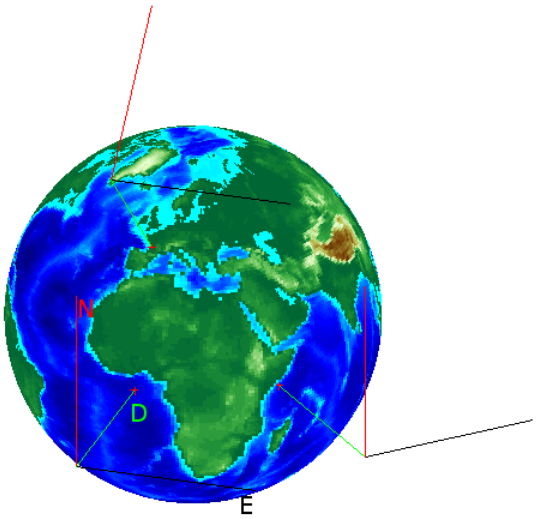


Figure 2.2 – NED reference frame.

2.1.2.3 Orbital frame

The origin of this frame coincides with the center of mass of the satellite. It rotates relative to the ECI frame, with a rate of ω_o depending on the altitude of the orbit. The X-Z plane is the orbital plane with unit vectors pointing one in the direction of the orbital velocity of the satellite and the other as the local vertical, while the Y-axis is orthogonal to this plane and complete the right-handed triad. The frame is denoted O.

It is possible to convert the coordinates of a point from the Orbital to the Inertial reference frame with the following rotation matrix:

$$\boldsymbol{R}_O^I = \begin{bmatrix} c(\Omega)c(\omega) - s(\Omega)s(\omega)c(i) & -c(\Omega)s(\omega) - s(\Omega)c(\omega)c(i) & s(\Omega)s(i) \\ s(\Omega)c(\omega) + c(\Omega)s(\omega)c(i) & -s(\Omega)s(\omega) + c(\Omega)c(\omega)c(i) & -c(\Omega)s(i) \\ s(\omega)s(i) & c(\omega)s(i) & c(i) \end{bmatrix} \tag{2.7}$$

with c and s compact notations for *cos* and *sin*. The other variables here used are defined in Section 2.3.1, in which is treated the description of the orbit.

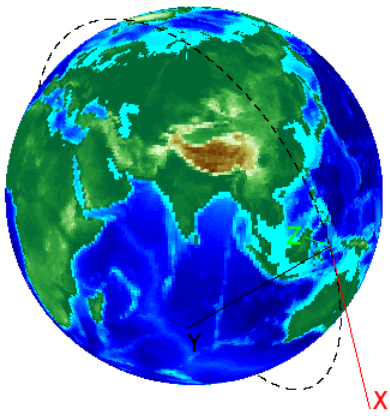


Figure 2.3 – Orbital reference frame.

2.1.2.4 Body frame

The body frame is fixed to the satellite and for practical reasons the origin is placed at the satellite's center of mass. The axes are locked in the satellite, X-axis is forward,

Z-axis is downwards and the Y-axis completes the right-hand orthogonal system. This frame is denoted B.

In order to transform a set of coordinates from the Body to the Orbital reference frame, or vice versa, the introduction of a rotation system is needed. The classical rotations used to describe these transformations are based on the use of the quaternions or the Euler Angles, both of them are described below.

2.2 Attitude representation

There are some different ways to represent the attitude of the satellite in a reference frame. These, along with tools to convert between the frames, are described here.

2.2.1 Euler Angles

The Euler Angles can be used to describe the angular position of a Body reference frame *XYZ*, with a set of rotations, relative to another reference frame *xyz* considered fixed. Here only rotations are considered, so the two reference frame are taken so that the origin is the same for both of them. If the *x*-*y* and *X*-*Y* planes do not coincide, a line of intersection will exist and it is called line of nodes (*N*). If they coincide then the line of nodes is taken coincident with the X axis. The Euler Angles are:

- α is the angle between the line of nodes and the x-axis, it is called *precession*;
- β is the angle between the z-axis and the Z-axis , it is called *nutation*;
- γ is the angle between the line of nodes and the X-axis, it is the intrinsic *rotation*.

The Euler Angles allow a representation of the rotation matrix in a easy form obtained with a multiplication of three rotation matrices. In other words the complete rotation described above can be done in three distinct passages:

- rotation around the z-axis of an angle α, to obtain the x-axis coinciding with the line of nodes N:

$$\boldsymbol{R}_\alpha = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

- rotation around the line of nodes N of an angle β:

$$\boldsymbol{R}_\beta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & \sin(\beta) \\ 0 & -\sin(\beta) & \cos(\beta) \end{bmatrix} \tag{2.9}$$

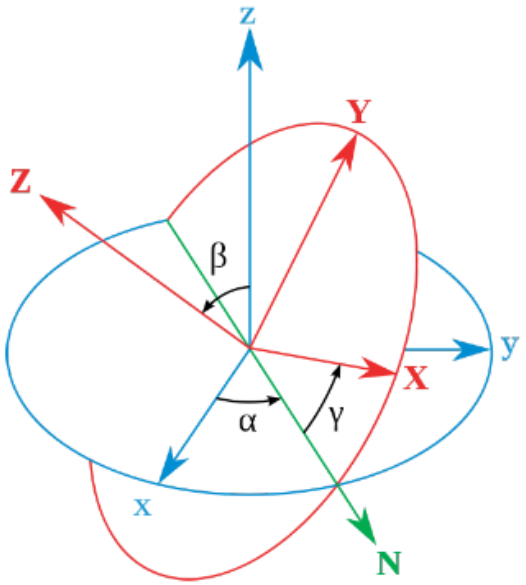


Figure 2.4 – Euler Angles: the fixed reference frame (xyz) is shown in blue, the rotated reference frame (XYZ) is shown in red and the line of nodes is shown in green.

- rotation around the Z-axis of an angle γ:

$$\boldsymbol{R}_\gamma = \begin{bmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

The rotation described above can be formalized as:

$$\begin{bmatrix} \hat{X} \\ \hat{Y} \\ \hat{Z} \end{bmatrix} = \boldsymbol{R}_\gamma \boldsymbol{R}_\beta \boldsymbol{R}_\alpha \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} \tag{2.11}$$

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} = \boldsymbol{R}_\alpha^T \boldsymbol{R}_\beta^T \boldsymbol{R}_\gamma^T \begin{bmatrix} \hat{X} \\ \hat{Y} \\ \hat{Z} \end{bmatrix} \tag{2.12}$$

The sequence described above is only one of the twelve possible sequences describing same rotation. It is called *ZXZ* from the axes around which the rotations take place. The other possibilities are *XZX*, *XYX*, *YXY*, *YZY*, *ZYZ*, *XZY*, *XYZ*, *YXZ*, *YZX*, *ZYX* and *ZXY*. These sequences are obtained from all the possible permutations of not consecutive equal axes.

A particular variant of the Euler Angles, used in aeronautic and robotic, is the Tait-Bryan Angles (Figure 2.5). In this case the angles ψ, ϕ and θ are named *yaw*, *roll* and *pitch* respectively.

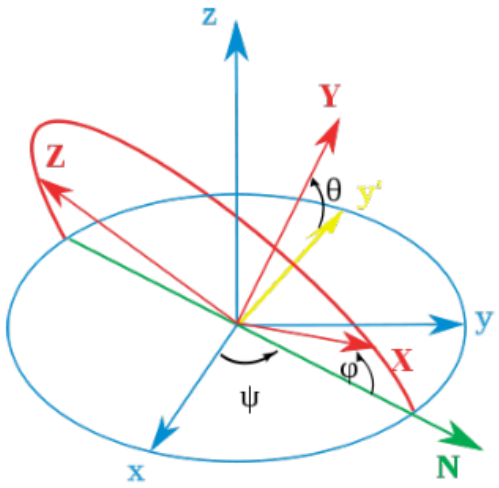


Figure 2.5 – Tait-Bryan Angles.

2.2.2 Quaternions

The main reason for using unit quaternions instead of Euler parameters is to avoid singularity which can occur when using Euler angles. In math quaternions are a numerical system used in order to extend the complex numbers. They were introduced for the first time by Sir William Rowan Hamilton in 1843 and successively applied to mechanics in three-dimensional spaces. One of the principal properties of quaternions is the fact that the product of two quaternions is not cumulative, which means that the product depends on the order of its terms. Hamilton defined a quaternion as the quotient of two vectors, but they can be also represented as sum of a scalar number and a vector. Quaternions are used vastly by theoretical and applied math, specially for rotations in the three-dimensional space. A generic quaternion can be written as:

$$\bar{q} = q_0 + q_1\hat{i} + q_2\hat{j} + q_3\hat{k} = a + b\hat{i} + c\hat{j} + d\hat{k} \tag{2.13}$$

with a, b, c and d real numbers. Quaternions contain naturally real numbers if $b = c = d = 0$ $(q = a)$ and complex numbers if $c = d = 0$ $(q = a + b\hat{i})$.

From the Euler theorem, that guarantees the possibility to rotate a fixed reference frame on another arbitrary reference frame with a simple rotation around an axis $\bar{a}$

(also called Euler rotation axis) fixed in both reference frames, it is possible to adopt quaternions to define any change of reference system in the three-dimensional space. Thanks to their properties it is possible to represent uniquely any rotation without having degenerating points, that are points in which at least a parameter lose its meaning.

The rotation of an angle α around a generic axis $\bar{u}$ can be described introducing the Euler parameters and obtaining the quaternion:

$$\bar{q} = \cos(\frac{\alpha}{2}) + \bar{u}\sin(\frac{\alpha}{2}) \tag{2.14}$$

or in matrix form:

$$\bar{q} = \begin{bmatrix} q_0 \\ \vec{q} \end{bmatrix} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(\frac{\alpha}{2}) \\ \hat{u_1}\sin(\frac{\alpha}{2}) \\ \hat{u_2}\sin(\frac{\alpha}{2}) \\ \hat{u_3}\sin(\frac{\alpha}{2}) \end{bmatrix} \tag{2.15}$$

Using quaternion above introduced, it is possible to define rotation matrix as:

$$\boldsymbol{R}_{q_0,\vec{q}} = \boldsymbol{I}_{3x3} + 2q_0\boldsymbol{S}(\vec{q}) + 2\boldsymbol{S}^2(\vec{q}) \tag{2.16}$$

where $\mathbf{S}(\vec{q})$ is the skew-symmetric matrix, that is:

$$\boldsymbol{S}(\vec{q}) = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \tag{2.17}$$

The Equation 2.16 allows to write rotation matrix from body to orbital frame as:

$$\boldsymbol{R}_B^O = \mathbf{1}_{3x3} + 2q_0\boldsymbol{S}(\vec{q}) + 2\boldsymbol{S}^2(\vec{q}) \tag{2.18}$$

so a representation of the rotation matrix from Orbital to Body frame can be calculated as follow:

$$\boldsymbol{R}_O^B = (\boldsymbol{R}_B^O)^T = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{2.19}$$

This rotation matrix can also be written as:

$$\boldsymbol{R}_O^B = \begin{bmatrix} c_1^B & c_2^B & c_3^B \end{bmatrix} \tag{2.20}$$

where $c_i^B = \begin{bmatrix} c_{ix}^B & c_{iy}^B & c_{iz}^B \end{bmatrix}^T$ are column vectors containing directional cosines.

2.3 Orbit

Based on the energy of the orbiting body, orbits can be closed and periodic or open and non-periodic [4]. It is possible to define different orbits:

Elliptic: the orbit is closed and it is an ellipse if the kinetic energy is less than the potential energy of the body. The orbits of the planets of the Solar System and of their satellites are elliptical. The circular orbit is a particular case of elliptic orbit;

Hyperbolic: the orbit is open and it is an hyperbole if the kinetic energy is more than the potential energy of the body. Orbits of the space probes sent out of the Solar System and some of probes sent towards external planets are hyperbolic;

Parabolic: if the kinetic energy is the same as the potential energy then the trajectory of the body is a parabola and this particular orbit is the type between the two families of closed and open orbits.

Spacecraft can be put into a number of different closed orbits around a planet. These are defined by a number of orbital characteristics, such as the height above the planet's surface, the inclination (a Keplerian elements descripted in 2.3.1) to the planet's equator and the direction in which the spacecraft orbits the planet [5]:

- Inclination

  equatorial: an orbit whose inclination is equal to zero with respect to equator of planet around which the satellite orbits;

  near equatorial: an orbit whose inclination with respect to the equatorial plane is nearly zero;

  polar: an orbit that passes above or nearly above both poles of the planet on each revolution, so it has an inclination of (or very close to) 90 degrees;

  polar Sun-synchronous: a nearly polar orbit that passes the equator at the same local solar time on every pass (SSO);

- Altitude (see Figure 2.6)

  LEO: low-Earth orbit, as the name implies, this is the lowest altitude a spacecraft must achieve in order to orbit the Earth. The altitude is between 200 and 2000 km;

GEO: geostationary (or geosynchronous) orbit, this is a much higher orbit and so takes a lot more energy to reach. However, once at the altitude of 35786 km, it takes the spacecraft a full 24 hours to orbit the Earth. Thus, the spacecraft moves at the same speed with which the Earth rotates and therefore appears to "hover" over the same spot on the ground;

MEO: medium-Earth orbits, these are between LEO and GEO orbits;

HEO: high-Earth orbits, normally are highly elliptical, but always at altitudes over the GEO;

- Direction

prograde: any orbit in which the spacecraft moves from west to east is termed prograde. This is the usual direction of rotation in our Solar System. Only a handful of objects orbit or rotate in the opposite direction;

retrograde: any orbit in which the spacecraft moves from east to west. This is the less usual direction in the Solar System; however, it is not impossible. For example, Venus has retrograde spin and some comets (notably comet Halley, which was encountered by ESA's Giotto spacecraft in 1986) also has a retrograde orbit.

2.3.1 Orbital parameters

The orbital elements or Keplerian orbital parameters are a set of parameters necessary in order to determine uniquely an orbit, given an ideal system of two masses that follow the Newton law of motion and the universal gravitational law. The set of traditional orbital parameters is:

e: eccentricity, that is (where a is the semimajor-axis and b is the semiminor-axis)

$$e = \sqrt{1 - \frac{b^2}{a^2}}; \tag{2.21}$$

i: inclination of the orbital plane referred to the equatorial plane;

Ω: Right Ascension of the Ascending node, RAAN, that is the angle, on the equatorial plane, between the First Point of Aries and the ascending node;

ω: argument of the periapsis, that is the angle, on the orbital plane, between the ascending node and the eccentricity vector. The eccentricity vector has the same direction as the line of apses (periapsis-apoapsis) and points from the apoapsis towards the periapsis;

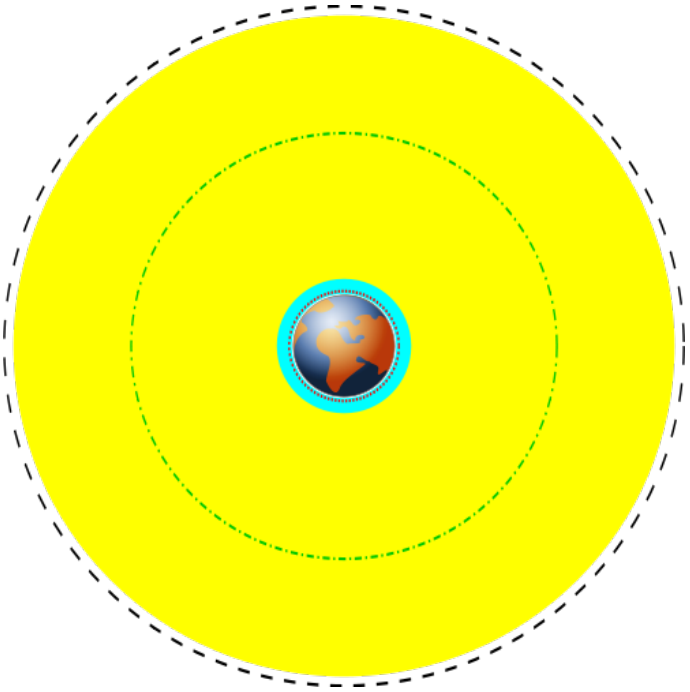


Figure 2.6 – Various Earth orbits to scale: innermost, the red dotted line represents the orbit of the International Space Station (ISS); cyan represents low Earth orbit, yellow represents medium Earth orbit, and the black dashed line represents geosynchronous orbit. The green dash-dot line represents the orbit of Global Positioning System (GPS) satellites.

T: orbital period, it represents the time required to complete an orbit. It is related to the semimajor-axis using Kepler's third law:

$$K = \frac{T^2}{a^3} \tag{2.22}$$

where K is the proportionality constant, almost the same for any planet around the Sun;

M: mean anomaly, it defines the position of the satellite in the ellipse and it is an angle that increases uniformly in time from 0° to 360° during one revolution.

The orbital period T can be replaced by the mayor semiaxis a, while the mean anomaly M by the true anomaly ν: this is the only variable parameter of the six because it describes position of the orbiting object on the orbit plane as an angle between the same object and the periapsis.

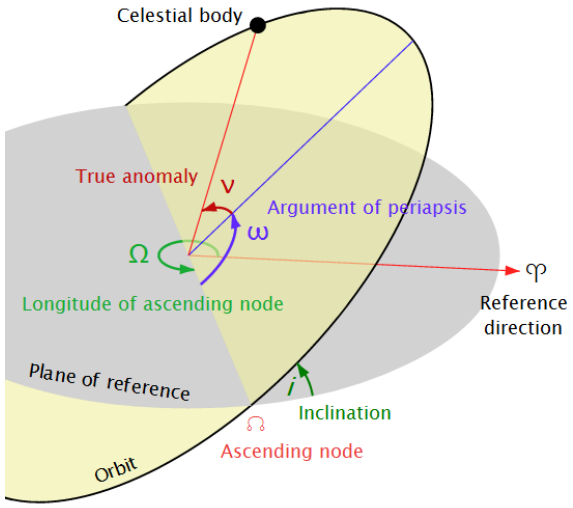


Figure 2.7 – Orbital parameters.

2.3.2 How to determine the orbit?

There are different ways to determine the orbit with different accuracies [6]:

real-time orbit determination: it provides the best estimate of where a satellite is at the present time and may be important for spacecraft and payload operations, such as accurate pointing at some target;

definitive orbit determination: it is the best estimate of the satellite position and orbital elements at some earlier time, it is done after gathering and processing all relevant observations;

orbit propagation: it refers to integrating the equations of motion to determine where a satellite will be at some other time. Usually orbit propagation refers to looking ahead in time from when the data was taken and is used either for planning or operations. Occasionally orbits will be propagated backward in time, either to determine where a satellite was in the past or to look at historical astronomical observations in the case of comets or planets.

We focus on the last type and there is also the possibility of using NORAD (more info in 2.3.2.2)

2.3.2.1 Orbit propagation

As just seen an orbit is completely defined given six orbital parameters, however this results as an ideal orbit in which are not considered the disturbances induced by

external factors. In the following treatment the disturbances induced on the orbit by the gravitational attraction of Earth, Moon and Sun and the presence of residual atmosphere that causes, by aerodynamic drag, the slowdown of the satellite are not considered. The orbital propagation follow the procedure below:

1. assignment of the orbital parameters, of the time and of the Earth constants;
2. calculation of the orbit properties from the orbital parameters;
3. variation of the orbital properties as function of the aerodynamic drag;
4. variation of the orbital properties as function of the gravitational attraction of Earth, Moon and Sun;
5. calculation of Ω (argument of latitude), distance from the Earth center and ν;
6. calculation of latitude and longitude from i, Ω and the argument of longitude;
7. calculation ECEF coordinates from latitude, longitude and distance from the Earth center.

In Figure 2.8 it is possible to observe the orbit propagation during 30 hours following the launch.

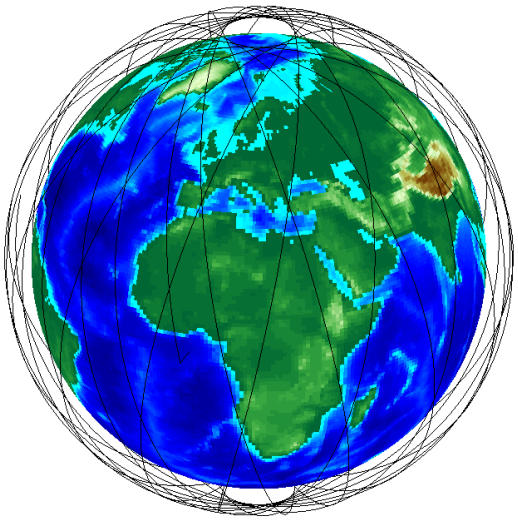


Figure 2.8 – Orbit propagation.

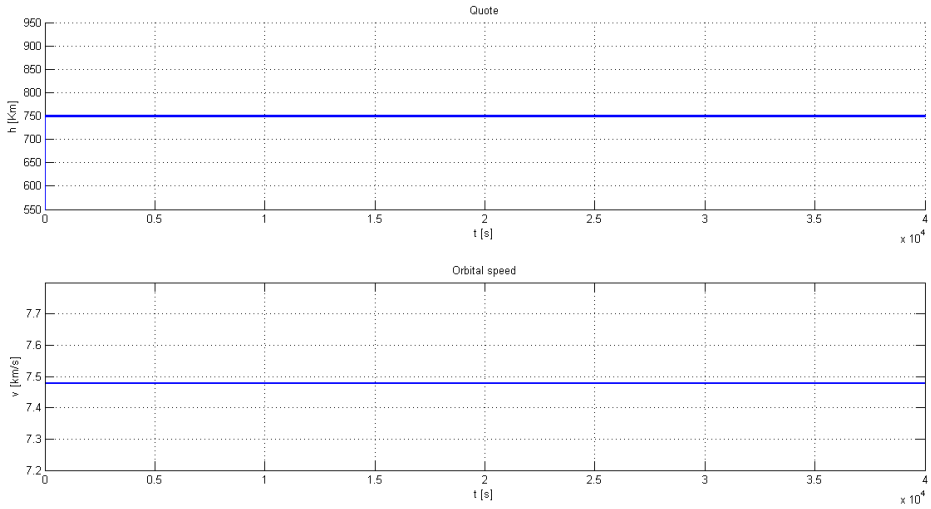


Figure 2.9 – Orbital speed and quote are constant because *3STAR* has a circular orbit.

2.3.2.2 NORAD

The North American Aerospace Defense Command is a joint organization of Canada and the United States that provides aerospace warning, air sovereignty and defense for the two countries. It was founded on May 12, 1958 (an effect of the Cold War) as a joint command between the governments of Canada and the United States, as the North American Air Defense Command.

In response to the emergence of the intercontinental ballistic missile and submarine-launched ballistic missile threat, a space surveillance and missile warning system was built to provide worldwide space detection, tracking and identification. The extension of NORAD's mission into space led to a name change, the North American Aerospace Defense Command in March 1981.

Considering the surveillance of the orbital objects around Earth, the NORAD, tracking every object bigger than a tennis ball, obtains for each of them a data set that describe its orbit. To simplify the notation also this data set is called TLE (see Figure 2.10).

2.4 Earth's magnetic field

Geomagnetism has a great importance for life on Earth. In fact, it extends for tens of thousands of kilometres into space, forming an area called the magnetosphere whose presence creates a sort of electromagnetic "shield" that deflects and reduces

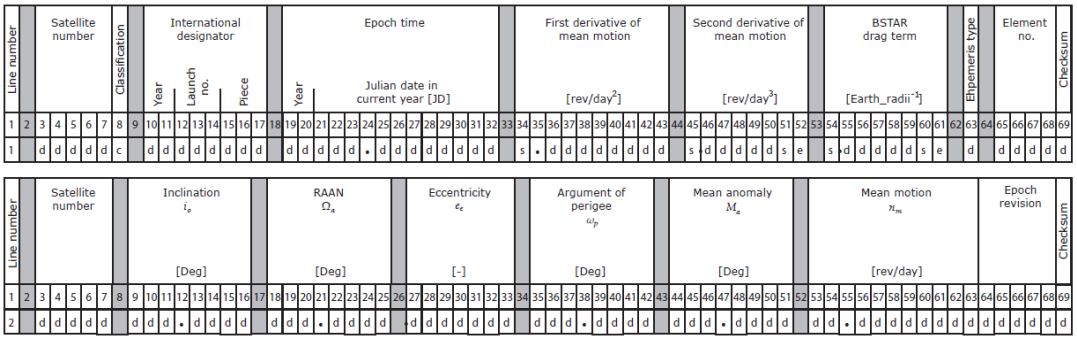


Figure 2.10 – TLE format where d is a decimal number, c is a character, s is a symbol and e is the exponent.

cosmic rays and, in general, all charged particles.

The geomagnetic field was the first Earth's field to be theorized and described,in fact its discovery is attributed to the observations of Pierre de Maricourt, French scientist of the thirteenth century, in his "*Epistula de magnete (Letter on the magnet)*", written in 1269.

The unit of the geomagnetic field, in International System of Units (SI, from French *Système international d'unités*), is Tesla (T), but being very small values of the field in this unit, in practice it is often used its submultiple nano-Tesla (nT), equal to 10^{-9} T, or the Gauss (G), equal to 10^{-4} T. From the equator to the poles, on the Earth's surface, the value of the magnetic field varies from about 20000 nT to about 70000 nT. The Earth's magnetic field is not constant, but undergoes significant changes in both direction and intensity. They have led, during the geological eras, to the drift of magnetic poles and repeated phenomena of inversion of the field, with mutual exchange of North and South magnetic poles. The Earth's magnetic field is subject to continuous variations of intensity and direction due to local or external causes. Long-term changes (5-10 years), also called secular variations, are due to changes in the deep sources and they are recognized by the data of magnetic observatories, archaeological and geological records. Changes in short term (a few minutes up to 5 years) are of external origin. A third category is that of very long period changes, related to the solar cycle of 11 years.

The secular variations are divided into a contribution due to the bipolar field and one due to the non-bipolar field (probably electric currents in the zones of contact between core and mantle as shown by investigations of magnetic tomography).

In summary, the secular variation is characterized by a average annual decrease in the dipole moment of 0.05%, a westward precession of the axis of dipole of 0.08° each year, an annual northward shift of 0.01°, drift of the western non-bipolar field of 0.2-0.3° per year and a variation of intensity of the field with a rate of about 10 nT per year. As for polarity reversals of the magnetic field (every 5000-10000

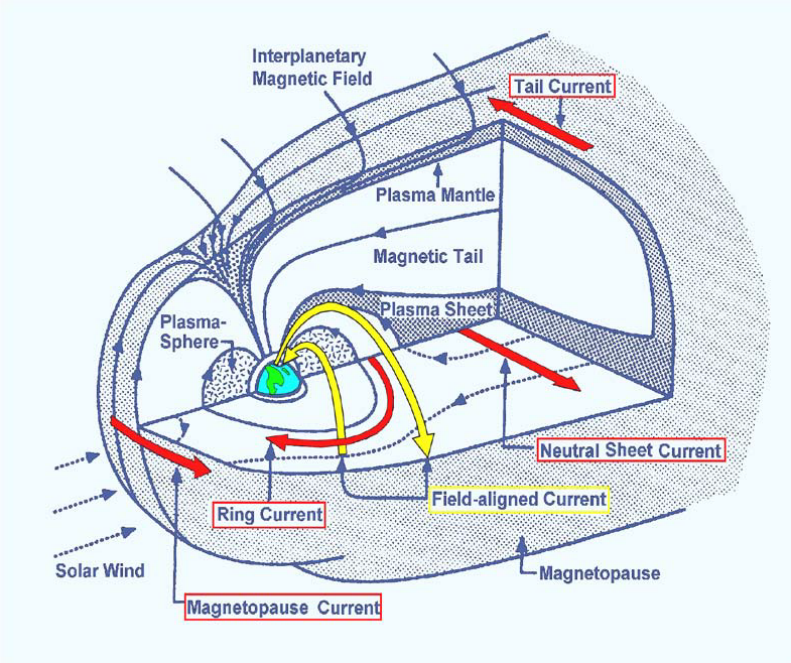


Figure 2.11 – Earth's magnetosphere.

years), they involve rapid changes in the declination of 180° and reversal of the tilt sign [9] [10].

2.4.1 Model

As it has been said so far, we understand that to properly model the Earth's magnetic field it is not enough an approximate model on theoretical grounds only, which could be the simple magnetic dipole model. It was therefore decided to rely on model *WMM* (World Magnetic field model) developed by the National Geophysical Data Center of the NOAA (National Oceanic and Atmospheric Administration) which is released every five years (the last one, WMM2010, was published 12/2009 [11]). These updates are designed so as to take into account with good approximation the variations of the geomagnetic field using linear approximations based on observations made in previous years and extrapolated for the next five years. In the Figure 2.13 and Figure 2.14 the errors estimation of declination of the model in 2010 and 2015 are shown.

