POLITECNICO DI TORINO

Master's Degree in Aerospace Engineering

Master's Degree Thesis

Implementation and testing of modular ADCS software using modular avionics test bench for cubesats

Supervisor

Manuela BATTIPEDE

Mentors

Ran QEDAR

Saish SRIDHARAN

Candidate

Riccardo SPARTÀ

Academic year 2023/2024

Summary

In recent years, the persistent boost in the New Space Economy is allowing more and more European realities to develop their different ideas: building their reusable launcher, miniaturized fleets of satellites, or commercial-off-the-shelf space components. Since the competition is growing exponentially, time is increasingly one of the most critical factors in the development process, particularly during the satellite subsystems' integration and testing phase. In addition, the increasing number of Space suppliers results in both a broader opportunity of choice during the design of a satellite and also in a wide variety of protocols and interfaces, which is necessary to deal with higher complexity, cost, and, again, longer time of development.

According to these demands, the primary objectives of this work are to enhance the reliability and validate the performances of a modular avionics test bench by implementing and testing a CubeSat software solution embedded into an On-Board Computer unit, capable of communicating with different components provided by various suppliers available in the market. The focus is to set up, integrate, and test a modular architecture on the bench for one of the most crucial subsystems of a satellite: the **ADCS**.

The choice to focus on *CubeSat* applications is based on the direction of the New Space Economy towards more cost-effective, flexible, agile, and technologically advanced solutions. Following *Modular* design principles aims to reduce development time and cost by designing individual building blocks or modules, which must be interchangeable and adaptable to accommodate various mission concepts.

The choice to test an *ADCS* application concerns the high level of complexity in the relative integration and testing phases, which involve different components from several European suppliers that have to cooperate and communicate with each other to meet the expected performance within the timeline available. For these purposes, the **modular avionic test bench** is an indispensable tool for rapid system integration and testing, providing a controlled and repeatable environment for any subsystem of small satellite applications. The Cubesat OBC unit also represents cutting-edge technology to ensure the capability to communicate with all the components integrated on the bench easily.

Hardware-in-the-loop (HIL) simulations, involving both software and hardware

setup to be tested, have allowed us to thoroughly assess the system's performance, resilience, and responsiveness, guaranteeing the system's robustness under varying conditions and scenarios.
In conclusion, the outcomes of this work contribute to the advancement of CubeSat testing technologies by offering a versatile and cost-efficient solution for ADCS development that is not limited to this application. Moving forward, further research may be needed to extend even more the possible applications and implications that this technology could have in the future Space Economy, increasing the degree of accuracy that could be provided. Overall, the study underscores the importance of technology's evolution in improving testing methodologies and tools for space projects.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof.Battipede, for the invaluable opportunity proposed to me and for the support and encouragement throughout the experience. Her humanity and availability have been instrumental in making this work greater.

I am also immensely thankful to my mentors at SPiN, Ran Qedar and Saish Sridharan, for their insightful professional teachings and suggestions that have enhanced the quality of the activities performed and have introduced me to the space industry environment. Thanks to them, I had the privilege to put into practice all the knowledge learned during my 5-year academic formation, directly interacting with cutting-edge technological solutions for Space sector.

Most heartfelt thanks to my parents, Cristina and Sebastiano, and grandparents, Fiorenza and Piero, for their unwavering love, encouragement, and understanding during this journey. Their omnipresent closeness has been the main source of strength and motivation for me. Specifically, I would like to point out the unreplaceable relationship and role of my mother along the whole path: without her, I would not have become the person that I am today.

I would also like to express my gratitude to Silvia, for being there for me throughout all of this, supporting every choice and decision, putting our wellness before everything else. And for sure, I do not forget Angelo, a great friend and comrade of a thousand battles, who has been able to teach me countless lessons about life and beyond, and above all the value of true friendship.

Lastly, I extend my appreciation to all my childhood friends and colleagues who have accompanied me along this challenging chapter of my life.

"The harder the battle, the sweeter the victory"
Less Brown

Table of Contents

List of Tables

List of Figures

Acronyms

ADCS

Attitude Determination and Control System

API

Application Program Interface

GUI

Graphical User Interface

HiL

Hardware-in-the-Loop

MA61C

Multipurpose Adapter Generic Interface Connector

MATB

Modular Avionic Test Bench

OBC

Onboard Computer

Chapter 1

Introduction

To introduce properly the work that will be presented, it is crucial to have an overview of the most important fields that are going to be touched throughout the following chapters: starting with the CubeSats environment, the Attitude Determination and Control subsystem, the importance of Test Bench, up to the projects taken as a reference for the study.
There will be presented also the main objectives and the outline of the discussion, to give an idea of what will be the logical flow followed.

1.1 General Contest

The Attitude Determination and Control System (ADCS) plays a crucial role in the successful operation of CubeSats. It is responsible for accurately determining and controlling the satellite's attitude or orientation in space. As CubeSats continue to gain popularity due to their compact size and low cost, the need for reliable and efficient ADCS solutions becomes paramount. In order to ensure the effectiveness and reliability of ADCS software, rigorous testing is essential.

1.1.1 The CubeSat Standard and Model

The aim of this section is to understand the "CubeSat" designation, compared to the other satellites' categories, using [1] as a reference.
A Small Satellite is generally considered to be any satellite that weighs less than 300 kg (1,100 lb). A CubeSat, however, must conform to specific criteria that control factors such as its shape, size, and weight. The very specific standards for CubeSats help reduce costs and make it possible for companies to mass-produce

components and offer off-the-shelf parts. As a result, the engineering and development of CubeSats becomes less costly than highly customized small satellites. The standardized shape and size also reduce costs associated with transporting to, and deploying them into, space.
CubeSats come in several sizes, which are based on the standard CubeSat “unit”, referred to as a 1U. A 1U CubeSat is a 10 cm cube with a mass of approximately 1 to 1.33 kg. In the years since the CubeSat’s inception, larger sizes have become popular, such as the 1.5U, 2U, 3U, 6U, and 12U.
One example of a set of general requirements for all Cubesats is CDS, CubeSat Design Specification.
Examples of a 1U and 3U are shown in figure 1.1.

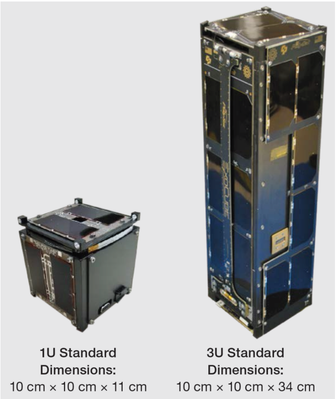


Figure 1.1: 1U Cubesat CP1 (left) and 3U Cubesat CP10 (right) [1]

1.1.2 Introduction to ADCS

The focus of the study, as already specified, will be in particular on the Attitude Determination and Control System, therefore it is important to have a clear idea of what it is aimed at.
Every spacecraft has 6 degrees of freedom (DOF): three of them are displacement, and the rest of them are rotation. The attitude determination and control system (ADCS) is responsible for the 3 DOF rotational control of the spacecraft.
The control of the satellite can be achieved passively or actively, and the present work will be focused on the second one.
Passive ADCS can be enough for CubeSats that do not need accurate positioning in their orbit, but the more complex missions require precise ADCS. Traditional ADCS, designed for full-scale satellites, cannot be used for nanosatellite needs due to their sizes, price, or power consumption.
CubeSat developers have to build ADCS using standard hardware or design their

own system.

To perform its assigned functions, the system consists of both hardware and software.

First, it is necessary to know what is the actual orientation of the vehicle: this process is called attitude determination. Then, a torque has to be applied to rotate the satellite to the desired orientation, called attitude control.

The first thing to notice in attitude determination is the “reference frame”.

The spacecraft’s attitude refers to its rotational relationship to some other reference frames adhered to an inertial frame like Earth-Centered Earth-Fixed Frame (ECEF) or Earth-Centered Inertial Frame (ECI), that will be discussed in the following chapter 2.

The most common attitude sensors used for satellites orbiting the earth are mainly (Coarse/Fine) Sun sensor, Earth sensor, IMU, Star tracker, and Geomagnetic sensor.

Regarding attitude control, the reaction wheel, Control Moment Gyro (CMG), thruster, and magnetorquer are the most popular actuators on the market. The main purposes of these actuators are to provide torque to compensate the disturbance torques on-orbit for stabilization, such as gravity-gradient torque, radiation pressure torque, aerodynamic drag torque, etc. and to apply yaw, roll, and pitch angle control for a satellite.

All the components listed are part of the hardware. As far as the software is concerned, this is divided into two sub-parts: the attitude determination algorithm, used to determine the current attitude of the satellite, starting from the input from the sensors, and the control algorithm, which takes the current attitude of the satellite as an input, processes it and, by comparing the current position with the desired one, calculates the actuation to bring the error between the two configurations to zero.

These two parts of the software are interdependent and constitute the control loop, for which an example is reported in figure 1.2.

The state-of-the-art for this subsystem is well and widely discussed in [3].

The future of this subsystem is going to be directed to an integrated solution, in order to satisfy most of the mission requirements while providing reliability and shortening up customer development time.

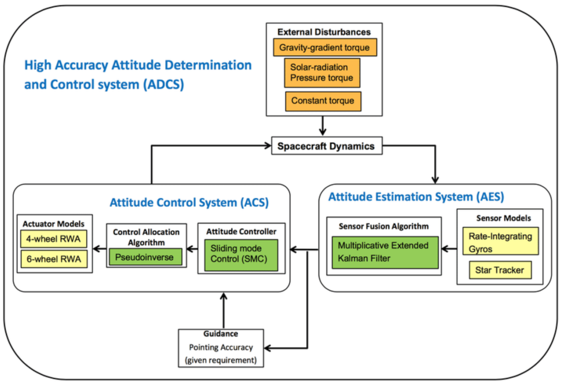


Figure 1.2: ADCS logic loop [2]

1.1.3 Test Bench for ADCS

A 'test bench' allows the overall working of the mission implementation to be simulated with software. When a particular piece of equipment is completed it can be plugged into this virtual mission platform to be tested.
An example from BepiColombo Engineering Test Bench (ETB) currently based in Airbus Defence and Space's Friedrichshafen facility can be seen in figure 1.3.

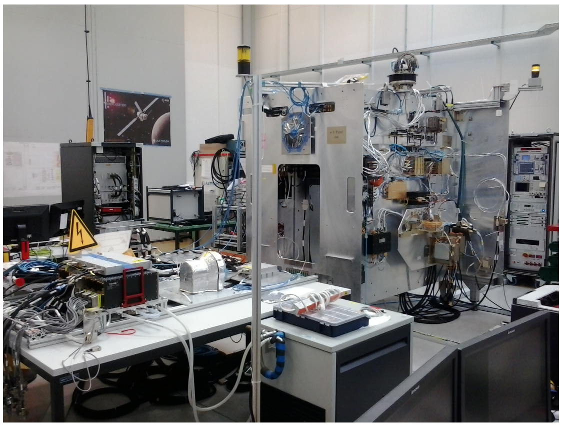


Figure 1.3: Example of Satellite Test Bench [4]

Such test benches offer many prospects for ground testing of CubeSats and their subsystems, and first of all Attitude Determination and Control System (ADCS). The test bench serves as a controlled environment for evaluating the performance and functionality of the ADCS software in simulated space conditions. It allows for comprehensive testing, verification, and validation of the software, ensuring its suitability for deployment in actual CubeSat missions.
The difficulty of ground satellite tests is to create conditions simulating the space

environment with given accuracy while limiting the effects that occur on ground facilities. For nanosatellites, particularly, the accuracy of ground tests has to be very high due to the reduced capability of actuators and high sensitivity to external forces.

This kind of test is essential especially for ADCS, as it is a sophisticated system which needs validation testing to prove its operability and verify software compatibility with hardware and other subsystems.

This procedure is widely used for full-size satellites, but testing of nanosatellites comes with many difficulties. The efforts required to set up nanosatellite ADCS ground testing are generally comparable to the efforts needed for building a CubeSat. Although every element of the system can be individually verified before flight and computer simulations can be made, only satellite operation in the space environment shows if all components work together correctly. Otherwise, one small mistake often leads to mission failure. Despite a CubeSat is relatively inexpensive, one failure on the orbit is costly in terms of money and time due to payload preparation and launch. After several CubeSats have been lost in orbit, the necessity of efficient ground testing system for nanosatellite-class ADCS became evident.

Full-scale satellite developers have arrived at the same conclusion when the first generation of satellites with complex orientation systems was built some fifty years ago. Since then, the technology of ADCS ground testing evolved and has become a common procedure for full-size satellites. For correct operation and valid results of the testing, the precision of the ADCS test bench has to correlate with the accuracy of the satellite control system, the capabilities of its actuators and the magnitudes of disturbing torques acting on the satellite in its orbit. Unfortunately, it cannot be simply scaled down. Due to this limitation, test benches designed for full-scale satellites are not efficient for CubeSats.

Further details about that matter can be found on [5].

1.2 SPiN Projects

It is undoubtedly important also to give context to the work presented, since it has been started from the previous development of the following projects described.

1.2.1 Multipurpose Adapter Generic Interface Connector (MA61C)

The Multipurpose Adapter Generic Interface Connector (MA61C) is an intelligent system that can scan connections, detect incoming data, route and convert them between interfaces. The bus interface allows connecting different subsystems to the

onboard computer without driver installation or user configuration. The adapter monitors data on the bus and sends it to connected units via a USB interface.
The core of MA61C is the GR712RC LEON3-FT SPARC V8 processor which supports multiple interfaces that can be used as both inputs and outputs. The processor has a built-in timer based on an onboard oscillator.
The MA61C holds a database of drivers, enabling plug-and-play functionalities, such as device recognition, self-configuration and driver installation. The MA61C is equipped with an internal memory of 64Mbit of SRAM and 64Mbit of FLASH for storage of onboard software, drivers, and data.
Ma61C is bundled with an Application Programming Interface (API) to command and monitor the units connected directly to the spacecraft onboard computer or connected to a PC through a USB interface. A Graphical User Interface (GUI) for Windows is developed to monitor and command connected sub-systems more easily.
Basically, the MA61C is an interface to the satellite subsystems, such as AOCS sensors and actuators, communication, power and payload subsystems, etc.
It adapts the communication interface and protocol standards of different suppliers to the unique standard of the onboard computer.
For further information regarding the initial problem statement, the motivation, the approach adopted in the choice of designing this solution and its first space application, it is useful to refer to [6] and [7].
It is crucial, as well, to choose the MA61C version that corresponds precisely to the particular project of interest, to connect all the subsystems in a plug-and-play fashion through the most commonly used space interfaces, without losing time and money on the integration. For that reason, there are mainly four versions:

- **MA61C Cubesat**, solution for 1U to 16U satellites, that automatically adapts between the communication interfaces and protocol standards of different suppliers to the unique standard of the onboard software. This simplifies the communication between the onboard software of the CubeSat and its subsystems such as AOCS sensors and actuators, communication, power and payload.

- **MA61C SmallSat**, solution for satellites up to 150 kg, that plays the same role as the version above but for SmallSats.

- **MA61C cPCI**, a solution for satellites designed with cPCI serial backplane technology, an industrial standard for modular computer systems.

- **MA61C EGSE**, a USB to some different interfaces adapter that holds electronic data sheet (EDS) to perform plug-and-play functionalities and could be used as a Swiss knife tool for early digital validation in different projects.

For more details of this solution, interesting insights are presented in [8].

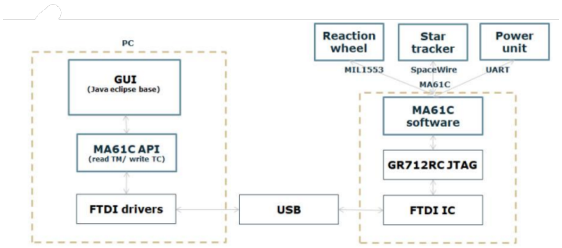


Figure 1.4: Block Diagram of MA61C EGSE to Subsystems [9]

In particular the study will be focused on the first version, related to the CubeSat solution, represented in figure 1.5.

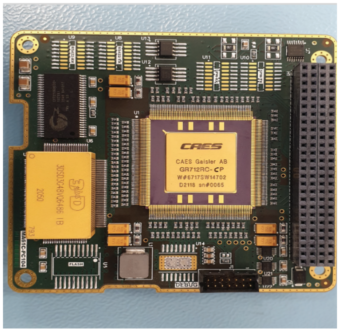


Figure 1.5: MA61C Cubesat version [10]

To have a clearer understanding of what will be described in 3, an overview of its embedded software is presented below, describing the most relevant parts, from the firmware to the processor.

MA61C Firmware

It is an abstraction layer between the application and the lower-level hardware control: it can be mounted directly on an embedded operating system such as RTEMS (Real-Time Executive for Multiprocessor Systems, an open source Real Time Operating System), and support one or more applications running in parallel. In the current design, the firmware is passive and only works when it is called upon by the main application.
The use of that firmware is open to all applications with no limitations and includes the following databases:

- **Telecommand**, which contains all commands and parameters needed to compose commanding packets or frames for the connected subsystems;
- **Telemetry**, which contains all the telemetry parameters and any command needed to pull those parameters;
- Generic, or PnP(Plug and Play), database, which contains all configuration settings for the interfaces, the subsystems and any auxiliary information;
- Packet database, which defines the packet composition, including which parameters of which subsystem to include in the packet;
- **ADCS** database, which holds all the calibration values, parameter settings and other auxiliary information needed by the ADCS application.