The model is based on the calculation of main magnetic field as potential field written as spatial gradient of a scalar potential field described by an expansion to twelfth grade of Earth's magnetic field based on time-varying Gaussian coefficients. Secular changes are described similarly as above for the following five years and

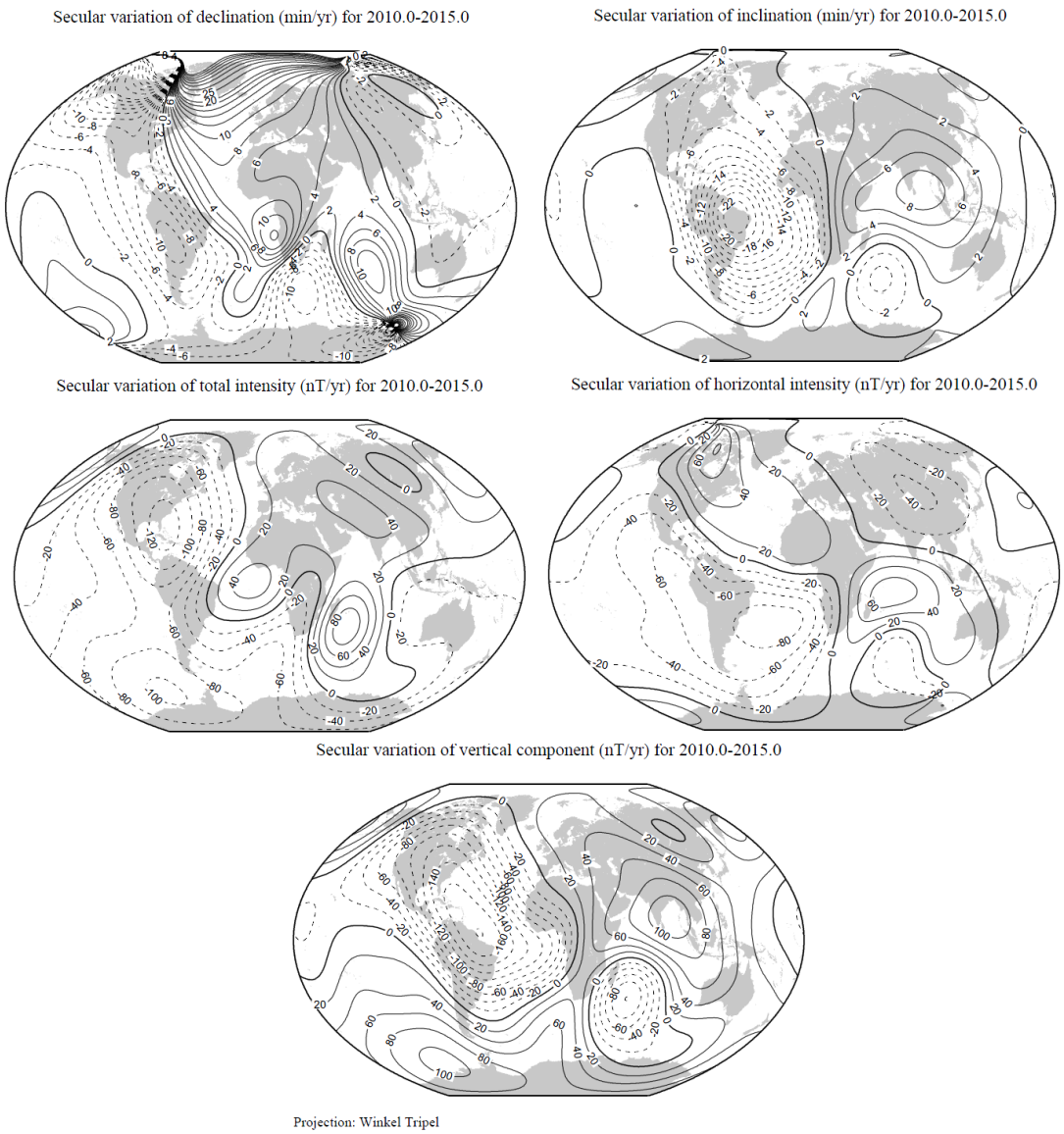


Figure 2.12 – Secular variation from 2010 to 2015 [8].

then superimposed on the main field as function of the input time (for the complete discussion of the model refers to: *The US/UK World Magnetic Model for 2010-2015* [12]). Below there is a brief analysis of magnetic field model used by analyzing the various parts and in Figure 2.15 it is possible to see the variation of magnetic field in orbit:

- input

  latitude: from -90° to 90°;

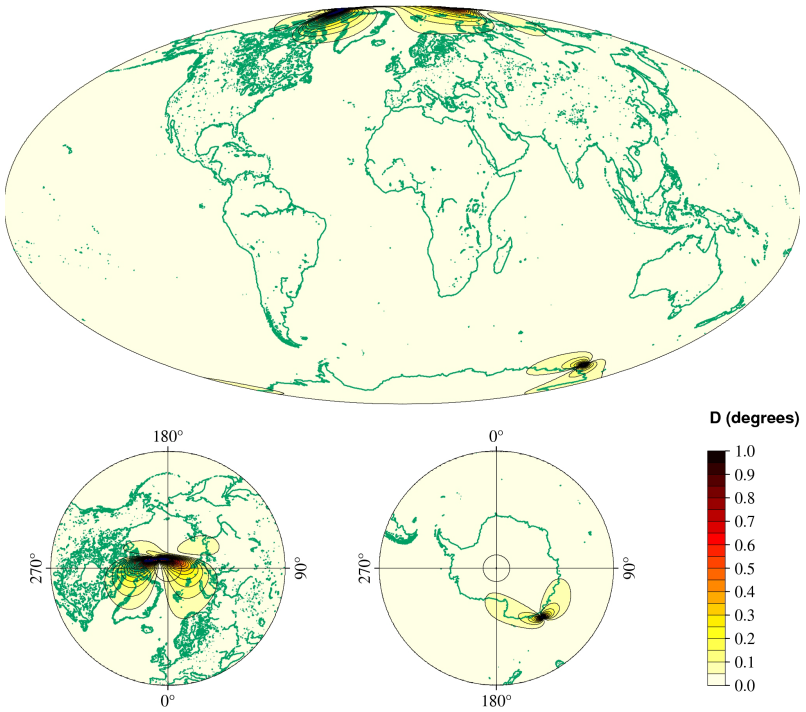


Figure 2.13 – Declination errors estimation for year 2010.

longitude: from -180° to 180°;

altitude: from the sea level to 1000 km;

date: from the base year of the model to five year later.

- output

  F: total intensity of the magnetic field;

  H: horizontal intensity;

  X: North intensity;

  Y: East intensity;

  Z: vertical intensity;

  I: geomagnetic inclination;

  D: geomagnetic declination;

2.4.2 Effects on satellites

We can identify two main effects on a satellite in orbit due to the Earth's magnetic field. The first is the creation of a disturbance torque due to the interaction between

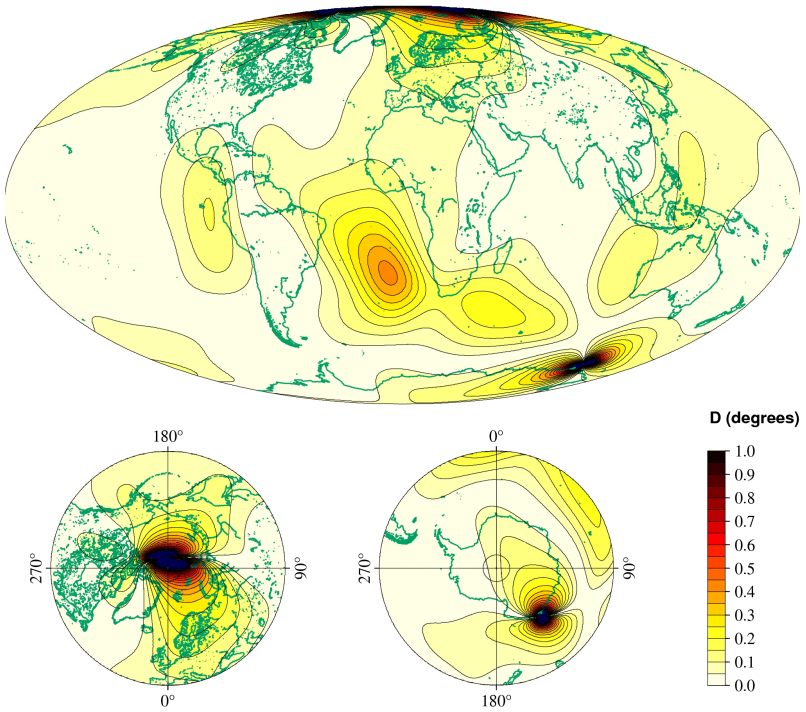


Figure 2.14 – Declination errors estimation for year 2015.

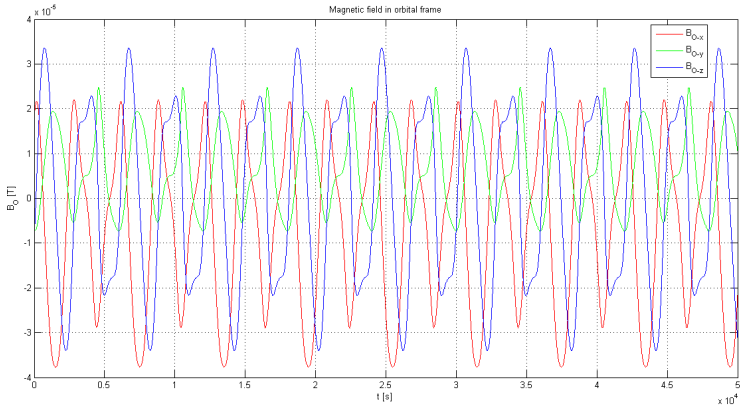


Figure 2.15 – Magnetic field measured in orbital frame.

the Earth's magnetic field and the magnetic field generated by the satellite itself, while the second is the shielding from the radiations and the charged particles from outside. If the first effect is clearly negative, and we will see in the section on the attitude control how to deal with it, on the contrary the second improves the conditions of the satellite in orbit allowing for greater operational life.

2.5 Control theory

Control theory is an interdisciplinary branch of engineering and mathematics that deals with the behavior of dynamical systems. The desired output of a system is called *reference*. When one or more output variables of a system need to follow a certain reference over time, a controller manipulates the inputs of a system to obtain the desired effect on the output of the system.

In control theory there are two basic types of control. These are feedforward[1] and feedback[2]. Feedback control usually results in intermediate periods where the controlled variable is not at the desired setpoint. Feedforward control can avoid the slowness of feedback control. With feedforward control, the disturbances are measured and accounted for before they have time to affect the system. The difficulty with feedforward control is that the effect of the disturbances on the system must be accurately predicted, and there must not be any unmeasured disturbances.

2.5.1 Main control strategies

To achieve the benefits of feedback control (controlling unknown disturbances and not having to know exactly how a system will respond to disturbances) and the benefits of feedforward control (responding to disturbances before they can affect the system), there are some combinations of feedback and feedforward control that can be used. Here there is a brief description for some of them:

Adaptive control: it uses on-line identification of the process parameters, or modification of controller gains, thereby obtaining strong robustness properties. Adaptive controls were applied for the first time in the aerospace industry in the 1950s, and they have found particular success in that field.

Hierarchic control: it is a type of control system in which a set of devices and governing software is arranged in a hierarchical tree. When the links in the tree are implemented by a computer network, then hierarchical control system is also a form of Networked control system.

[1] *Feedforward* is a term describing an element or pathway within a control system which passes a controlling signal from a source in the external environment of control system to a load elsewhere in its external environment. A control system which has only feedforward behavior responds to its control signal in a predefined way without responding to how the load reacts.

[2] *Feedback* describes the situation when the output from an event or phenomenon in the past will influence an occurrence of the same event in the present or future. When an event is part of a chain of cause-and-effect that forms a circuit or loop, then the event is said to "feed back" into itself.

Intelligent control: it uses various AI computing approaches like neural networks, Bayesian probability, fuzzy logic, machine learning, evolutionary computation and genetic algorithms to control a dynamic system.

Robust control: it deals explicitly with uncertainty in its approach to controller design. Controllers designed using robust control methods tend to be able to cope with small differences between true system and nominal model used for design. The early methods of Bode and others were fairly robust; the state-space methods invented in the 1960s and 1970s were sometimes found to lack robustness. A modern example of a robust control technique is *H-infinity Loop-Shaping* developed by Duncan McFarlane and Keith Glover of Cambridge University, United Kingdom. Robust methods aim to achieve robust performance and/or stability in the presence of small modelling errors.

Stochastic control: it deals with control design with uncertainty in the model. In typical stochastic control problems, it is assumed that there exist random noise and disturbances in the model and the controller so the control design must take into account these random deviations.

Optimal control: it is a particular control technique in which the control signal optimizes a certain "cost index". Two optimal control design methods have been widely used in industrial applications, as it has been shown they can guarantee closedloop stability. These are Model Predictive Control (MPC) and Linear-Quadratic-Gaussian control (LQG). The first can more explicitly take into account constraints on the signals in the system, which is an important feature in many industrial processes. However, the "optimal control" structure in MPC is only a means to achieve such a result, as it does not optimize a true performance index of the closed-loop control system. Together with PID controllers, MPC systems are the most widely used control technique in process control.

2.5.2 Types of Controllers

Tuning a control loop is the adjustment of its control parameters to reach the optimum values for the desired control response. Stability is a basic requirement, but beyond that, different systems have different behavior, different applications have different requirements. Furthermore, some processes have a degree of non-linearity and for this parameters that work well at full-load conditions, don't work when the process is starting up from no-load. Some of the types of controllers are now presented with a brief description:

Open-loop: it can be used in systems sufficiently well-characterized as to predict what outputs will necessarily achieve the desired states. Drawbacks of open-loop control is that it requires perfect knowledge of the system and it assumes there are no disturbances to the system;

Proportional: it is also called P controller. With this type of controller, the controller output is proportional to the error of the measured variable. In feedback control, it is standard to define the error as the difference between the desired value y_s and the current value y. If the error is large, then the control action is large. Mathematically:

$$u(t) = K_c e(t) + u_0 \tag{2.23}$$

where $u(t)$ represents the controller output, $e(t) = y_s(t) - y(t)$ represents the error, K_c represents the controller gain and u_0 represents the steady state control action (bias). It is important that the control action $u(t)$ counteracts the change in the controlled variable $y(t)$ (negative feedback).

Proportional-Integral-Derivative: it is also called *PID* controller. It is a generic control loop feedback mechanism widely used in industrial control systems. A PID controller calculates an "error" value as the difference between a measured process variable and a desired setpoint. The controller attempts to minimize the error by adjusting the process control inputs. The PID controller algorithm involves three separate parameters, in fact it is sometimes called *three-term* control: the proportional, the integral and derivative values, denoted P, I, and D. The proportional value determines the reaction to the current error, the integral value determines the reaction based on the sum of recent errors, and the derivative value determines the reaction based on the rate at which the error has been changing. The weighted sum of these three actions is used to adjust the process. Heuristically, these values can be interpreted in terms of time: P depends on the present error, I on the accumulation of past errors, and D is a prediction of future errors, based on current rate of change. The proportional, integral, and derivative terms are summed to calculate the output of the PID controller. Defining $u(t)$ as the controller output, the final form of the PID algorithm is:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \tag{2.24}$$

where the control parameters are K_p, K_i and K_d:

- larger values of the proportional gain typically mean faster response since the larger the error, the larger the proportional term compensation. An

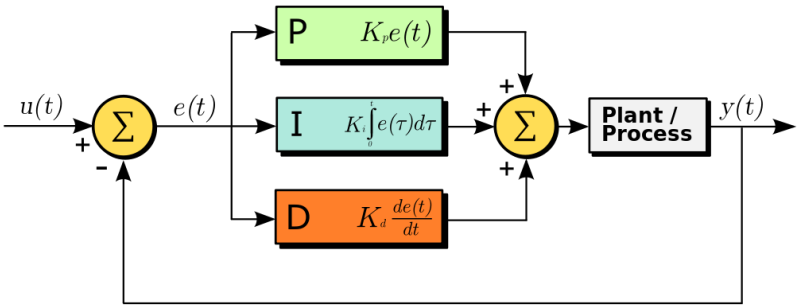


Figure 2.16 – A typical block diagram of a PID controller.

excessively large proportional gain will lead to process instability and oscillation;

- larger values of the integral gain imply that steady state errors are eliminated more quickly. The disadvantage is represented by a larger overshoot;
- larger values of the derivative gain decrease overshoot, but slow down transient response and may lead to instability due to signal noise amplification in the differentiation of the error.

PID controllers often provide acceptable control even in the absence of tuning, but performance can generally be improved by careful tuning and they may be unacceptable with poor tuning.

Proportional-Derivative: it is also called *PD* controller. Only the proportional and derivative feedback of the PID controller are active, that is the gain of the integral feedback is set to zero. An example of PD controller is the one implemented for the detumbling attitude control in Figure 4.7.

Linear-Quadratic-Gaussian: it is also called *LQG* controller. It is simply the combination of a Kalman filter, i.e. a Linear-Quadratic Estimator (LQE), with a Linear-Quadratic Regulator (LQR). The separation principle guarantees that these can be designed and computed independently. LQG control applies to both linear time-invariant systems as well as linear time-varying systems. The application to linear time-varying systems enables the design of linear feedback controllers for non-linear uncertain systems.

The LQG controller itself is a dynamic system like the system it controls. Both systems have the same state dimension. Therefore implementing the LQG controller may be problematic if the dimension of the system state is large. The reduced-order LQG problem (fixed-order LQG problem) overcomes this by fixing a-priori the number of states of the LQG controller. This problem

is more difficult to solve because it is no longer separable. Also the solution is no longer unique. Despite these facts numerical algorithms are available to solve the associated optimal projection equations which constitute necessary and sufficient conditions for a locally optimal reduced-order LQG controller.

LQG optimality does not automatically ensure good robustness properties. The robust stability of the closed loop system must be checked separately after the LQG controller has been designed. To promote robustness some of the system parameters may be assumed stochastic instead of deterministic. Consider the linear dynamic system:

$$\dot{\bar{x}}(t) = \boldsymbol{A}(t)\bar{x}(t) + \boldsymbol{B}(t)\bar{u}(t) + \bar{v}(t) \tag{2.25}$$

$$\bar{y}(t) = \boldsymbol{C}(t)\bar{x}(t) + \bar{w}(t) \tag{2.26}$$

where $\bar{x}$ represents the vector of state variables of the system, $\bar{u}$ the vector of control inputs and $\bar{y}$ the vector of measured outputs available for feedback. Both added noise, that is $\bar{w}(t)$ affect the system. Given this system, the aim is to find control input history $\bar{u}(t)$ which at every time t may depend only on the past measurements such that the following cost function is minimized ($\mathbb{E}$ denotes the expected value):

$$J = \mathbb{E}\Big(\bar{x}'(T)\boldsymbol{F}\bar{x}(T) + \int_0^T \big(\bar{x}'(T)\boldsymbol{Q}\bar{x}(T) + \bar{u}'(T)\boldsymbol{R}\bar{u}(T)dt\big)\Big) \tag{2.27}$$

The LQG controller that solves the LQG control problem is specified by the following equations:

$$\dot{\hat{x}}(t) = \boldsymbol{A}(t)\hat{x}(t) + \boldsymbol{B}(t)\bar{u}(t) + \boldsymbol{K}(t)(\bar{y}(t) - \boldsymbol{C}(t)\hat{x}(tv)) \tag{2.28}$$

$$\bar{u}(t) = -\boldsymbol{L}(t)\hat{x}(t) \tag{2.29}$$

The matrix $\boldsymbol{K}(t)$ is called *Kalman gain* of the associated Kalman filter represented by the first equation. At each time t this filter generates estimates $\hat{x}(t)$ of the state $\bar{x}(t)$ using the past measurements and inputs. The Kalman gain $\boldsymbol{K}(t)$ is computed from the matrices $\boldsymbol{A}(t)$, $\boldsymbol{C}(t)$, the two intensity matrices $\boldsymbol{V}(t)$, $\boldsymbol{W}(t)$ associated to the white Gaussian noises $\bar{v}(t)$ and $\bar{w}(t)$ and finally $P(0) = \mathbb{E}(\bar{x}(0)\bar{x}'(0))$. These five matrices determine the Kalman gain through the following Riccati differential equation:

$$\dot{\boldsymbol{P}}(t) = \boldsymbol{A}(t)\boldsymbol{P}(t) + \boldsymbol{P}(t)\boldsymbol{A}'(t) - \boldsymbol{P}(t)\boldsymbol{C}'(t)\boldsymbol{W}^{-1}(t)\boldsymbol{C}(t)\boldsymbol{P}(t) + \boldsymbol{V}(t) \tag{2.30}$$

Given the solution $\boldsymbol{P}(t)$ the Kalman gain is equal to:

$$\boldsymbol{K}(t) = \boldsymbol{P}(t)\boldsymbol{C}'(t)\boldsymbol{W}^{-1}(t) \tag{2.31}$$

The matrix $\boldsymbol{L}(t)$ is called *feedback gain matrix*. This matrix is determined by the matrices $\boldsymbol{A}(t)$,$\boldsymbol{B}(t)$, $\boldsymbol{Q}(t)$, $\boldsymbol{R}(t)$ and $\boldsymbol{F}$ through the following associated matrix Riccati differential equation:

$$\dot{\boldsymbol{S}}(t) = \boldsymbol{A}'(t)\boldsymbol{S}(t) + \boldsymbol{S}(t)\boldsymbol{A}(t) - \boldsymbol{S}(t)\boldsymbol{B}(t)\boldsymbol{R}^{-1}(t)\boldsymbol{B}'(t)\boldsymbol{S}(t) + \boldsymbol{Q}(t) \tag{2.32}$$

Given the solution $\boldsymbol{S}(t)$ the feedback gain is equal to:

$$\boldsymbol{L}(t) = \boldsymbol{R}^{-1}(t)\boldsymbol{B}'(t)\boldsymbol{S}(t) \tag{2.33}$$

It's important to observe the similarity of the two matrix Riccati differential equations, the first one running forward in time, the second one running backward in time: this similarity is called *duality*. The first matrix Riccati differential equation solves the LQE problem while the second solves the LQR problem. These problems are dual and together they solve the LQG control problem.

Chapter 3

System design

In order to guarantee the satellite is able to control adequately its attitude, it is necessary to develop a model quite accurate, that is a system capable of representing the real physic that the satellite will face when it is in orbit. The project has been divided into several phases:

1. requirements definition;
2. functional analysis;
3. development and implementation of a basic mathematical model;
4. trade-off between different possible configurations;
5. improvements to the mathematical model.

3STAR is a 3U cubesat, a category not very common; in fact, while in orbit there is a considerable number of units of the classical type (1U), including *e-st@r* (previous project of Politecnico di Torino) from 13th February 2012, the number of 3U type in orbit is smaller than ten. Among these, three have been designed by NASA (therefore for obvious differences in the available budget, they can not be a source of information to design the attitude determination and control system) and most of the remaining have a passive ADCS. Consequently, an active ADCS for this category represents an important innovation and challenge, just because the background is not as wide as the 1U category.

3.1 Requirements

The *attitude determination and control subsystem* measures and controls the spacecraft's angular orientation, that is pointing direction. Seeing at mission statement

and objectives, it is evident that the system must be able to ensure appropriate pointing and adequate reorientation when required, so it is necessary an active control system, because the pointing accuracy provided by passive systems is not suitable for this type of mission. Obviously the ADCS requirements are closely tied to mission needs and characteristics of other subsystems, as it is possible to see in Table 3.1.

Table 3.1 – *3STAR* requirements.

Code	*Requirement description*
	Mission requirements
3S.MR-230	*3STAR* complexity shall be minimized
3S.MR-240	*3STAR* cost shall be minimized
3S.MR-360	*3STAR* mission shall be performed in a LEO
	Space segment requirements
	Space segment general requirements
SSR.GR-010	The system shall be placed on a LEO orbit
SSR.GR-020	The system shall remain in a LEO orbit between 500 and 750 km during all the mission exploitation time
SSR.GR-100	*3STAR* shall be equipped with an attitude determination and control system
SSR.GR-101	The attitude control system shall be partially or completely active to fulfill the antenna pointing requirements
	Mechanical requirements
SSR.MCR-040	*3STAR* mass shall not exceed 4 kg
SSR.MCR-130	The ADCS mass shall not exceed TBD g
	Thermal requirements
SSR.TR-030	*3STAR* temperature shall be mantained in TBD range during all mission phases
	Functional requirements
	P-GRESSION payload
PG-FR-020	*3STAR* system shall provide the right Zenit antenna pointing
PG-FR-030	*3STAR* system shall provide the right Nadir antenna pointing

Table 3.1: continues on next page

Table 3.1: continues from the previous page

Code	*Requirement description*
PG-FR-040	*3STAR* system shall provide the right Limb antenna pointing
PG-FR-050	*3STAR* system shall provide an active ADCS to allow the antennas pointing
	Bus
B.FR-405	The spacecraft shall be able to determine its attitude (at least its roll)
B.FR-410	The ADCS shall determine the attitude via IMU and magnetometer
B.FR-411	The IMU shall provide angular velocity measurement
B.FR-412	The IMU shall provide acceleration measurement
B.FR-413	The magnetometer measurement accuracy shall be at last TBD mT
B.FR-414	The IMU velocity mesurement accuracy shall be at last TBD rad/s
B.FR-415	The IMU acceleration measurement accuracy shall be at last TBD m/s^2
B.FR-420	The ADCS shall eventually determine the attitude via sun sensor
B.FR-430	The ADCS shall control the attitude via magnetic coils or micro reaction wheels
B.FR-440	The ADCS pointing accuracy shall be lower than TBD°
B.FR-450	The moment generated shall be higher than TBD Nm
B.FR-451	Maximum actuated angular velocity shall be at last TBD rad/s
B.FR-460	The ADCS shall have a dedicated microprocessor for data elaboration
B.FR-470	The ADCS microprocessor shall communicate with the OBC via a TBD port
B.FR-480	The ADCS microprocessor shall communicate with the OBC with TBD protocol
B.FR-490	Actuators shall be controlled via PWM
B.FR-500	The ADCS shall choose the best measurement set provided by sensors
B.FR-510	The magnetometer shall not be jammed by the magnetic field generated inside the spacecraft

Table 3.1: continues on next page

Table 3.1: continues from the previous page

Code	*Requirement description*
B.FR-520	ADCS shall be able to detumbling the satellite after the deployment
B.FR-530	The ADCS actuators maximum consumption shall be TBD W
	Operational requirements
SSR.OR-020	The cubesat shall be able to receive a command to change his attitude
SSR.OR-100	The cubesat shall be able to receive a command to reboot/shut down the ADCS
	Interface requirements
	Physical interfaces
SSR.PINT-060	EPS-PCDU, ADCS, and COMSYS shall be attached to the 3STAR bus, on the OBC board
SSR.PINT-070	The OBC, COMSYS, ADCS, and EPS-PCDU shall be attached to the main structure
	Functional interfaces
SSR.FINT-020	EPS-PCDU, ADCS, OBC, and COMSYS shall communicate via the 3STAR bus, using a defined communication protocol
SSR.FINT-070	The ADCS shall provide the OBC with IMU, Magnetometer and if present the Sun Sensor telemetry, and MT command information and housekeeping data

Table 3.1: ends from the previous page

3.2 Functional analysis

The attitude determination and control subsystem (ADCS) stabilizes the vehicle and orients it in desired directions during the mission despite the external disturbance torques acting on it. This requires that the vehicle is able to determine its attitude, using sensors (like magnetometer, inertial measurement unit and sun sensor), and to control it using actuators (for *3STAR* , requirements impose to adopt magnetic torquers or/and reaction wheels). After a careful evaluation of all the requirements, it has been possible to define functions that ADCS subsystem must be able to guarantee. The most important are (Figure 3.1 to see all functions, Figure 3.2 to see

functions-devices matrix and Figure 3.3 to see a preliminary scheme of ADCS):

- to determine attitude;
- to control attitude (counteracting disturbance torques);
- to guarantee correct pointing;
- to communicate health-status to OBC;

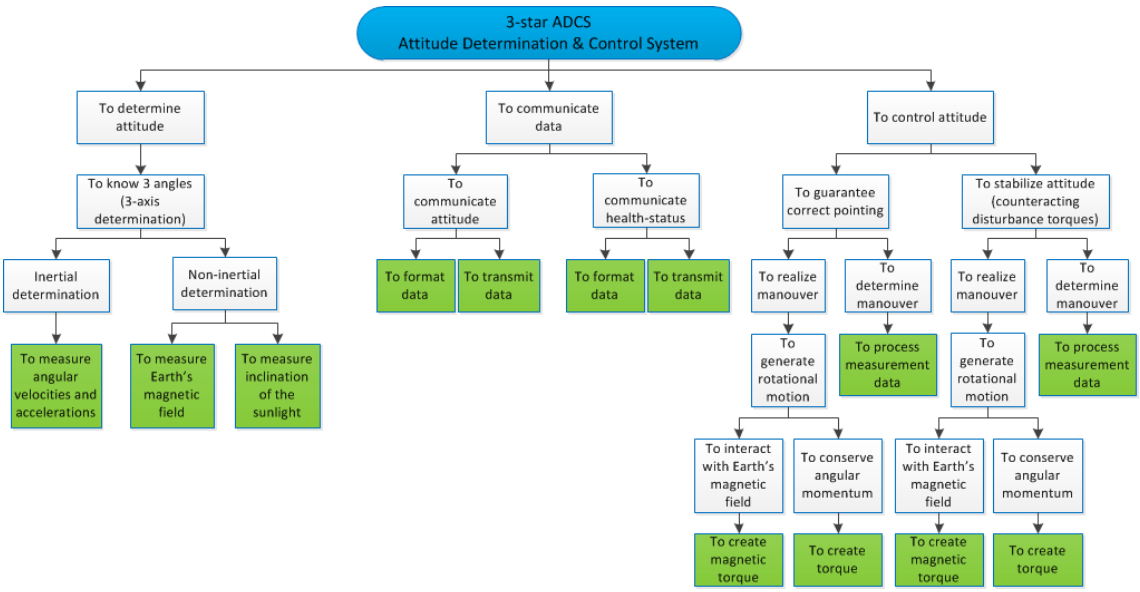


Figure 3.1 – ADCS functional tree.

	Sensors			Actuators		Microprocessor
Devices→ ***Functions↓***	IMU	Magnetometers	Sun sensors or solar panels	Magnetic torquers	Reaction wheels	ARM
To measure angular velocities and accelerations	X					
To measure Earth's magnetic field		X				
To measure inclination of the sunlight			X			
To format data						X
To transmit data						X
To process measurement data						X
To create magnetic torque				X		
To create torque					X	

Figure 3.2 – Functions-devices matrix.

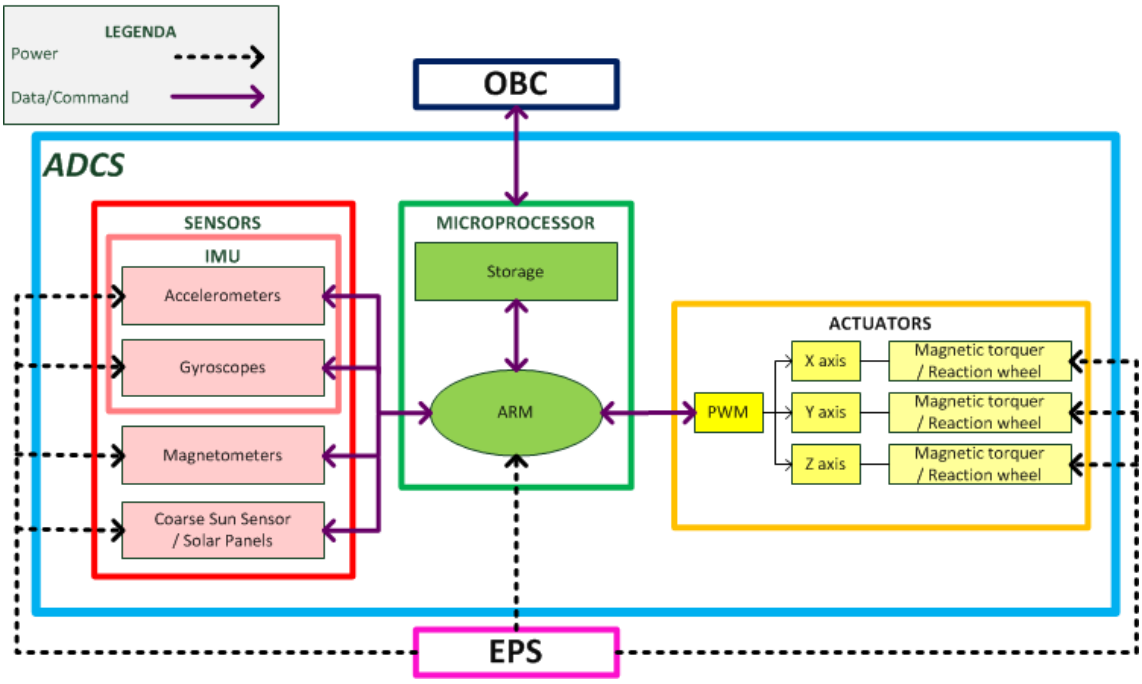


Figure 3.3 – Preliminary scheme of ADCS.