Electronic Data Sheets (EDS)

The EDS are written in JSON (JavaScript Object Notation) files, a lightweight data interchange format easy for humans to read and write, and for machines to parse and generate. It is usually used to transmit data between a server and a web application or between different parts of an application.

The structure of the file is based on a subset of the JavaScript programming language, but it is language-independent and can be used with virtually any programming language.

The possible applications are wide and extended: the most useful in this case are for storage and for the serialization/deserialization process.

In particular, the latter has been exploited in the work presented, on the programming side: first the data have been converted to JSON format for storage and transmission (serialization), and then converted back, via a proper Python script, to specific databases (deserialization), used in the embedded functions of the software.

Coming back to the EDS, using that format, they can be easily processed by programming languages such as C and Python, and they store the following parameters:

- Device ID;
- Meta Data;
- Subsystem information such as name, interface, protocol, address, commands, telemetry, arguments, etc.

As mentioned before, Python code is used to process these sheets and convert them into a format accepted by the onboard software. This process can be seen in the following figure 1.6 and can be used both on the ground and in orbit for updating parameters.

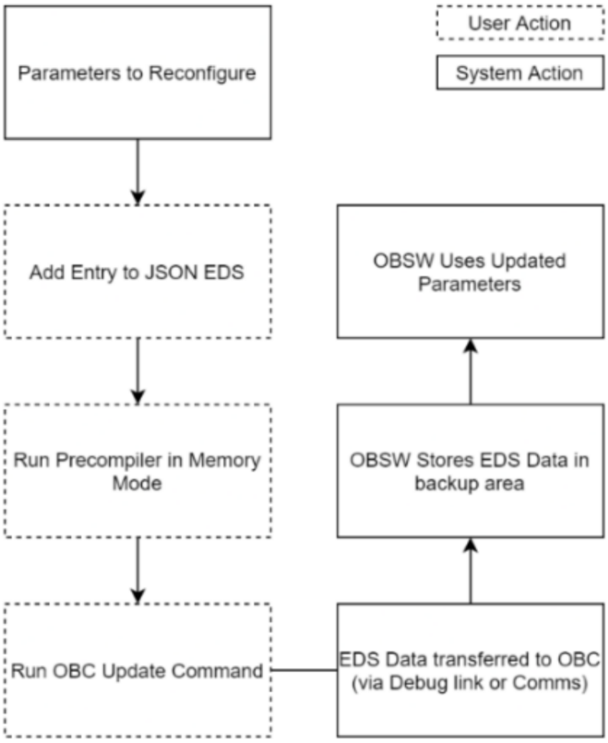

User Action
System Action
Parameters to Reconfigure
Add Entry to JSON EDS
Run Precompiler in Memory Mode
Run OBC Update Command
EDS Data transferred to OBC (via Debug link or Comms)
OBSW Stores EDS Data in backup area
OBSW Uses Updated Parameters

Figure 1.6: Processing Software [11]

Protocol definition

As introduced in the previous chapter, section 2.2.3, understanding and managing protocols of the different subsystems is essential to handle the communication within a satellite.
In this particular case, the protocols have to follow the defined template shown below; to understand it better it would be useful to describe the parameters involved:

- interface, which represents the type of interface that the protocol is applicable to;
- interaction method, which is the topology on which the protocol is based, e.g. bus or point-to-point;
- character encoding, about how the data is encoded;
- packet header, which means the function that composes the start of the packet;
- additional, or any added features between the header and the data;

- packet data, the actual internal definition of the content;
- packet footer, which identifies each packaging, CRC and CS.

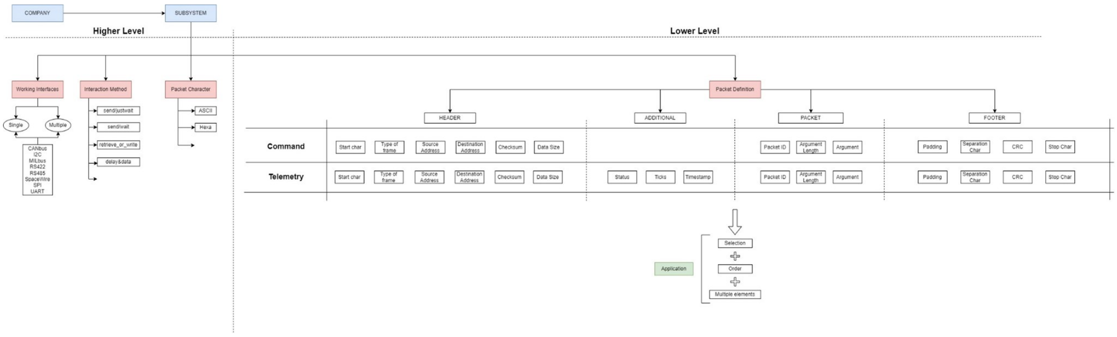


Figure 1.7: Protocol Internal definition [11]

MA61C API and GUI

These two important interfaces are both really useful for the onboard software itself.
The Application Program Interface, or **API**, is used in the application software to know which functions have to be used to send commands, get telemetry or change the settings of the device.
The Graphical User Interface, or **GUI**, instead, is introduced to provide easy usage, allowing to command and monitor the subsystems: it displays the devices connected to the board and all the commands available to be sent.
This interface could be used also for early verification of the subsystem with the OBC unit before assembling it to the MATB chain.

GR712RC and GRMON

It could be also useful to have a better insight and comprehension of the processor with which the main Cubesat OBC board, that has been used, is made of.
Described in [12], the radiation-hard **GR712RC** Dual-Core LEON3FT SPARC V8 processor, is based on LEON3FT and IP cores from the GRLIB IP library, implemented with the Ramon RadSafe 180 nm cell library on Tower Semiconductors 180nm CMOS process. Among all the numerous on-chip features present, it is interesting to highlight the six SpaceWire Links, six UARTs, two CAN 2.0 controllers, SPI and I2C master controller, 2x32 bits General Purpose I/O, a JTAG debug link and more others. In the figure below it is possible to see a block diagram with all the interfaces and features highlighted. **GRMON** instead, as described in

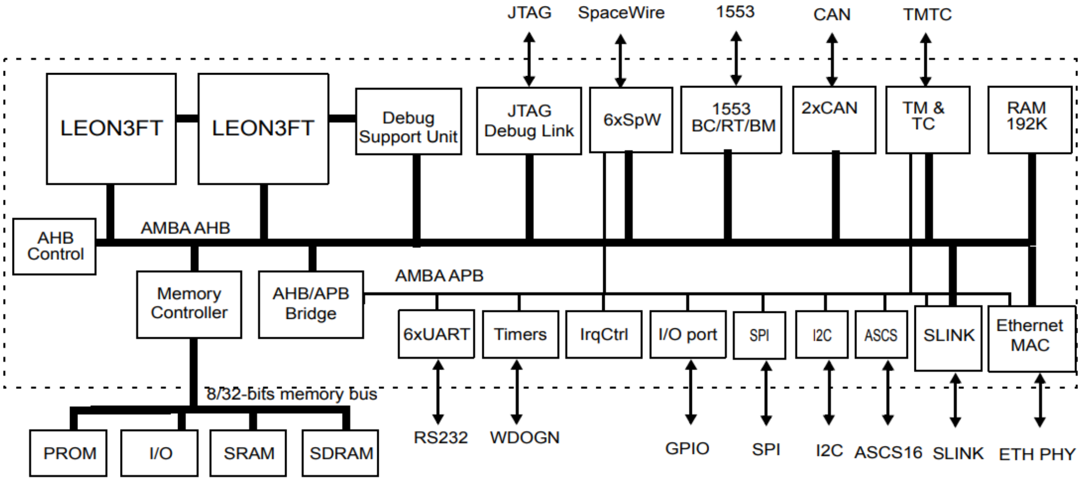


Figure 1.8: GR712RC Block Diagram [12]

[13], is a general debug monitor for the LEON processor, and for system-on-chip (SOC) designs based on the GRLIB IP library; it has several different functions such as read/write access to all system registers and memory, support for USB, JTAG, RS232, PCI, Ethernet and SpaceWire debug links and so on.
The monitor connects to a dedicated debug interface on the target hardware, through which it can perform read and write cycles on the on-chip bus (AHB). The debug interface can be of various types: the LEON3/4 processor supports debugging over a serial UART, 32-bit PCI, JTAG, Ethernet and SpaceWire (using the GRESB Ethernet to SpaceWire bridge) debug interfaces. On the target system, all debug interfaces are realized as AHB masters with the Debug protocol implemented in hardware. There is thus no software support necessary to debug a LEON system, and a target system does in fact not even need to have a processor present.
GRMON can operate in two modes: command-line mode and GDB mode. In command-line mode, GRMON commands are entered manually through a terminal window. In GDB mode, GRMON acts as a GDB gateway and translates the GDB extended-remote protocol to debug commands on the target system. In the case of interest, only the command-line mode has been used.
The image displays a concept overview of the relationship between the debug monitor described and a general LEON SOC Target system.

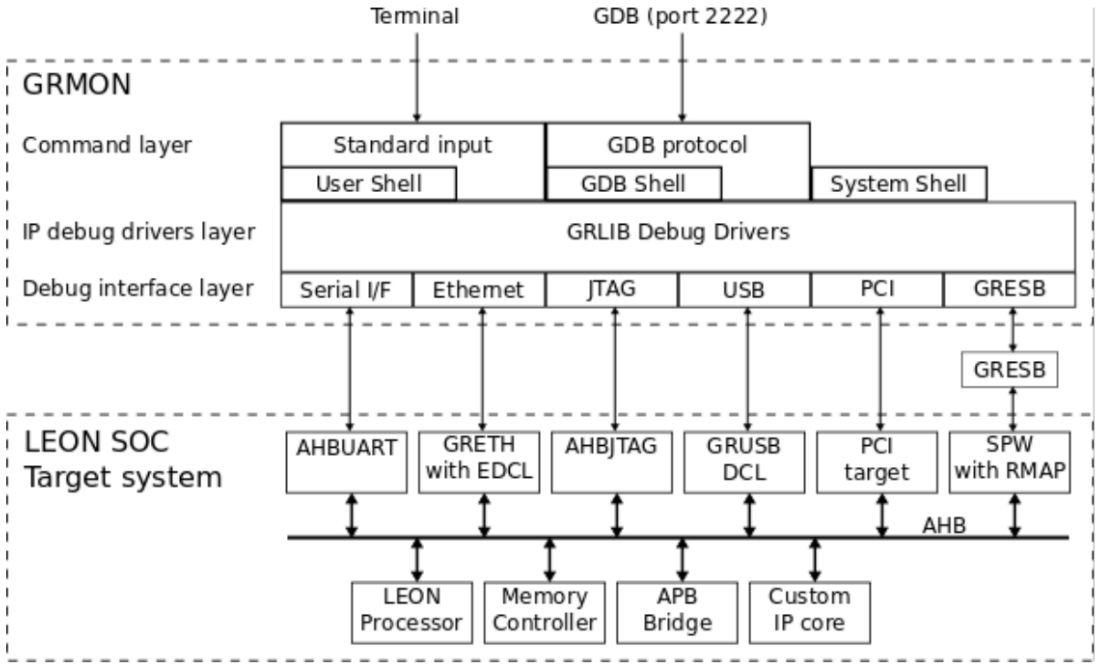


Figure 1.9: GRMON concept overview [13]

1.2.2 Modular Avionic Test Bench (MATB)

The modular avionics test bench serves as the platform for implementing and testing the ADCS software. It consists of hardware components, simulation tools, and software interfaces that collectively enable the emulation of CubeSat's onboard avionics systems and the simulation of space conditions. The test bench facilitates the execution of various test scenarios, including nominal operations, fault detection and recovery, and anomaly response.

By employing a modular avionics test bench, the software's functionality and performance in a controlled environment can be verified before deploying it on an actual CubeSat. The test bench allows for iterative testing and debugging, ensuring the identification and resolution of any software issues or deficiencies prior to the operational phase.

The purpose of the Modular Avionics Test Bench Project in particular is to map all the interconnectivity technologies offered by European suppliers, design the Central Data Handling System (CDHS) hardware and software to match those technologies, test the design against the supplier's components and provide a final product that is 80 % compatible with the EU Cubesat subsystem suppliers.

To do this, 3 flatsats are set up with components from more than 10 suppliers and validate the CDHS based on plug-and-play (PnP) technology.

The design of the flatsat will revolve around MA61C-CubeSat.
The final flatsat hardware and support library software, which is MA61C-CubeSat Software, is tested and validated against fully qualified off-the-shelf Cubesat subsystems (power, communications, attitude, thermal, payload, etc.).
The flatsat facilitates a modular plug-and-play system, is easy to use, and is reconfigurable to support different mission scenarios.

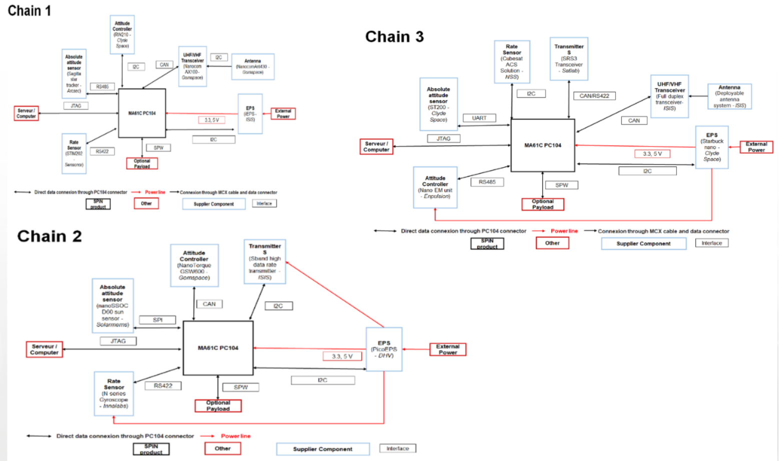

Chain 1
Attitude Controller (RW210 - Clyde Space)
Absolute attitude sensor (Sagitta star tracker - Arcsec)
UHF/VHF Transceiver (Nanocom AX100 - Gomspace)
Antenna (NanocomAnt430 - Gomspace)
I2C
CAN
I2C
RS485
JTAG
Serveur / Computer
MA61C PC104
EPS (iEPS - ISIS)
External Power
3.3, 5 V
I2C
Rate Sensor (STIM202 - Sensonor)
RS422
SPW
Optional Payload
Direct data connexion through PC104 connector
Power line
Connexion through MCX cable and data connector
SPiN product
Other
Supplier Component
Interface
Chain 2
Attitude Controller (NanoTorque GSW600 - Gomspace)
Transmitter S (Sband high data rate transmitter - ISIS)
Absolute attitude sensor (nanoSSOC D60 sun sensor - Solarmems)
CAN
I2C
SPI
JTAG
Serveur / Computer
MA61C PC104
EPS (PicoEPS - DHV)
External Power
3.3, 5 V
I2C
Rate Sensor (N series Gyroscope - Innalabs)
RS422
SPW
Optional Payload
Direct data connexion through PC104 connector
Power line
SPiN product
Other
Supplier Component
Interface
Chain 3
Rate Sensor (Cubesat ACS Solution - NSS)
Transmitter S (SRS3 Transceiver - Satlab)
Absolute attitude sensor (ST200 - Clyde Space)
UHF/VHF Transceiver (Full duplex transceiver- ISIS)
Antenna (Deployable antenna system - ISIS)
I2C
CAN/RS422
UART
CAN
JTAG
Serveur / Computer
MA61C PC104
EPS (Starbuck nano - Clyde Space)
External Power
3.3, 5 V
I2C
Attitude Controller (Nano EM unit - Enpulsion)
RS485
SPW
Optional Payload
Direct data connexion through PC104 connector
Power line
Connexion through MCX cable and data connector
SPiN product
Other
Supplier Component
Interface

Figure 1.10: MATB Reference Setup [11]

The CAD model of the test benches designed are shown in the figure below.

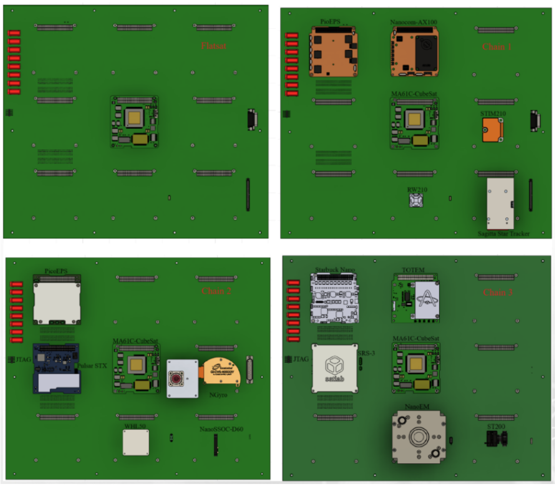


Figure 1.11: Test Bench CAD model [11]

1.2.3 Modular ADCS

Modularity is a key aspect when developing ADCS software for CubeSats. The software is typically composed of various modules, each responsible for specific tasks such as sensor data processing, attitude estimation algorithms, control algorithms, and actuator commands. Modular software architecture offers flexibility, reusability, and ease of maintenance, allowing for efficient development and evolution of the ADCS software.