3.3 Mathematical model

This section presents satellite's preliminary mathematical model, used to simulate *3STAR* satellite in its environment so to be able to choose the most powerful configuration among those assumed.

3.3.1 Dynamics

Some assumptions are considered to obtain the dynamic model of the satellite. For example, it can be considered as an ideal rigid body, so the dynamic model is derived using a Newton-Euler formulation, where the angular momentum changes related to applied torques. In the generic case of three-axial spinning satellite, it is possible to write:

$$\dot{\bar{h}}^B + \bar{\omega}_{IB}^B \times \bar{h}^B = \bar{T}^B \tag{3.1}$$

where $\bar{h}^B$ is the momentum vector expressed in the Body frame, $\bar{\omega}_{IB}^B$ the angular velocity of the Body frame relative to the ECI frame expressed in the Body frame and $\bar{T}^B$ the sum of all torques acting on the satellite. If there are no internal moving parts the previous equation can be rewritten, considering $\mathbf{I}$ inertia matrix, as:

$$\mathbf{I}\dot{\bar{\omega}}_{IB}^B + \bar{\omega}_{IB}^B \times \mathbf{I}\bar{\omega}_{IB}^B = \bar{T}^B \tag{3.2}$$

that gives the equation for the angular acceleration:

$$\dot{\bar{\omega}}_{IB}^{B} = \mathbf{I}^{-1}(\bar{T}^{B} - \bar{\omega}_{IB}^{B} \times \mathbf{I}\bar{\omega}_{IB}^{B}) \tag{3.3}$$

Integrating the acceleration over time it is possible to calculate the angular velocity of the satellite. Moreover, the angular velocity $\bar{\omega}_{IB}^{B}$ can be written as the sum of two angular velocities, as

$$\bar{\omega}_{IB}^{B} = \bar{\omega}_{IO}^{B} + \bar{\omega}_{OB}^{B} = \boldsymbol{R}_{O}^{B}\bar{\omega}_{IO}^{O} + \bar{\omega}_{OB}^{B} \tag{3.4}$$

where $\boldsymbol{R}_{O}^{B}$ is rotation matrix from orbit frame to body frame (Equation 2.19) and and $\bar{\omega}_{IO}^{O} = \begin{bmatrix} 0 & \omega_o & 0 \end{bmatrix}$ is the known angular velocity of the Orbit frame relative to the ECI frame, expressed in Orbit frame. This velocity depends only on the altitude of the orbit, and can be calculated as

$$\omega_o = \sqrt{\frac{GM_e}{R^3}} \tag{3.5}$$

where G is the gravitational constant, M_e is the mass of the Earth and R is the distance from the center of the Earth to the satellite. The implementation in Matlab Simulink is represented in the Figure 3.4.

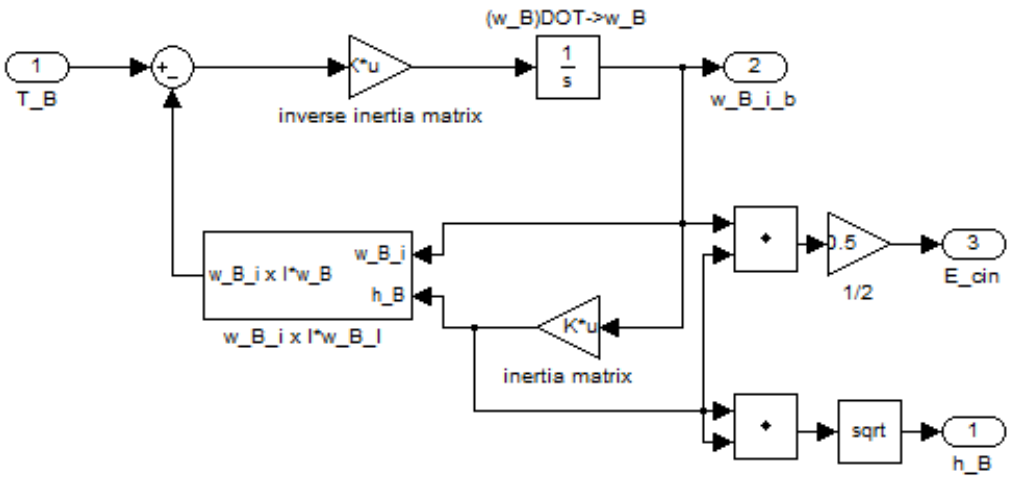


Figure 3.4 – Simulink model for the computation of the dynamics of *3STAR* .

3.3.2 Kinematics

Kinematics of the satellite describes the orientation of the satellite and its calculation is simply obtained through integration over time of the angular velocity. Using

quaternions to describe the attitude, it is possible to write:

$$\bar{q} = \int \dot{\bar{q}} dt \tag{3.6}$$

$$\dot{\bar{q}} = \begin{bmatrix} -\frac{1}{2}\vec{q} \cdot \bar{\omega}_{OB}^{B} \\ \frac{1}{2} q_0 \bar{\omega}_{OB}^{B} + \frac{1}{2}\vec{q} \times \bar{\omega}_{OB}^{B} \end{bmatrix} \tag{3.7}$$

where $\vec{q}$ is the the vectorial part of the quaternion (Equation 2.15) and $\bar{\omega}_{OB}^{B}$ is the angular velocity of the Body frame relative to the Orbit frame expressed in the Body frame (it can be obtained from Equation 3.4). The implementation in Matlab Simulink is represented in the Figure 3.5.

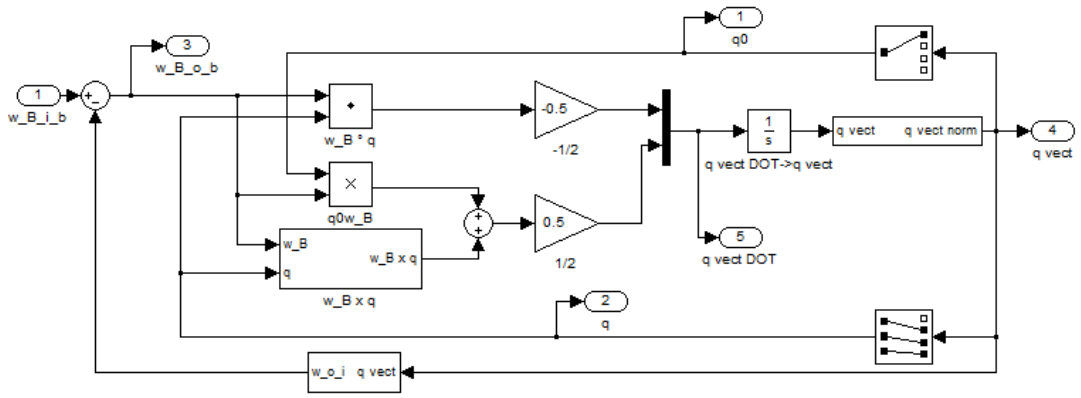


Figure 3.5 – Simulink model for the computation of the kinematics of *3STAR* .

3.3.3 Torques acting on *3STAR*

The attitude of a satellite is influenced by control torques and a number of different disturbances. The first depend on the attitude control system that, if properly developed, should be capable to place the satellite in the desired attitude. The second however, as the name implies, are disturbances caused by internal or external sources that interact with the satellite to change its attitude, in fact in both cases they act on the satellite as torques. The dominant external disturbances are caused by the magnetic and gravitational field, the aerodynamic drag in the upper atmosphere, solar radiation pressure and pressure from impacts of micrometeorites. For our mission, the aerodynamic drag can be considered negligible and the last mentioned before can however be considered highly unlikely in low orbits around the Earth [13].

3.3.3.1 Disturbance from the Earth's gravitational field

A non-symmetrical satellite orbiting the Earth is subjected to a gravitational torque due to the Earth's non-uniform gravitational field. Assuming that the satellite is only influenced by the Earth's gravitational field, the satellite consists of a single body and that both the Earth and the satellite are assumed to be two point masses, it is possible to calculate the gravitational force exerted by the Earth on the satellite, using Newton's law of gravitation. In general, it is possible to define the intensity of a gravitational field generated by presence of a mass as

$$\bar{g} = -\frac{GM}{r^2}\hat{r} \tag{3.8}$$

where G is the gravitational constant, M is the mass and r is the distance from the center of mass. Any object immersed in the gravitational field undergoes an acceleration, so a force greater in its parts closer to the body that generates the field, compared to more distant parts. This difference of forces applied to various parts of a satellite generates a torque that tries to align the major axis of the same with the local vertical. This torque is calculated as:

$$\bar{T_g} = -3\omega_o^2 \hat{r} \times \mathbf{I} \cdot \hat{r} \tag{3.9}$$

3.3.3.2 Disturbance from atmospheric drag

The aerodynamic drag disturbance originates from atmospheric molecules colliding with the surface of the satellite: it is the cause of the variation of the attitude and the reducing of the height of the orbit. If there is a difference between the center of mass and the center of pressure of the satellite (indicated with $\bar{r}_{cp}$), there will be a resulting torque given by:

$$\bar{T_a} = \bar{r}_{cp} \times \bar{F}_a \tag{3.10}$$

where the aerodynamic force vector $\bar{F}_a$ is

$$\bar{F}_a = \frac{1}{2}\rho V^2 C_D S \frac{\bar{V}}{V} \tag{3.11}$$

3.3.3.3 Disturbance from the satellite's magnetic residual

During its orbit, the satellite is situated in the Earth's magnetosphere that protects it from cosmic radiation. However it has to be considered that the satellite has a residual magnetic field primarily originates from currents in the on-board electronics and hysteresis effects in ferromagnetic materials. The interaction of this residual magnetic dipole moment of the satellite (m_r) and the Earth's magnetic field (B) generates a magnetic disturbance torque that can be quantified as:

$$\bar{T}_{mr}^B = \bar{m}_r^B \times \bar{B}^B \tag{3.12}$$

3.3.3.4 Control with magnetic torquers

This type of control is based on the same principle that generates the disturbance magnetic torques with the substantial difference that in this case the dipole moment is created intentionally to modify the attitude. The desired torque is obtained as

$$\bar{T}_m^B = \bar{m}^B \times \bar{B}^B \tag{3.13}$$

It is possible to calculate the dipole moment $\bar{m}$ as

$$\bar{m} = NIA\hat{n}_A \tag{3.14}$$

where N is the number of coils, I the current flowing in them, A the area inscribed by the coils and $\hat{n}_A$ the unit vector perpendicular to the plane of the coils. The model implemented is shown in Figure 3.6.

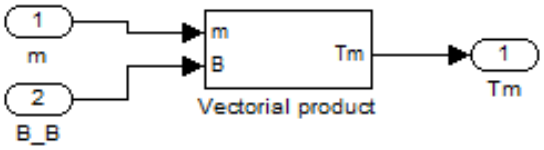


Figure 3.6 – Simulink model of magnetic torquers.

3.3.3.5 Control with reaction wheels

Reaction wheels are a common choice for active spacecraft attitude control. In this mode of control, an electric motor attached to the spacecraft spins a small, freely rotating wheel, the rotational axis of which is aligned with a vehicle control axis. Obviously the electric motor drives the wheel in response to a correction command computed as part of the spacecraft's feedback control loop. They are capable of generating internal torques only; with such a system, the wheel rotates one way and the spacecraft the opposite way in response to torques imposed externally on the spacecraft. From application of Euler's momentum equation, the integral of the net torque applied over a period of time will produce a particular value of total angular momentum stored onboard the spacecraft, resident in the rotating wheel or wheels, depending on how many axes are controlled. When it is spinning as fast as it can with the given motor drive, the wheel becomes "saturated" and cannot further compensate external torques [6].

The reaction wheel configuration in X, Y and Z axes is generally descripted by the following equation:

$$\bar{T}_{rw}^B = \dot{\bar{h}}_{rw}^B + \bar{\omega}_{IB}^B \times \bar{h}_{rw}^B - \bar{T}_{friction}^B \tag{3.15}$$

where $\bar{T}^B_{rw}$ is the torque caused by reaction wheel, $\bar{h}^B_{rw}$ is the total moment vector of reaction wheel, and $\bar{T}^B_{friction}$ is the frictional torque caused by wheels and usually assumed to be zero.

The model [7] implemented is shown in Figure 3.7.

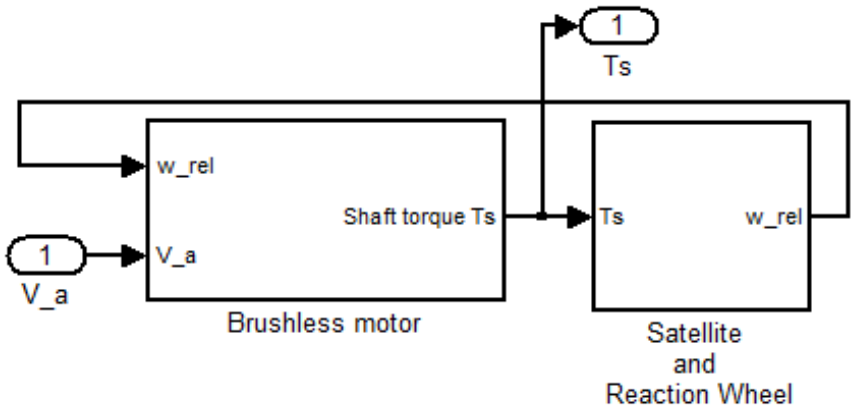


Figure 3.7 – Simulink model of reaction wheel.

3.4 Trade-off of possible configurations

During the early stages, i.e. the analysis of mission requirements and functional analysis, it was decided to adopt two possible types of controllers, magnetic torquers and reaction wheels. Obviously they can be positioned in different ways along the axes of reference and in different numbers, therefore it is necessary to do a trade-off between different configurations, also using the mathematical model developed previously (running multiple simulations for configurations assumed). The configurations investigated are:

- a magnetorquer for each axis;
- a magnetorquer for each axis and a reaction wheel on X axis;
- a magnetorquer for each axis and a reaction wheel on Y axis;
- a magnetorquer for each axis and a reaction wheel on Z axis;
- a magnetorquer for each axis and two reaction wheels, on X and Y axes;
- a magnetorquer for each axis and two reaction wheels, on X and Z axes;
- a magnetorquer for each axis and two reaction wheels, on Y and Z axes;

- a magnetorquer on X axis and a reaction wheel for each axis;
- a magnetorquer on Y axis and a reaction wheel for each axis;
- a magnetorquer on Z axis and a reaction wheel for each axis;

The instrument adopted to decide which configuration to take is based on a scoring system, in which marks are assigned to the various criteria and it is also given the "weight" of a criterion with respect to others. Weighting factors are obtained firts considering which of the criteria is more important than others. The table is produced asking "X (parameter on row) is more or less important than Y (parameter on column)?" and assigning 1 if X is more important than Y, 0 if they have the same importance and -1 if X is less important. Finally, the Equation 3.16 shows how the weighting factor is obtained (N_i is the sum of the elements on the row, N is the sum of the absolute values of N_i and J is the number of criteria):

$$W_{f-i} = \frac{N_i + N}{\sum N_i + NJ} \tag{3.16}$$

Marks are assigned from 0 (worst rating) to 10 (best score) based on personal knowledge, previous practical experience and information gathered on specialized texts.

Criteria evaluated are:

- mass;
- reliability;
- size;
- power consumption;
- cost;
- accuracy;
- stabilizing time.

Because of the limited budget, the cost has been considered in general the most important parameter, followed closely by reliability, power consumption and accuracy. The stabilization time is regarded as the less important since, in order to remain in those that are the limits of cost, mass and electric power, it is also disposed to have longer stabilization times than those that would occur, for example with components more powerful.

Thanks to this decision tool (the results thus obtained are shown in the Figure 3.8), it has been decided therefore to adopt the configuration with *a magnetic torquer for each axis and a reaction wheel only on Z axis* which reaches a score of 5.473, far superior to all other configurations (for example, the configuration with the second score is by far superior in terms of mass and cost but fails to stabilize the satellite).

This table compares the various characteristics considered, the table should be read by asking "X (parameter on row) is more or less important than Y (parameter on column)?" and the answer is 1 if it is more important, 0 if they have the same importance and -1 if it is less important.

	Mass	Reliability	Size	Power consumption	Cost	Accuracy	Stabilization time	N_i	Wf
Mass		-1	0	0	-1	-1	0	-3	0,116
Reliability	1		1	0	0	0	0	2	0,161
Size	0	-1		-1	0	0	1	-1	0,134
Power consumption	0	0	1		-1	1	0	1	0,152
Cost	1	0	0	1		0	1	3	0,170
Accuracy	1	0	0	-1	0		1	1	0,152
Stabilization time	0	0	-1	0	-1	-1		-3	0,116
							N =	14	
							J =	7	

KEY	
symbols	*explanation*
N_i	Sum of the coefficients of the i-th row
Wf	Weighting factors
N	Sum of absolute values of N_i column
J	Criteria number of trade-off
MT	Magnetic Torquer
RW	Reaction Wheel
..._X	...(MT or RW) on X axis
..._Y	...(MT or RW) on Y axis
..._Z	...(MT or RW) on Z axis
..._3	...(MT o RW) on all axes

In this table there are ratings of the characteristics for different configurations of actuators assumed, the values range is from 0 (worst rating) to 10 (best score)

		MT_3		*MT_3 + RW_X*		*MT_3 + RW_Y*		*MT_3 + RW_Z*		*MT_3 + RW_X + RW_Y*	*MT_3 + RW_X + RW_Z*	*MT_3 + RW_Y + RW_Z*	*MT_X + RW_3*		*MT_Y + RW_3*		*MT_Z + RW_3*	
	Wf	Mark	Mark*Wf	Mark	Mark*Wf	Mark	Mark*Wf	Mark	Mark*Wf	Mark	Mark*Wf	Mark*Wf	Mark*Wf	Mark	Mark*Wf	Mark	Mark*Wf	Mark	Mark*Wf
Mass	0,116	9	0,964	7	0,750	7	0,750	7	0,750	5	0,536	5	0,536	5	0,536	3	0,321	3	0,321	3	0,321
Reliability	0,161	6	0,964	5	0,804	5	0,804	5	0,804	4	0,643	4	0,643	4	0,643	4	0,643	4	0,643	4	0,643
Size	0,134	7	0,938	5	0,670	5	0,670	5	0,670	3	0,402	3	0,402	3	0,402	1	0,134	1	0,134	1	0,134
Power consumption	0,152	4	0,571	3	0,429	3	0,429	4	0,571	4	0,571	3	0,429	3	0,429	2	0,286	2	0,286	2	0,286
Cost	0,170	7	1,188	5	0,848	5	0,848	5	0,848	3	0,509	3	0,509	3	0,509	2	0,339	2	0,339	2	0,339
Accuracy	0,152	0	0,000	0	0,000	0	0,000	6	1,018	6	1,018	0	0,000	0	0,000	0	0,000	4	0,679	5	0,848
Stabilization time	0,116	0	0,000	0	0,000	0	0,000	7	0,813	6	0,696	0	0,000	0	0,000	0	0,000	2	0,232	6	0,696
Score			4,625		3,500		3,500		5,473		4,375		2,518		2,518		1,723		2,634		3,268

Figure 3.8 – Trade-off decision tool.

Chapter 4

Development of the model, simulations and results

The model examined until now can be considered a good model but since then there is the need to test it with the Hardware In the Loop simulation technique, it is necessary to replace some physical parts of the satellite with the simulated ones increasing the complexity of the model but thus improving the results of simulations. In this chapter the model described in the previous chapter will be improved with the aim to better simulate the real physical system behavior without using it. In order to do this, the model is firstly linearized (because Kalman filter and LQR controller work on linear model), then modified adding simply white noise blocks and more accurate models of the physical parts.

4.1 Linearization of the mathematical model

The mathematical model of the system has to be linearized because linear controller techniques has been selected for the attitude control system. The linearization point is selected as given in following equation:

$$\bar{q} = \begin{bmatrix} q_0 \\ \vec{q} \end{bmatrix} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.1}$$

4.1.1 Kinematics

In Equation 3.7, the kinematic model of the satellite is given, so ,applying the conditions of linearization points (Equation 4.1), it becomes:

$$\dot{\bar{q}} = \frac{1}{2}\begin{bmatrix} -(q_1+q_2+q_3) \\ q_0\begin{bmatrix}1 & & \\ & 1 & \\ & & 1\end{bmatrix} + \begin{bmatrix}0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0\end{bmatrix} \end{bmatrix} \bar{\omega}_{OB}^{B} \approx \begin{bmatrix} 0 \\ \frac{1}{2}\bar{\omega}_{OB}^{B} \end{bmatrix} \tag{4.2}$$

It is easy to see from Equation 4.2 that $\bar{\omega}_{OB}^{B} = 2\dot{\bar{q}}$

4.1.2 Rotation matrix

If the rotation matrix between body and orbit frame given in Equation 2.19 is linearized around point given in Equation 4.1, it becomes:

$$\boldsymbol{R}_{O}^{B} \approx 2\begin{bmatrix} \frac{1}{2} & q_3 & -q_2 \\ -q_3 & \frac{1}{2} & q_1 \\ q_2 & -q_1 & \frac{1}{2} \end{bmatrix} \tag{4.3}$$

4.1.3 Angular velocity

By applying Equation 4.3 and Equation 4.2 into Equation 3.4, linearized model of $\bar{\omega}_{IB}^{B}$ is derived as:

$$\bar{\omega}_{IB}^{B} = \begin{bmatrix}\omega_x \\ \omega_y \\ \omega_z\end{bmatrix} \approx \begin{bmatrix}2\dot{q}_1 + 2q_3\omega_o \\ 2\dot{q}_2 + \omega_o \\ 2\dot{q}_3 - 2q_1\omega_o\end{bmatrix} \tag{4.4}$$

The time derivative of $\bar{\omega}_{IB}^{B}$ is hence obtained as:

$$\dot{\bar{\omega}}_{IB}^{B} = \begin{bmatrix}\dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z\end{bmatrix} \approx \begin{bmatrix}2\ddot{q}_1 + 2\dot{q}_3\omega_o \\ 2\ddot{q}_2 \\ 2\ddot{q}_3 - 2\dot{q}_1\omega_o\end{bmatrix} \tag{4.5}$$

The Equations 4.4 and 4.5 will be used to derive the linearized dynamics equation.

4.1.4 Gravitational torque

The gravitational torque is written in Equation 3.9, using Equation 2.20 it can be linearized as:

$$\bar{T}_{g}^{B} = 3\omega_o\hat{c}_3^{B} \times (\mathbf{I}\hat{c}_3^{B}) \approx 3\omega_o\begin{bmatrix}2(I_z + I_{rw} - I_y)q_1 \\ 2(I_x - I_z - I_{rw})q_2 \\ 0\end{bmatrix} \tag{4.6}$$

4.1.5 Magnetic torquer

The torque produced by a magnetic torquer is given in Equation 3.13 but it can also be expressed as shown below by using skew-symmetric matrix formulation:

$$\bar{T}_m^B = \bar{m}^B \times \bar{B}^B = \boldsymbol{S}(\bar{m}^B)\bar{B}^B = \boldsymbol{S}(\bar{m}^B)\boldsymbol{R}_O^B\bar{B}^O \approx \boldsymbol{S}(\bar{m}^B)\bar{B}^O = \begin{bmatrix} B_z^O m_y - B_y^O m_z \\ B_x^O m_z - B_z^O m_x \\ B_y^O m_x - B_x^O m_y \end{bmatrix} \tag{4.7}$$

4.1.6 Reaction wheel

Reaction wheels dynamic equation is linearized around the point where $\bar{\omega}_{IB}^B$ is almost equal to 0, so considering that *3STAR* has only a reaction wheel on Z axis the Equation 3.15 becomes:

$$\begin{aligned} \bar{T}_{rw}^B &= \dot{\bar{h}}_{rw}^B + \bar{\omega}_{IB}^B \times \bar{h}_{rw}^B - \bar{T}_{friction}^B = \begin{bmatrix} T_{rw-x} \\ T_{rw-y} \\ T_{rw-z} \end{bmatrix} = \\ &= \begin{bmatrix} 0 \\ 0 \\ \dot{h}_{rw-z} + h_{rw-y}\omega_x - h_{rw-x}\omega_y \end{bmatrix} \approx \begin{bmatrix} 0 \\ 0 \\ \dot{h}_{rw-z} \end{bmatrix} \end{aligned} \tag{4.8}$$

4.1.7 Complete model

Starting from the Equation 3.2 and replacing all the linearized equations, it is possible to obtain the complete linearized model; the system can be expressed by state-space representation in linear form given by following equation:

$$\dot{\bar{x}}(t) = \mathbf{A}\bar{x}(t) + \mathbf{B}(t)\bar{u}(t) \tag{4.9}$$

where $\bar{x}$ is the state vector (defined as $\bar{x} = \begin{bmatrix} q_1 & q_2 & q_3 & \dot{q}_1 & \dot{q}_2 & \dot{q}_3 \end{bmatrix}^T$), $\bar{u}$ the input vector (defined as $\bar{u} = \begin{bmatrix} m_x & m_y & m_z & T_{rw} \end{bmatrix}^T$), then $\mathbf{A}$ and $\mathbf{B}$ matrices can be written as in following equations:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -4\sigma_x\omega_o^2 & 0 & 0 & 0 & 0 & (1-\sigma_x)\omega_o \\ 0 & -3\sigma_y\omega_o^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\sigma_z\omega_o^2 & -(1-\sigma_z)\omega_o & 0 & 0 \end{bmatrix} \tag{4.10}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{B_z^O}{2I_x} & -\frac{B_y^O}{2I_x} & 0 \\ -\frac{B_z^O}{2I_y} & 0 & \frac{B_x^O}{2I_y} & 0 \\ \frac{B_y^O}{2(I_z+I_{rw})} & -\frac{B_x^O}{2(I_z+I_{rw})} & 0 & \frac{1}{2(I_z+I_{rw})} \end{bmatrix} \tag{4.11}$$

where:

$$\sigma_x = \frac{I_y - (I_z + I_{rw})}{I_x} \quad \sigma_y = \frac{I_z + I_{rw} - I_x}{I_y} \quad \sigma_z = \frac{I_x - I_y}{I_z + I_{rw}} \tag{4.12}$$

4.2 Magnetometer

Magnetometers are simple, reliable, lightweight sensors that measure both the direction and size of the Earth's magnetic field in the proximity of the instrument. When compared to the Earth's known field, their output helps us establish the spacecraft's attitude. But their accuracy is not very high when compared to other types of sensors, in fact it is of the order of 0.5-3 deg [6].

The first improvement of the system introduced is the magnetometer, which has been implemented firstly using white noise and then using a specific model (Figure 4.1). The magnetometer gives the values of the components of the magnetic field in Body oordinates and its readings are affected of a noise of about 0.5 mGauss, as reported in the Magnetometer datasheet. Said $\bar{H}$ the real magnetic field vector, the measured

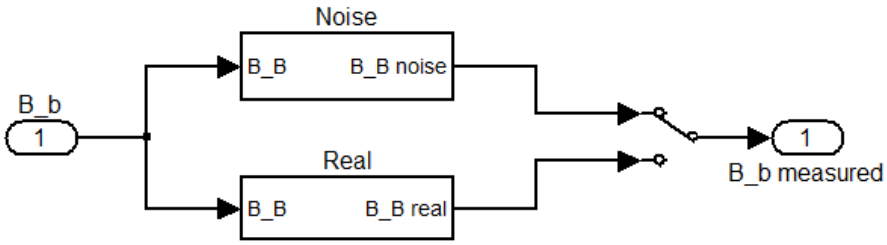


Figure 4.1 – Magnetometer implemented.

one, called $\bar{B}$, can be obtained with a model [15] based on the equation:

$$\bar{B} = \mathbf{KTC}\bar{H} + \bar{b} + \bar{v} \tag{4.13}$$

where $\mathbf{K}$ is the scale factor matrix, $\mathbf{T}$ the magnet sensor sensitivity rotation matrix, $\mathbf{C}$ the assembly error matrix, $\bar{b}$ the biases vector and $\bar{v}$ the noises vector.

The scale factor matrix is a diagonal matrix containing the gain values, in this case it is:

$$\mathbf{K} = \begin{bmatrix} 9715 & 0 & 0 \\ 0 & 8584 & 0 \\ 0 & 0 & 7392 \end{bmatrix} \tag{4.14}$$

The magnet sensor sensitivity rotation matrix represent the misalignment of the sensor due to the manufacturing phases and it is:

$$\mathbf{T} = \begin{bmatrix} 1.000 & -0.007 & -0.006 \\ -0.060 & 0.998 & -0.009 \\ -0.051 & 0.010 & 0.999 \end{bmatrix} \tag{4.15}$$

Like to the previous one, the assembly error matrix includes the misalignment of the sensor, but in this case the cause is the assembly of the sensor on the satellite. This matrix can be written in the general case as:

$$\mathbf{C} = \begin{bmatrix} c(\alpha)c(\gamma) - c(\beta)s(\alpha)s(\gamma) & c(\gamma)s(\alpha) + c(\alpha)c(\beta)s(\gamma) & s(\beta)s(\gamma) \\ -c(\beta)c(\gamma)s(\alpha) - c(\alpha)s(\gamma) & c(\alpha)c(\beta)c(\gamma) - s(\alpha)s(\gamma) & c(\gamma)s(\beta) \\ s(\alpha)s(\beta) & -c(\alpha)s(\beta) & c(\beta) \end{bmatrix} \tag{4.16}$$

with c and s compact notations for *cos* and *sin*.

The biases vector contains the values that the sensor considers as zero, in this case it is equal to:

$$\bar{b} = \begin{bmatrix} 32400 \\ 32568 \\ 32698 \end{bmatrix} \tag{4.17}$$

The noises vector contains noises affecting sensor during measurements and it can be written as:

$$\bar{v} = 2 \begin{bmatrix} 0.5 - rand \\ 0.5 - rand \\ 0.5 - rand \end{bmatrix} \frac{\left\|\bar{H}\right\|}{100} \tag{4.18}$$

where it is generated a three element vector each of which has zero mean and is between $-\frac{\left\|\bar{H}\right\|}{100}$ and $\frac{\left\|\bar{H}\right\|}{100}$, that is a noise of 1% of the signal intensity.

Summing all the disturbances, the measured magnetic vector is obtained; in fact the sensor has a built-in software that eliminates known misalignments and filter the output so that the effective output has a greater accuracy then the measured one (see Figure 4.2) and also because there is no need to use additional filters. This can be represented, in a simplified form, with the equation:

$$\bar{B}_{out} = (\mathbf{KTC})^{-1}(\bar{B} - \bar{b}) \tag{4.19}$$

In Figure 4.2 it is shown the evolution of the magnetic field calculated and measured with the two different magnetometers implemented: the difference between calculated and measured with real magnetometer is very small, that is about $o(10^{-11})$

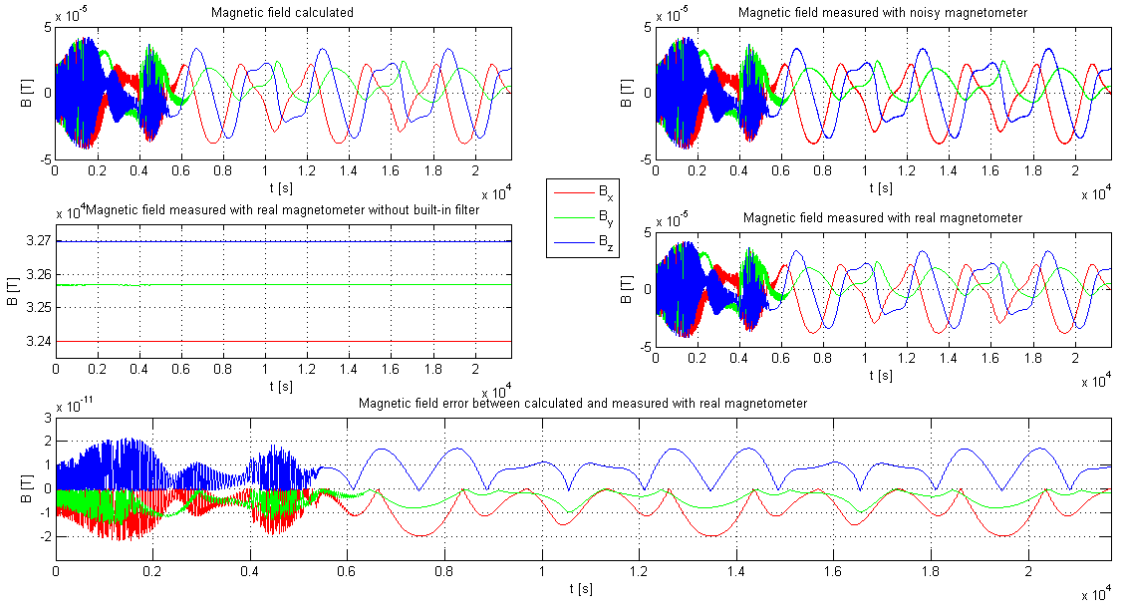


Figure 4.2 – Results of different magnetometers.

4.3 IMU

IMU stands for *Inertial Measurement Unit*, that is a system for the direct measurement of accelerations and angular velocities, with respect to three coordinate axes by means of accelerometers and gyroscopes, and for the calculation of velocity and position through the integration of the quantities measured by a processor. Gyroscopes are inertial sensors which measure the speed or angle of rotation from an initial reference, but without any knowledge of an external, absolute reference. They are used on spacecraft for precision attitude sensing, in fact their accuracy is very high, about 0.01 deg/hr. Manufacturers use a variety of physical phenomena, from simple spinning wheels (iron gyros using ball or gas bearings) to ring lasers, hemispherical resonating surfaces and laser fibre optic bundles [6].

As seen previously for the magnetometer, also for the IMU has been made a first version adding Gaussian noise (in this case the noise is of 0.05 deg/sec [17]) and then a suitable model has been implemented [16]; this model represent the gyroscopes of an IMU considering these disturbances:

gravity gradient: its effect is due to different forces on the sensor than the calibration ones and it can be considered using this formulation:

$$\bar{f}_g = \bar{f} + \mathbf{T}\bar{g} - \mathbf{T}_0\bar{g}_0 \tag{4.20}$$

where $\bar{f}$ are the forces applied on the IMU, $\mathbf{T}$ the rotation matrix from the

IMU to the external reference frame, $\bar{g}$ the local gravitational force, $\mathbf{T}_0$ the calibration rotation matrix and $\bar{g}_0$ the calibration gravitational forces;

misalignment: this effect is due to misalignment between the sensor and the IMU reference frame, causing that the specific force components are not aligned to each input axis. It can be expressed as:

$$\bar{f}_m = \mathbf{T}_m \bar{f} \tag{4.21}$$

where $\mathbf{T}_m$ is the gyroscopes misalignment matrix. It can be written as:

$$\mathbf{T}_m = \begin{bmatrix} c(\beta_{xy})c(\beta_{xz}) & -c(\beta_{yx})s(\beta_{yz}) & c(\beta_{zx})s(\beta_{zy}) \\ c(\beta_{xy})s(\beta_{xz}) & c(\beta_{yx})c(\beta_{yz}) & -s(\beta_{zx}) \\ -s(\beta_{xy}) & s(\beta_{yx}) & c(\beta_{zx})c(\beta_{zy}) \end{bmatrix} \tag{4.22}$$

with c and s compact notations for *cos* and *sin*, moreover $\beta_{(..)}$ are the six components of the misalignment vector that can be obtained using this equation:

$$\bar{\beta} = \bar{\beta}_n + \delta_d \bar{\beta}_n + \delta_r \bar{\beta}_n \tag{4.23}$$

where $\bar{\beta}_n$ is the base misalignments vector, δ_d the day-to-day stability parameter and δ_r the in-run stability parameter and they can be obtained as:

$$\delta_{(.)} \bar{\beta}_n = \sigma_{(.)} \bar{\beta}_n rand \tag{4.24}$$

with $rand$ a random number between 0 and 1 and $\sigma_{(.)}$ are standard deviations (function of the nominal values β and given parameters for stability day-to-day β_d and in-run β_r);

cross coupling: this effect is that the value measured on an axis is influenced by the values of the others and it can be represented using the following matrix:

$$\mathbf{K}_{xc} = \begin{bmatrix} 0 & xc_{xy} & xc_{xz} \\ xc_{yx} & 0 & xc_{yz} \\ xc_{zx} & xc_{zy} & 0 \end{bmatrix} \tag{4.25}$$

where $xc_{(..)}$ are the cross coupling coefficients that are obtained, similarly as in Equation 4.23, as:

$$\bar{xc} = \bar{xc}_n + \delta_d \bar{xc}_n + \delta_r \bar{xc}_n \tag{4.26}$$

where $\bar{xc}_n$ is the nominal cross coupling vector and the day-to-day stability parameter (δ_d) and the in-run stability parameter (δ_r) can be obtained as in Equation 4.24:

$$\delta_{(.)} \bar{xc}_n = \sigma_{(.)} \bar{xc}_n rand \tag{4.27}$$

g-sensitivity: it is due to the linear influence of the actuating force on the input of the gyros and it can be accounted for implementing the rotation given by the matrix:

$$\mathbf{K}_{gs} = \mathbf{gs}_n + \mathbf{gs}_d + \mathbf{gs}_r \tag{4.28}$$

where:

$$\mathbf{gs}_n = \begin{bmatrix} gs_{xx} & gs_{xy} & gs_{xz} \\ gs_{yx} & gs_{yy} & gs_{yz} \\ gs_{zx} & gs_{zy} & gs_{zz} \end{bmatrix} \tag{4.29}$$

$$\mathbf{gs}_d = \delta_d \mathbf{gs}_n rand \tag{4.30}$$

$$\mathbf{gs}_r = \delta_r \mathbf{gs}_n rand \tag{4.31}$$

g2-sensitivity: this effect is due to the square influence of the actuating force on the input of the gyros and it can be implemented computing the rotation matrices:

$$\mathbf{Kx}_{g2s} = \begin{bmatrix} g2sx_{xx} & g2sx_{xy} & g2sx_{xz} \\ 0 & g2sx_{yy} & g2sx_{yz} \\ 0 & 0 & g2sx_{zz} \end{bmatrix} \tag{4.32}$$

$$\mathbf{Ky}_{g2s} = \begin{bmatrix} g2sy_{xx} & g2sy_{xy} & g2sy_{xz} \\ 0 & g2sy_{yy} & g2sy_{yz} \\ 0 & 0 & g2sy_{zz} \end{bmatrix} \tag{4.33}$$

$$\mathbf{Kz}_{g2s} = \begin{bmatrix} g2sz_{xx} & g2sz_{xy} & g2sz_{xz} \\ 0 & g2sz_{yy} & g2sz_{yz} \\ 0 & 0 & g2sz_{zz} \end{bmatrix} \tag{4.34}$$

where:

$$g2s(\cdot)_{(\cdot\cdot)} = g2s(\cdot)_{(\cdot\cdot)n} + g2s(\cdot)_{(\cdot\cdot)d} + g2s(\cdot)_{(\cdot\cdot)r} \tag{4.35}$$

in which $g2s(\cdot)_{(\cdot\cdot)d}$ and $g2s(\cdot)_{(\cdot\cdot)r}$ are the day-to-day and in-run stability parameters and they are calculated as:

$$g2s(\cdot)_{(\cdot\cdot)d} = \delta_d g2s(\cdot)_{(\cdot\cdot)n} rand \tag{4.36}$$

$$g2s(\cdot)_{(\cdot\cdot)r} = \delta_r g2s(\cdot)_{(\cdot\cdot)n} rand \tag{4.37}$$

So, the measured angular velocity can be written as:

$$\bar{\omega}_{out} = \mathbf{T}_m \bar{\omega} + \mathbf{K}_{xc}\mathbf{T}_m \bar{\omega} + \mathbf{K}_{gs}\mathbf{T}_m \bar{f}_g + \begin{bmatrix} (\mathbf{T}_m \bar{f}_g)^T \mathbf{Kx}_{g2s} (\mathbf{T}_m \bar{f}_g) \\ (\mathbf{T}_m \bar{f}_g)^T \mathbf{Ky}_{g2s} (\mathbf{T}_m \bar{f}_g) \\ (\mathbf{T}_m \bar{f}_g)^T \mathbf{Kz}_{g2s} (\mathbf{T}_m \bar{f}_g) \end{bmatrix} \tag{4.38}$$

The results obtained using parameters of IMU that probably will be used for *3STAR* are shown in the Figure 4.3 and Figure 4.4: it is very interesting as the error is highest in the first phase (that is detumbling phase), where it reaches a peak of $-0.02 rad/s$ while, going towards stabilization, the error decreases so much until it becomes a static error.

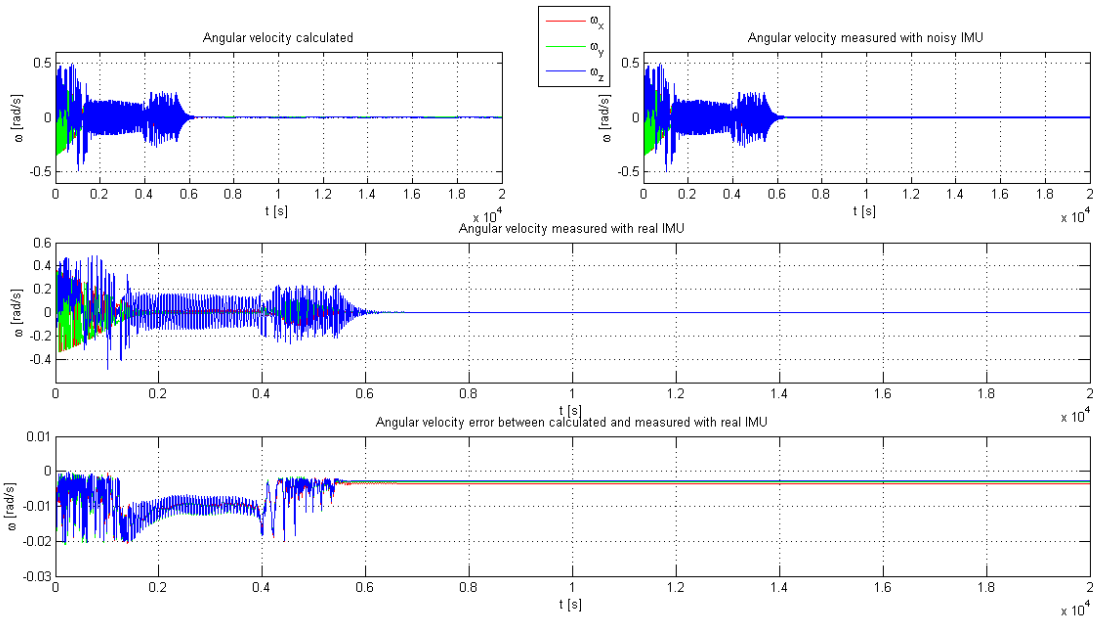


Figure 4.3 – Angular velocity measurements obtained with different models of IMU.

4.4 Kalman filter

The Kalman filter, also known as linear quadratic estimator (LQE), is an algorithm which uses a series of measurements observed over time, containing noise (random variations) and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those that would be based on a single measurement alone. More formally, the Kalman filter operates recursively on streams of noisy input data to produce a statistically optimal estimate of the underlying system state: thanks to its properties, it is an optimal filter for Gaussian errors with zero mean acting on the system, in fact the main assumption of the Kalman filter is that the underlying system is a linear dynamical system and that all error terms and measurements have a Gaussian distribution.

The algorithm works in a two-step process (see Figure 4.5): in the prediction step, the Kalman filter produces estimates of the current state variables, along with

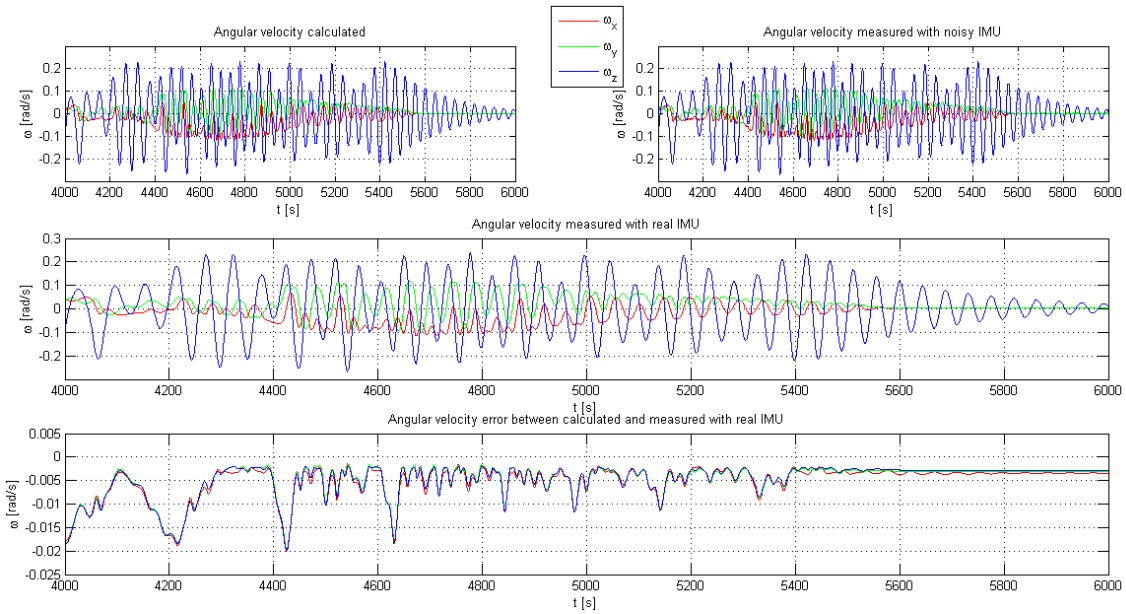


Figure 4.4 – Particular of angular velocity measurements.

their uncertainties. Once the outcome of the next measurement (obviously corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty (in fact the weights are calculated from the covariance, a measure of the estimated uncertainty of the prediction of the system's state). The result of the weighted average is a new state estimate which is between the predicted and measured state, and has a better estimated uncertainty than either alone. Because of the algorithm's recursive nature, it can run in real time using only the present input measurements and the previously calculated state: the entire history of a system's state is not required.

Because the certainty of the measurements is often difficult to measure precisely, it is common to discuss the filter's behavior in terms of gain. The Kalman gain is a function of the relative certainty of the measurements and current state estimate, and it can be "tuned" to achieve particular performance. With a high gain, the filter places more weight on the measurements, and thus follows them more closely. With a low gain, the filter follows the model predictions more closely, smoothing out noise but decreasing the responsiveness. At the extremes, a gain of one causes the filter to ignore the state estimate entirely, while a gain of zero causes the measurements to be ignored. When performing the actual calculations for the filter, the state estimate and covariances are coded into matrices to handle the multiple dimensions involved in a single set of calculations. This allows for representation of linear relationships

between different state variables in any of the transition models or covariances.

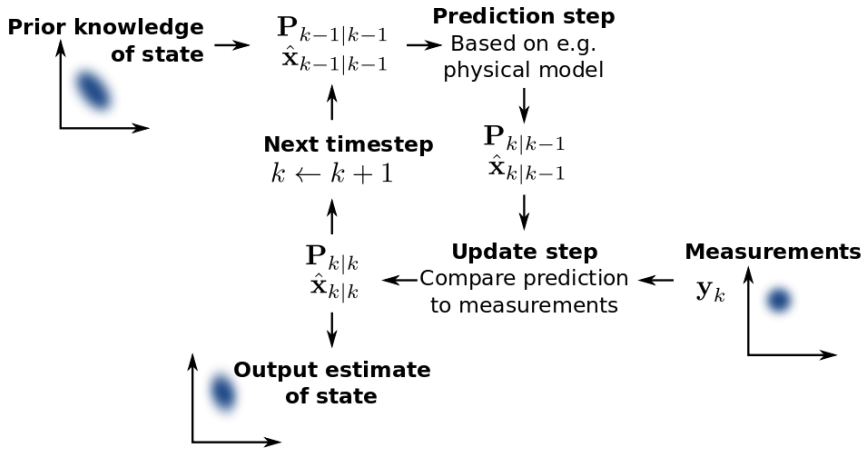


Figure 4.5 – Schematic operation of a Kalman filter.

4.4.1 Generic dynamic system model

In order to use the Kalman filter to estimate the internal state of a process given only a sequence of noisy observations, the process must be modeled in accordance with the framework of the Kalman filter. This means specifying the following matrices (for each time-step, k):

$\mathbf{F}_k$: the state-transition model;

$\mathbf{H}_k$: the observation model;

$\mathbf{Q}_k$: the covariance of the process noise;

$\mathbf{R}_k$: the covariance of the observation noise;

$\mathbf{B}_k$: the control-input model.

The Kalman filter model assumes the true state at time k is evolved from the state at *(k - 1)* according to:

$$\bar{x}_k = \mathbf{F}_k \bar{x}_{k-1} + \mathbf{B}_k \bar{u}_k + \bar{w}_k \tag{4.39}$$

where $\bar{x}_k$ is the state vector, $\bar{u}_k$ is the control vector and $\bar{w}_k$ is the process noise vector, which is assumed to be drawn from a zero mean multivariate normal distribution with covariance $\mathbf{Q}_k$. At time k a measurement $\bar{z}_k$ of the true state $\bar{x}_k$ is made according to:

$$\bar{z}_k = \mathbf{H}_k \bar{x}_k + \bar{v}_k \tag{4.40}$$

where $\mathbf{H}_k$ is the observation model which maps the true state space into the observed space and $\bar{v}_k$ is the observation noise which is assumed to be zero mean Gaussian white noise with covariance $\mathbf{R}_k$. The initial state and the noise vectors at each step are all assumed to be mutually independent.

4.4.2 Equations

The Kalman filter can be written as a single equation, however it is most often conceptualized as two distinct phases, "predict" and "update":

predict: the predict phase uses the state estimate from the previous timestep to produce an estimate of the state at the current timestep. This predicted state estimate is also known as the *a priori* state estimate because, although it is an estimate of the state at the current timestep, it does not include observation information from the current timestep;

update: in the update phase, the current *a priori* prediction is combined with current observation information to refine the state estimate. This improved estimate is termed the *a posteriori* state estimate.

Typically, the two phases alternate, with the prediction advancing the state until the next scheduled observation, and the update incorporating the observation. However, this is not necessary: if an observation is unavailable for some reason, the update may be skipped and multiple prediction steps performed. Likewise, if multiple independent observations are available at the same time, multiple update steps may be performed.

The state of the filter is represented by two variables:

$\hat{\bar{x}}_{k|k}$**:** the *a posteriori* state estimate at time k given observations up to and including at time k;

$\mathbf{P}_{k|k}$**:** the *a posteriori* error covariance matrix, that is a measure of the estimated accuracy of the state estimate.

4.4.2.1 Predict

Predicted (*a priori*) state estimate:

$$\hat{\bar{x}}_{k|k-1} = \mathbf{F}_k \hat{\bar{x}}_{k-1|k-1} + \mathbf{B}_k \bar{u}_k \tag{4.41}$$

with $\hat{\bar{x}}_{k-1|k-1}$ the *a posteriori* state estimation at time *k-1* given the measurements at time *k-1*. Predicted (*a priori*) estimate covariance:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \tag{4.42}$$

where $\mathbf{P}_{k-1|k-1}$ is the *a priori* estimate covariance at time *k-1*.

4.4.2.2 Update

Innovation or measurement residual:

$$\tilde{\tilde{y}}_k = \bar{z}_k - \mathbf{H}_k \hat{\hat{x}}_{k|k-1} \tag{4.43}$$

Innovation (or residual) covariance:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k \tag{4.44}$$

Optimal Kalman gain:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1} \tag{4.45}$$

Updated (*a posteriori*) state estimate:

$$\hat{\hat{x}}_{k|k} = \hat{\hat{x}}_{k|k-1} + \mathbf{K}_k \tilde{\tilde{y}}_k \tag{4.46}$$

Updated (*a posteriori*) estimate covariance:

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \tag{4.47}$$

4.4.3 Results

The Kalman filter has been used in the model using the *kalman* function of MATLAB®, which requires special input for the operation [14]:

```
[KEST,L,P] = kalman(SYS,QN,RN,NN)
```

where `KEST` is a Kalman estimator, `L` is the Kalman gain, `P` the solution of corresponding algebraic Riccati equation, `SYS` the state-space model (it has been used the model described in Section 4.1.7) and `QN`, `RN` and `NN`, are the covariance matrices. Using the parameters of *3STAR* ADCS, the results obtained are displayed in Figure 4.6.

4.5 Controls

To obtain the stabilization of the satellite, it was decided to divide the control in two different parts: the first is intended to reduce the angular velocity of the satellite after the release by the launcher, also called *detumbling* while the second to bring the satellite in the correct attitude, that is the *stabilization*. The transition from first to second stage is managed by a suitable control logic based on the angular velocity of the satellite.

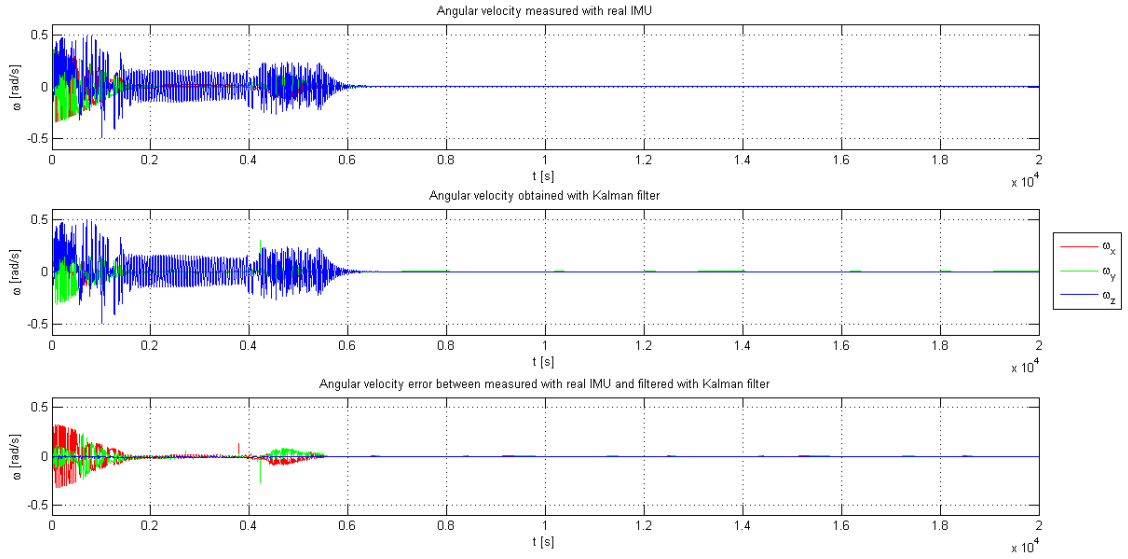


Figure 4.6 – Results of Kalman filter.

4.5.1 Detumbling

This stage, immediately after the release from the launcher, is characterized by a more or less high angular velocity depending on the conditions of release: the objective therefore is to reduce it relatively fast (and compatibly with the other subsystems of the satellite, for example EPS, since it can not surely be a high demand of electrical power). The controller used is PD (it uses a different proportional and derivative gain for each axis) because of its high robustness and it is shown in Figure 4.7.

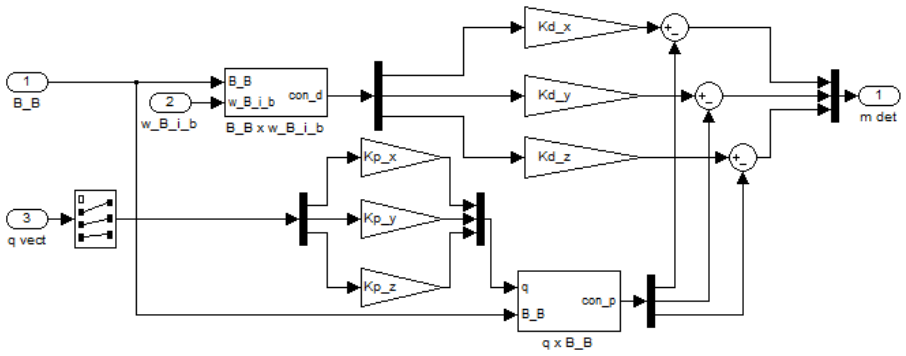


Figure 4.7 – Simulink model for the detumbling phase controller.

The controls are calculated from the local magnetic field expressed in body coordinates, the angular velocity and the angular position written in quaternions;

the equation of controller is:

$$\bar{m}_{det} = \bar{K}_p[\bar{B}_B \times (\vec{q} - \vec{q}_{ref})] + \bar{K}_d[\bar{B}_B \times (\bar{\omega}^B_{IB} - \bar{\omega}^B_{IBref})] \tag{4.48}$$

It has been decided to consider the angular position $\bar{q} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$ as the target position, so the Equation 4.48 becomes:

$$\bar{m}_{det} = \bar{K}_p[\bar{B}_B \times \vec{q}] + \bar{K}_d[\bar{B}_B \times \bar{\omega}^B_{IB}] \tag{4.49}$$

Considering an initial (that is when *3STAR* will be released from launcher vehicle) angular velocity of $\omega^B_{IB} = \begin{bmatrix} 0.25 & 0.25 & 0.25 \end{bmatrix}^T$, the detumbling phase ends after just over 3900 seconds, as shown in Figure 4.8, where on the second plot there is a parameter displaying the passage from detumbling to stabilization phase. This represents quite a good result because, under these conditions, the detumbling phase would be completed in a short time.

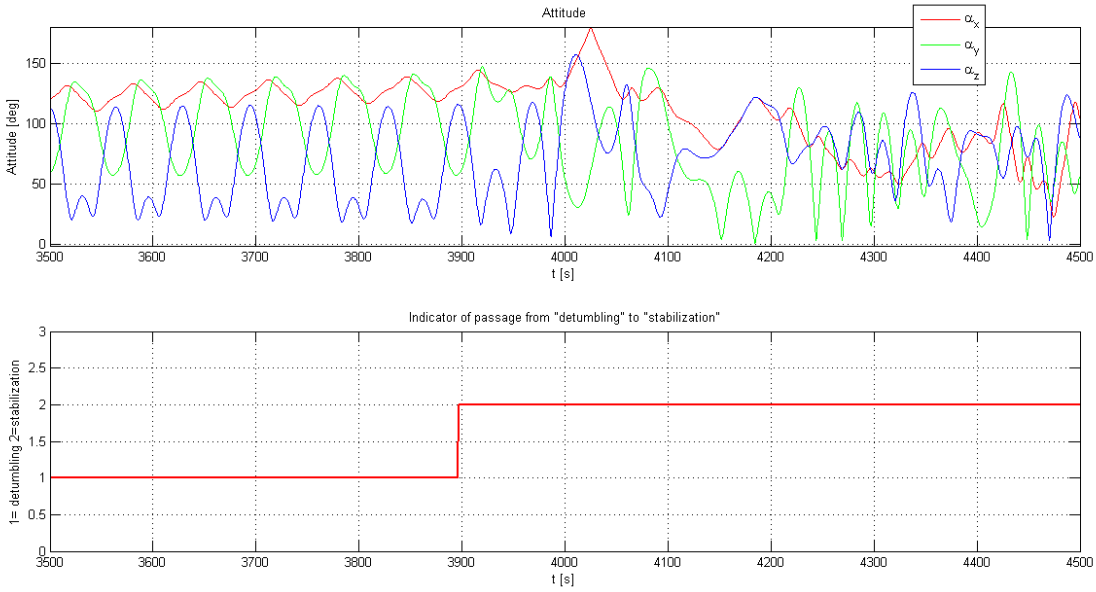


Figure 4.8 – Passage from *detumbling* to *stabilization* phase.

4.5.2 Stabilization

For this phase it has been decided to use a type of control different from the previous, in fact it has been used the Linear-Quadratic Regulator, which combined with a Linear-Quadratic Estimator (that is Kalman filter) takes the name of Linear-Quadratic-Gaussian control or Optimal control (see Section 2.5.2). The controller

has been implemented in the model using the *lqr* function of MATLAB®, which requires, as seen previously for *kalman* function, special input for the operation [14]:

```
[K,S,E] = lqr(SYS,Q,R,N)
```

where `K` is the optimal gain matrix, `S` the solution of the associated algebraic Riccati equation, `E` the closed-loop eigenvalues, `SYS` the state-space model (also for this function it has been used the model described in Section 4.1.7) and `Q`, `R` and `N` (sometimes omitted), are the matrices present in the cost function (Equation 2.27). As for Kalman filter, the determination of matrix elements has been relatively complex and it required many simulations (more than one hundred) to achieve a fine tuning and so reaching a satisfactory result.