The implementation and testing of modular ADCS software on the test bench involve the integration and evaluation of individual software modules. Each module is tested independently to ensure its proper functioning and compatibility with other modules. Integration testing is then performed to verify the seamless operation and communication between the modules within the ADCS software.

The Modular ADCS project, taken as a reference for the study, is focused on a modular integrated Attitude Determination and Control Subsystem (ADCS) for Small Satellites based on MA61C adapter and ADCS software, developed by an external company. The architecture is composed of a multipurpose hardware board that holds 7 different communication interfaces, a plug-and-play adaptive software and universal ADCS software.

The system has the following capabilities:

- Attitude determination;
- Attitude control;
- Boot and self-recovery;
- Plug and play interface to sensor and actuators;
- Fault Detection, Isolation & Recovery (FDIR);
- Data management and storage;
- External control interface.

The system can connect to multiple sensors and actuators that exist in the new space SmallSat market and provide an independent ADCS subsystem that is compatible with multisession, including but not limited to Telecommunication, Navigation, In-situ measurements, Earth observation, and Space exploration.

The design is based on existing components from the consortium partners which need only little modification and verification. So it is possible to create a cost-efficient, short lead time and COTS integrated ADCS system for small satellites.

Further information about the mission scenarios considered, the tests done and the relative performance results can be found in [9].

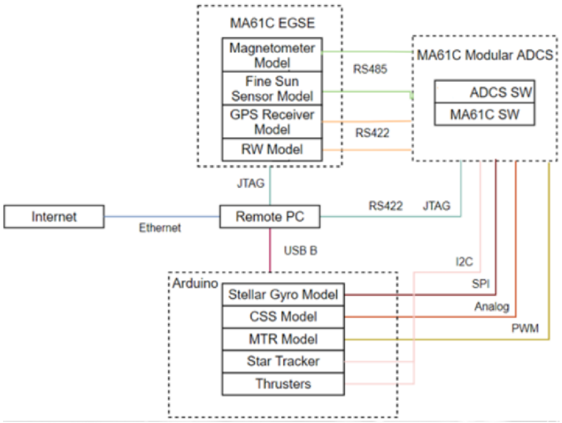


Figure 1.12: Starting Point of the Modular ADCS software [9]

In figure 1.12 it is shown the first approach on the Modular ADCS software.
As it has been done for MA61C in section 1.2.1, especially for this reference project is interesting and important to provide an overview of the software under exam and to report the changes applied between the two different versions, from both the hardware and software side, of a modular Attitude Determination and Control System, one for SmallSat, already finalized and tested properly in the previous project 1.2.3, and the other for CubeSat applications, the focus of this study.
The following insight would be useful to understand the main differences between the two models and the corrections and updates that have to be applied to the new version in order to be implemented and tested, once completed, using the modular avionic test bench available.

Models Comparison

First of all, it could be worth providing a deeper view into these models:

- **SmallSat mADCS**: this version is part of a project 1.2.3 focused on SmallSats, with the aim to create a modular ADCS subsystem, offering quick solutions to mission designers; the most important thing to highlight is that the setup of this version, represented in figure 1.13, is composed by:
  - an external company simulation desktop, used to get and analyse data;
  - a SPiN test desktop, for initializing the test procedure;
  - a MA61C EGSE, which holds the emulated equipment model of sensors and actuators;
  - an Arduino board, to send input data to the emulator;

  - a MA61C Modular ADCS Node, that is the main core of the test, containing both the model of the satellite and the ADC embedded software.

It is important to focus, between all these elements, on the importance of the MA61C EGSE device: it allows to connect, test and validate all the digital inputs and outputs before integrating the final system, removing the need for dedicated hardware for hardware-in-the-loop tests, and to cover the functional verification fully.

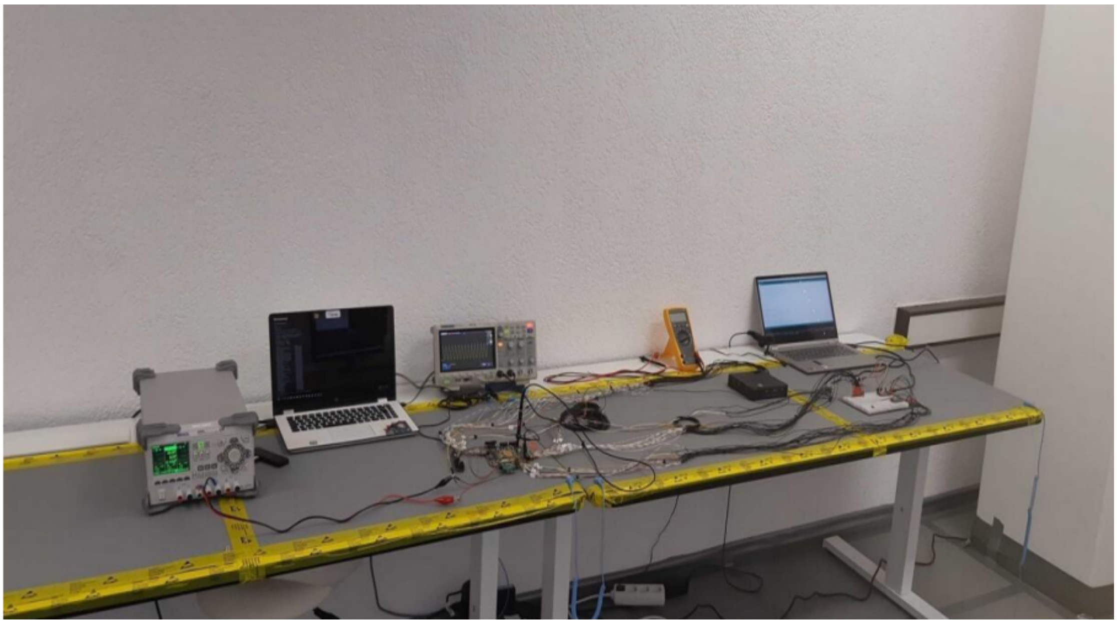


Figure 1.13: SmallSat mADCS setup [9]

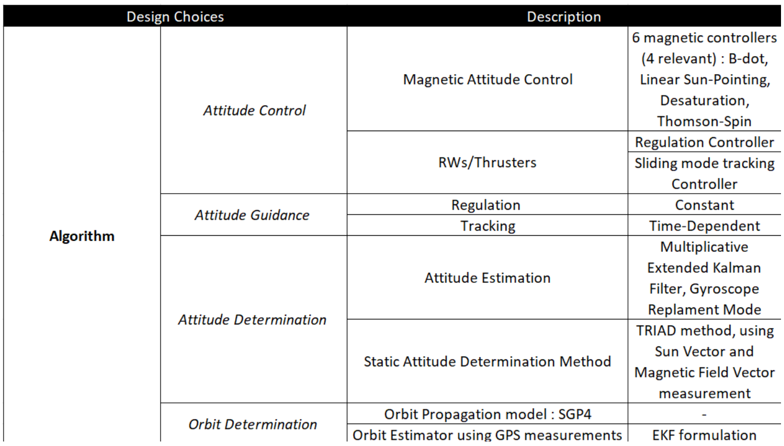
- **Cubesat mADCS**: this new version is meant as an extension and evolution of the previous model with a few major changes applied:
  - the focus is moved to the CubeSat model, with the relative changes in the Software and Hardware aspects;
  - the idea is not to use the MA61C EGSE unit to emulate the equipment models of sensors and actuators but to implement and test it in the avionic test bench with real space components, provided by external suppliers.

Overall, besides the version considered, the structure of the software is kept the same and is composed of the following main items:

- ADCS_performance.cpp file represents the main file, which includes all the useful folders, compiled and run to test the performance of the solution proposed under different scenarios and operating modes.

- *src* (source) folder includes all the *c* files needed for the correct operation of the Ma61c OBC Unit, from databases to parser files.
- *lib* folder includes all the header files related to the correspondent source folder.
- *components* folder includes all the functions related to attitude determination, guidance, control algorithms and models for Sun and Earth Magnetic Field.
- *executables* folder includes all the functions related to the modes available.

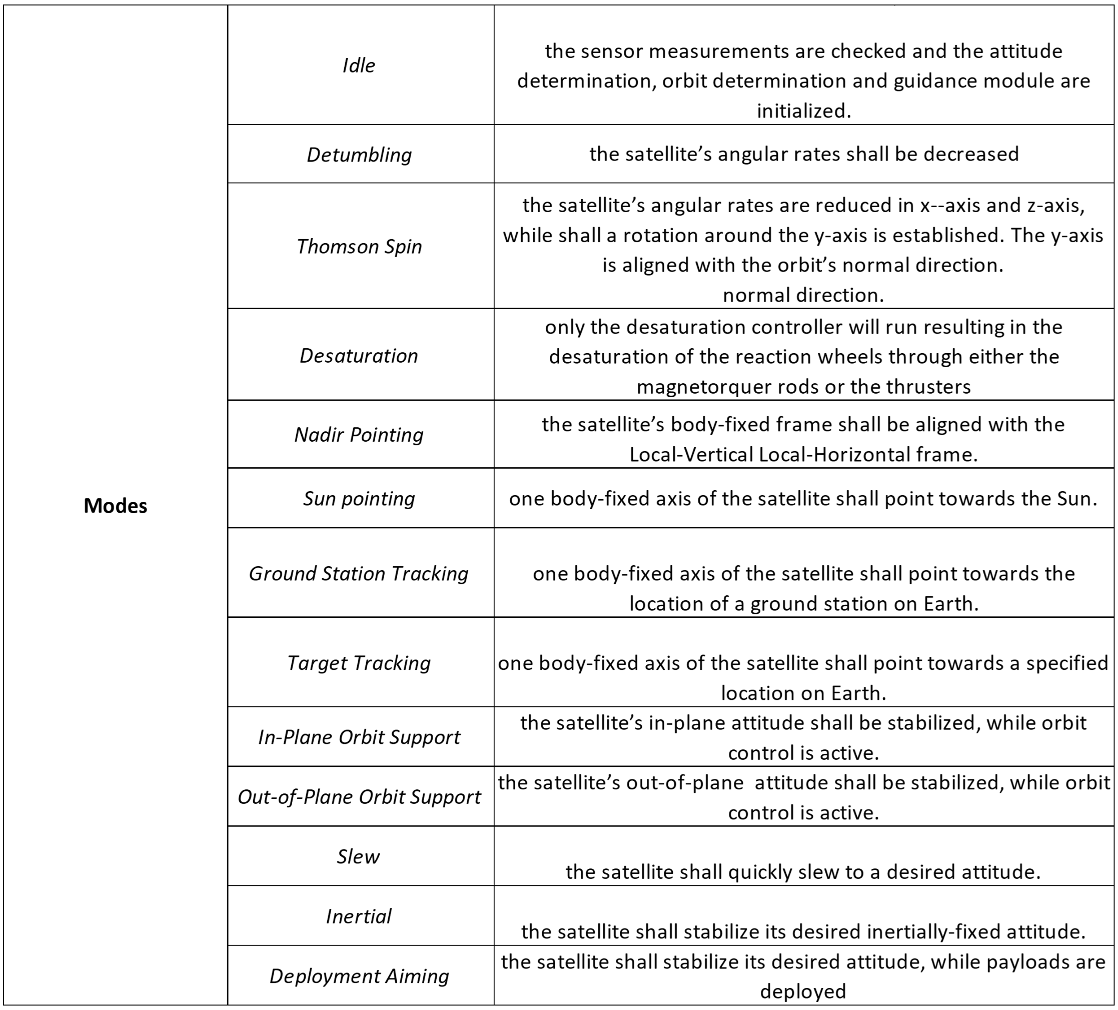
To conclude the Modular ADCS Software presentation, although it has not been part of the development process, since it was already designed and tested in a previous project, it is appropriate to provide a brief overview of the ADCS design choices and operational modes.
Figure 1.14 briefly summarizes the main design choices for the subsystem.



Design Choices		Description	
Algorithm	*Attitude Control*	Magnetic Attitude Control	6 magnetic controllers (4 relevant) : B-dot, Linear Sun-Pointing, Desaturation, Thomson-Spin
		RWs/Thrusters	Regulation Controller
			Sliding mode tracking Controller
	Attitude Guidance	Regulation	Constant
		Tracking	Time-Dependent
	Attitude Determination	Attitude Estimation	Multiplicative Extended Kalman Filter, Gyroscope Replament Mode
		Static Attitude Determination Method	TRIAD method, using Sun Vector and Magnetic Field Vector measurement
	Orbit Determination	Orbit Propagation model : SGP4	-
		Orbit Estimator using GPS measurements	EKF formulation

Figure 1.14: ADCS Design Choices

In figure 1.15 instead, a brief description of the modes considered is presented.



Modes	*Idle*	the sensor measurements are checked and the attitude determination, orbit determination and guidance module are initialized.
	Detumbling	the satellite's angular rates shall be decreased
	Thomson Spin	the satellite's angular rates are reduced in x--axis and z-axis, while shall a rotation around the y-axis is established. The y-axis is aligned with the orbit's normal direction. normal direction.
	Desaturation	only the desaturation controller will run resulting in the desaturation of the reaction wheels through either the magnetorquer rods or the thrusters
	Nadir Pointing	the satellite's body-fixed frame shall be aligned with the Local-Vertical Local-Horizontal frame.
	Sun pointing	one body-fixed axis of the satellite shall point towards the Sun.
	Ground Station Tracking	one body-fixed axis of the satellite shall point towards the location of a ground station on Earth.
	Target Tracking	one body-fixed axis of the satellite shall point towards a specified location on Earth.
	In-Plane Orbit Support	the satellite's in-plane attitude shall be stabilized, while orbit control is active.
	Out-of-Plane Orbit Support	the satellite's out-of-plane attitude shall be stabilized, while orbit control is active.
	Slew	the satellite shall quickly slew to a desired attitude.
	Inertial	the satellite shall stabilize its desired inertially-fixed attitude.
	Deployment Aiming	the satellite shall stabilize its desired attitude, while payloads are deployed

Figure 1.15: ADCS Modes Description

1.3 Objectives of the study

Given all the information provided in the previous sections, this discussion aims to pursue the following objectives:

- develop and implement a modular ADCS software architecture that encompasses different software modules responsible for tasks such as sensor data processing, attitude estimation algorithms, control algorithms, and actuator commands, to create a flexible and modular software framework that allows for easy integration and scalability;
- design and set up a modular avionics test bench that replicates the onboard avionics systems of a CubeSat and simulates space conditions. This involves

selecting appropriate hardware components, simulation tools, and software interfaces to create a controlled testing environment;

- implement and integrate ADCS software modules within the test bench environment. Each module should be developed and tested independently to ensure its proper functioning and compatibility with other modules. Integration testing should be conducted to verify the seamless operation and communication between the modules;
- perform comprehensive testing and validation of the ADCS software using the modular avionics test bench. This includes executing various test scenarios. The objective is to evaluate the solution's performance, robustness, and adherence to specified requirements;
- analyze and evaluate the results obtained from the modular avionics test bench, to assess the software's performance, identify any anomalies or deficiencies, and propose improvements or optimizations if necessary. This analysis should demonstrate the effectiveness and reliability of the implemented ADCS software.

In conclusion, the implementation and testing of modular ADCS software using a modular avionics test bench for CubeSats are vital aspects of ensuring the reliability and effectiveness of the ADCS system. The test bench provides a controlled environment that emulates space conditions, enabling comprehensive evaluation and validation of the ADCS software. Through modular software architecture and the utilization of a test bench, developers can enhance the development process, verify module compatibility, and detect and rectify any software anomalies.
The following chapters will delve further into the details of the test bench setup, testing methodologies, and the significance of rigorous testing for ADCS software in CubeSat missions.

1.4 Outline

In this last introductory section it has been chosen to present how the work will be developed into the different chapters:

- Chapter 2: **Preliminaries**, giving an overview of all the main concepts, terminologies and tools that are going to be exploited during all the work;
- Chapter 3: **Software Development and Implementation**, providing an introduction about the MA61C Cubesat SW and the reference Modular ADCS software, then presenting the changes in the adapted software version, to conclude with the explanation of the development of the Simulator;

- Chapter 4: **Modular Avionics Test Bench Design and Setup**, explaining the requirements and design of the bench, the selection and the integration of hardware components;
- Chapter 5: **Testing Process Description**, presenting the test scenarios, the testing methodologies employed for the space environment simulation, describing the assumptions and limitations taken into consideration, explaining the execution of the tests, and the software interfaces used for debugging;
- Chapter 6: **Results and Analysis**, presenting the different tests gradually performed and the results obtained, analyzing the performance of the software and discussing any anomalies or deficiencies observed;
- Chapter 7: **Conclusion**, summarizing the key findings, discussing the implications of the research, highlighting the limitations and challenges and providing recommendations for improving the architecture tested, suggesting enhancements or modifications to the test bench setup.

Chapter 2

Preliminaries

The aim of this chapter is mainly to give a brief but complete overview of the concepts that have to be known before going deeper into the work that will be presented in the following chapters.

2.1 Spacecraft and Orbital Mechanics

To develop and implement ADCS software and the relative orbital simulator needed for the Hardware in the Loop simulation, it is important to have a clear understanding of how the spacecraft mechanics work, along the orbit: starting from the reference frames taken into consideration for the simulation, then presenting the attitude representations used in the software and concluding with the orbital and satellite state models implemented in the Simulator. For this section, the main reference has been taken by the complete discussions presented in [14], [15] and [16]. It is suggested to consult them for further details about the subjects presented.

2.1.1 Reference Frames

To perform attitude analysis there are several reference frames to be taken into account according to the particular application or phase of the mission. In general, a reference frame is defined by the location of its origin and the orientation of its coordinate axis.
In Figure 2.1 the most significant ones for the application are represented.

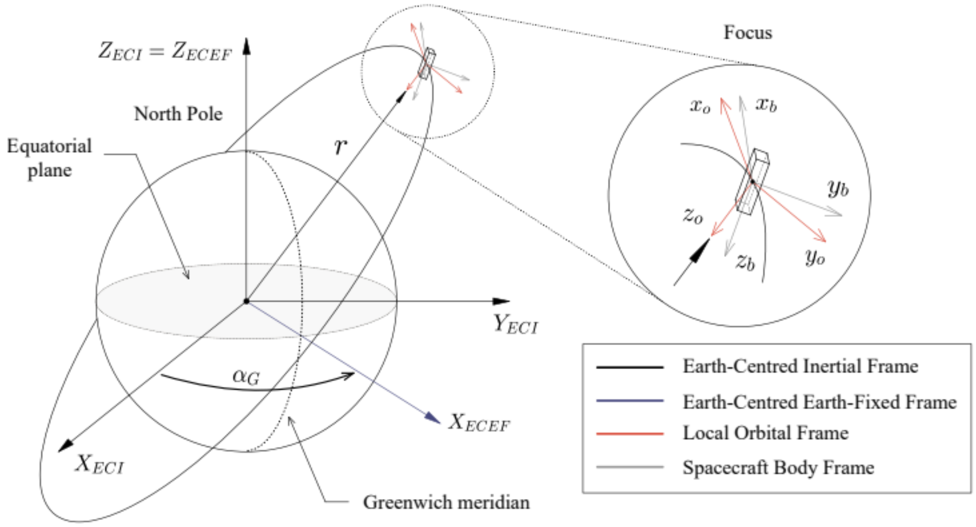


Figure 2.1: Main reference frames for satellites orbiting around Earth [16]

Inertial Reference Frame

An inertial reference frame is a frame in which Newton's laws of motion are valid; furthermore, any frame moving at constant velocity and without rotation with respect to an inertial frame is also inertial.

Celestial reference frames with their axes fixed relative to distant "fixed" stars are the best realization of inertial frames. The current standard is the **International Celestial Reference System(ICRF)**, with its axes fixed with respect to the positions of several hundred distant extra-galactic sources of radio waves, determined by very long baseline interferometry. The z axis of this frame is aligned with the Earth's North pole, and the x axis with the *vernal equinox*, the intersection of the Earth's equatorial plane with the plane of the Earth's orbit around the Sun, in the direction of the Sun's position relative to the Earth on the first day of spring. Since neither the polar axis nor the ecliptic plane are inertially fixed, the ICRF axes are defined to be the *mean* orientation of the pole and the vernal equinox at some fixed epoch time. The origin of the ICRF is located at the center of mass of the solar system.

An approximate inertial frame, known as the $Geocentric\ Inertial\ Frame(GCI)$ or **Earth-Centered Inertial**, has its origin at the center of mass of the Earth and has a linear acceleration because of the Earth's circular orbit around the Sun, negligible for attitude analysis. The axes of a "mean of epoch" GCI frame are aligned with the mean North pole and mean vernal equinox at some epoch.

A variation of this frame is the **Earth-Centered/Earth-Fixed** frame, which is similar to the GCI frame with the same z axis, but with the x axis pointing

in the direction of the Earth's prime meridian, and the y axis completing the right-handed system. Unlike the GCI frame, the ECEF one rotates with the Earth and the rotation angle is known as the Greenwich Mean Sidereal Time(GMST) angle, denoted by α_G in 2.1. This angle represents the time elapsed since the Greenwich meridian passed through the vernal equinox.
Determining the GMST angle requires the Julian Date (JD), which could be calculated using the following equation: for a given year Y (between 1901 and 2099), month M, day D, hour h, minute m and second s

$$JD(Y,M,D,h,m,s) = 1{,}721{,}013.5 + 367 \cdot Y - INT\left\{\frac{7}{4} \cdot \left[Y + INT\left(\frac{M+9}{12}\right)\right]\right\} + INT\left(\frac{275 \cdot M}{9}\right) + D + \frac{60 \cdot h + m + \frac{s}{60*}}{1440} \tag{2.1}$$

with INT denoting the integer part and $60*$ denoting the use of 61 s for days with a leap second.
After JD, it is needed to compute T_0, the number of Julian centuries elapsed from epoch J2000, taken as a reference, to zero hours of the date in question:

$$T_0 = \frac{JD(Y,M,D,0,0,0) - 2{,}451{,}545}{36{,}525} \tag{2.2}$$

The Greenwich sidereal time α_{G_0} at 0 h UT can be found, expressed in degrees, using this formula:

$$\alpha_{G_0} = 100.4606148 + 36{,}000.77004 \cdot T_0 + 0.000387933 \cdot T_0^2 - 2.583 \cdot 10^{-8} \cdot T_0^3 \tag{2.3}$$

The Greenwich sidereal time α_G, at the end, is found using the relation

$$\alpha_G = \alpha_{G_0} + 360.98564724 \cdot \frac{UT}{24} \tag{2.4}$$

with UT defined as

$$UT = h + \frac{minutes}{60} + \frac{seconds}{3600} \tag{2.5}$$

Local Orbital Frame

This frame is commonly used to describe motions with respect to the moving position and direction towards the center of the Earth of an orbiting body. The use of this reference is mainly convenient for Earth-pointing spacecraft. Also known as Local Vertical/Local Horizontal Frame, it has the z axis radial and pointing from the spacecraft center of mass to the Earth's center, the y axis normal to the orbital plane and pointing in the opposite direction of the angular momentum vector of

the orbit and the x axis defined to complete the right-handed system. For circular orbits, this axis is aligned with the orbital velocity vector.
The origin is located at the center of the mass of the spacecraft.

Spacecraft Body Frame

A spacecraft body frame is defined by an origin at a specified point in the spacecraft body and three Cartesian axes, and it is used to align the various components during spacecraft assembly. As shown in 2.1, this frame is used to describe all rotations of the spacecraft and the attitude with respect to the Local Orbital frame. Usually, the origin is the center of mass of the spacecraft and its axes are defined with the z one pointing in the direction of the face where the observation payload is located, the x nominally pointing in the direction of the orbital velocity vector and the y completing the right-handed system.
For nanosatellites, the center of mass rarely moves from the nominal point, so this frame could be considered fixed.

Sensor Frame

A sensor frame for space satellites is a coordinate system that is attached to a specific sensor on the satellite, such as a camera or an attitude sensor. The sensor frame defines the orientation of the sensor relative to the satellite body frame, fixed to the satellite structure. The sensor frame is useful for calibrating the sensor measurements and relating them to other reference frames, such as the Earth-centered inertial (ECI) frame. The sensor frame is important for attitude determination and control, especially during the process of estimating and controlling the satellite orientation using sensors and actuators.

2.1.2 Attitude Representations

To describe the attitude of the satellite and its evolution over time, it is necessary to introduce an attitude representation and Euler's equation of motion. The position of the spacecraft body frame with respect to the local orbital frame, previously described, completely defines the attitude of the spacecraft. Mathematically this orientation can be described using a coordinate transformation matrix, defined by 3 parameters.

Euler Angles

One of the possible solutions is the one based on the Euler angles set, which defines a sequence of rotations to make the starting reference frame coincide with the body reference frame: this sequence of rotations is not unique and the rotations are not commutative.
The sequence originally proposed by Euler, so-called "Proper Euler" or 3-1-3, shown in figure 2.2 is based on the three following rotations to make the starting frame, defined by the unit vectors $\hat{E}_1, \hat{E}_2, \hat{E}_3$, coincide with the body frame, defined by

the unit vectors $\hat{e_1}, \hat{e_2}, \hat{e_3}$:

- The first rotation is about the third axis of the initial frame $\hat{E_3}$; the rotation angle is called precession angle Ψ;
- The second rotation is about the first axis transformed after the first rotation $\hat{e_1}'$; the rotation angle is called nutation angle Θ;
- The final rotation is about $\hat{e_3}$; the rotation angle is called spin angle Φ.

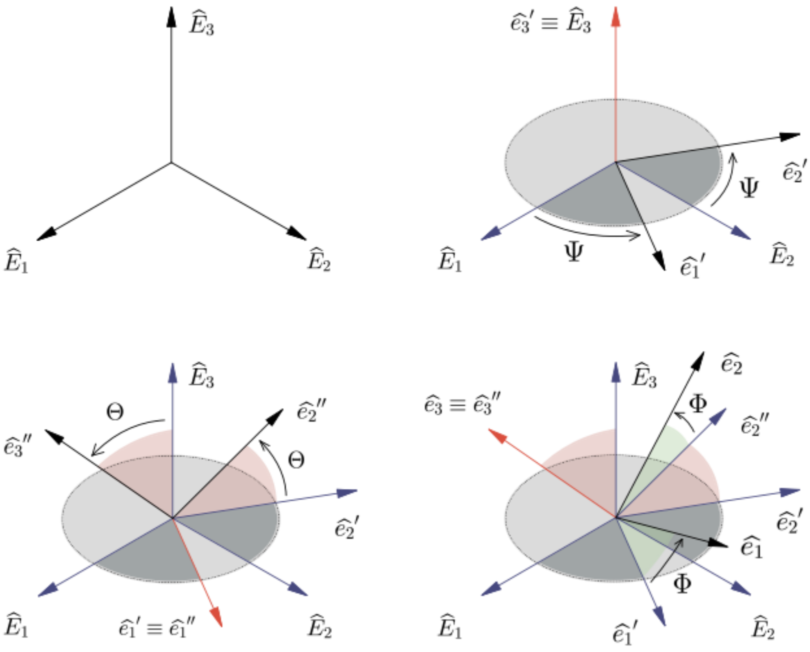


Figure 2.2: Euler angles and rotations [16]

The three angles represent the attitude of the body frame with respect to the initial frame. It is possible to obtain the transformation matrix, describing the attitude of the satellite, by properly multiplying the rotation matrices for the sequence of 3 elementary Euler rotations, defined as

$$A_3(\Psi) = \begin{bmatrix} \cos\Psi & \sin\Psi & 0 \\ -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

$$A_1(\Theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Theta & \sin\Theta \\ 0 & -\sin\Theta & \cos\Theta \end{bmatrix} \tag{2.7}$$

$$A_3(\Phi) = \begin{bmatrix} \cos\Phi & \sin\Phi & 0 \\ -\sin\Phi & \cos\Phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

Therefore the coordinate transformation matrix is

$$A_{3,1,3}(\Psi,\Theta,\Phi) = A_3(\Phi)\cdot A_1(\Theta)\cdot A_3(\Psi) \tag{2.9}$$

The weakest feature of this representation is that singularities could occur: this means that there may be situations in which the relative attitude between the two reference frames can be described in different ways and not uniquely; when a singularity occurs, some angles go to infinity causing serious consequences in the attitude representation. For that reason, in some applications, quaternions are usually preferred.

Quaternions

To avoid and overcome singularity issues, the spacecraft attitude is parameterized using 4 parameters known as quaternions.

A quaternion can be represented in two forms

$$\overline{q} = \begin{bmatrix} q_0 \\ \vec{q} \end{bmatrix} \qquad \overline{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

Stating the Euler Eigenaxis rotation theorem, as it is possible to rotate a fixed frame onto another frame with a simple rotation around an axis $\hat{a}$ fixed in both frames, it is possible to use quaternions to describe changes in attitude: the rotation from the inertial frame to the body one is then the rotation of the inertial frame about the unit vector $\hat{a}$ through angle α. According to this statement, the quaternion can be also defined as

$$\overline{q} = \begin{bmatrix} \cos\frac{\alpha}{2} \\ a_1\cdot\sin\frac{\alpha}{2} \\ a_2\cdot\sin\frac{\alpha}{2} \\ a_3\cdot\sin\frac{\alpha}{2} \end{bmatrix} \tag{2.10}$$

It is possible to demonstrate that the coordinate transformation matrix between the two frames is

$$A_i^B = (q_0^2 - \vec{q}\cdot\vec{q})\cdot\mathbf{1} + 2\cdot(\vec{q})\cdot(\vec{q})^T - 2\cdot q_0\cdot\mathbf{S}(\vec{q}) \tag{2.11}$$

where $\mathbf{S}(\vec{q})$ is the cross-product matrix equivalent, or skew matrix, defined as

$$\mathbf{S}(\vec{q}) = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \tag{2.12}$$

Coordinate Transformations
Having defined the reference frames and two different ways of representing the attitude, it is possible to consider and define the transformation matrices that allow one reference frame to be rotated on another, bringing them to coincide.
The most interesting rotations to be considered for the work are:

- rotation from ECEF to ECI, for which it is necessary to know the rotation angle, or the GMST already defined.

- rotation from Sensor frame to Spacecraft Body frame, for which is necessary to calculate the Direction Cosine Matrix(DCM) from the current quaternion detected.

2.1.3 Orbit and Reference Models

To understand fully how the orbital simulator has been built, it is interesting to give an insight also about the characteristics of the orbital propagator and the reference models used.
Two-Body Problem
This represents the classical and simplest problem of determining the motion of two bodies due solely to their own mutual gravitational attraction and it is resolved by Newton's law of Universal Gravitation. As well discussed in [16], by considering an inertial frame, such as the ECI defined, making some simplifying assumptions, and by applying Newton's second law, the path of one of the masses relative to the other is a conic section, i.e. circle, ellipse, parabola or hyperbola, whose shape is determined by the eccentricity.
The key vector differential equation of the relative motion for the problem presented is the following

$$\ddot{\overline{r}} = -\frac{G(M+m)}{r^3} \cdot \overline{r} \tag{2.13}$$

where G is the universal gravitational constant, M is the mass of the main body and m is the mass of the secondary body.
In the case of interest, the main body is represented by the Earth and the secondary body by the spacecraft, therefore the mass m is much less than M, so it follows

$$G(M+m) \simeq GM \equiv \mu \tag{2.14}$$

where μ is defined as the gravitational parameter and for Earth is

$$\mu_{\oplus} = 3.986004418 \cdot 10^{14} \ \frac{m^3}{s^2} \tag{2.15}$$

According to this simplification, the Equation 2.13 becomes

$$\ddot{\overline{r}} = -\frac{\mu}{r^3} \cdot \overline{r} \tag{2.16}$$

Defining the *state vector* of the spacecraft, which includes its position $\overline{r}$ and its velocity $\overline{v}$, the time derivative of the state vector returns the velocity and acceleration that is provided by 2.16. By integrating the six components of this vector, the instantaneous position and velocity of the spacecraft relative to the inertial frame can be calculated to define the characteristics of the orbit: the six components of the state vector are sufficient to uniquely determine an orbit.
A valid alternative to the six components of the state vector is represented by the six *classical orbital parameters*, represented in figure 2.3, which can be used to describe the orbit and the position of the spacecraft:

- **a, semi-major axis**, constant defining the size of the orbit;
- **e, eccentricity**, constant defining the shape of the orbit;
- **i, inclination**, angle between the $\hat{K}$ unit vector and the angular momentum vector $\overline{h}$;
- **Ω, longitude of the ascending node**, or RAAN angle, in the fundamental plane, between the $\hat{I}$ unit vector and the point where the spacecraft crosses through the fundamental plane in a northerly direction measured counter-clockwise when viewed from the north side of the fundamental plane;
- **ω, argument of periapsis**, angle, in the plane of the spacecraft's orbit, between the ascending node and the periapsis point, measured in the direction of the satellite's motion;
- **ν, true anomaly at epoch**, angle, in the plane of the spacecraft's orbit, between the periapsis and the position of the spacecraft at a particular time, t, called the "epoch".

Five of them $(a, e, i, \Omega, \omega)$ are sufficient to define the shape, size and orientation of the orbit, while the sixth (ν) is required to specify the position of the spacecraft along the orbit.