4.5.3 Selector

The choice of which controller to use is made using the angular velocity of the satellite as shown in Figure 4.9, in which the following equation is implemented:

$$selector = 1+((\left|\omega^B_{IB-x}\right| < 0.001)and(\left|\omega^B_{IB-y}\right| < 0.001)and(\left|\omega^B_{IB-z}\right| < 0.001)) \quad (4.50)$$

where $<$ and *and* have to be considered as logical operators so that they give 1 if all the three absolute values of the velocity vector are less than the threshold. In this way, the value of selector will be 1 if at least one of the three velocities is greater than 0.001 and otherwise 2. The hysteresis block *relay* has been implemented in order not to permit the back pass from the stabilization to the detumbling controller: this trick is very useful during test because mainly in the early stages of development of the system often it is received unwanted and not real behavior, but it is not necessary in real application because it is illogic that the satellite once stabilized will return to conditions similar to whose after the launcher separation.

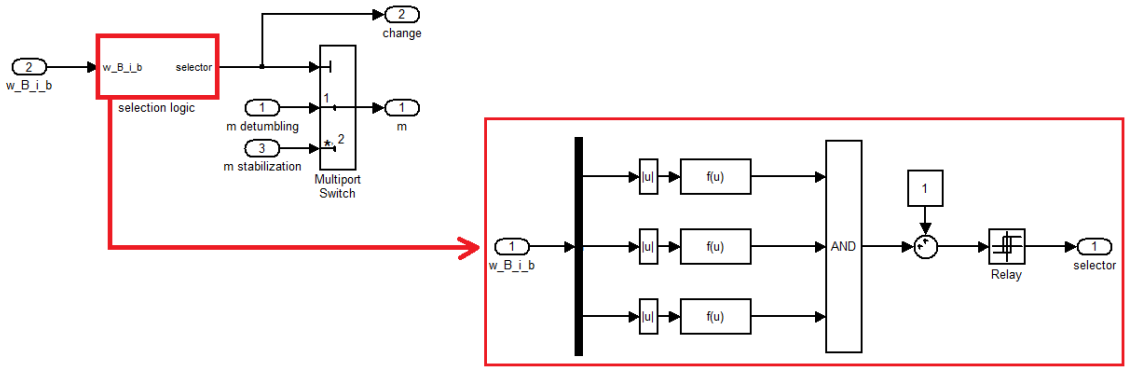


Figure 4.9 – Simulink model for the selector.

Combining two models of the controllers with selector, the complete control logic is obtained, as shown in Figure 4.10. The magnetic torque is followed by a saturation block in order to limit the control intensity into the maximum range available, so to ensure the respect of the physical limits of the system.

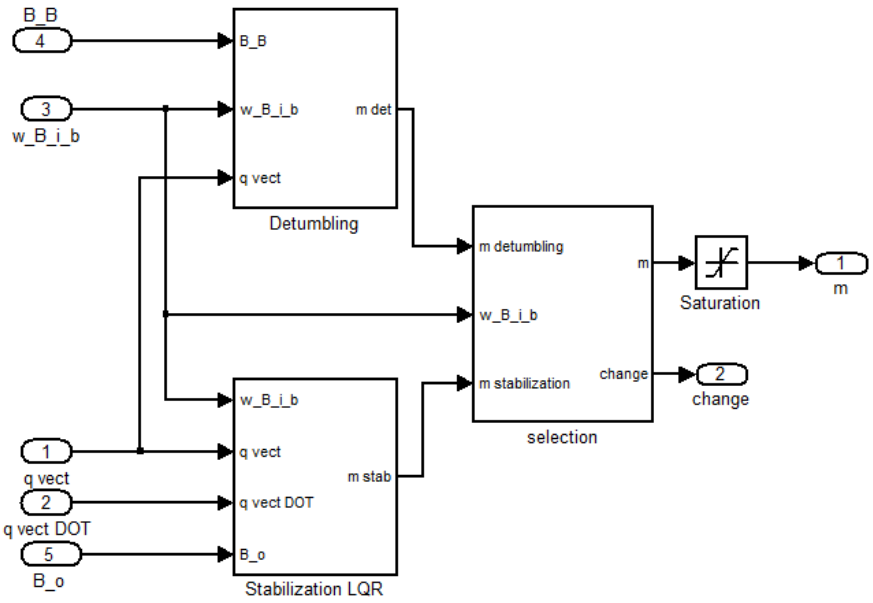


Figure 4.10 – Simulink model for the complete control logic.

4.6 Complete model

By joining the models described so far with the model of the orbit and of the magnetic field (Figure 4.11), the complete model of the system is obtained as shown in Figure 4.12.

The results are shown in Figure 4.13, where it is possible to see how the attitude, after detumbling phase, reaches desired values in a short time, as shown in Figure 4.14 relatively to quaternions and in Figure 4.15 for angular velocities (detumbling phase is completed in about $3900s$ while stabilization is completed in about $8000s$). In Figure 4.17 there is the dipole moment required to magnetic torquers while in Figure 4.18 there are the power consumptions of them: it is possible to note that they are heavily used in the phase of detumbling and until stabilization is reached while reaction wheel (Figure 4.16) is used only in detumbling.

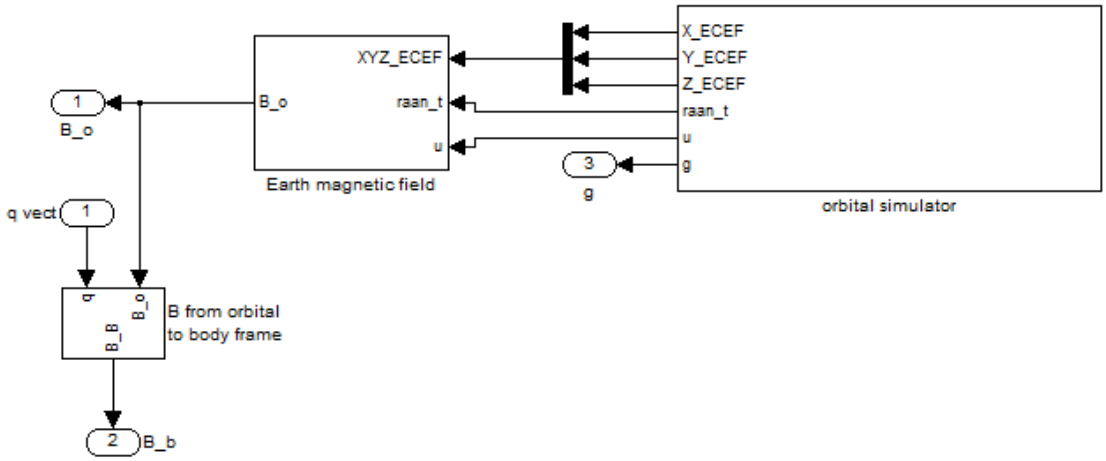


Figure 4.11 – Simulink model for the orbit and the magnetic field.

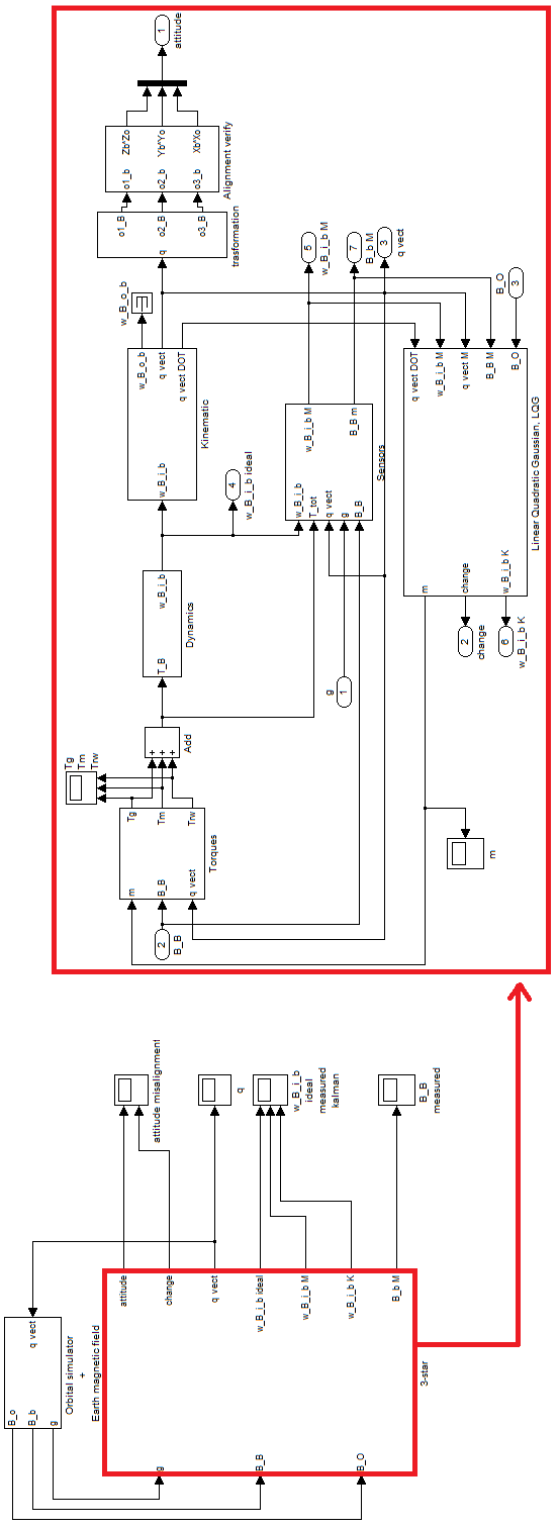


Figure 4.12 – Complete Simulink model for the ADCS of *3STAR* .

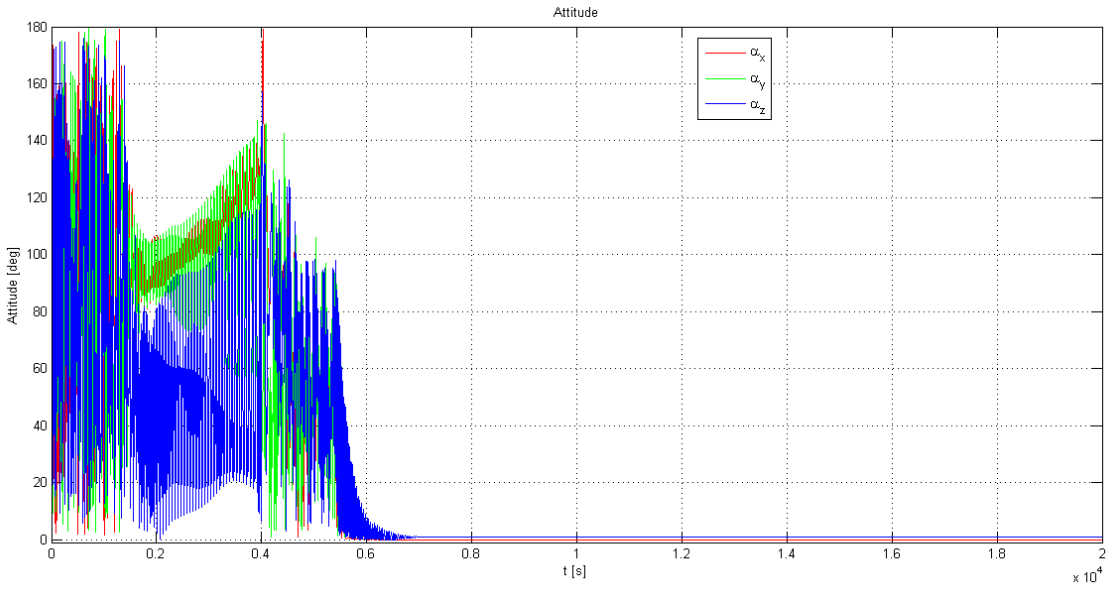


Figure 4.13 – *3STAR* attitude

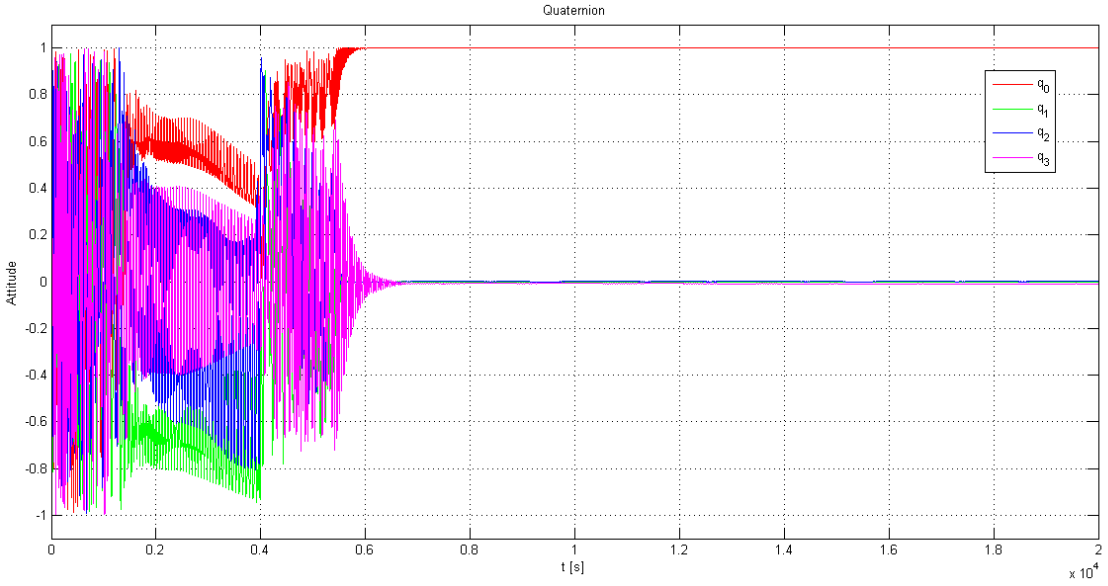


Figure 4.14 – Quaternions.

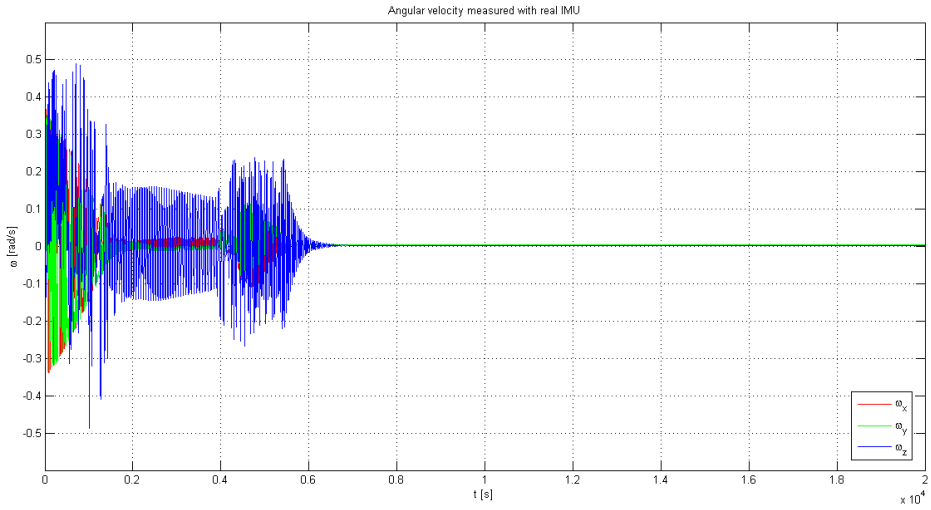


Figure 4.15 – Angular velocity.

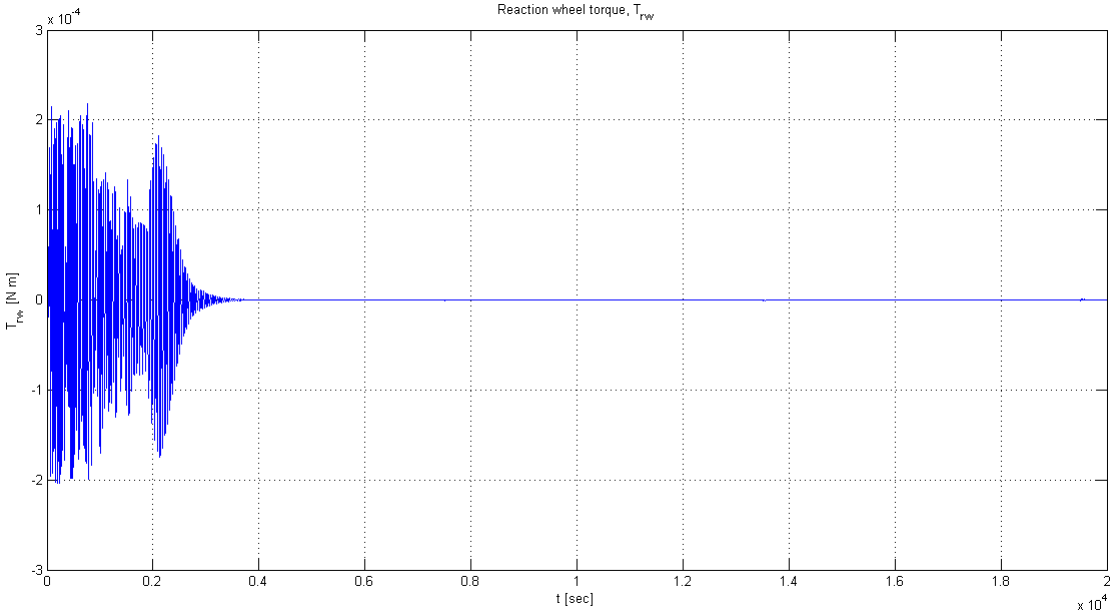
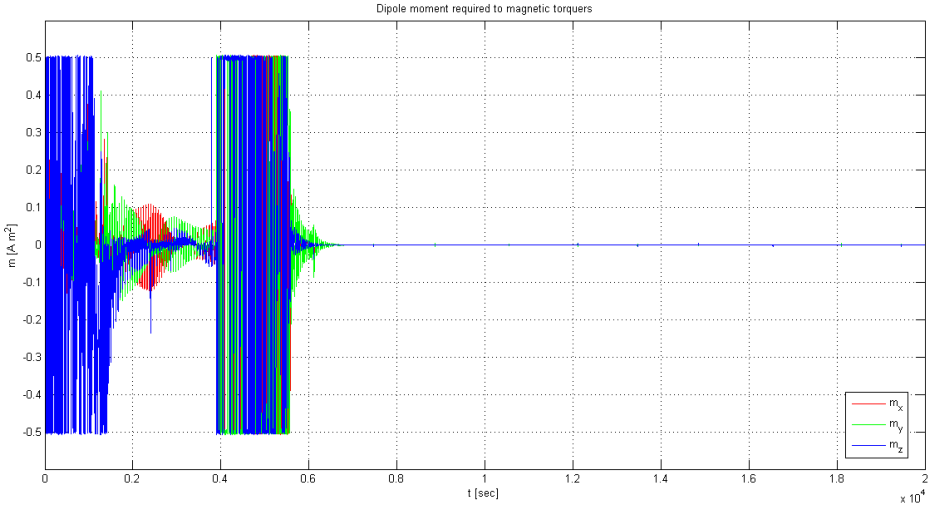
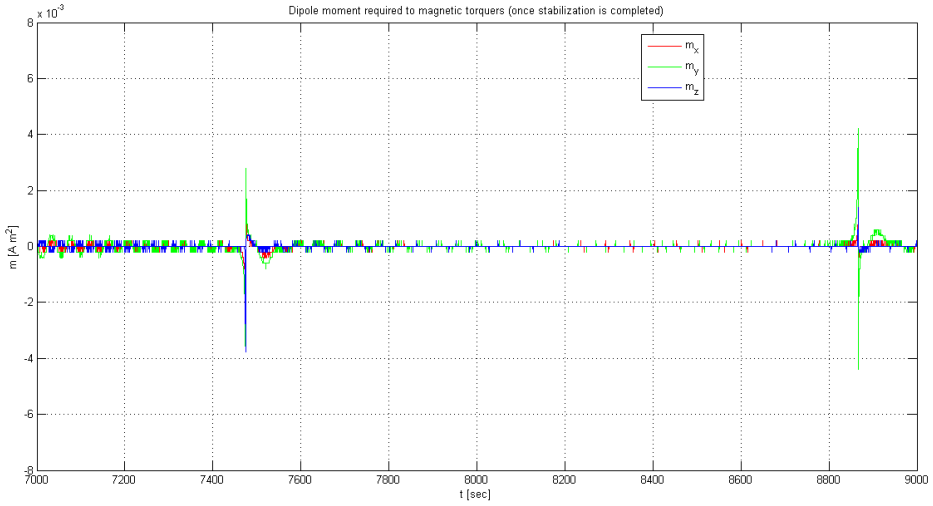


Figure 4.16 – Reaction wheel torque.

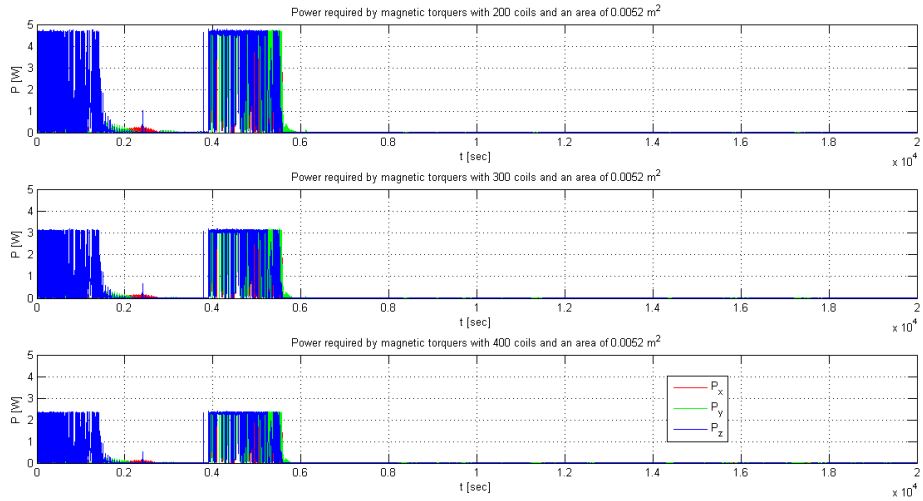


(a) Global view.

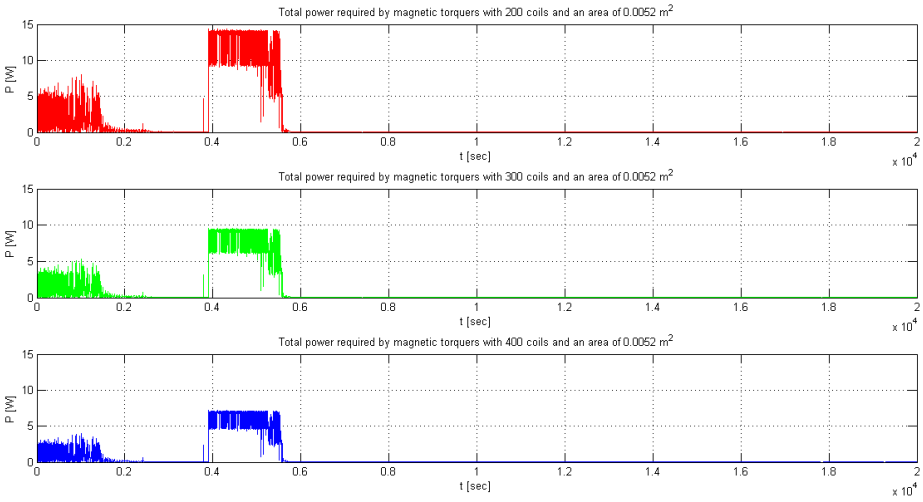


(b) Once stabilization is completed.

Figure 4.17 – Dipole moment required to magnetic torquers.



(a) Power required by MT.



(b) Total power required by MT.

Figure 4.18 – Power consumptions of MT.

Chapter 5

Hardware test

Before being able to achieve what is the effective system that will be installed on *3STAR* (see Figure 5.1 for the ADCS scheme and Figure 5.2 to see the logical process followed by the system), it is necessary a phase of characterization of the individual sensors and actuators that will be used and then, of course, an implementation in on-board software, which must be able to handle all the various components in order to ensure a good operation of the system.

In the following sections it will be presented the ADCS processor (ARM9), the test done for the hardware (like IMU) and then relative software as implemented on the satellite. Obviously to implement the control logic on the satellite there is the necessity to write a code that is understandable by the processor installed on board: being impossible of installing MATLAB®, it has been adopted C programming language, properly compiled on UNIX platform for ARM platform using a cross-compiler (*ELDK*, Embedded Linux Development Kit).

5.1 Brief description of *ARM architecture* and *C* programming language

The ARM is a 32-bit reduced instruction set computer (RISC) instruction set architecture (ISA) developed by ARM Holdings. It was named Advanced RISC Machine and, before that, Acorn RISC Machine. The ARM architecture is the most widely used 32-bit ISA in terms of numbers produced. They were originally conceived as a processor for desktop personal computers by Acorn Computers, a market now dominated by the x86 family used by IBM PC compatible computers. The relative simplicity of ARM processors made them suitable for low power applications. This has made them dominant in the mobile and embedded electronics market as relatively low cost and small microprocessors and microcontrollers.

In 2005, about 98% of the more than one billion mobile phones sold each year use

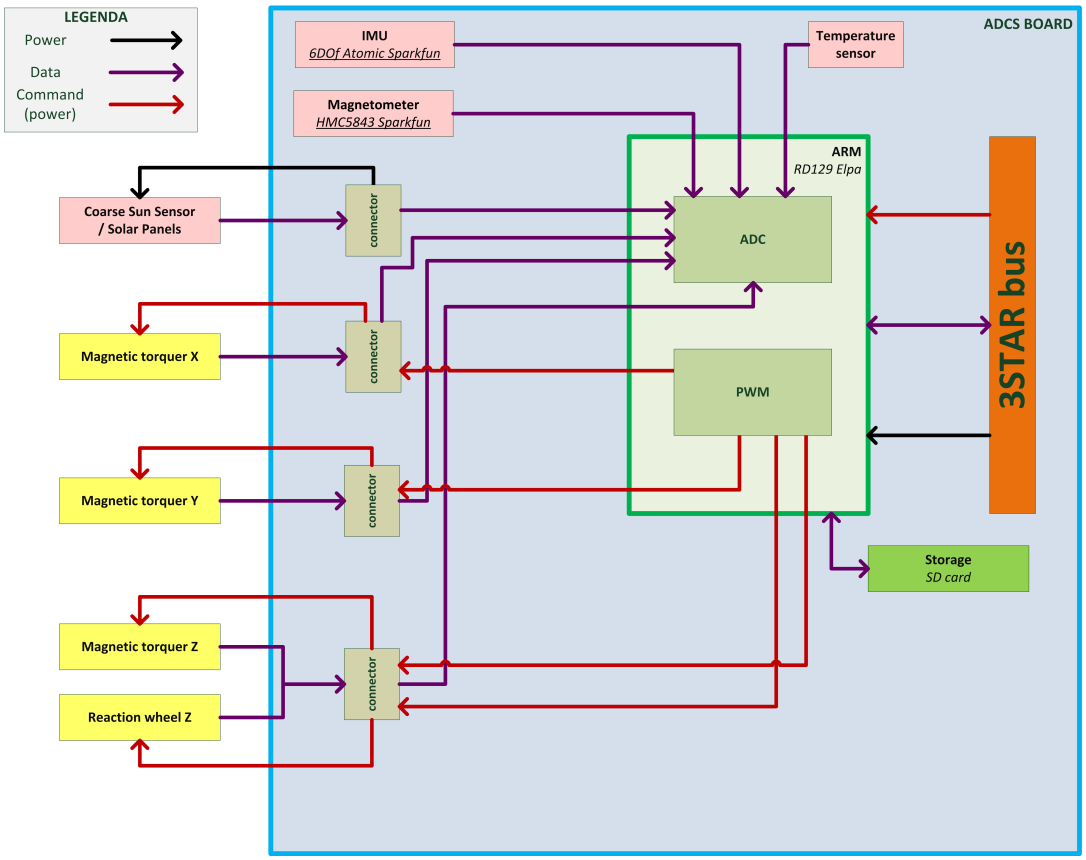


Figure 5.1 – ADCS scheme.

at least one ARM processor. As of 2009, ARM processors account for approximately 90% of all embedded 32-bit RISC processors. ARM processors are used extensively in consumer electronics, including PDAs, mobile phones, digital media and music players, hand-held game consoles, calculators and computer peripherals such as hard drives and routers. ARM9 is an ARM architecture 32-bit RISC CPU family: with this design generation, ARM moved from a von Neumann architecture (Princeton architecture) to a Harvard architecture with separate instruction and data buses (and caches), significantly increasing its potential speed. Most silicon chips integrating these cores will package them as modified Harvard architecture chips, combining the two address buses on the other side of separated CPU caches and tightly coupled memories.

C is a general-purpose computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system. It is an imperative systems implementation language. It was designed to be compiled using a relatively straightforward compiler, to provide low-level

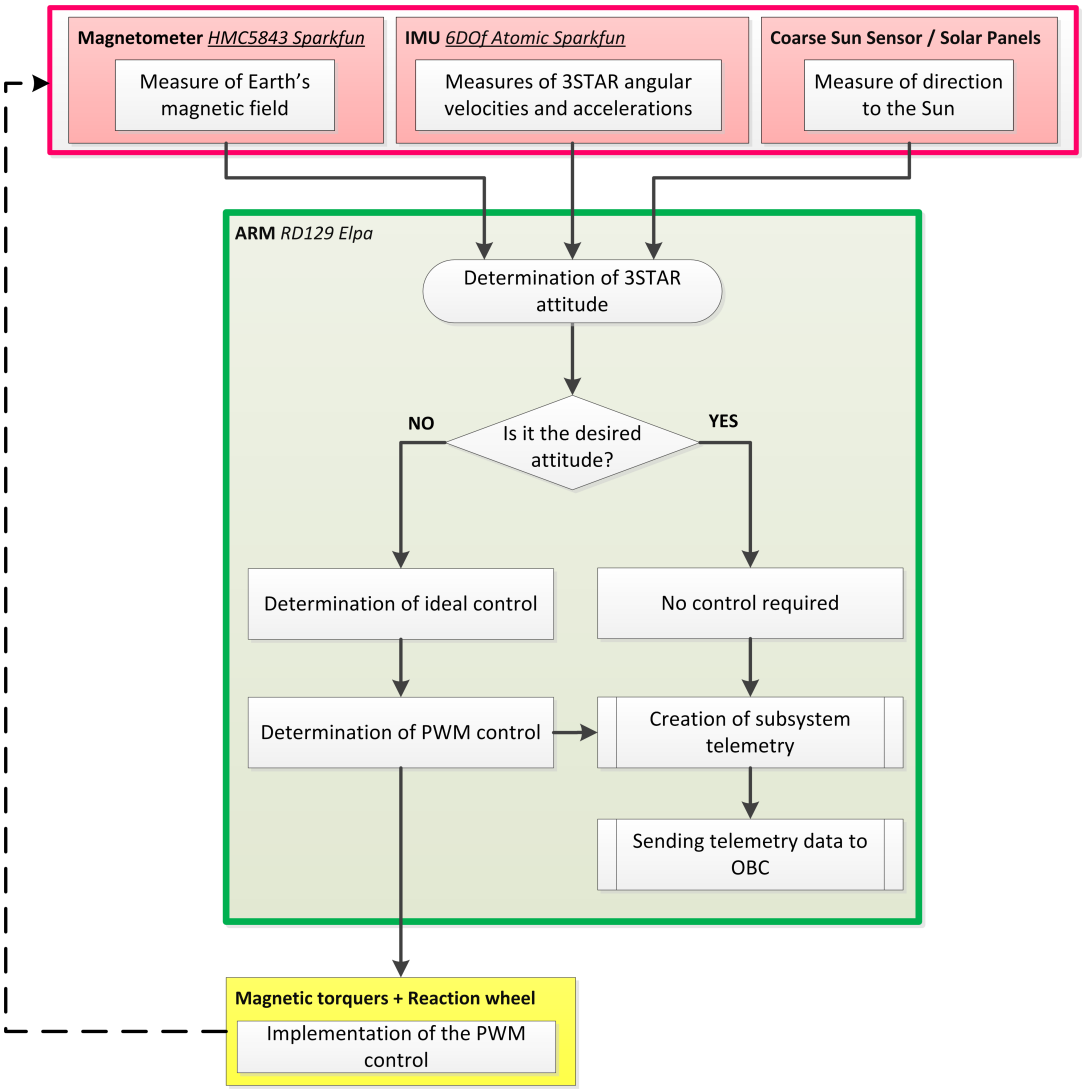


Figure 5.2 – Process followed by ADCS.

access to memory, to provide language constructs that map efficiently to machine instructions, and to require minimal run-time support. *C* was therefore useful for many applications that had formerly been coded in assembly language.