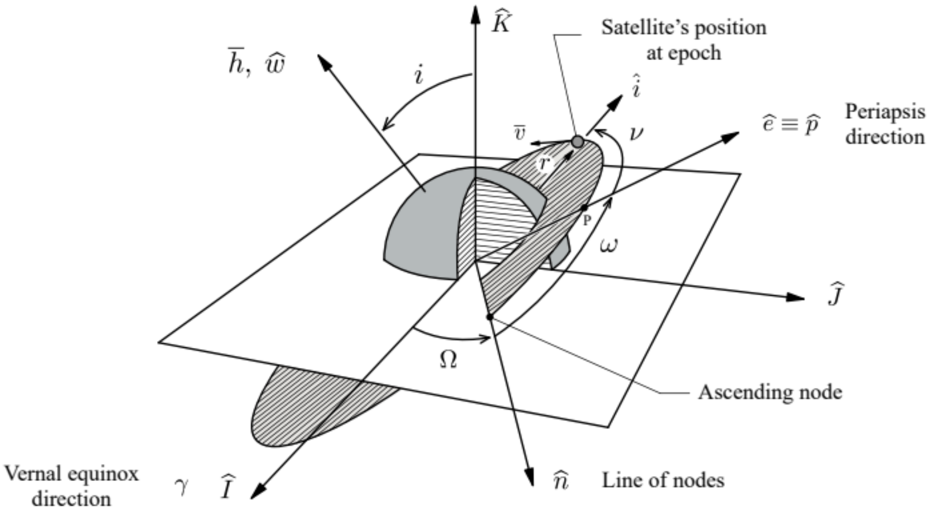


Figure 2.3: Classical Orbital elements [16]

An orbit that can be described by these six parameters, constant over time (except for ν), is called a Keplerian orbit. This type of orbit is the closed-form solution of the two-body equation of relative motion, based on the following assumptions:

- only two bodies in space are considered;
- their gravitational fields are spherically symmetric;
- the only source of interaction between the bodies taken into account is the mutual gravitational attraction.

Any kind of effect that causes the motion to deviate from a Keplerian trajectory is known as a perturbation, added in the equation 2.16

$$\ddot{\overline{r}} = -\frac{\mu}{r^3} \cdot \overline{r} + \overline{a_p} \tag{2.17}$$

where the vector $\overline{a_p}$ is the net perturbative acceleration from all sources other than the spherically symmetric gravitational attraction between the two bodies. An example of perturbations of two-body motion could be

- gravitational interactions with celestial objects like the moon and the sun;
- non-spherical central body;
- solar radiation pressure;

- atmospheric drag.

There are two main categories of perturbation techniques:

- Special perturbations: techniques which deal with the direct numerical integration of the equations of motion including all necessary perturbing accelerations, e.g. Cowell, Encke, Variation elements techniques;

- General perturbations which involve an analytic integration of series expansions of the perturbing accelerations, e.g. SGP, SGP4, BL.

In the thesis work presented, in the Orbital Simulator developed, attention has been focused on General Perturbations and in particular in the SGP4 method. Further details can be found in the relative section 3.2.1 and in the dedicated reference [17].

2.1.4 Attitude Dynamics and Kinematics

In this final section of the Spacecraft Mechanics introduction, the focus is on the kinematics and dynamics of a satellite.
The whole rigorous discussion is clear and complete in [14], while here there are going to be reported only the most interesting results and equations for the work.
Attitude Kinematics
To retrieve and understand the equation for the evolution of the quaternion in time, first of all, it is necessary to define the different angular velocities involved, taking into consideration the reference frames presented in 2.1:

- $\overline{\omega}_{OB}^{B}$ represents the angular velocity of the spacecraft relative to the local orbital frame expressed in the Body frame;

- $\overline{\omega}_{IB}^{B}$ represents the angular velocity of the spacecraft relative to the inertial frame expressed in the Body frame;

- $\overline{\omega}_{IO}^{O}$ represents the angular velocity of the orbital frame relative to the inertial frame expressed in the orbital frame and is defined as

  $$\overline{\omega}_{IO}^{O} = \begin{bmatrix} 0 & -\omega_0 & 0 \end{bmatrix} \tag{2.18}$$

  with ω_0, the scalar value of the angular velocity of the spacecraft about the Earth's center, calculated as

  $$\omega_0 = \sqrt{\frac{\mu_{\oplus}}{r^3}} \tag{2.19}$$

These three angular velocities are correlated via the following equation

$$\overline{\omega}_{IB}^{B} = \overline{\omega}_{OB}^{B} + \mathbf{A}_{O}^{B} \cdot \overline{\omega}_{IO}^{O} \tag{2.20}$$

with $\mathbf{A}_O^B$ transformation matrix needed to convert $\overline{\omega}_{IO}^O$ expressed in orbital frame, into body frame.
Considering the definitions provided and using quaternions as attitude representation, the attitude evolution is given by integrating the following equations:

$$\begin{aligned} \dot{q_0} &= -\frac{1}{2} \cdot \overline{\omega}_{OB}^B \cdot \vec{q} \\ \dot{\vec{q}} &= \frac{1}{2} \cdot q_0 \cdot \overline{\omega}_{OB}^B + \frac{1}{2} \cdot \vec{q} \; x \; \overline{\omega}_{OB}^B \end{aligned} \tag{2.21}$$

where $\vec{q}$ and q_0 are respectively the vector and scalar part of the quaternion, as previously defined.

Attitude Dynamics

For the study of the dynamics, instead, besides the introduction of the angular velocities, it is essential to define the angular momentum of the spacecraft, measured in the body frame:

$$\overline{h}^B = \mathbf{I} \cdot \overline{\omega}_{IB}^B \tag{2.22}$$

where $\mathbf{I}$ is the inertia matrix of the spacecraft.
Assuming the spacecraft as a rigid body, it implies that the time derivative of the inertia matrix is equal to zero, therefore the time derivative of the angular momentum is given by

$$\dot{\overline{h}}^B = \mathbf{I} \cdot \dot{\overline{\omega}}_{IB}^B \tag{2.23}$$

This expression would be more useful after having introduced the well-known *Euler′s equation*, which characterizes the dynamic of a rigid body about its center of mass with respect to an inertial frame, indicating how the angular momentum varies over time, taking into account the variations due to the applied torques:

$$\dot{\overline{h}}^B + \overline{\omega}_{IB}^B \times \overline{h}^B = \overline{T}^B \tag{2.24}$$

where $\overline{T}^B$ includes all the external torques acting on the spacecraft with respect to its center of mass expressed in body frame.
By substituting the expression 2.23 in 2.24, the following equation for the angular acceleration is retrieved:

$$\dot{\overline{\omega}}_{IB}^B = \mathbf{I}^{-1} \; (\overline{T}^B \; - \overline{\omega}_{IB}^B \times \mathbf{I} \; \overline{\omega}_{IB}^B) \tag{2.25}$$

Integrating 2.25 over time allows to get the angular velocity of the spacecraft.

Disturbance torques

As introduced in 2.24, external torques acting on the spacecraft contribute to its dynamic, therefore, it is important to take into account all the torques, generated by the space environment, which perturbs the satellite's dynamic. Some of these

disturbances generate forces which in turn yield non-negligible perturbation torques. For the orbits of interest and satellites orbiting around Earth, usually four of them are taken into consideration, which are the ones due to gravity, Earth's magnetic field, solar radiation and aerodynamic drag effects.
How these will be modeled is presented in the dedicated section 3.2 in chapter 3.

2.2 Data Transmission

After having given a brief introduction to the Orbital mechanics and Space environment, since the work concerns even communication aspects between space devices inside a satellite, it is fundamental to discuss also the basic concepts of data transmission.
The first point to be introduced is how the communication between the board and the components is managed, starting from the explanation of the communication models and ending with the description of the different most used protocols.

2.2.1 Communication models

Two communication models allow to describe the network access to resources and communication:

- **Master/Slave**: one device acts as a master device establishing the timing and controlling communication. The master initiates communications with the other devices, the slave cannot do the same with the master device nor can it communicate with the other slave devices;

- **Peer-to-Peer**: the devices behave equally, so any device/node can initiate communication; depending on the network connections, the data is exchanged with one peer or all the peers. When the number of nodes increases, the resource also increases but the performance suffers during heavy communication loads.

2.2.2 Serial and Parallel Communication

There are two ways of transmitting a byte between devices:

- **Serial**: the bits are transmitted as a series of pulses; these interfaces have low data rates and high reliability. Usually used for device control.

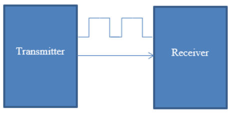


Figure 2.4: Series Interface [18]

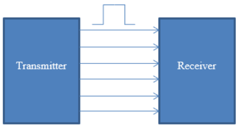
- 

Figure 2.5: Parallel Interface [18]

Parallel: each bit has a single wire devoted to it and all the bits are transmitted at the same time; these interfaces have high data rates. Usually used for large data usage like image sensor or camera.

Serial Communication

There are two types of serial data transmission:

- **Synchronous** Data Transmission: data is transmitted one bit at a time, using a clock to maintain integrity between words. The main advantages are that only one (half) or two (full duplex) wires are required to send/receive data and low cost due to low number of wires. The main disadvantages are lower speeds than parallel transmissions and difficulty in maintaining data integrity due to problems with synchronizing clocks.
  The sender sends a clock signal along with data at every rising/falling edge of the clock, so the data value is read by the receiver.

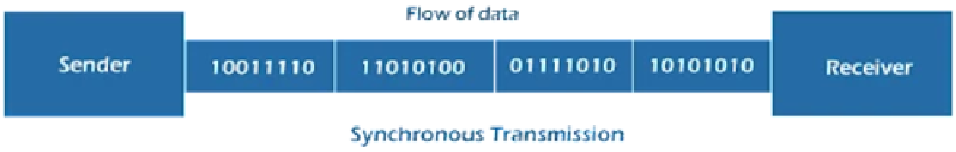


Figure 2.6: Synchronous Mode [19]

- **Asynchronous** Data Transmission: data is transmitted one bit at a time using start bits and stop bits to maintain integrity between words. The main disadvantage is lower speeds than parallel transmissions.

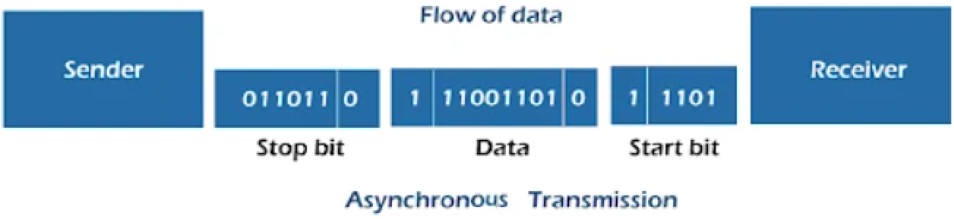


Figure 2.7: Asynchronous Mode [19]

Transmission modes

Three main ways to transmit the data:

- One-way transmission

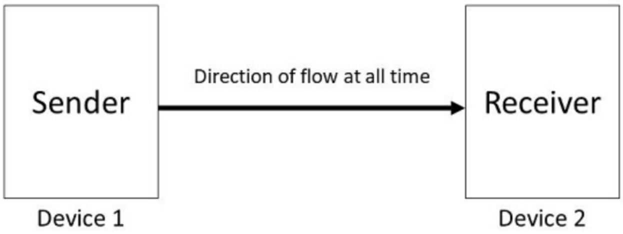


Figure 2.8: One Way Transmission Mode [20]

- Two-way transmission (half-duplex): only one end can communicate at a time;

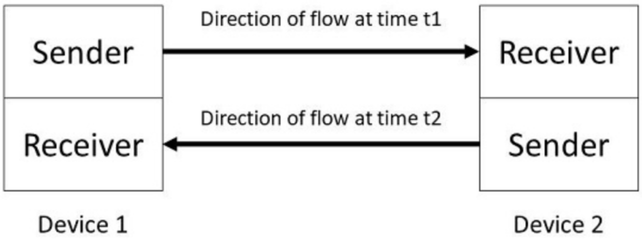


Figure 2.9: Half-Duplex Transmission Mode [20]

- Two-way transmission (full-duplex): both ends can communicate simultaneously.

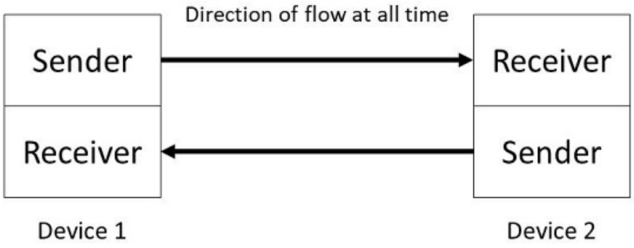


Figure 2.10: Full-duplex Transmission Mode [20]

2.2.3 Communication protocols

The concept of protocol itself is nothing new for anyone, as it is used in each field of everyday life but this section aims to focus on what is a protocol for a space component.
In the context of space systems and spacecraft, a protocol typically refers to a set of rules and conventions that dictate how different components or subsystems communicate with each other. These protocols are crucial for ensuring that various parts of a spacecraft or space system can work together seamlessly.
So protocols define the standards for communication between different components, such as sensors, instruments, control systems, and data processors. These standards specify how data is transmitted, received, and processed, so the format of data messages, including how data is structured, the encoding used e.g. binary or ASCII, and any error-checking mechanisms, e.g. checksums, to ensure data integrity.
There is the possibility that protocols also include timing and synchronization requirements to ensure that data is transmitted and received at the right time and in the correct sequence, as precise timing is critical for mission success. These may be used in some cases also to define how errors are detected and handled, involving the mechanism of re-transmitting data in case of transmission errors or for notifying the system of errors.
Usually, the details of these protocols are documented in user manuals and Interface Control Documents (ICDs): user manuals provide information to users and operators of the space component, while ICDs define the interfaces and interactions between different components within a space system.
Many space agencies and organizations, such as NASA or the European Space Agency (ESA), develop and adhere to specific communication protocols, e.g. Consultative Committee for Space Data Systems (CCSDS) protocols, MIL-STD-1553, and SpaceWire among others, to ensure interoperability between different space components and missions, but sometimes, depending on the specific mission and the components involved, these may be customized or modified to meet the mission's unique requirements.

Series Interfaces of interest for the work:

- **SPI** or Serial Peripheral Interface: a synchronous protocol that allows a master device to initiate communication with a slave device; data is exchanged between these devices. This interface is both synchronous and full duplex.
  There can be multiple nodes in the network: one of them is the master and the rest are slaves, with the communication always initiated with the master; the slaves can only communicate with the master.

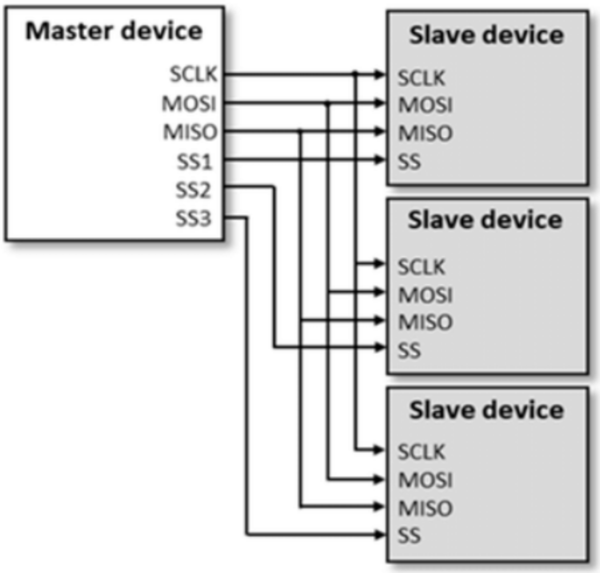
  

  Figure 2.11: Example of SPI schematics with multiple slaves [21]

  The main actors displayed in the schematics in 2.11 are:

  - CLK is generated by Master and is used as the mode is synchronous;
  - MOSI is Master Out Slave In: Data sent by master to slave;
  - MISO is Master In Slave Out: Data sent by slave to master;
  - SS is slave select: the slave communicates with master only if this pin is set as "Low".

  The communication protocol works as represented in figure 2.12.
  The number of lines in an SPI increases with the number of devices, making this one of the main disadvantages.
  The SPI mode defines which SCLK edge is used for toggling data and which SCLK edge is used for sampling data.