Despite its low-level capabilities, the language was designed to encourage cross-platform programming. A standards-compliant and portably written C program can be compiled for a very wide variety of computer platforms and operating systems with little or no change to its source code. The language has become available on a very wide range of platforms, from embedded microcontrollers to supercomputers. In addition to system software, C has long been the dominant language in a variety of

other applications characterized by strong emphasis on efficiency. Typical examples are telecommunications, industrial process control and real-time software. Today, the dominance of C in these contexts is partly decreased due to the advent of significant competitors, first of all the C++, however, it is not an obsolete programming language.

5.2 Inertial Measurement Unit

As written in Section 4.3, the task of the IMU is to measure accelerations and angular velocities by means of accelerometers and gyroscopes, nevertheless for our mathematical model (see Section 3.3) only angular velocities are needed, so only gyroscopes of the platform has been investigated and tested.

Rate sensors determine the attitude by measuring the rate of rotation of the spacecraft. They are located internal to the spacecraft and work at all points in an orbit. Since they measure a change instead of absolute attitude, gyroscopes must be used along with other attitude hardware to obtain full measurements. Rate gyros are simplest and least expensive gyros; their accuracy is usually good and although they provide only rotation rate information, their output could be fed into on-board computers and integrated to give angular displacement from some reference time or position.

The real IMU adopted for *3STAR* is the *Atomic IMU 6 Degrees of Freedom* (Figure 5.3) produced by SparkFun Electronics: it is a stripped-down IMU, designed to give good performance at a low price; the unit can run as a hard-wired UART interface (0-3.3V, 115200bps) and the processor is an Atmel ATMega328 running at 8MHz with 6 dedicated 10-bit ADC channels reading the sensors. The 6-DOF Atomic uses a Freescale MMA7361L triple-axis accelerometer, which is configurable to 1.5 or 6g sensitivity. Riding along with the MMA7361L are three ST Microelectronics LISY300AL single-axis, 300°/s gyros [17]. Other features of this sensor are:

Dimensions: 1.85 x 1.45 x 0.975 inches (47 x 37 x 25 mm);

Input voltage: from 3.4V to 10V DC;

Current consumption: 24mA hardwired with UART;

5.2.1 Test bench

A special test bench has been created by me in collaboration with Davide Falsetti (MSc student of Automotive Engineering) in the Aerospace System Laboratory to properly test the behavior of the IMU; it consists of:

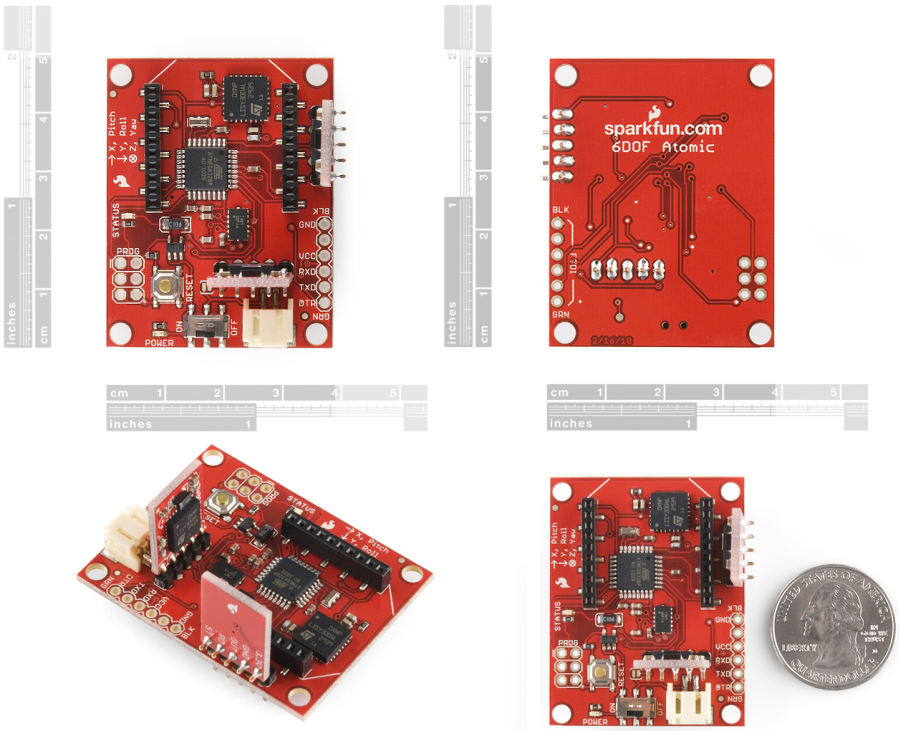
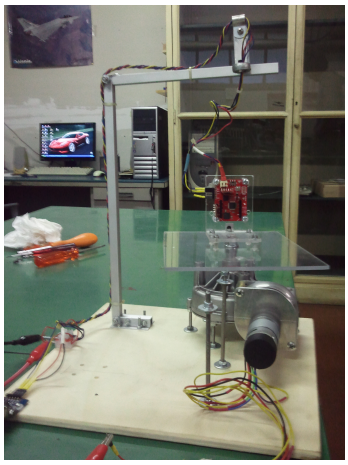


Figure 5.3 – Atomic IMU 6 Degrees of Freedom.

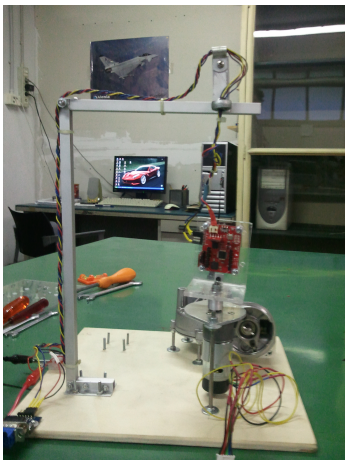
- an electric motor EMG30 (used with and without gearbox);
- facilities supporting cables necessary to power the motor, the platform and of course for serial connection to PC;
- a RS232 adapter with the task of converting the voltage at the desired levels;
- a rigid surface in plexiglass for the installation of the platform.

The final result is shown in Figure 5.4, where it is possible to see some details.

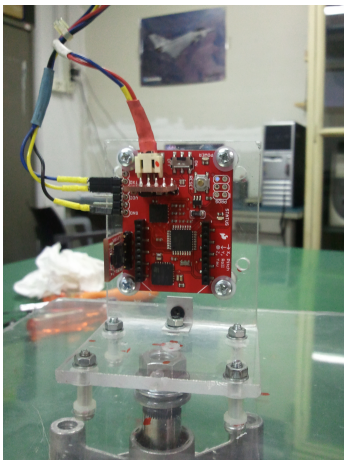
This type of instrument is capable of testing only one axis at a time, then there were carried out various types of installation of IMU on the platform so as to align the rotation axis of the engine with the X-axis, the Y-axis or the Z-axis. Of course, before to analyze IMU behavior, it was necessary to characterize the engine in the configuration both with the gearbox and without it. The result thus obtained is shown in the diagram in the Figure 5.5, where is possible to observe a substantially linear behavior for both configurations analyzed.



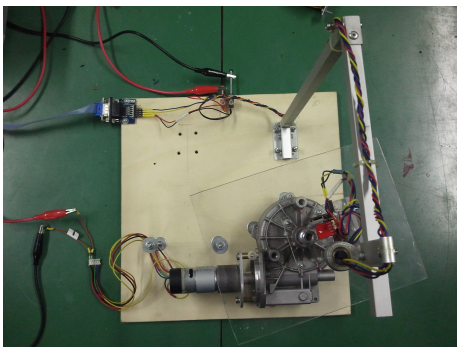
(a) With gearbox.



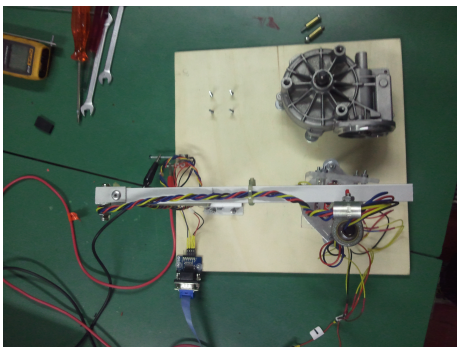
(b) Without gearbox.



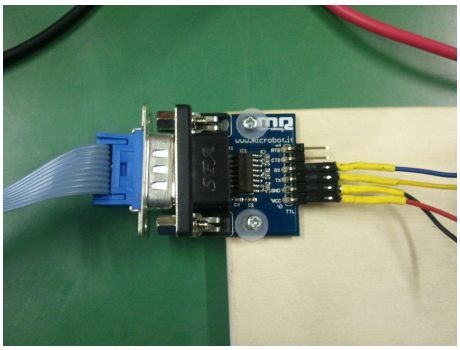
(c) IMU installed on platform.



(d) With gearbox (top view).



(e) Without gearbox (top view).



(f) RS232 adapter.

Figure 5.4 – Some figures of test bench of the IMU.

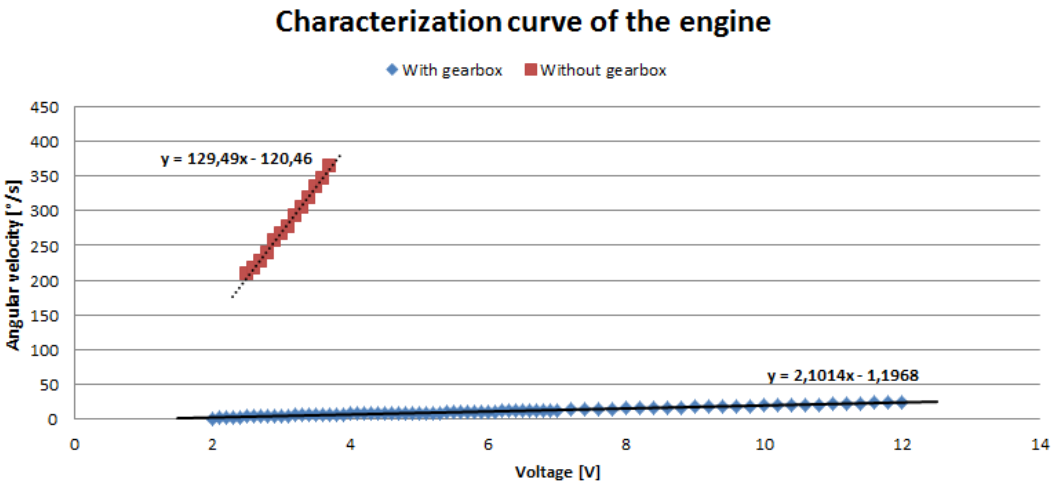


Figure 5.5 – Characterization curve of the engine.

5.2.2 Characterization

After testing the behavior of the engine, it has been possible to analyze IMU performances. As mentioned previously, the IMU is connected to the computer through a serial interface and a RS232 adapter (see Figure 5.4(f)), so firstly it is useful to look at the string of data received by the PC.

This type of electrical board provides an output in ASCII or binary and it can be configured from a menu reachable via a terminal like *HyperTerminal* on Windows platform or *Minicom* on UNIX platform; the strings obtained are shown in Table 5.1 and Table 5.2 where:

A: is the header flag;

c: is a counter;

ax: is the accelerometer on X axis;

ay: is the accelerometer on Y axis;

az: is the accelerometer on Z axis;

gx: is the gyroscope on X axis;

gy: is the gyroscope on Y axis;

gz: is the gyroscope on Z axis;

Z: is the footer flag.

	Binary string															
# byte	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
data	A	c_1	c_2	ax_1	ax_2	ay_1	ay_2	az_1	az_2	gx_1	gx_2	gy_1	gy_2	gz_1	gz_2	Z

Table 5.1 – Binary string output of IMU.

	ASCII string															
# byte	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
data	A	-	c_1	c_2	-	ax_1	ax_2	-	ay_1	ay_2	-	az_1	az_2	-	gx_1	gx_2
	16	*17*	*18*	*19*	*20*	*21*	*22*	*23*								
	-	gy_1	gy_2	-	gz_1	gz_2	-	Z								

Table 5.2 – ASCII string output of IMU.

Furthermore, the measures provided (this particular IMU provides measurements of angular velocities as voltages) require a proper conversion from digital to analog measure of voltage using the following relationship:

$$V_d = \frac{V_a \cdot res}{V_{ref}} \tag{5.1}$$

where V_a is the analog voltage measure, V_d is the digital voltage measure (provided by IMU), res is the resolution ($res = 2^{10} = 1024$) and $V_{ref} = 3.3[V]$ is the reference voltage.

After this, the next step was the verification of output of the electrical board in conditions first of zero rate (although this information is provided on the datasheet, obviously for manifacturing defects or other similar cases, the behavior of the specific IMU may differ from that which is the nominal case, that is, an output of 1.65 V under conditions of precisely zero rate) and then at some different speeds, roughly covering the entire measurement range (according to the characteristics and potentialities of the engine).

For the characterization it has been used a *C* code that, by exploiting a large number of measurements made from IMU, calculates the average to obtain the reference values (see Listing 5.1).

As mentioned previously, the tests has been conducted by covering a large part of the measuring range of the platform (this explains the use of the motor with gearbox, to test low angular velocities, and without a gearbox, for the high angular velocities) and especially both with a positive rotation that with a negative.

The results thus obtained (Figure 5.6) show a good response at both low and high angular speeds, denoting a completely linear behavior in the whole measuring range.

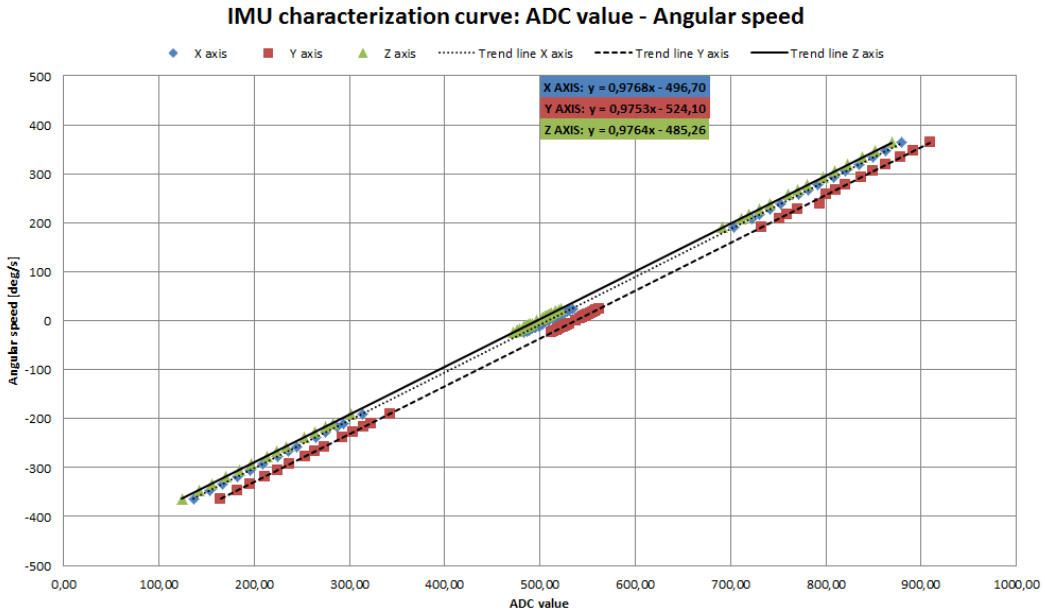


Figure 5.6 – Graph of angular speed as a function of digital measure (used in C implementation to obtain angular speed measured from output of IMU).

Listing 5.1 – C code used to calibrate IMU.

```
serial = serial_init();
if (serial < 0)
{ printf("errore apertura seriale\n");
  return 0;}
// Commands to activate IMU
write(serial,'41', 1);//sets the sample frequency to 50Hz
write(serial,'35', 1);//starts the unit running in binary
   mode with all channels active
//
unsigned char buffer[150] = { "" };
while (1)
{ read(serial, buffer, 150);
  usleep(100000);
  i = 0;
  while (i < 150)
  { if (buffer[i] == 'A')
    { c = 1;
      count1 = buffer[i + 1];
      count2 = buffer[i + 2];
```

```
19        accx1 = buffer[i + 3];
          accx2 = buffer[i + 4];
21        accy1 = buffer[i + 5];
          accy2 = buffer[i + 6];
23        accz1 = buffer[i + 7];
          accz2 = buffer[i + 8];
25        pitch1 = buffer[i + 9];
          pitch2 = buffer[i + 10];
27        roll1 = buffer[i + 11];
          roll2 = buffer[i + 12];
29        yaw1 = buffer[i + 13];
          yaw2 = buffer[i + 14];
31        count = (int) (count1 * 256 + count2);
          accx = (int) (accx1 * 256 + accx2);
33        acc_x = accx*3.3/1024-1.65;
          accy = (int) (accy1 * 256 + accy2);
35        acc_y = accy*3.3/1024-1.65;
          accz = (int) (accz1 * 256 + accz2);
37        acc_z = accz*3.3/1024-1.65;
          gyrox = (int) (pitch1 * 256 + pitch2);
39        gyro_x = gyrox*3.3/1024-1.65;
          gyroy = (int) (roll1 * 256 + roll2);
41        gyro_y = gyroy*3.3/1024-1.65;
          gyroz = (int) (yaw1 * 256 + yaw2);
43        gyro_z = gyroz*3.3/1024-1.65;
//mostra le misure in ADC
45        printf("%c\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%c\n", buffer[i
             ], count,
              accx, accy, accz, gyrox, gyroy, gyroz, buffer[i +
                 15]);
47        i = 150;}
      else
49      i = i + 1;
      if (c == 1)
51      { if (k<countcalib)
          { //Giroscopi - valori ADC
53          cal_gyrox = cal_gyrox+gyrox;
            cal_gyroy = cal_gyroy+gyroy;
55          cal_gyroz = cal_gyroz+gyroz;}
          else if (k == countcalib)
57        { //Giroscopi - valori ADC
            cal_gyrox = (cal_gyrox+gyrox)/countcalib;
59          cal_gyroy = (cal_gyroy+gyroy)/countcalib;
```

```
        cal_gyroz = (cal_gyroz+gyroz)/countcalib;
61      printf("I coefficienti di calibrazione sono stati
           trovati:\n");
        printf("Asse X=%g\n",cal_gyrox);
63      printf("Asse Y=%g\n",cal_gyroy);
        printf("Asse Z=%g\n",cal_gyroz);}
65    }
  }
67  k = k+1;
}
```

5.2.3 C implementation

The next step was the creation of a program written in C language, in such a way as to be implemented on board the satellite, which was able to take the measurements of IMU, provided in voltage levels, convert them into angular velocities so to be used for the control of the satellite. The script created (Listing 5.2) is similar to the previous one, but it does not include the part of calibration but simply that of conversion in radians per second (that is the unit required by the script that is responsible for the control).

Listing 5.2 – C code used to obtain measures from IMU.

```
serial = serial_init();
2 if (serial < 0)
{
4   printf("errore apertura seriale\n");
  return 0;
6 }
// Commands to activate IMU
8 write(serial,'41', 1);//sets the sample frequency to 50Hz
write(serial,'35', 1);//starts the unit running in binary
   mode with all channels active
10 //
unsigned char buffer[150] = { "" };
12 while (1)
{
14   read(serial, buffer, 150);
  usleep(100000);
16   i = 0;
  while (i < 150)
18   {
    if (buffer[i] == 'A')
```

```
20      {
          c = 1;
22        count1 = buffer[i + 1];
          count2 = buffer[i + 2];
24        accx1 = buffer[i + 3];
          accx2 = buffer[i + 4];
26        accy1 = buffer[i + 5];
          accy2 = buffer[i + 6];
28        accz1 = buffer[i + 7];
          accz2 = buffer[i + 8];
30        pitch1 = buffer[i + 9];
          pitch2 = buffer[i + 10];
32        roll1 = buffer[i + 11];
          roll2 = buffer[i + 12];
34        yaw1 = buffer[i + 13];
          yaw2 = buffer[i + 14];
36        count = (int) (count1 * 256 + count2);
          accx = (int) (accx1 * 256 + accx2);
38        acc_x = accx*3.3/1024-1.65;
          accy = (int) (accy1 * 256 + accy2);
40        acc_y = accy*3.3/1024-1.65;
          accz = (int) (accz1 * 256 + accz2);
42        acc_z = accz*3.3/1024-1.65;
          gyrox = (int) (pitch1 * 256 + pitch2);
44        gyro_x = gyrox*3.3/1024-1.65;
          gyroy = (int) (roll1 * 256 + roll2);
46        gyro_y = gyroy*3.3/1024-1.65;
          gyroz = (int) (yaw1 * 256 + yaw2);
48        gyro_z = gyroz*3.3/1024-1.65;
          //mostra le misure in ADC
50        //printf("%c\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%c\n", buffer
             [i],
          count, accx, accy, accz, gyrox, gyroy, gyroz, buffer[i
             + 15]);
52        //conversione da ADC a velocità angolari (deg/s)
             sfruttando
          //la linea di tendenza dalla caratterizzazione
54        omegaxDEG=0.9768*gyrox-496.70;
          omegayDEG=0.9753*gyroy-524.10;
56        omegazDEG=0.9764*gyroz-485.26;
          //conversione in radianti
58        omegax=omegaxDEG*PIGRECO/180;
          omegay=omegayDEG*PIGRECO/180;
```

```
60        omegaz=omegazDEG*PIGRECO/180;
          // mostra le misure in gradi
62        printf("%c\t%d\t%g\t%g\t%g\t%c\n", buffer[i], count,
             omegaxDEG,
          omegayDEG, omegazDEG, buffer[i + 15]);
64        // mostra le misure in radianti
          printf("%c\t%d\t%g\t%g\t%g\t%c\n", buffer[i], count,
             omegax,
66        megay, omegaz, buffer[i + 15]);
          i = 150;
68      } else
        i = i + 1;
70    }
```

5.3 PWM and ADC on ARM9 processor

Pulse-width modulation (PWM) is a commonly used technique for controlling power to inertial electrical devices, made practical by modern electronic power switches. The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast pace: the longer the switch is on compared to the off periods, the higher the power supplied to the load is. The term duty cycle describes the proportion of "on" time to the regular interval or "period" of time; a low duty cycle corresponds to low power, because the power is off for most of the time. Duty cycle is often expressed in percent, 100% being fully on. The main advantage of PWM is that power loss in the switching devices is very low: when a switch is off there is practically no current, and when it is on, there is almost no voltage drop across the switch, so power loss, being the product of voltage and current, is thus in both cases close to zero. PWM also works well with digital controls, which, because of their on/off nature, can easily set the needed duty cycle.

Pulse-width modulation uses a rectangular pulse wave whose pulse width is modulated resulting in the variation of the average value of the waveform. If we consider a pulse waveform $f(t)$ with a low value y_{min}, a high value y_{max} and a duty cycle D (see Figure 5.7), the average value of the waveform is given by:

$$\bar{y} = \frac{1}{T}\int_0^T f(t)dt \tag{5.2}$$

As $f(t)$ is a pulse wave, its value is y_{max} for $0 < t < D \cdot T$ and y_{min} for $D \cdot T < t < T$,

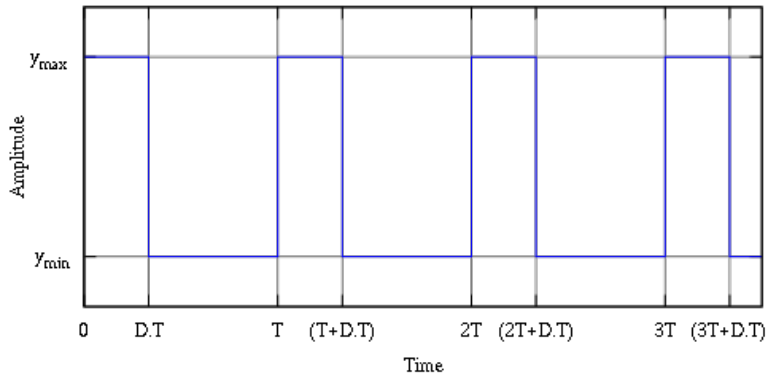


Figure 5.7 – Example of PWM.

the Equation 5.2 then becomes:

$$\begin{aligned} \bar{y} &= \frac{1}{T}\left(\int_0^{DT} y_{max}dt + \int_{DT}^{T} y_{min}dt\right) = \\ &= \frac{D \cdot T \cdot y_{max} + T(1-D)y_{min}}{T} = \\ &= D \cdot y_{max} + (1-D)y_{min} \end{aligned} \tag{5.3}$$

This latter equation can be fairly simplified in many cases where $y_{min} = 0$ as $\bar{y} = D \cdot y_{max}$. From this, it is obvious that the average value of the signal $\bar{y}$ is directly dependent on the duty cycle D.

The simplest way to generate a PWM signal is the intersective method, which requires only a sawtooth or a triangle waveform (easily generated using a simple oscillator) and a comparator. When the value of the reference signal (the red sine wave in Figure 5.8) is more than the modulation waveform (blue), the PWM signal (magenta) is in the high state, otherwise it is in the low state.

With regard to the ADC, it is nothing more than an acronym for Analog to Digital Converter: it is a device that uses sampling to convert a continuous quantity to a discrete time representation in digital form. An ADC may also provide an isolated measurement such as an electronic device that converts an input analog voltage or current to a digital number proportional to the magnitude of the voltage or current. The digital output may use different coding schemes. Typically the digital output will be a two's complement binary number that is proportional to the input,as in this case. Generically, an ADC is characterized by its resolution (the number of discrete values it can produce over the range of analog values), response type (linear or logarithmic), quantization error (it is the difference between the original signal and the digitized signal) and non-linearity (physical imperfections cause output to deviate from a linear function of input).

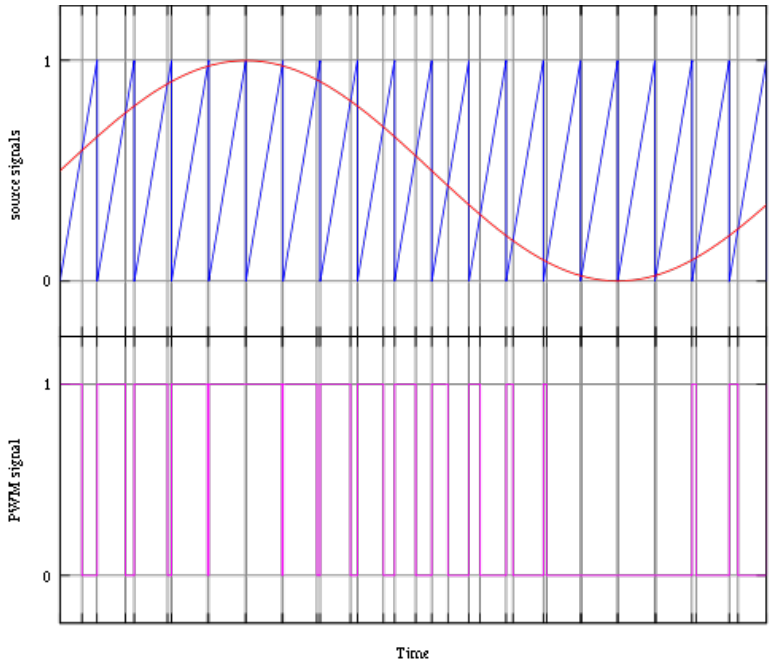


Figure 5.8 – A simple method to generate the PWM pulse train corresponding to a given signal is the intersective PWM: the signal (here the red sinewave) is compared with a sawtooth waveform (blue). When the latter is less than the former, the PWM signal (magenta) is in high state (1). Otherwise it is in the low state (0).