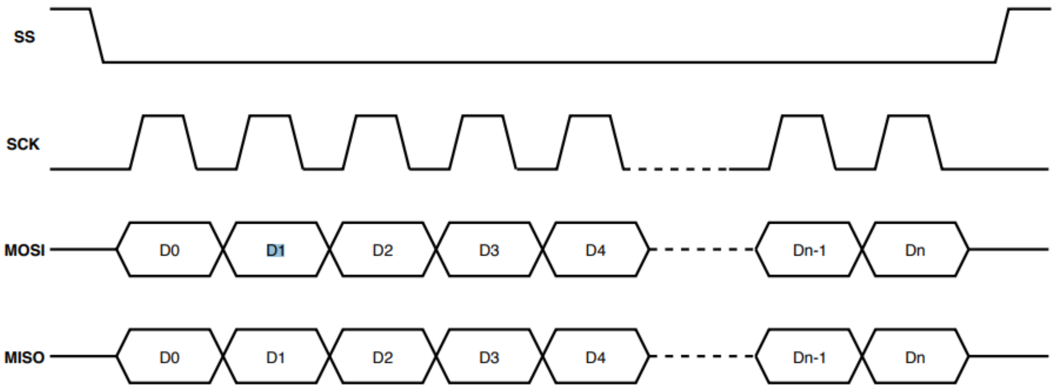
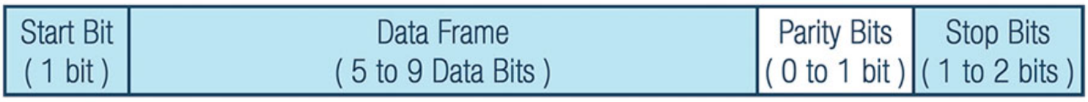


Figure 2.12: Example of simple SPI communication [22]

- **UART**, Universal Asynchronous Receiver-Transmitter, is a simple, asynchronous serial communication protocol used for basic data exchange between components, used for data rates below 100 kbps for a single device and when the total quantity of platform having a 1553 interface is less than about 20; are point-to-point links and require more hardware in the RTU and more harness for each device than a data bus.
  Used for devices where low power consumption and mass is required.
  The communication standards used for UART communication are RS-232, RS-422 and RS-485:

  - RS-232 is a common serial communication standard, used for low-speed data transfer between COTS components in space systems and often used for configuration, monitoring, and control purposes;
  - RS-422/RS-485 are serial communication standards that offer better noise immunity and longer communication distances compared to RS-232, used for data transmission between COTS components within a spacecraft; they specify only the electrical characteristics of the interface.

  The main differences between these standards are summarised in table 6.3.
  A connection with this bus requires one wire for the transmission of data; therefore clock line is not used and the data is transferred by pulling down the line to send data.
  The transmission starts with a "start bit" followed by 7 data bits; a parity bit is included depending on the design and the transmission ends with a 'stop bit', as shown in figure 2.13.
  The UART's real data transmission rate is 70 percent.



Start Bit (1 bit)	Data Frame (5 to 9 Data Bits)	Parity Bits (0 to 1 bit)	Stop Bits (1 to 2 bits)

Figure 2.13: UART data transmission structure [23]

- **I2C**, Inter-Integrated Circuit, is a multi-master protocol that uses two signal lines, called 'serial data' (SDA) and 'serial clock'; as displayed in figure 2.14, virtually any number of masters and slaves can be connected to these lines.

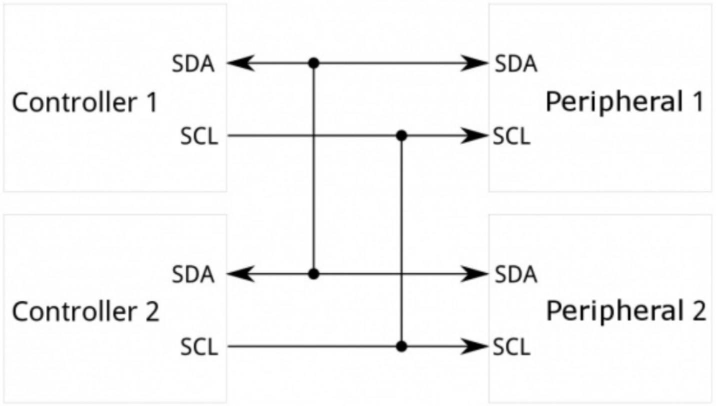


Figure 2.14: Example of I2C schematics diagram [24]

These communicate with each other using a protocol that defines: 7-bit slave addresses, one for each device connected to the bus, data is divided into 8-bit bytes, a few bits for controlling the communication start, end, direction, and for acknowledgement.
The SDA and SCL are connected to the entire device on the I2C bus: the master is the device that drives the SCL clock line, the slaves respond to the master and cannot initiate a transfer over this bus.

I2C nominal protocol: as shown in figure 2.15, when the master wishes to talk with the slave, it issues a start sequence on the I2C bus; the start and the stop sequence are special in a way that it is in these places the SDA is allowed to change when the SCL line is high. When the data is transferred, the SDA should not change whilst the SCL is high.

- **MIL-STD-1553**: traditional choice for a data bus in space, developed by US and used for data rates below 100 kbps for a single device and when the total quantity of platform having a 1553 interface is less than about 25 (slow for

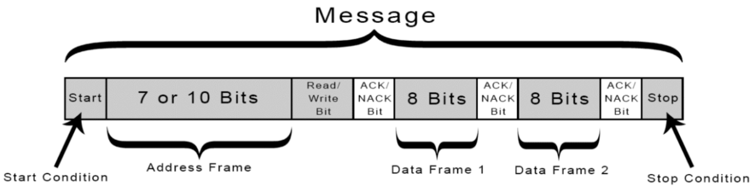


Figure 2.15: I2C protocol [25]

modern satellite applications).
Typical satellite usage today is 10-15 units connected to a 1553 bus.
Used for devices where 3W peak power consumption **can** be tolerated.

- **CAN**: Controller Area Network, is a multi-master system, therefore increasing the complexity of the system; used for data rates below 50 kbps for a single device and when the total quantity of platform having a 1553 interface is more than 20/25; used for devices where 3W peak power consumption **cannot** be tolerated.
  The CAN bus system, in figure 2.16, consists of a bus wire, bus terminations, and a bus station that contains a micro-controller, CAN controller, and a bus driver: the micro-controller edits the data received or transmitted and controls the communication, the CAN controller is an interface to the physical bus wire, it realizes the functions of CAN bus such as bus arbitration, error handling, bit stuffing, etc.

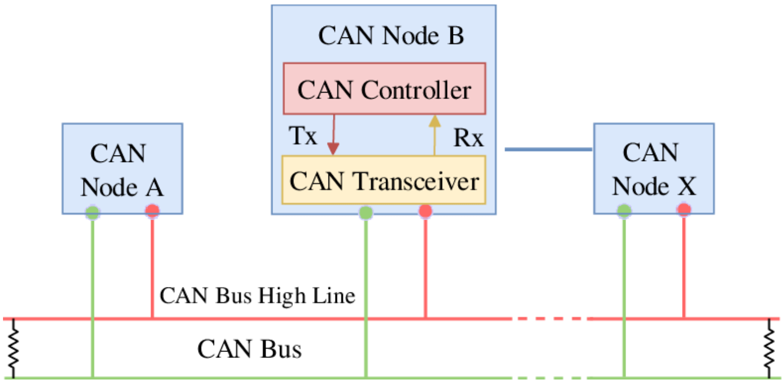


Figure 2.16: CAN bus architecture [26]

- **Space-Wire**: originally developed by ESA to provide data link with a high bandwidth to complex missions, also defined in ECSS-E-ST-50-12C, used for most european and american missions.

It connects different nodes with low latency, full duplex, and point-to-point connections and it is used for data rates above 100 kbps for a single device and when the total quantity of platforms having this interface is less than 10-14; to extend beyond this number, large Space Wire routers are needed; used for devices with low power consumption and low masses.
It is a point-to-point serial data link encoding data using two differential pairs in each direction, for a total of eight signal wires: two signal pairs are for data in and out while the other two wire pairs carry data strobe in and out.
Designed and developed to meet the EMC specifications of a satellite.

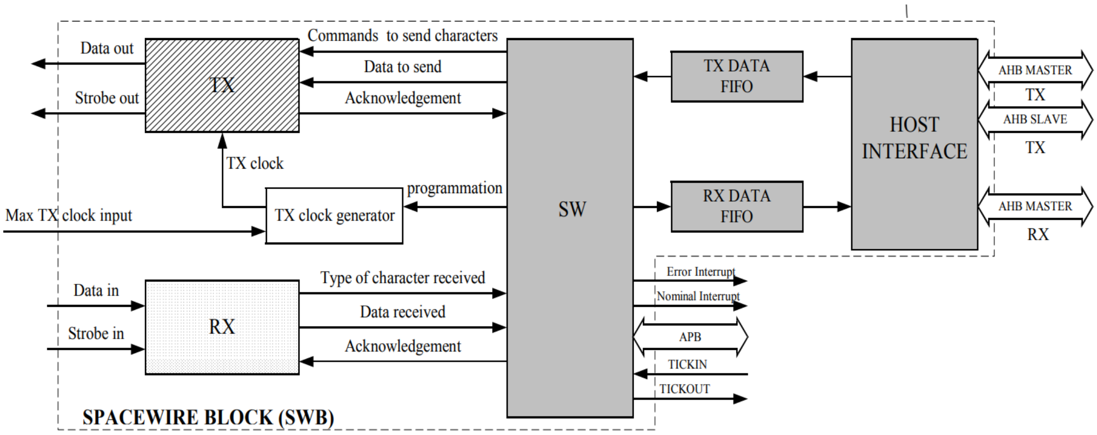


Figure 2.17: SpaceWire block diagram [27]

- Ethernet: with its associated TCP/IP protocol suite, is increasingly used in space applications, especially for high-data-rate communication, payload data handling, and telemetry.

In the work presented, UART interfaces are the most used, therefore it could be interesting to further highlight the different methods of communication and the differences between the standards already mentioned.

Comparison between UART TTL communication and RS-232 communication

Many UARTs (built-in micro-controllers) receive and transmit data serially, they transmit one bit at a time at a specified data rate: a method called TTL serial. Serial communication at a TTL level is always between 0 and Vcc which is earlier 5V or 3.3V, a logic high '1' is represented by Vc, while a logic low is 0V.
RS-232 signals are similar to microcontrollers with respect to the idea they transmit one bit at a time, with a specified baud rate, with or without parity and/or stop

bits; for the RS-232, logic high '1' represents a negative voltage between -3 to -25V and a low logic '0' transmits a positive voltage between +3 to +25V.
RS-232 can travel longer distances than the TTL counterparts while providing a reliable data transmission.
To connect the two ports, one has to invert the signals and also regulate the voltage so that it does not destroy the microcontroller's serial pins.
MAX-232 IC uses transistors or inverters to flip the signals and charge pumps for RS-232 voltage.

Comparison between RS-232, RS-485, RS-422 serial interfaces

	Line Configuration	**Mode of operation**	**Max Data Rate [kbits/s]**
RS-232	Single-ended	Simple or Full duplex	20
RS-485	Differential	Simple or Half duplex	10000
RS-422	Differential	Simple of Half duplex	30000

Table 2.1: UART Comparison

When communicating with high data rates or over long distances, single-ended methods are not enough: differential signals help nullify the effects of ground shits and induce noise signals that can appear as common mode voltages on the network.

2.3 Simulations

As the last part of the work will involve simulation tools and setups for testing the modular ADCS application implemented, it is useful to introduce also the differences between types of simulation available, according to the phase of the project and the purpose of the activity.

2.3.1 SiL and HiL simulations

Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) simulations are crucial testing methodologies that help engineers and scientists validate and verify the functionality and performance of complex systems before they are deployed in real-world environments, ensuring their reliability and safety.
It is interesting to highlight the differences between them and also the several reasons why they are so useful, especially in this sector.
Software-in-the-loop simulation involves testing the software component of a system in a virtual environment without the actual hardware. In other words, it runs the software on a computer to simulate how it would interact with the

hardware and external systems. It is very useful for different reasons, e.g. enabling early development and debugging of software components, for cost-effectiveness because it does not require physical hardware and so accessible for testing and development throughout the project's lifecycle, for the repeatability, essential for verifying the software's behavior under various conditions, for the safety, as it helps ensure that software functions as intended, minimizing the risk of software-related errors or failures once the hardware is integrated and at the end also for the scalability, which allows to simulate the behavior of various parts of the space systems, making it easier also to test complex, interconnected systems.

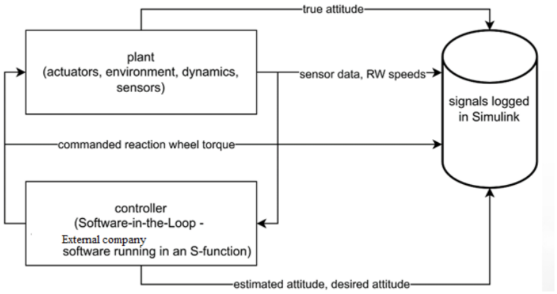

true attitude
plant
(actuators, environment, dynamics, sensors)
sensor data, RW speeds
signals logged in Simulink
commanded reaction wheel torque
controller
(Software-in-the-Loop -
External company
software running in an S-function)
estimated attitude, desired attitude

Figure 2.18: Example of SiL workflow [9]

Hardware-in-the-Loop simulation instead involves testing a system's hardware components by connecting them to a simulated environment that mimics the real-world conditions they will encounter. It interfaces real hardware with simulated or emulated components.
Although it is a more expensive, complex and longer process, enables the testing of actual hardware components in a controlled, repeatable environment, providing insights into their performance and reliability and how these interact with each other before they are physically integrated into the complete space system. Another interesting aspect is the possibility, with that kind of simulation, to inject faults or anomalies into the simulated environment to assess how the hardware and the software respond, helping to identify vulnerabilities and develop fault tolerance mechanisms.
Furthermore its value concerns also the opportunity to replicate extreme environmental conditions that space systems will encounter and also to validate that the hardware behaves as expected and meets the project's requirements, ensuring mission success.

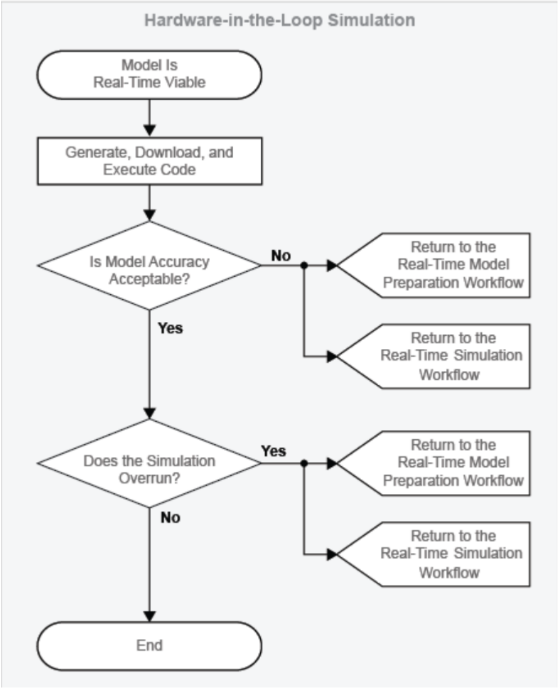

Hardware-in-the-Loop Simulation
Model Is Real-Time Viable
Generate, Download, and Execute Code
Is Model Accuracy Acceptable?
No
Return to the Real-Time Model Preparation Workflow
Return to the Real-Time Simulation Workflow
Yes
Does the Simulation Overrun?
Yes
Return to the Real-Time Model Preparation Workflow
Return to the Real-Time Simulation Workflow
No
End

Figure 2.19: Example of HiL workflow [28]

In conclusion, both the simulations presented are essential to help identify and mitigate issues early in the development process, increasing the chances of mission success, the safety of every kind of payload and contributing also to cost savings, by reducing the need for physical prototypes and allowing for iterative development and testing.

Chapter 3

Software Development and Implementation

Since an overview of the MA61C software has been already discussed in section 1.2.1 and the main design of the reference modular ADCS project has been handled in section 1.2.3, in this chapter there are going to be discussed all the main actions applied on the software side for the work, giving also an insight into how the software works and interacts with the hardware.
First, the focus is going to be on the onboard computer (OBC) software together with the modular ADCS one, and lastly the Orbital Simulator.

3.1 OBC and ADCS Software

On the Software side, it could be relevant to report all the main updates applied with respect to the original SmallSat version of the project.
In particular, it is interesting to focus on the following aspects, which represents also the different versions of the software that are going to be tested and explained in chapter 5:

- First of all the necessary changes made to obtain a compilable version of the code to load and run into the hardware;

- Interfaces, in particular focusing on a code which allows to test the correct operation of all the interfaces on the on-board computer, that will be used for the communication with the components.

- With both a compilable file properly modified and an interface test code, it has been necessary to create a communication test code, trying to send some random data and receive the response, even without any component or EGSE connected, just for checking that the communication line is operating properly.

- After the communication test code, the next step was to update the software to exchange data with real components integrated on the bench and connected to the OBC. Proper sections of the code have to be added to receive and read correctly the outputs from the physical sensors, and eventually to send commands to the actuators.
- The ultimate step has been to assemble a complete version of the software which includes all the elements needed to run the simulation, involving also the communication with the orbital simulator, explained in section 3.2.
- In parallel with the previous activities, there have been followed also other software developments, such as:
  - Json files update;
  - Protocols update.

Since the whole code is under the property of the company, there are going to be discussed, conceptually, all the actions applied to adapt the reference version of the software, without mentioning or showing the particular functions or sections of the code.