The CPU board adopted for *3STAR* is the *RD129 - ARM9 Embedded CPU* (see Figure 5.9(a)) produced by ELPA: it is an ARM9 CPU, designed to give good performance at a very low price for this kind of item; the same company produces also a development board (*RD126*, Figure 5.9(b)) for this CPU that gives the possibility to make all operations more convenient, providing many inputs for various devices. Features of this CPU are [18]:

Technical features:

- 32-bits ARM 9 CPU, clocked at 240MHz;
- 32MB of SDRAM (64MB optional) at 120MHz (32-bits interfaced);
- 64MB of on board Flash memory;
- 3 Serial interfaces;
- 2 USB host interfaces (1 switchable to USB device);
- 10-bits ADC converter;

- IIC bus interface;
- 4 internal PWM timers;
- Watchdog timer;

Dimensions: very small sizes, 45x40x8 mm;

Power: 0.5W typical, 1W max; only a single 3.3Vdc 5% is required;

Operating temperature range: from -25°C to +85°C;

Development board features:

- 3.3V switching power supply;
- level translator for 2 RS232 serial interfaces;
- USB to Ethernet converter;
- USB Host connector;
- MMC or SD flash memory card connector;
- IIC bus eeprom;

5.3.1 Characterization

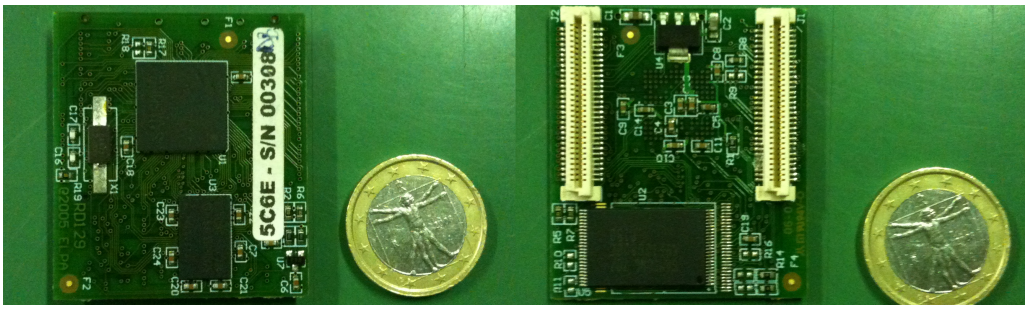
Before to perform PWM control, it is necessary to properly configure I/O CPU pins. In this case, all operations are performed by reading/writing virtual ascii files located into directory */sys/devices/platform/s3c2410-gpio/*. There are different groups of these files:

pin status files: with same name of pin (ex: g0), can be used to read or write the pin status (0 or 1);

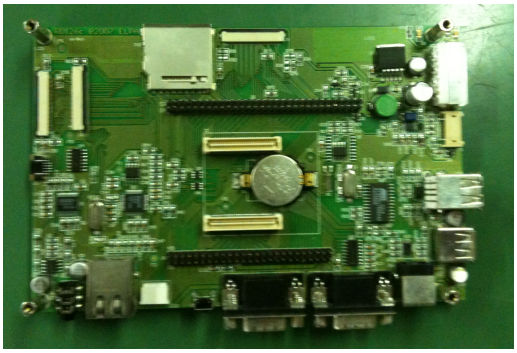
pin configuration files: with name of pin followed by suffix *-cfg*, can be used to read or change the pin mode, that can be: *input*, *output* (not allowed on all pins), *fn1* or *fn2* where *fn1* is the main peripheral function of pin while *fn2* is the auxiliary function (only some pins have it);

pullup configuration file: with name of pin followed by suffix *-pup*, can be used to read or change the pullup status (1=on, 0=off);

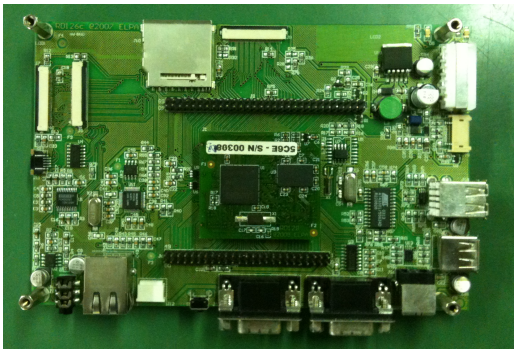
external interrupt configuration file: named "eintX-cfg", it reflects the external interrupt mode. It can have the values "rising", "falling" or "both" and selects which signal's edge triggers the interrupt. These files appear only when relative pins are configured for external interrupt function.



(a) RD129 CPU.



(b) Development board RD126.



(c) Development board RD126 with RD129 CPU.

Figure 5.9 – Some figures of CPU adopted and its development board.

The next step is to configure the timer drivers: they gives user access to internal timers and can be used in 2 ways, to generate an external signal or to wait for a defined time. All operations are performed by reading/writing virtual ascii files located into directory */sys/devices/platform/um-timer.X/*, where X ranges from 0 to 3 (timer 4 is used exclusively by UNIX' scheduler). In this directory there are 3 virtual files:

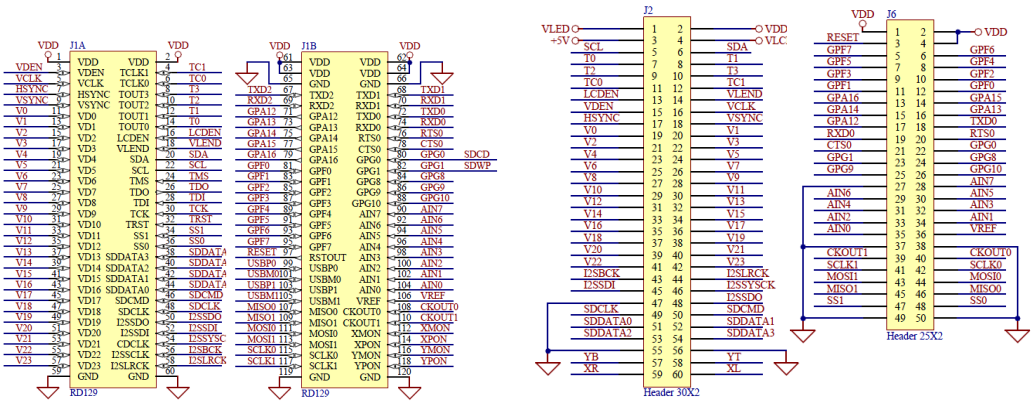
frequency: sets the frequency (in Hz) you want to generate on dedicated pin. When you write to this file, the duty-cycle is setted to 50%;

duty: sets the duty-cycle of the signal you want to generate on dedicated pin. Range is from 0 to 1000;

wait: when written it programs the time required in microseconds and starts the timer, when read waits until the programmed time expires.

To take measure from ADC pins, simply must read virtual ascii files located into directory *"/sys/devices/platform/s3c2410-adc/ainX*, where X ranges from 0 to 7.

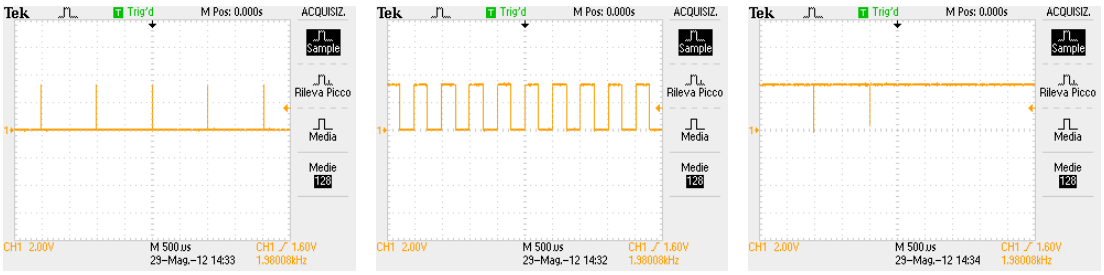
Pins used as output for PWM are shown in Figure 5.10: they are # 8, # 10 and # 12 for RD129 CPU and # 8, # 9 and # 10 for RD126 development board. Pins used to read measures (ADC pins) are # 90, # 92, # 94, # 96, # 98, # 100, # 102 and # 104 for RD129 CPU and # 28, # 29, # 30, # 31, # 32, # 33, # 34 and # 35 for RD126 development board. In Figure 5.11 it is possible to see the output



(a) RD129 CPU. (b) RD126 development board.

Figure 5.10 – Schema of pins of RD126 and RD129.

obtained with Listing 5.3 (it has been used a terminal to communicate with RD129 CPU installed on RD126 development board connected to pc through a serial-to-usb converter) by varying duty cycle. The images are captured using the TDS2004C Oscilloscope produced by Tektronix.



(a) Duty cycle = 1 (0.1%). (b) Duty cycle = 500 (50%). (c) Duty cycle = 999 (99.9%).

Figure 5.11 – PWM output from pins.

Listing 5.3 – Command used to try PWM output pin and ADC pin.

```
cd /sys/devices/platform/s3c2410-gpio  REM path
echo fn1 > b1-cfg  REM to properly set pin output
```

```
cat b1-cfg  REM to verify pin setting
cd /sys/devices/platform/um_timer.X  REM X stands for 1,2 or
   3
echo 2000 > frequency  REM to set frequency
echo xxx > duty  REM to set duty cycle, xxx stands for a
   value between 1(that is 0.1%) and 999(that is 99,9%)
REM -------------------------
cd /sys/devices/platform/s3c2410-adc/ainX REM X stands for
   0,1,2,3,4,5,6 or 7
```

The real characterization has been done using the ADCS board (see Figure 5.12) and magnetic torquers used on *e-st@r*, because those of *3STAR* has not yet been realized (but presumably it will be very similar) and because there was a need to test the behavior of the ARM CPU on a "real" electrical board and not on the development board as they have very different circuits.

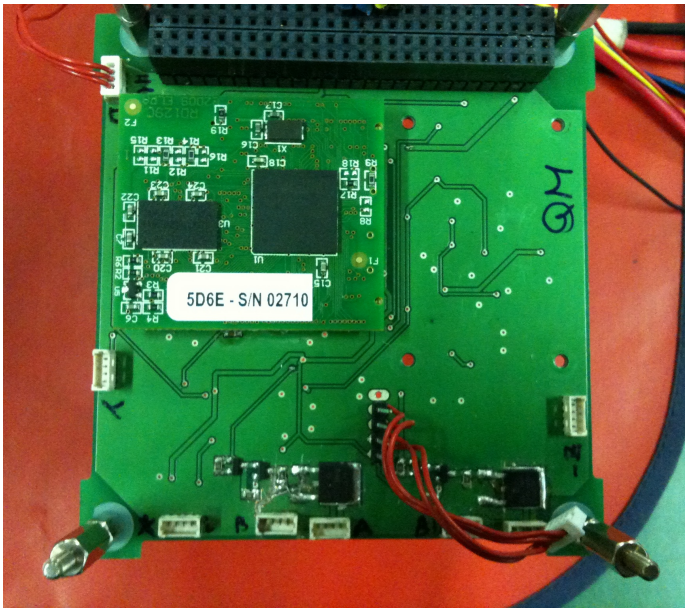


Figure 5.12 – ADCS board used to test PWM control with ARM9

The settings used to do this characterization and then implemented on board are shown in Listing 5.4: in this way both PWM output and related ADC has been tested to understand their behavior.

Listing 5.4 – C code used to properly test RD129 (PWM and ADC).

```
#define PWM_1 "/sys/devices/platform/um_timer.1/frequency"
#define SET_PORT_PWM1 "/sys/devices/platform/s3c2410-gpio/b1-
   cfg"
```

```
#define PWM_2 "/sys/devices/platform/um_timer.2/frequency"
#define SET_PORT_PWM2 "/sys/devices/platform/s3c2410-gpio/b2-
    cfg"
#define PWM_3 "/sys/devices/platform/um_timer.3/frequency"
#define SET_PORT_PWM3 "/sys/devices/platform/s3c2410-gpio/b3-
    cfg"
#define PWM_DUTY1 "/sys/devices/platform/um_timer.1/duty"
#define PWM_DUTY2 "/sys/devices/platform/um_timer.2/duty"
#define PWM_DUTY3 "/sys/devices/platform/um_timer.3/duty"
#define ADC_1 "/sys/devices/platform/s3c2410-adc/ain0"
#define ADC_2 "/sys/devices/platform/s3c2410-adc/ain1"
#define ADC_3 "/sys/devices/platform/s3c2410-adc/ain2"

FILE *pwm1;
FILE *pwmset1;
FILE *pwm2;
FILE *pwmset2;
FILE *pwm3;
FILE *pwmset3;
FILE *pwmduty1;
FILE *pwmduty2;
FILE *pwmduty3;
FILE *adc1;
FILE *adc2;
FILE *adc3;
FILE *risultati;

printf("QUI COMINCIA LA PROCEDURA PER TESTARE PWM E ADC DELL'
    ARM RD129\n");
printf("Indicare la frequenza [Hz] per il PWM-1: ");
scanf("%d", &frequepwm1);
printf("Indicare la frequenza [Hz] per il PWM-2: ");
scanf("%d", &frequepwm2);
printf("Indicare la frequenza [Hz] per il PWM-3: ");
scanf("%d", &frequepwm3);
//----inizio scrittura frequenze su file virtuali----
pwmset1 = fopen(SET_PORT_PWM1, "w");
if(pwmset1 == NULL) {
  printf("mancata apertura gpio\n");
}
else {
  fprintf (pwmset1,"fn1");
}
```

```
fclose( pwmset1);
pwm1 = fopen(PWM_1, "w");
if(pwm1 == NULL) {
  printf("mancata apertura\n");
}
else {
  fprintf (pwm1,"%d", frequepwm1);
}
fclose( pwm1);

pwmset2 = fopen(SET_PORT_PWM2, "w");
if(pwmset2 == NULL) {
  printf("mancata apertura gpio\n");
}
else {
  fprintf (pwmset2,"fn1");
}
fclose( pwmset2);
pwm2 = fopen(PWM_2, "w");
if(pwm2 == NULL) {
  printf("mancata apertura\n");
}
else {
  fprintf (pwm2,frequepwm2);
}
fclose( pwm2);

pwmset3 = fopen(SET_PORT_PWM3, "w");
if(pwmset3== NULL) {
  printf("mancata apertura gpio\n");
}
else {
  fprintf (pwmset3,"fn1");
}
fclose( pwmset3);
pwm3 = fopen(PWM_3, "w");
if(pwm3== NULL) {
  printf("mancata apertura\n");
}
else {
  fprintf (pwm3,frequepwm3);
}
fclose( pwm3);
```

```
86 //----fine scrittura frequenze su file virtuali----
   //----inizio scrittura duty cycle su file virtuali----
88 risultati = fopen("risultati", "w+");
   if(risultati<0) {
90   printf("File dei risultati non aperto correttamente\n");
   }
92 else {
     for(i=1;i<=999;i++)
94   {
       pwmduty1=fopen (PWM_DUTY1 , "w");
96     if(pwmduty1 == NULL) {
         printf("mancata apertura\n");
98     }
       else {
100      duty1=i;
         fprintf (pwmduty1,"%d", duty1);
102    }
       fclose(pwmduty1);
104    adc1=fopen (ADC_1, "r");
       if(adc1 == NULL) {
106      printf("Errore apertura ADC1\n");
       }
108    else {
         fscanf (adc1, "%d", &misura1);
110    }
       fclose(adc1);
112    pwmduty2=fopen (PWM_DUTY2 , "w");
       if(pwmduty2== NULL) {
114      printf("mancata apertura\n");
       }
116    else {
         duty2=i;
118      fprintf (pwmduty2,"%d", duty2);
       }
120    fclose(pwmduty2);
       adc2=fopen (ADC_2, "r");
122    if(adc2 == NULL) {
         printf("Errore apertura ADC2\n");
124    }
       else {
126      fscanf (adc2, "%d", &misura2);
       }
128    fclose(adc2);
```

```
    pwmduty3=fopen (PWM_DUTY3 , "w");
    if(pwmduty3 == NULL) {
      printf("mancata apertura\n");
    }
    else {
      duty3=i;
      fprintf (pwmduty3,"%d", duty3);
    }
    fclose(pwmduty3);
    adc3=fopen (ADC_3, "r");
    if(adc3 == NULL) {
      printf("Errore apertura ADC3\n");
    }
    else {
      fscanf (adc3, "%d", &misura3);
    }
    fclose(adc3);
  }
  fprintf(risultati,"%d\t%d\t%d\t%d\t%d\t%d\n", duty1,
     misura1,duty2, misura2,duty3, misura3);
  printf ("%d\t%d\t%d\t%d\t%d\t%d\n", duty1, misura1,duty2,
     misura2,duty3, misura3);
  usleep(100000);
  fclose(risultati);
}
return 0;
}
```

Results in Figure 5.13 show an approximately linear trend of the ADC with respect to duty cycle, except for the initial part (corresponding to low duty cycle values) where instead is present a saturation: this problem will be fixed on the *3STAR* ADCS board with different circuits so as to avoid reaching this saturation; in Figure 5.14 it is possible to see the relationships between duty cycle values and the corresponding measured voltages and currents at PWM outputs, using as loads the magnetic torquers used for e-st@r (they have 176 coils and a roughly square area of $0.0048m^2$).

5.3.2 C implementation

After the characterization, has been created a *C* program that can set the values of duty cycle (on the basis of controls calculated as explained in Section 4.5) and to detect ADC measures fully automated with without action from the outside through

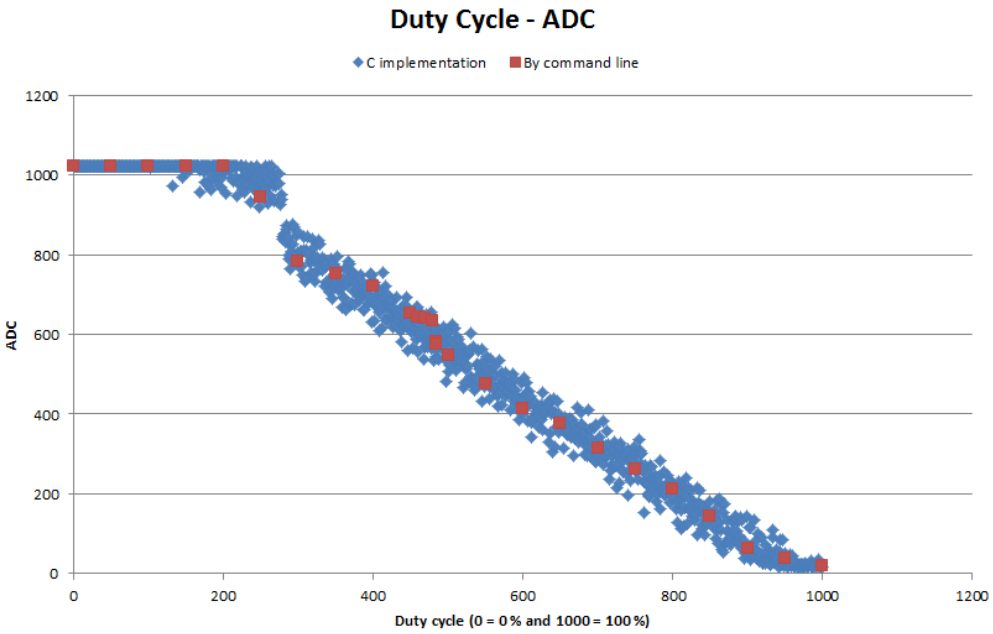


Figure 5.13 – Characterization curve Duty cycle - ADC of ADCS board.

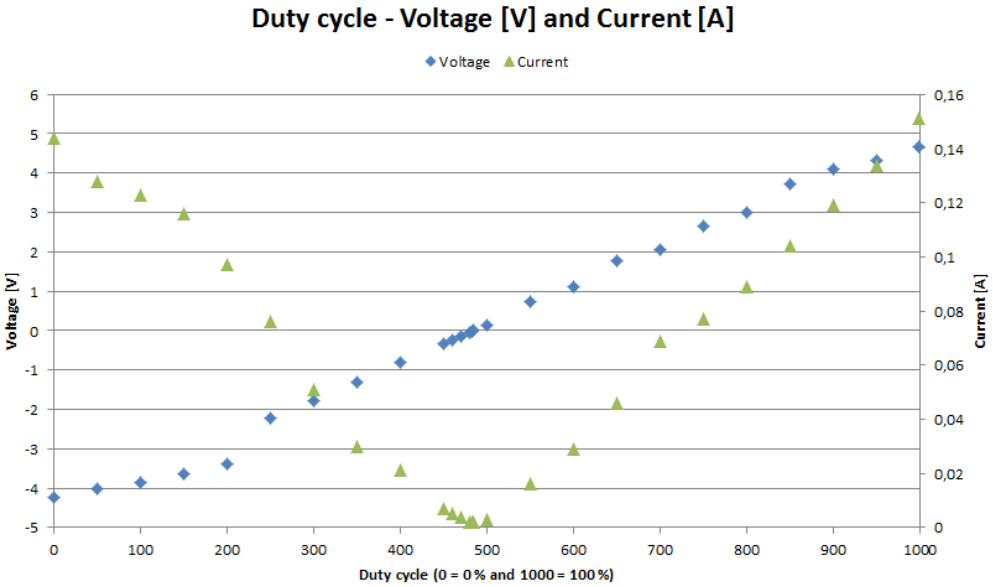


Figure 5.14 – Characterization curve Duty cycle - Voltage & Current of ADCS board.

the terminal. The script created (Listing 5.5) is similar to the previous one, but it does not include testing procedure.

Listing 5.5 – C code used to properly set PWM and ADC.

```
#define PWM_1 "/sys/devices/platform/um_timer.1/frequency"
#define SET_PORT_PWM1 "/sys/devices/platform/s3c2410-gpio/b1-
    cfg"
#define PWM_2 "/sys/devices/platform/um_timer.2/frequency"
#define SET_PORT_PWM2 "/sys/devices/platform/s3c2410-gpio/b2-
    cfg"
#define PWM_3 "/sys/devices/platform/um_timer.3/frequency"
#define SET_PORT_PWM3 "/sys/devices/platform/s3c2410-gpio/b3-
    cfg"
#define PWM_DUTY1 "/sys/devices/platform/um_timer.1/duty"
#define PWM_DUTY2 "/sys/devices/platform/um_timer.2/duty"
#define PWM_DUTY3 "/sys/devices/platform/um_timer.3/duty"
#define ADC_1 "/sys/devices/platform/s3c2410-adc/ain0"
#define ADC_2 "/sys/devices/platform/s3c2410-adc/ain1"
#define ADC_3 "/sys/devices/platform/s3c2410-adc/ain2"

FILE *pwm1;
FILE *pwmset1;
FILE *pwm2;
FILE *pwmset2;
FILE *pwm3;
FILE *pwmset3;
FILE *pwmduty1;
FILE *pwmduty2;
FILE *pwmduty3;
FILE *adc1;
FILE *adc2;
FILE *adc3;
FILE *risultati;

pwmset1 = fopen(SET_PORT_PWM1, "w");
if(pwmset1 == NULL) {
  printf("mancata apertura gpio\n");
}
else {
  fprintf (pwmset1,"fn1");
}
fclose( pwmset1);
pwm1 = fopen(PWM_1, "w");
```

```
37 if(pwm1 == NULL) {
     printf("mancata apertura\n");
39 }
   else {
41   fprintf (pwm1,"%d", frequepwm1);
   }
43 fclose( pwm1);

45 pwmset2 = fopen(SET_PORT_PWM2, "w");
   if(pwmset2 == NULL) {
47   printf("mancata apertura gpio\n");
   }
49 else {
     fprintf (pwmset2,"fn1");
51 }
   fclose( pwmset2);
53 pwm2 = fopen(PWM_2, "w");
   if(pwm2 == NULL) {
55   printf("mancata apertura\n");
   }
57 else {
     fprintf (pwm2,frequepwm2);
59 }
   fclose(pwm2);
61

   pwmset3 = fopen(SET_PORT_PWM3, "w");
63 if(pwmset3== NULL) {
     printf("mancata apertura gpio\n");
65 }
   else {
67   fprintf (pwmset3,"fn1");
   }
69 fclose( pwmset3);
   pwm3 = fopen(PWM_3, "w");
71 if(pwm3== NULL) {
     printf("mancata apertura\n");
73 }
   else {
75   fprintf (pwm3,frequepwm3);
   }
77 fclose( pwm3);
   risultati = fopen("risultati", "w+");
79 if(risultati<0) {
```

```
  printf("File dei risultati non aperto correttamente\n");
}
else {
    pwmduty1=fopen (PWM_DUTY1 , "w");
    if(pwmduty1 == NULL) {
      printf("mancata apertura\n");
    }
    else {
      fprintf (pwmduty1,"%d", duty1);
    }
    fclose(pwmduty1);
    adc1=fopen (ADC_1, "r");
    if(adc1 == NULL) {
      printf("Errore apertura ADC1\n");
    }
    else {
      fscanf (adc1, "%d", &misura1);
    }
    fclose(adc1);
    pwmduty2=fopen (PWM_DUTY2 , "w");
    if(pwmduty2== NULL) {
      printf("mancata apertura\n");
    }
    else {
      fprintf (pwmduty2,"%d", duty2);
    }
    fclose(pwmduty2);
    adc2=fopen (ADC_2, "r");
    if(adc2 == NULL) {
      printf("Errore apertura ADC2\n");
    }
    else {
      fscanf (adc2, "%d", &misura2);
    }
    fclose(adc2);
    pwmduty3=fopen (PWM_DUTY3 , "w");
    if(pwmduty3 == NULL) {
      printf("mancata apertura\n");
    }
    else {
      fprintf (pwmduty3,"%d", duty3);
    }
    fclose(pwmduty3);
```

```
123     adc3=fopen (ADC_3, "r");
        if(adc3 == NULL) {
125       printf("Errore apertura ADC3\n");
        }
127     else {
          fscanf (adc3, "%d", &misura3);
129     }
        fclose(adc3);
131   }
      fprintf(risultati,"%d\t%d\t%d\t%d\t%d\t%d\n", duty1,
         misura1,duty2, misura2,duty3, misura3);
133   usleep(100000);
      fclose(risultati);
135 return 0;
}
```

Chapter 6

Hardware In the Loop

Hardware-in-the-loop (HIL) simulation is a technique used in the development and test of complex real-time embedded systems: HIL simulation provides an effective platform by adding the complexity of the plant under control to the test, through a mathematical representation of all related dynamic systems, to the test platform. These mathematical representations are referred to as the "plant simulation" and the embedded system to be tested interacts with this plant simulation.

An HIL simulation includes electrical emulation of sensors and actuators. These electrical emulations act as the interface between the plant simulation and the embedded system under test. The value of each electrically emulated sensor is controlled by the plant simulation and is read by the embedded system under test (feedback). Likewise, the embedded system under test implements its control algorithms by outputting actuator control signals. Changes in the control signals result in changes to variable values in the plant simulation.

In many cases, the most effective way to develop an embedded system is to connect the embedded system to the real plant while in other cases, HIL simulation is more efficient. The metric of development and test efficiency is typically a formula that includes the following factors:

cost: the cost of the approach will be a measure of the cost of all tools and effort;

duration: the duration of development and test affects the time-to-market for a planned product;

safety: the safety factor and duration are typically equated to a cost measure;

feasibility: for *3STAR* project, probably this is the most important feature just because a satellite spends a large part of the operating life in an environment that is difficult to reproduce on the ground.

Specific conditions that warrant the use of HIL simulation include the following:

enhancing the quality of testing: usage of HIL enhances the quality of the testing by increasing the scope of the testing. An ideal condition to test the embedded system is to test it against the real plant but most of the time real plant itself imposes limitations in terms of the scope of the testing;

tight development schedules: often the tight development schedules do not allow embedded system testing to wait for a prototype to be available;

high-burden-rate plant: in many cases, the plant is more expensive than a high fidelity, real-time simulator and therefore has a higher-burden rate. Therefore, it is more economical to develop and test while connected to a HIL simulator than the real plant.

For *3STAR* project, the Hardware In the Loop is particularly useful because the satellite will operate in an hardly reproducible environment, that is the space. Obviously more accurate is the model more accurately the response of the system will be. Only with an HIL simulation is possible to reproduce all the environmental conditions that the satellite will face in its operative life, which would be more expensive and more difficult to realize that without an HIL simulation. It is easily understandable the impossibility to test the *3STAR* ADCS in our laboratory without the HIL simulation, because the laboratory should be equipped to recreate the same conditions satellite would experience in orbit, that are:

- vacuum;
- apparent Sun position;
- low gravity;
- Earth magnetic field;
- orbital speed.

Using HIL simulation it is possible to avoid these problems by implementing a numerical model for parts that are not physically realizable.

To verify the system via HIL simulation two components can be identified:

test object: ADCS board with ARM RD129. It must be able to receive data from sensors, to process and filter them to determine attitude and at the end to compute control;

simulated system: physical parts and environment, comprehensive of dynamics and kinematics of the satellite, sensors (IMU, magnetometer, solar panels as sun sensors) and actuators (magnetic torquers and reaction wheel). All simulated parts are:

- dynamics and kinematics;
- IMU;
- magnetometer;
- solar panels;
- orbital position;
- magnetic torquers (coils);
- reaction wheel.

Knowing the orbital position it is possible to obtain the local sun vector and, together with the simulated magnetometer, the local magnetic field, while the simulated IMU gives the angular velocities of the satellite. In the same way, knowing the sun vector, it is possible to calculate the power fluxes and the temperatures on the solar panels. Combining this data, ARM is able to determine attitude, to compare it with desired attitude and so to compute necessary control to reach the target.

The configuration adopted for HIL simulation (whose design has been done by Lorenzo Feruglio, MSc student of Aerospace Engineering) is shown in Figure 6.1, where starting from the left is possibile to see:

simulator console: it is the computer on which the HIL (or SIL) simulation is run;

core: simulator core has been used to correctly launch the process for HIL simulation in hard real-time mode (here is shown only the scheduler, but there are also other modules). Links between modules in the core and in the HIL process are software links;

HIL process: it includes the parts that need to be simulated, that is environment, dynamics, kinematics, sensors and actuators. Its operation is managed by the scheduler and it transmits to the ARM RD129, through serial port SAC0, data (angular velocity and Earth magnetic field) required for determining attitude and for calculating necessary control to reach desired configuration;

ARM RD129: as said before, this is the test object so is tested the goodness of the algorithms of attitude determination and control. Actually it send data (duty cycle for magnetic torquers and reaction wheel) using debug port to HIL process to operate the parts simulated, but in future HIL simulations the real IMU and real magnetic torquers will be added.

In Figure 6.2 is possible to see the real configuration: on the left there is a laptop running HIL process, on the center the ADCS board with ARM RD129 while on the

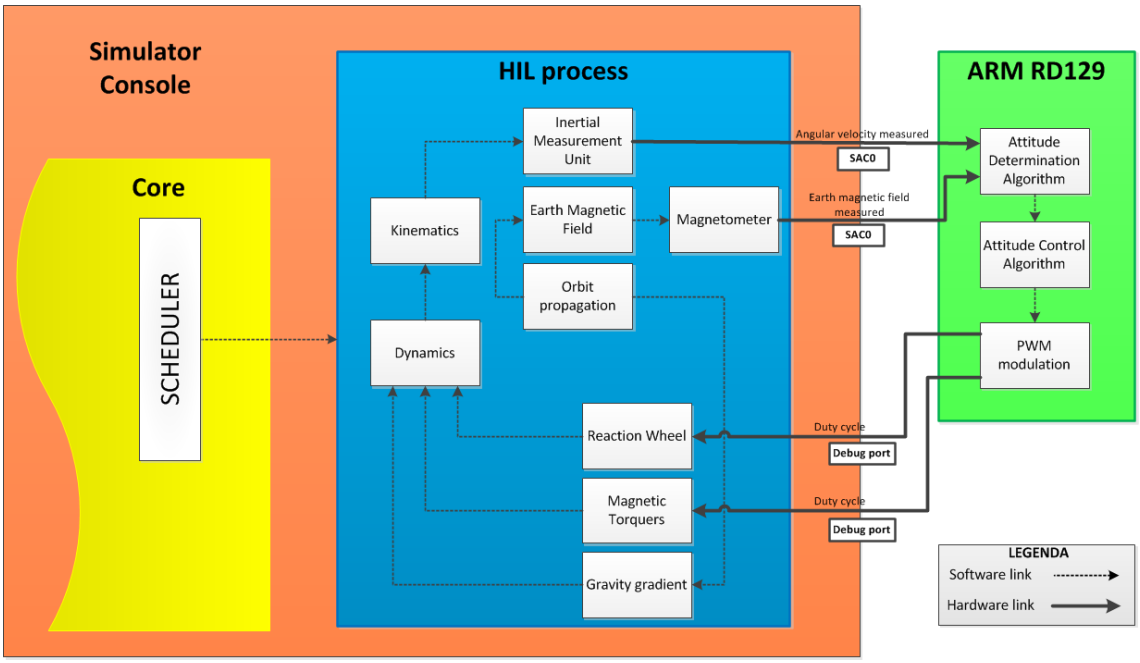


Figure 6.1 – Configuration of HIL simulation.

right there is the qualification model of *e-st@r* with magnetic torquers used for this simulation.

Before of the HIL simulation, a SIL simulation has been realized (Figure 6.3): it is possible to recognize that both the HIL process and the control process are coded in target software language but are run on the simulator machine. The inter-process communication has been implemented using named pipes to simulate the two serial ports and cables on which the normal simulation would transmit data. This test step has the potentiality of allowing several tests being run at once: in this configuration, the only thing needed was a UNIX PC to run an instance of this simulation. This way, it has been possible to run up to 5 instances (this number dictated by the availability of PC in the laboratory). It has not been tried to run more than one instance per computer, despite it being possible, for stability reason. Once verified that the entire software (HIL process and control software on ARM RD129) has no problems (bugs), it is proceeded to validate the configuration on a Hardware in the Loop simulation.

6.1 *C* control logic

In previous chapter, control theory and its Matlab implementation have been analysed, now there is a brief description of the entire control logic used for HIL simulation:

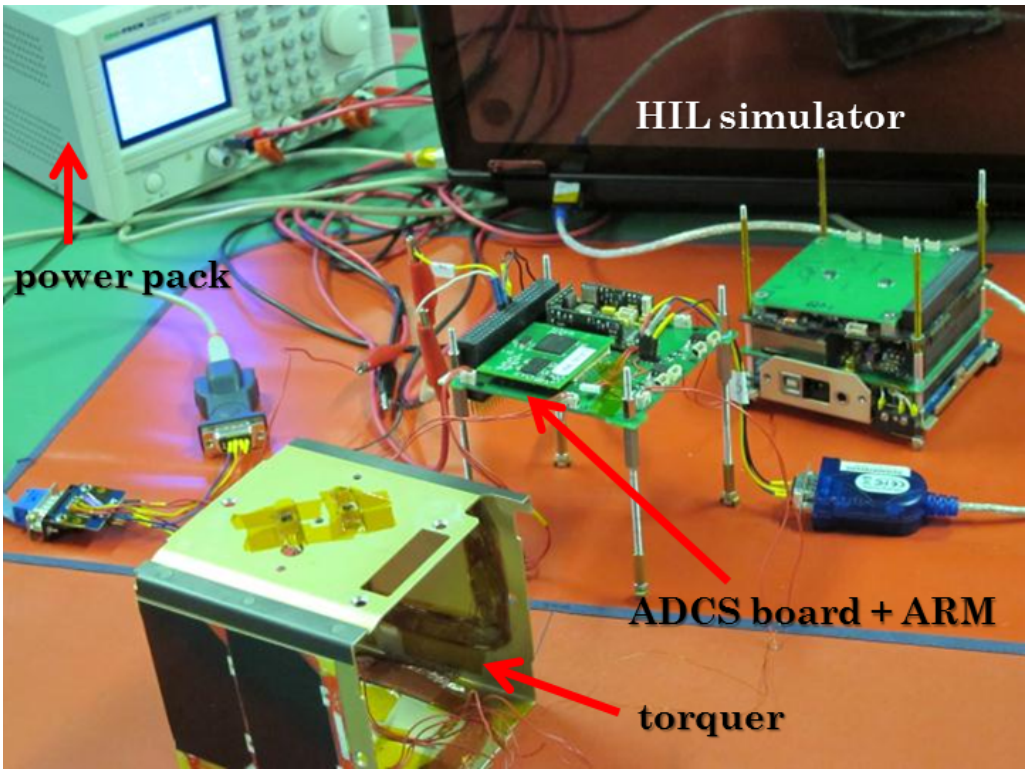


Figure 6.2 – Photo of HIL simulation.

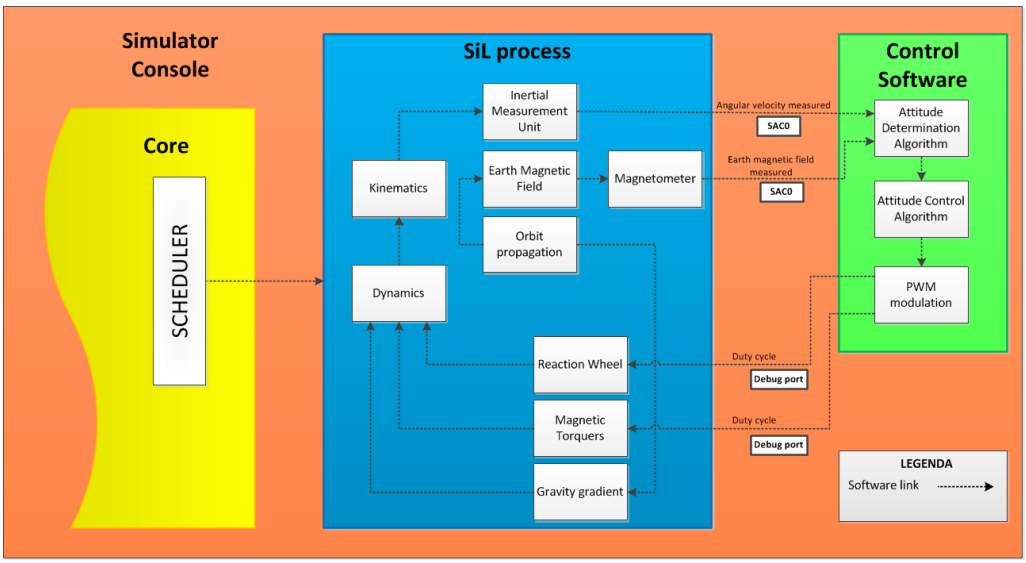


Figure 6.3 – Configuration of SIL simulation.

it was completely written in C, but a part of functions performed is elaborated by HIL process (then they are compiled to run on UNIX platform) and a part to be performed by ARM RD129 (so they cross-compiled for embedded UNIX).

The outputs of the program for the attitude control are the commands for the magnetic torquers and reaction wheel given as percentage of the maximum applicable control, that is duty cycle, while needed inputs are:

time: given by the ARM RD129 as date and milliseconds from the power-up (it is very important to synchronize functions execution between the HIL process and ARM RD129);

magnetic field: in Body coordinates given by the magnetometer (simulated by HIL process);

inertial angular velocity: in Body coordinates given by the IMU (simulated by HIL process);

norad: it is a vector containing orbital parameters and it is evaluated by HIL process;

q_d**:** it is desired attitude, expressed by quaternion;

Conceptually, the simulation follows this logical process (see Figure 6.4):

1. only at first cycle, control software send a string through debug port with default values, its task is to activate the HIL process;

2. control software, loaded on ARM processor, starts integration with initial data set on its memory and then it determines ideal control;

3. real controls (duty cycle) are calculated by PWM function and sent via hardware link (a debug port) to the simulator, more precisely to the magnetic torquers model and to the reaction wheel model;

4. output of these modules are used by dynamics and kinematics modules, where integrations are performed;

5. results of this integrations are passed to the IMU model (see Section 4.3), that takes as input the ideal angular velocities obtained from the integrations and gives as output measured velocities;

6. at the same time magnetometer model(see Section 4.2) works on input (ideal Earth magnetic field calculated with a proper function) to obtain a measured output, which is;

7. loop is now closed, since the measured velocities from the IMU and measured Earth magnetic field from magnetometer are sent via hardware link (a serial cable) to the ARM, on which is loaded attitude determination algorithm.

For this test the construction of the string sent from HIL process to ARM RD129 is slightly different from that seen previously (that is the string sent from IMU to ARM RD129, Table 6.1), because for simulation purpose it has been necessary to include magnetometer measurements: they have been added to this string, after the Z (ending character of previous string) and followed by a M that symbolize the end of that portion of string. Last piece of string includes the communication of the dt needed for integration (in reality this is compared with that calculated by the processor) and the new ending character is T.

Binary string from HIL to ARM										
# byte	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
data	A	c_1	c_2	ax_1	ax_2	ay_1	ay_2	az_1	az_2	gx_1
	10	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*
	gx_2	gy_1	gy_2	gz_1	gz_2	Z	Bx_{1-1}	Bx_{1-2}	Bx_{1-3}	Bx_{1-4}
	20	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*
	Bx_{2-1}	Bx_{2-2}	Bx_{2-3}	Bx_{2-4}	Bx_{3-1}	Bx_{3-2}	Bx_{3-3}	Bx_{3-4}	M	dt_1
	30	*31*	*32*	*33*	*34*	*35*	*36*	*37*		
	dt_2	dt_3	dt_4	dt_5	dt_6	dt_7	dt_8	T		

Table 6.1 – Binary string from HIL process to ARM RD129.

On the contrary, string sent from ARM RD129 to HIL process to contains only 4 starting character (H, I, T, L) that allow detection of the string and then the duty cycle values determined by ARM RD129 with control algorithm. Its structure is shown in Table 6.2).

Binary string from ARM to HIL												
# byte	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
data	H	I	T	L	$dcMTx_1$	$dcMTx_2$	$dcMTy_1$	$dcMTy_2$	$dcMTz_1$	$dcMTz_2$	$dcRW_1$	$dcRW_2$

Table 6.2 – Binary string from ARM RD129 to HIL process.

Now the different functions used will be analyzed, following the logic flow in Figure 6.4.

6.1.1 Earth magnetic field and orbit propagation

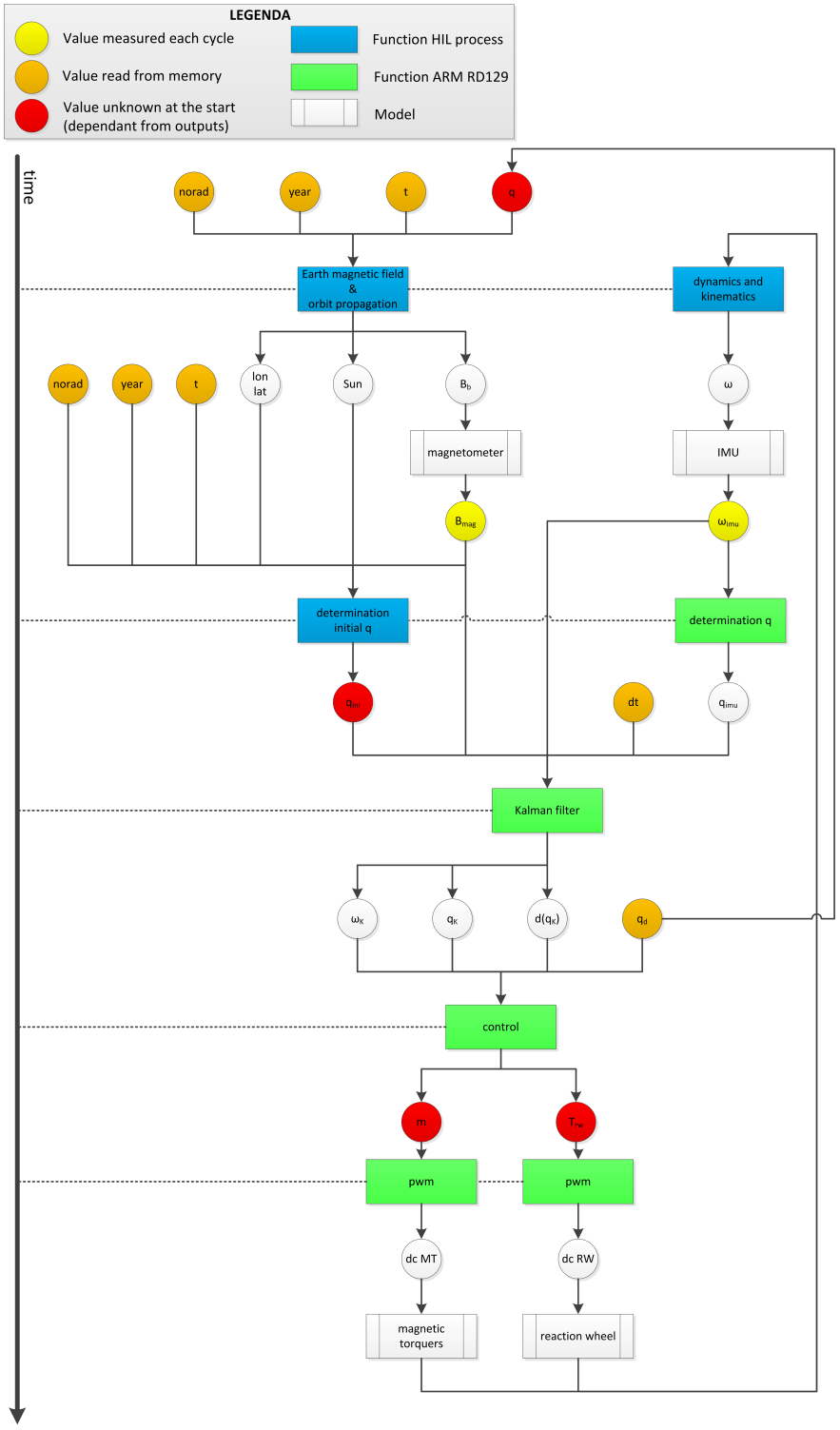


Figure 6.4 – Timeline of functions performed by HIL simulation.

Listing 6.1 – Earth magnetic field and orbit propagation functions.

```
(Sb[3],Bb[3],LL[3]) = magf(norad[8],anno,t,q[4])
```

The calculation of the orbital position as latitude, longitude and altitude (LL vector) is done by the same code that calculate the sun vector (Sb vector) and magnetic field vector (Bb vector) in the Body reference frame. However during the first cycle of the program this is not possible because there is not information about the angular position given by the quaternion (q vector) so the code evaluate a magnetic field vector based on a casual quaternion.

The calculation of the orbital position is the same as proceeding seen in Section 2.3.2.1 and the magnetic field vector in local coordinates (NED) is obtained from the WMM as in Section 2.4.1. The rotation from the NED to the Body reference frame is done with these sequential rotations (previously defined in Section 2.1.1 and Section 2.1.2):

- from NED to ECEF;
- from ECEF to Inertial;
- from Inertial to Orbital;
- from Orbital to Body.

Regarding the sun vector calculation, is used a model that gives the vector in NED coordinates and then it is rotated, as the magnetic field vector, in Body coordinates. The algorithm for the calculation of the sun vector needs as inputs the satellite position as latitude, longitude and altitude and gives as outputs the three components of the vector in the local reference system (NED) using an algorithm using an algorithm that computes:

- Julian date (jd) and Julian day (d);
- Keplerian elements of the Sun;
- auxiliary angle;
- rectangular coordinates in the ecliptic plane;
- distance and true anomaly;
- Sun longitude;
- rectangular coordinates of the ecliptic;
- rotation of the coordinates to the equatorial rectangular reference frame;

- conversion to right ascension and declination;
- number of days from the 1/01/2000;
- local sidereal time;
- substitution of the right ascension with the hours angle;
- conversion to rectangular coordinates;
- rotation along the east-west axis;
- Azimuth e Elevation;
- NED coordinates.

6.1.2 Dynamics, kinematics and determination of *q*

These functions are a translation in *C* code of equations (omitted here) seen in Section 3.3, so it uses all torques acting on satellite to evaluate its behavior (angular velocity) and then, after an appropriate integration, it determines attitude (quaternion).

6.1.3 Determination initial *q*

Listing 6.2 – Determination initial *q* function.

```
qini[4] = detq(B[3],S[3],LL[3],anno,t,norad[8])
```

The attitude determination based on measures of magnetic field and sun vector is necessary because the satellite, following the CubeSat standard, will be switched-on some minutes after being released by the P-POD. This sets the problem to know the initial attitude of *3STAR* (*qini* vector). The calculation of the attitude of the satellite is based on the work of Markley [19]: it is possible, from two measured vectors in a reference systems and knowing the same vectors in another reference system, calculate the rotation present between these two reference systems.

6.1.4 Kalman filter

Listing 6.3 – Kalman filter function.

```
(qK[4],qKdot[4],wKib[4]) = kalman(m[3],wMib[3],dt,Bb[3],Trw)
```

This function is a (not so simply) translation in C code of theory and Matlab implementation analyzed in Section 4.4: it uses as input controls (m vector, that is dipole moment of mangetic torquers, and Trw, that is reaction wheel torque), angular velocity measured ($wMib$ vector) to evaluate quaternion vector qK, derivative of quaternion vector $qKdot$ and angular velocity $wKib$.

6.1.5 Control

Listing 6.4 – Control function.

```
(m[3],Trw) = controllosat(qc[4],qK[4],qKdot[4],wKib[3],Bb[3])
```

As previously seen for MATLAB®-Simulink®model, there are two different algorithms for attitude control, one for the detumbling phase (PD controller) and the other for the stabilization phase (LQR controller) of the satellite. Obviously algorithms implemented are the same seen in Section 4.5. Regarding the choice of which controller is used, the value of angular velocity of the satellite is analyzed with Listing 6.5: if at least one component exceeds the threshold, the controller uses the detumbling algorithm, otherwise goes to the stabilization one.

Listing 6.5 – Control selector code.

```
for(ii=0;ii<3;ii++){
  if(w[ii]<0){
    ww[ii]=-w[ii];
  }else{
    ww[ii]=w[ii];
  }
}
if(ww[0]>0.001 && ww[1]>0.001 && ww[2]>0.001){
  //PD detumbling controller
  }else{
  //stabilization controller
}
```

All controls evaluated, for both magnetic torquers and reaction wheel, have a saturation: for magnetic torquer tha saturation is on maximum dipole moment while for reaction wheel it is on supply voltage (Listing 6.6).

Listing 6.6 – Control saturation code.

```
for(ii=0;ii<3;ii++){
  if(m[ii]>0.5){
    m[ii] = 0.5;
  }
```

```
  if(m[ii]<-0.5){
    m[ii] = -0.1;
  }
}
if (Vrw > Vsat) {
    Vrw = Vsat;
    }
if (Vrw < Vsat) {
    Vrw = -Vsat;
    }
```

6.1.6 PWM

Listing 6.7 – PWM function.

```
dc[4] = torquers(m[3],Vrw)
```

The last step of the code is the calculation, from ideal control values, of the duty-cycle to be applied to magnetic torquers and reaction wheel to obtain the desired controls: the code is based on the discretization of the controls, considering the maximum available current for magnetic torquers and maximum supply voltage for reaction wheele, on an equispaced range from 0 to 1000, where the value 500 corresponds with null control. Indeed, because PWM pins of ARM RD129 allow only a value from 1 to 999, minimum control (duty cycle equal to 0) and maximum control (duty cycle equal to 1000) are slightly changed as in Listing 6.8.

Listing 6.8 – PWM refining code.

```
if (dc[0]==0)
   dc[0]=1;
if (dc[1]==0)
   dc[1]=1;
if (dc[2]==0)
   dc[2]=1;
if (dc[3]==0)
   dc[3]=1;
if (dc[0]==1000)
   dc[0]=999;
if (dc[1]==1000)
   dc[1]=999;
if (dc[2]==1000)
   dc[2]=999;
if (dc[3]==1000)
```

```
dc[3]=999;
```

6.2 HIL procedure

The implementation of the algorithms previously defined on the *3STAR* ADCS unit and on HIL simulation laptop completes the preparations for the HIL simulation. The procedure for the *3STAR* HIL simulation is summarized in Table 6.3 and it defines the verification activity flow and it provides detailed instructions about specific steps QA personnel has to follow.

Table 6.3 – ADCS HIL simulation procedure.

# step	***Activity description***
1	*Test set up*: take the antistatic pad, lay it on the table, and make all the necessary electrical connections to ground; wear the antistatic bracelet; wear the latex gloves.
2	*Assembly*: connect the four rods in the apposite housings; insert four spacers; install the card on the appropriate support structure for the test.
3	*Connections*: start simulation PC; connect cable 0-3(0) to USB0 port on simulation laptop; connect cable 2 to USB2 port on simulation laptop; connect cable 2 to IMU port on the ADCS board; connect cable 0-3(out) to RS232 adapter; connect RS232 adapter output cable on connector C1 on the ADCS; connect cable debug to USB0 por on debug ADCS laptop; connect power cable 1 of power pack to 5V pin of ADCS board (settings: 5$[V]$, 0.5$[A]$); connect power cable 2 of power pack to 3.3V pin of ADCS board (settings: 3.3$[V]$, 0.5$[A]$); connect power cable 3 of power pack to RS232 adapter (settings: 4$[V]$, 0.03$[A]$); open *minicom* terminal on debug ADCS laptop on ttyUSB0 port.

Table 6.3: continues on next page

Table 6.3: continues from the previous page

# step	*Activity description*
4	*Start test*: turn on power pack; start HIL code on simulation laptop start ADCS code on satellite through *minicom* on ttyUSB0

6.3 Results

The analysis of the Hardware In the Loop simulation results for the *3STAR* ADCS is now considered. Test data are collected through three different channels:

- HIL process log file, saved on HIL simulation laptop;
- ARM RD129 log file, saved directly on ARM;
- ADCS debug screen, it shows the instantaneous ADCS status and it does not have a log file but it allows the system debug and to control correct behavior of the ADCS program compared to the simulations.

A simulation of about 28 hours has been performed in real time via Hardware In the Loop simulation and the attitude trend obtained is reported in Figure 6.5.

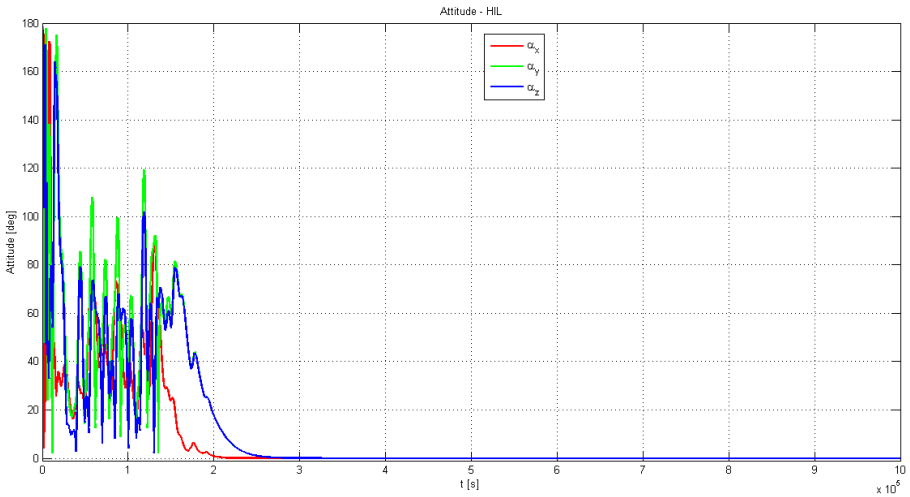


Figure 6.5 – Attitude trend obtained via HIL simulation (ARM RD129 log file).

As noticeable from the curves reported above the HIL simulation gives a behavior similar to that obtained with the MATLAB®-Simulink®simulations of Section 4.6.

Obviously these are approximated trends because data are saved every fixed times and not continuously as in MATLAB®-Simulink®. In addition the differences are due, however, by the fact that the presence of real components introduces a series of problems related to the compatibility and the delays in communications.

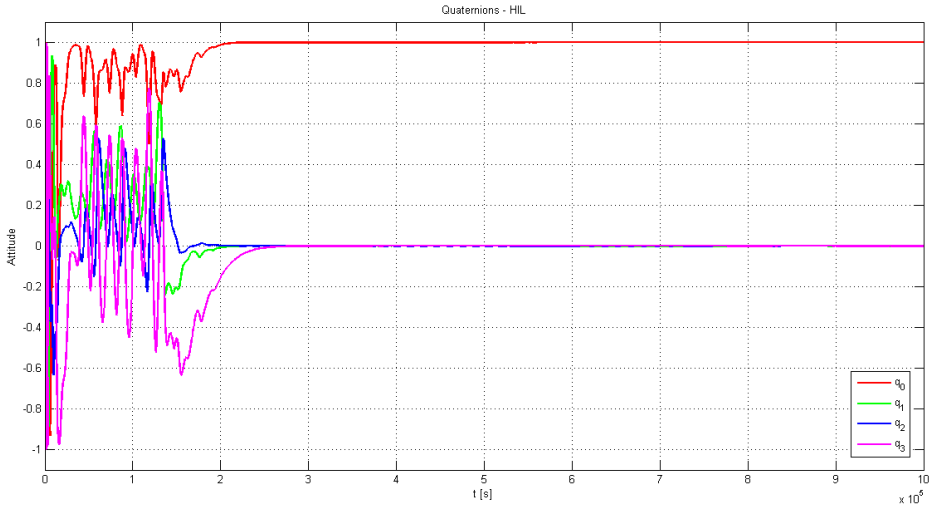


Figure 6.6 – Quaternion trend obtained via HIL simulation (ARM RD129 log file).

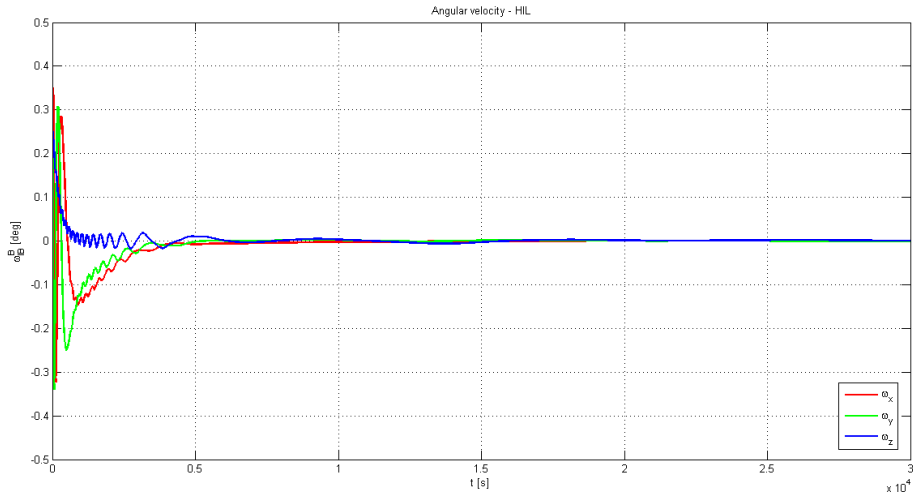


Figure 6.7 – Angular velocity trend obtained via HIL simulation (ARM RD129 log file).

Chapter 7

Conclusions

The ADCS is a limit of CubeSats for future commercial applications, as to accommodate a paying payload is necessary to ensure a certain pointing accuracy. In anticipation of use of CubeSats, as well as for scientific purposes but also for commercial purposes, the creation of an active ADCS is a fundamental point.

An active Attitude Determination and Control System for nanosatellite has been developed at the Systems and Technologies for Aerospace Research (STAR) laboratory of DIMEAS; individual components and entire subsystem have been tested via Hardware In the Loop simulation.

A simulation model of the complete system was carried on MATLAB®-Simulink®to study the behavior of the satellite and also to optimize different algorithms (in particularly control algorithms) before the future implementation on board.

Finally, after a test phase of individual equipments, like Inertial Measurement Unit (Atomic IMU 6 Degrees of Freedom) and microprocessor (ARM RD129), ADCS control logic has been installed on a prototype of *3STAR* ADCS board and tested with an Hardware In the Loop simulation.

The results, those of MATLAB®-Simulink®simulations and those of HIL simulation, are analyzed and show a good behavior, with stabilization time and power consumption not very high and therefore acceptable.

Starting from models obtained and components tested, is possible to optimize ADCS improving them, eg. increasing the complexity of models or introducing voting functions, always using HIL simulation to properly observe system behavior; moreover, an optimization of the satellite ADCS is possible both for what the power consumption and the manoeuvres capability are concerned.

Bibliography

[1] The CubeSat Program (2009), *CubeSat Design Specification Rev. 12*, California Polytechnic State University.

[2] Darío Hermida, Jorge Iglesias and Marcos Arias (2010), *HUMSAT mission requirements document, HUM-22000-MRD-001-UVIGO*, Universidade de Vigo.

[3] Darío Hermida, Jorge Iglesias and Marcos Arias (2010), *HUMSAT system requirements document, HUM-22000-SRD-001-UVIGO*, Universidade de Vigo.

[4] Jean Meeus (2005), *Astronomical algorithms*, Hardbound.

[5] ESA (2005), *Science programme*, http://www.esa.int/esaSC/.

[6] Sabrina Corpino (2009), *Sistemi spaziali*, Politecnico di Torino.

[7] Nicole Viola (2006), *Modellazione, simulazione e sperimentazione di sistemi aerospaziali*, Politecnico di Torino.

[8] British Geological Survey (2010), *Overview of Geomagnetism*, http://www.geomag.bgs.ac.uk/education/earthmag.html.

[9] ECSS Secretariat ESA-ESTEC (2008), *ECSS-E-ST-10-04C*, Requirements & Standards Division Noordwijk, The Netherlands..

[10] Alan C. Tribble (2003), *The Space Environment: Implications for Spacecraft Design*, Princeton University Press.

[11] NOAA National Geophysical Data Center (2009), *World Magnetic Model 2010-2015*, http://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml.

[12] Stefan Maus, Susan Macmillan, Susan McLean, Brian Hamilton, Manoj Nair, Alan Thomson and Craig Rollins (NGA) (2009), *The US/UK World Magnetic Model for 2010-2015*, NOAA National Geophysical Data Center.

[13] J. R. Wertz (1994), *Spacecraft Attitude Determination and Control - 1st Edition*, Kluwer Academic Publishers.

[14] The MathWorks Inc. (2012), *Matlab R2012a Documentation*, http://www.mathworks.it/help/toolbox/control/ref/kalman.html.

[15] Pages 371-372 (2009), *Journal of Aerospace Engineering: Proceedings of the Institution of Mechanical En- gineers Part G*, Professional Engineer Publishing.

[16] G. Baldo Carvalho, S. Theil and H. Koiti Kuga. (2009), *IMU: generic model development approach*, V Simpòsio Brasileiro de Engenharia Inercial.

[17] SparkFun Electronics (2012), *Atomic IMU 6 Degrees of Freedom*, http://www.sparkfun.com/products/9812.

[18] ELPA (2011), *CPU Board RD129 manual*, http://www.elpa.it/rd129it.html.

[19] F. Landis Markley. (2002), *Fast quaternion attitude estimation from two vector measurements*, NASAs Goddard Space Flight Center, Guidance, Navigation and Control Systems Engineering Branch.