3.1.1 Simulation Software

The major changes applied to the main code could be summarised in the following points:

- The files in the *source* folder of the original version have been compared to the ones in the CubeSat source folder:
  - All the useful databases present in the mADCS project have been added, including the one containing the main settings for the ADCS software;
  - The databases related to commands and telemetries have to be replaced with the ones from the subsystems of the MATB project, eventually adding the external ones (refer to 3.1.3).
  - The parser files, which manage the communication with all the interfaces available, have to be modified according to the devices available: initialization, sending, and receiving functions have to be properly customized.
  -

- To modify specific functions of the ADCS software, according to the need, it is necessary to identify the macro-folder which contains the files of interest, mainly among *executables* for *mode* updates and *components* for determination and control algorithms, and then directly apply the modifications into the desired file.

- In the *main* file (ADCS_performance.cpp) several changes have been applied:
  - Some libraries have been included in the header section at the beginning of the code, according to the functions added;
  - in the *recall* section of the source files for the board, almost the same files have been added, changing the name of the relative folders;
  - in the section for the configuration of the UART interfaces, the inputs of the functions have been updated with the correct values for the new components chosen;
  - in the section for *setting variables*, all the additional required initialization have been added, according to the new parameters introduced;
  - in the section for *initializing* the third-party ADCS software, a few changes have to be made:
    - mode and scenario to be simulated are overwritten;
    - values of time parameters have been updated according to the orbit of the scenario chosen;
    - specific physical components are properly initialized by setting determined operating modes or configuration parameters.
  - in the section to retrieve the telemetry from the connected components, for each of them, these actions are introduced:
    - a function to send the adequate command, providing the relative number, to a specific subsystem, for requesting the telemetry;
    - a section for processing the data, further explained in chapter **??**, received from the subsystem;
    - a section for storing the processed data in the variables employed in the ADCS software itself.
  - in the section for calling the ADCS software
    - the functions to run the software have to be kept as they are;
    - the displayed output can be changed according to the need for the specific test;
    - for the simulation, as will be further discussed later in the work i.e. 3.2.3, it can be also useful to send some computed parameters, from

the OBC hardware to the orbital simulator, since the simulation will be closed-loop: for this purpose, first these parameters have to be stored into proper variables, converted into floating values and then sent via specific functions, through the chosen interface, to the end user.

- For the compiling of the main file, different commands have been used in the prompt: for the original version it has been used the following one

  ```
  "sparc-gaisler-rtems5-g++ -Wall -O2 -I -L components/preprocessor/preProcessor.cpp configuration/control.cpp components/orbitDetermination/*.cpp components/attitudeDetermination/*.cpp components/controllers/torqueController/*.cpp components/controllers/magnetorquerController/*.cpp tools/math/*.cpp components/models/*.cpp components/guidanceModule/*.cpp tools/models/*.cpp components/torqueAllocation/*.cpp components/schmittTrigger/*.cpp spin/*.c -mcpu=leon3 -qbsp=leon3 ADCS_performance.cpp -o ADCS_performance_test"
  ```

  This command has been composed by taking into consideration the user manual of the RCC, cross-compilation system for LEON processors (for further details refer to [29]). Besides the add-ons, $sparc - gaisler - rtems5 - g + +$ is the command to compile *.cpp* files; after $-L$ all the files needed for the compilation process are included, both ADCS and OBC related ones; after $-o$ the name of the compiled file is provided, which will be the output of the command, in the same folder which collects the main file.
  To compile the new version of the software instead, the same command has been used but without the inclusion of the *.c* files in the source folder, already included in the *.cpp* file to be compiled;

3.1.2 Interface and communication test code

In this section the aim is to report the steps followed to update the Interface Test code, already existing, in order to test the correct operation of all the interfaces available, before using them for the communication between the components, integrated on the test bench, and the MA61C Cubesat board, as On-Board Computer. The previous version of the code already includes the procedures to test the following interfaces: SpaceWire, CANbus, MIL1553B and UART; therefore in this section, there will be described only the remaining ones:

- I2C;

- SPI;
- GPIO;
- GPI;
- MT pins.

Not all of them will be useful for this work, but the additional ones may be needed to extend the range of the future applications of the solution. Focusing on each interface analyzed, one by one, the procedure followed to develop and test them with the MA61C Cubesat board is going to be explained.
It is important to point out that for the development of each interface test code, two main documents have been taken as a reference:

- MA61C CubeSat Verification document: as a guide for the connections between the interface and the board, and for the former manual test procedure, already written and used until the automated software version is ready.
- GR712RC User Manual: as a guide for the processor of the board, to understand the addresses and the functioning of each interface throughout the board's point of view.

I2C
Following the order already present in the code, after the UART, there is the I2C interface.
On the development side, first of all, the starting step has been to create a section to interact with the user, guiding him to set, through the dedicated GUI for the I2C interface, the proper parameters of the device connected, such as the address, the bit rate and the mode. After having done that preliminary setting, it is necessary to understand if the device has been properly identified by the board, using the *i2cScan* function available in the source folder, which also initializes the speed of the interface.
After scanning, the address identified is returned by the function and a confirmation is asked to the user, before the writing step.
As it has been done for the scanning, also for the writing it has been used an embedded function $i2cWriteSingle$, which takes in input the data that it is desired to write and the address to which the data has to be sent; the message is sent to the transaction log of the GUI of the interface and again a confirmation is asked to the user.
After the writing check, it remains only the reading one, achieved by using the *i2cRead*, taking in input the address from which it has to be read, after having told the user to send a message from the GUI of the interface device to MA61C. Also for that last step, a confirmation from the user is required to proceed.

If all the confirmations have gone correctly, the test procedure for that interface is passed.

SPI

For this interface, the device initially used has been the same as the one for the I2C interface, since it can be both an I2C and SPI adapter to the computer. Unfortunately, after having set it up with the connections for SPI mode, there have been some issues in the communication with MA61C board, in particular in the writing action from MA61C to the device.
That issue forced to change the device used to a "Bus pirate v3.6", but unfortunately even with that one, any communication has been established.
The last and successful attempt has concerned the Arduino UNO device, by setting it in a Slave Mode, with the proper pinouts available.
Therefore, in Arduino IDE, two different codes have been written: a code to initialize proper SPI master parameters to Ma61c, to write a test message, send it to the slave device (Arduino Unit) and after that to read the message sent back from the slave; the other code refers to the initialization of the SPI slave device, the receiving process of the message sent from the master and after receiving, the sending back action to the master.
In that way, both the communication lines, writing and reading, are tested and validated.

GPIO

The next interface test developed is the GPIO one, for which the testing procedure is different from the others: voltage measurements, using a multimeter, are necessary to verify the correct operation.
Before entering into details about the development of the procedure, it is necessary to point out that to allow the enabling and the possibility to use each one of the available GPIO pins in the board it has been required to disable the interfaces that operate using the same line, at least while this interface is going to be tested.
To achieve that, the procedure presented on the user manual of the processor, embedded in the board, to gate the clock for a core has to be followed: first, it has to be enabled the relative unlock register, then the core reset register, and after that disable the clock, enable the register of the core of interest, to conclude with the disabling of the unlock register.
For the test of this interface, there have been disabled the following cores of the processor: Space Wire, CAN, CCDS Telecommand, MIL-STD-1553.
Starting the development explanation of that code, first of all, since this interface is present in each one of the versions of the MA61C board, the user has to choose which board is connected for the test.
The procedure has been prepared only for the case of interest, therefore only for the Cubesat board.

The second choice that the user has to make is the GPIO pin that has to be tested: after that, via a proper function, a precise procedure for each of the different pins is followed to set at high or low levels the relative register bits and with them the output and the port direction registers. After all these operations, the user is requested for a confirmation, to check in the multimeter the voltage measurements according to the different cases tested.

GPI

The GPI interface is very similar to the previous one, except for the fact that in this case the communication is only in one direction, the input one.

The setup for this interface testing is a bit different because there is the need for an ARDUINO board, which has to be connected to the power and it has to provide the correct voltage to the pin of interest in the board to be tested.

According to the point highlighted above, also the interface test code is simpler than before and limited to checking for the correct value of the register bit, different for each case selected (as explained in the previous procedure). In case of power provided, the register bit has to be set to high, instead of low.

MT pins

The last interface test code added is related to the MT pins, whose procedure is very similar to the one of the GPIO pins.

Actually, the interface test code developed has the same structure as the one for the GPIO pins; the main changes are related to the different pins that have to be tested and with these also the commands that have to be sent to the board in order to set high or low values to the register bits in the registers of interest.

After having verified and validated that all the interfaces to be used are working well, the attention has been focused on the communication test code.

The main actions applied could be summarised in the following way:

- the functions useful for sending and receiving packets to and from the hardware, via UART interfaces, have been updated according to the CubeSat version protocols, different from the ones of the former SmallSat version;

- in the main file a dedicated section has been added, applying the following measures:

  - The port ID to which the request and the effective data are sent and received has been set and configured according to the device used for the data transfer between hardware and computer;
  - Due to an internal conflict, before sending and receiving anything, it has been necessary to disable the UART 5 interface of the hardware, which is not associated with any interface in the Cubesat version. It is furthermore

required, after that step, to ensure that the GPIO pin related to the UART 2 interface (RS485) is correctly enabled;

- since in the new configuration there is only the Ma61c Cubesat hardware, without the MA61c EGSE connected, it is necessary to set a proper waiting time between the send and receive packets action; this has been done just by putting the receive function inside a loop, which allows to wait enough time to receive the response back.

After having applied these precautions, the only action remaining is to build the message or random data in a format recognizable by the software as a good input, emulating the attitude of a real component.
According to that, it has been necessary to check in the former file, used for the emulation of the components, with which protocols the commands were sent to the emulated components, e.g. regarding the fine sun sensors, the protocol used is the NSP one, therefore data to be sent have to be first encoded according to that protocol. Having encoded data properly, via a specific debug tool used for UART interface communication (Termite), it has been possible to see data received and send a response back to the software every time the correct request data sequence arrives.

3.1.3 Json files and Protocols update

After having created a new version of the main code that is going to manage the communication between the onboard computer and the components on the bench, the next action is related to the update of the JSON file and the protocols of the devices chosen.
For the JSON file, better explained in the previous chapter, it has been necessary to take as a reference the interface control documents and the user manuals available for the components and extract from them the commands and the telemetries for each of these, to get, via a proper python code, the correct databases for Command and Telemetry. Since for that work there have been used some of the components already present in the Modular Avionic Test Bench project, the JSON file for these has been inherited and integrated with the additional devices required for this work, such as the analog sun sensor and the magnetometer (for this choice refer to 4.2).
Regarding the protocols, a similar approach has been applied: since most of the protocols and drivers have already been written for the parallel project, and the same for the magnetometer, used in previous activities, the only one that has had to be introduced is the sun sensor one. Therefore, referring to the data sheet of the specific version of the light sensor procured, it has been possible to understand the correct sequence of commands needed to send/write and receive/read data to/from

the device and then implement it on a software level.
After having updated all the protocols, the necessary step is to test them in order to understand if they allow to communicate in the correct way with the devices. The outcomes of these tests are reported in the dedicated section 6.1.5.

3.2 Simulator development

It is necessary also to provide proper inputs to the physical components mounted on the bench, therefore an orbital simulator has been developed.
This section describes the software components that form the reduced simulative verification environment. All components are implemented in MathWorks Simulink. The Simulink model in figure 3.7 does not show the complete ADCS control loop but it includes only what is needed for the specific application:

- the **Plant**, which is made of Space Environment, Disturbances and Satellite State blocks;
- **Sensor and Actuator models**: since not all the components needed for the correct operation of the software are physically available on the bench, some of them have to be properly modeled to provide additional telemetry or parameters requested.
- **Communication** blocks: since the data simulated has to be sent to the software under testing, and the output from the software has to be received as input into the simulator itself, proper blocks to exchange data between the hardware and the model need to be included.

All the other elements of the control loop, i.e. filters for navigation, control algorithms, etc. do not have to be simulated as they are embedded inside the On-board computer Software, which allows to command, control, and monitor the system during the simulation.

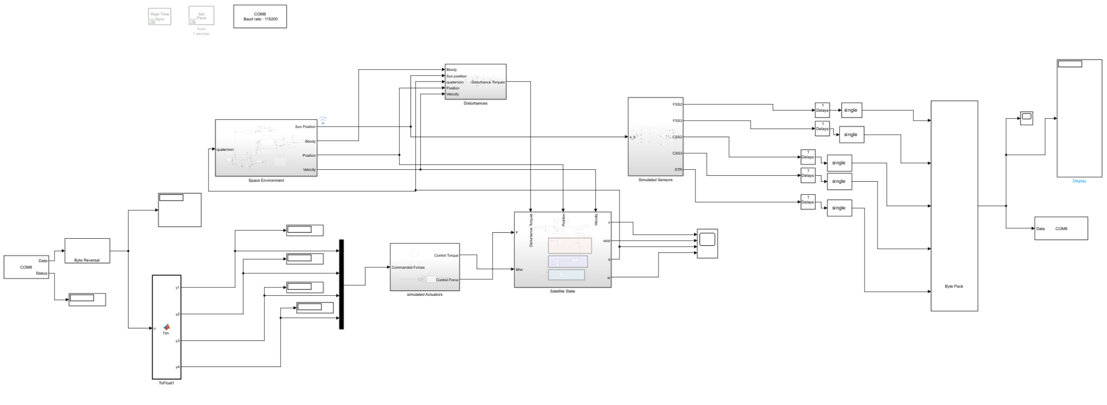


Figure 3.1: Overview of the Orbital Simulator

For the reasons explained, the main blocks which have been necessary to build are the following:

- Space Environment which includes a model for the Sun and Earth location, Earth Magnetic field and an orbit propagator, to determine the evolution of position and velocity vectors of the spacecraft along its orbit;
- Disturbances which includes all the external forces applied to the spacecraft along its orbit, to be evaluated and taken into consideration for the study of its Dynamic;
- Satellite State which includes Attitude Dynamics and Kinematics, involving both the external torques calculated and the ones applied by the actuators to correct the attitude as desired;
- Actuator Models which includes attitude controllers such as Reaction Wheels and Thrusters;
- Sensor Models which include mainly attitude sensors, such as digital and analog sun sensors;
- Transmission blocks which include sending and receiving tools, using Serial Interface.

The sample rate for the plant is continuous and the solver will make the step size as small as required for the model to fulfill the maximum tolerance of the model, which is 1E-12.

3.2.1 Plant

SPACE ENVIRONMENT
The Space Environment block designed contains, as already introduced, a time propagator, useful to update the Julian date; an orbit propagator, implementing the SGP4 algorithm, for the evolution in time of the state of the satellite into its orbit; a Sun position and Earth shadow model, determining the normalized position of the Sun and whether the satellite is under sunlight; and a magnetic field model, which estimates a magnetic field vector via IGRF model.

Time propagator
To update the Julian date for the simulation it has been enough to simply add to the Julian date of the starting date (i.e. 22nd of September 2021, the same date used in the former simulator), each running second, represented by a digital clock block, divided by the number of seconds in a day. In this way, it is possible to get the updated value and provide it as an output useful for software operations.

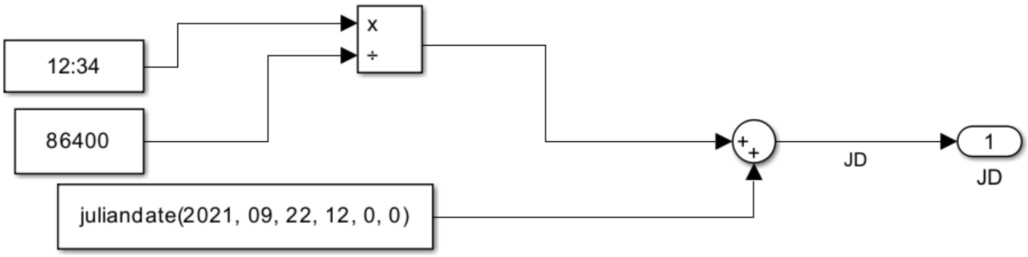


Figure 3.2: Time Propagator

Orbit Propagator

The most important block of the Environment subsystem is surely the propagator one because it allows to determine for the whole simulation time the current position and velocity vectors of the satellite along the orbit.
Several methods could be used, which take into consideration more or less factors, disturbances, perturbations, and in the end differ for the complexity and the accuracy that allow to achieve; in the case presented, in particular, the criteria followed for the choice has been to keep coherence with the project taken as a reference, to be able to compare the results of the simulation among these two more validly and loyally.
For this reason, as it was used for the SmallSat version of the project, the SGP4 model has been chosen: the Simplified General Perturbations model is a mathematical model used to calculate orbital state vectors of satellites and space debris relative to the Earth-centered inertial coordinate system; this model predict the effect of perturbations caused by the Earth's shape, drag, radiation and gravitation effects from other bodies, such as the sun and the moon and can be applied to near-earth objects with an orbital period of less than 225 minutes. In general, the model was developed by Ken Cranford in 1970, obtained by simplification of the more extensive analytical theory of Lane and Cranford, which uses the solution of Brouwer, for its gravitational model and a power density function for its atmospheric model. To go more in-depth into the development of this theory these are some of the references to follow: [30] and [31].
To implement this model, a code developed by professor Meysam Mahooti(2023) ([17]) has been used, inside a Matlab function block, which takes as input the satellite data and the time since the epoch: the satellite data are provided via a Two-line element format file, which is a data format encoding a list of orbital elements of an Earth-orbiting object for a given point in time, the so-called epoch; the second input is the time since the epoch, as a reference, at which it is requested to estimate the state of the satellite.

The TLE used as an input depends on the scenario chosen to be simulated, i.e. for the Space Tug scenario, referring to Spaceflight Sherpa FX-2, it is the following:
1 48958U 21059CH 21264.60415120 .00000642 00000-0 38926-4 0 9999
2 48958 97.5233 32.3239 0016080 313.6609 46.3289 15.14014873 13280
From this file, it is possible to retrieve the epoch, which in that particular case is 2021-Sep-21 14:29:58.6637 UTC: knowing the epoch and the desired starting date and time of the simulation, it is easy to determine the time since epoch to provide to the model, which in minutes is around 1290.7.
As already mentioned, the outputs of the propagator model are the state vectors, representing the position and velocity at this particular time frame considered, in TEME, or True Equator Mean Equinox, reference frame, an Earth-centered inertial coordinate frame, where the origin is placed at the center of the mass of Earth and the coordinate frame is fixed with respect to the stars.

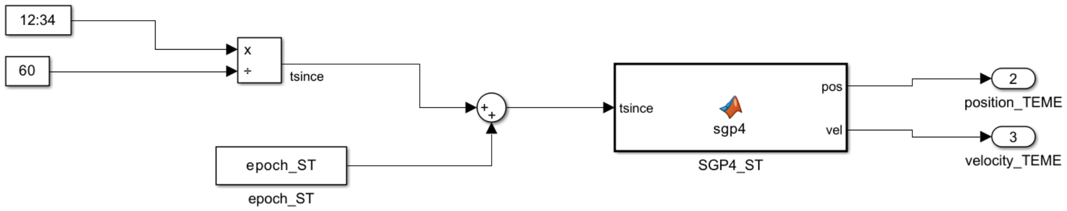


Figure 3.3: SGP4 Propagator

Since the TEME is an inertial frame but not the most used one, it has been useful also to convert the state vectors into ECI (Earth Centered Inertial) or ICRF (International Celestial Reference System, the current standard celestial reference system adopted by the International Astronomical Union.
For doing that conversion, it has been necessary, in a dedicated subsystem, to consider the Earth orbital parameters and calculate the precession and nutation contributions accordingly; after that, rotate the current vectors into the desired reference frame.
In this way there have been obtained the state vectors both with respect to TEME and ECI frame, to be used according to the application.

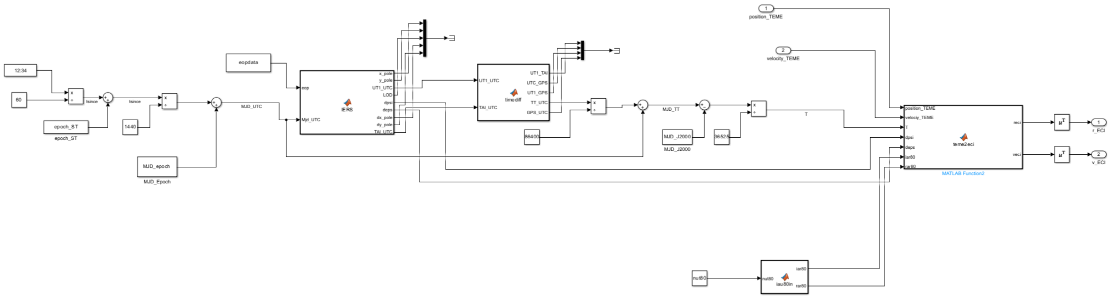


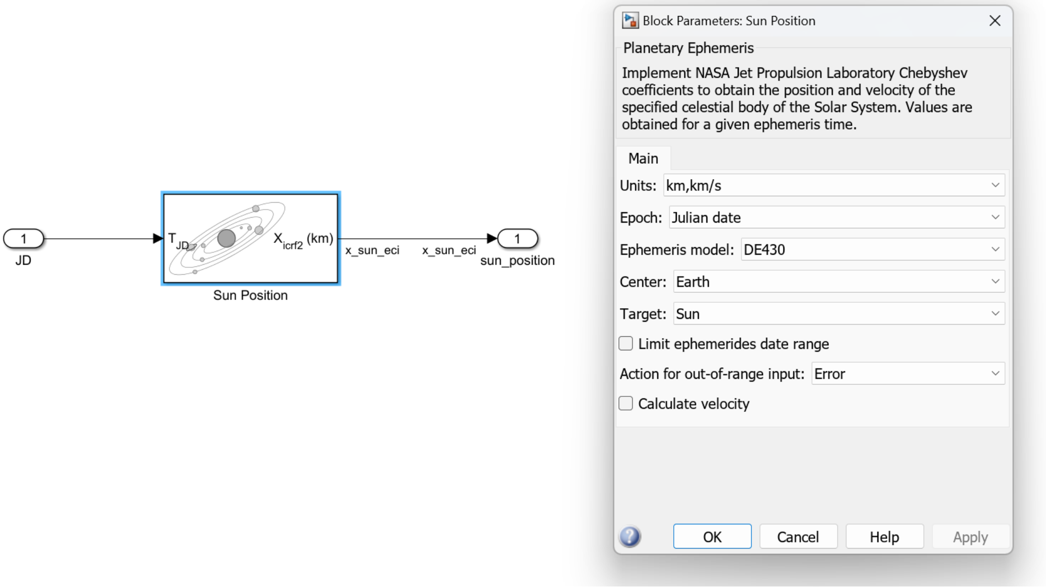
Figure 3.4: TEME to ECI reference Frame Conversion

Sun Position Model

For the determination of the position of the Sun, with respect to the Earth, two approaches could be used:

- analytical one: using proper equations to determine an approximate value, first in the ecliptic coordinate system and then converting it into equatorial and horizontal ones, for the observer's local time and location;
- Planetary Ephemeris block, embedded in Simulink toolbox, which uses Chebyshev coefficients to implement the position and velocity of the target object relative to the specified center object for a given Julian Date, taken as the only input of the block.
  For the use of this resource, besides the Julian Date of interest, it is necessary to specify even the Target parameter or the astronomical object for which we want to know the position and the Center parameter, the astronomical object taken as the reference.
  The output provided is in the International Celestial Reference Frame.

Both the methods have been implemented and the results have been compared, noticing that the Planetary Ephemeris model provides values closer to the ones obtained in the previous project, so this approach has been chosen for the simulator.



Block Parameters: Sun Position

Planetary Ephemeris

Implement NASA Jet Propulsion Laboratory Chebyshev coefficients to obtain the position and velocity of the specified celestial body of the Solar System. Values are obtained for a given ephemeris time.

Main

Units: km,km/s

Epoch: Julian date

Ephemeris model: DE430

Center: Earth

Target: Sun

□ Limit ephemerides date range

Action for out-of-range input: Error

□ Calculate velocity

Figure 3.5: Sun model

Eclipse Shadow Model

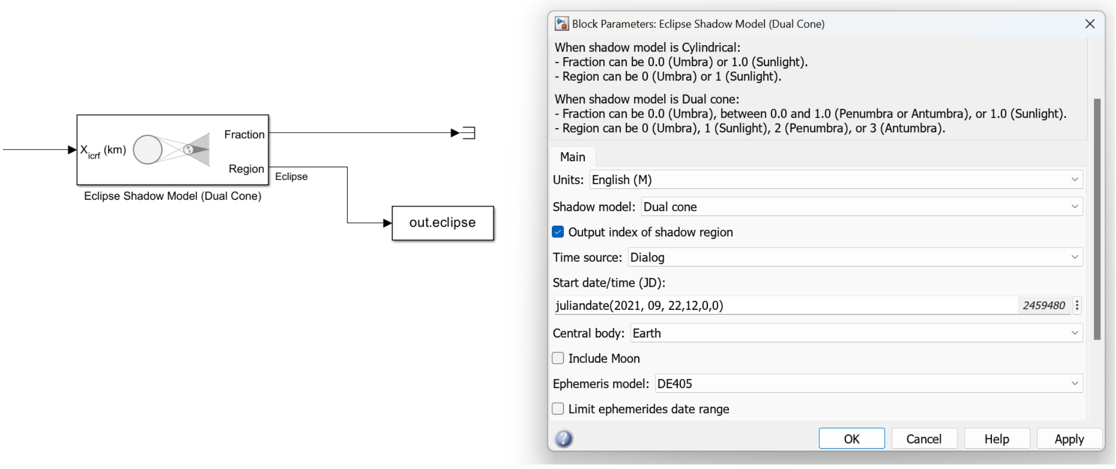
To determine whether the satellite is under sunlight or in eclipse along its orbit, it is necessary to build a block that models the Earth Shadow, in order to calculate the fraction of the solar disk that is visible from the inertial positions, provided as an input and previously calculated in the Orbit Propagator block.

For this purpose, it has been used the embedded block of Simulink, called the Eclipse Shadow Model, which considers the assumption of spherical shape for both occulted bodies and the Sun.

There are two shadow models available to choose from: a dual cone model and a cylindrical one. The first one differentiates between partial, annular and total eclipse, identified by specific numbers, which means the spacecraft could be in sunlight (1), penumbra, antumbra (0 to 1) or umbra(0); with that division, the eclipse is partial in penumbra, annular in antumbra, and total in umbra.

The second model is the cylindrical one, which differentiates only between total eclipse (umbra, 0) and full sunlight (1).

For the application presented, contrary to the reference project, it has been chosen to adopt the dual cone model, in order to exploit the different modes of the Sun Light Simulator, that has been used for the Hardware in the Loop Simulation, and will be discussed more in-depth in the dedicated chapter 5.2.



Block Parameters: Eclipse Shadow Model (Dual Cone)

When shadow model is Cylindrical:
- Fraction can be 0.0 (Umbra) or 1.0 (Sunlight).
- Region can be 0 (Umbra) or 1 (Sunlight).

When shadow model is Dual cone:
- Fraction can be 0.0 (Umbra), between 0.0 and 1.0 (Penumbra or Antumbra), or 1.0 (Sunlight).
- Region can be 0 (Umbra), 1 (Sunlight), 2 (Penumbra), or 3 (Antumbra).

Main

Units: English (M)

Shadow model: Dual cone

☑ Output index of shadow region

Time source: Dialog

Start date/time (JD):

juliandate(2021, 09, 22,12,0,0) 2459480

Central body: Earth

□ Include Moon

Ephemeris model: DE405

□ Limit ephemerides date range

Figure 3.6: Eclipse Shadow model

Earth Magnetic Field

The last thing remaining for the Space environment is to get measurements of the Earth's Magnetic Field varying with the position of the satellite and time. For this aim, the International Geomagnetic Reference Field block of Simulink has been used, which calculates the Earth Magnetic Field and secular variation using the selected IGRF generation, at a particular time and position, provided as an input. For the time, a reference date has to be introduced, directly inside the block, and this has been chosen to be the starting date of the simulation; for the position instead, as Longitude Latitude Altitude coordinates are requested, it is necessary to convert before the inertial position, in ECI frame, into LLA and then provide the resulting vector to the block.

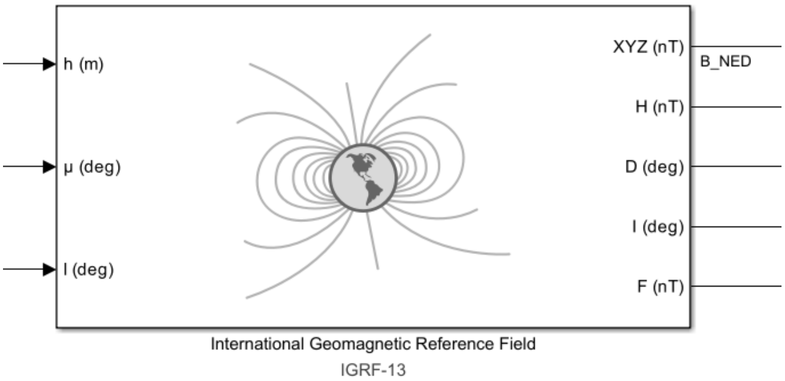


Figure 3.7: International Geomagnetic Reference Field model

The main limits of this block are related to the applicability area, valid between the heights of -1000 m and 5.6 Earth radii, and the validity in time, because the model used, IGRF-13, is valid until 2025. Nevertheless, for the application needed, it has been a good and quite accurate choice.
Two considerations are important to point out about the outputs obtained for the Magnetic Field vector: these are in nanoTesla unit and in the north-east-down, or NED, reference frame, so a further step has been necessary to get consistent results. First converting the output units from nanoTesla to Tesla and then changing the reference frame from NED to ECI, inertial, and body. Body measurements are useful to calculate the torque disturbance, relative to the magnetic contribution, in the body frame.
To do this reference frame conversion, Direction Cosine Matrixes have been used: first the one from NED to ECEF, then the one from ECEF to ECI in the inertial case, only the one from NED to Body for the body frame case.

DISTURBANCES

Before entering into the analysis of the dynamics of the satellite, this block is necessary to determine all the main disturbance torques that affect the attitude of the satellite, along its orbit. No forces have been evaluated separately because they affect only the evolution of the position on the orbit, and not properly the attitude; therefore for the application developed there have been necessary just the torques involved.
The main contributions taken into consideration are the following:

- gravity gradient torque;
- atmospheric drag torque;
- solar radiation torque;
- residual magnetic dipole torque.

The procedure followed to determine them has been guided by the work done in [14], refer to that for further details.

Gravity Gradient Torque

The origin of that disturbance arises from the fact that the satellite is travelling in a non-uniform central force field resulting in slightly different attraction of gravity across the satellite.
The model used to calculate it is based on the two-body approximation, i.e. Earth and spacecraft, with the Earth having a spherically symmetric mass distribution and the spacecraft assumed to be rigid and small compared to its distance from

the center of the Earth.
The general expression is the following:

$$T_{GG} = 3 \cdot \omega_0^2 \cdot [o_3^b \ \times \ I^b \cdot o_3] \tag{3.1}$$

in which o_3 represents the local vertical, third unit base vector of the orbital frame, in the body frame. This detail implies that the gravity gradient effect is a function of the attitude and that no torque is exerted about the local vertical direction.
This torque is often expressed in the principal reference system in which some of the terms related to the products of inertia in the inertia tensor vanish.
It is possible to write the three components of the torque in terms of 3-2-1 Euler angles, which result in the following form:

$$T_{GG,1} = \frac{3\mu}{2R^3} \cdot (I_3 - I_2) \cdot sin(2\phi)cos(\theta)^2 \tag{3.2}$$

$$T_{GG,2} = \frac{3\mu}{2R^3} \cdot (I_3 - I_1) \cdot sin(2\theta)cos(\phi) \tag{3.3}$$

$$T_{GG,3} = \frac{3\mu}{2R^3} \cdot (I_1 - I_2) \cdot sin(2\theta)sin(\theta) \tag{3.4}$$

Looking at the elements needed to calculate the components of the vector, it is evident that, besides the constants, the only inputs necessary are the inertia tensor of the satellite and the current orientation, in terms of Euler angles.

Atmospheric Drag Torque
The reason why this disturbance has to be considered is because satellites, at LEO altitudes, experience forces and torques which are a result of aerodynamic drag, since a spacecraft travels along the outer fringe of the Earth's atmosphere where the atmospheric density ρ is greater than zero. Relatively to the density profile to be chosen, there is more than one available but usually the one recommended by the ECSS standard atmosphere model, MSISE-90, is preferred.
This contribution could be evaluated in different ways: the simplest model consists of a scalar evaluation of the aerodynamic drag force imposed on the center of pressure with an offset from the center of gravity, ignoring the fact that the flow regime at LEO altitudes is far from continuous and considering only normal pressure forces; for simulation purposes, instead, it is preferable to have a model which evaluates the secular aerodynamic torque as a vector, expressed in the following way:

$$T_a = \frac{1}{2}c_D \cdot \rho \cdot v^2 \cdot \sum_{k=1}^{n} \vec{r_{s,k}} \times (n_k^T \cdot V) \cdot V \cdot A_k \tag{3.5}$$

in which n is equal to 6 and is the number of orthonormal surfaces into which the convex body shape in exam, a cube, is divided. Therefore the total aerodynamic

torque is obtained by summation of the individual torque contributions.
A_k is the area of the k-th surface, V is the normalized velocity vector in the body frame, n_k is the normal vector of the k-th surface and $r_{s,k}$ is the vector from the CG to the area center of the k-th surface.
The coefficient of drag c_D is predicted via the Newtonian slipstream theory of rarefied gas dynamics, i.e. exactly 2.0 for a spherically shaped body, slightly higher for a cubic body; in the exam case, a value of 2.2 has been assumed.
For more realistic results, it could have been also possible to evaluate the torque considering the velocity relative to the Earth's atmosphere, v_{ECEF}, assuming a static, Earth-fixed atmosphere.
To conclude, besides the constants, all the input required are the velocity vector, the orientation of the spacecraft in time and the areas of the different exposed surfaces.
In the table 3.1 the constant parameters used are reported:

Parameter	Value	Unit
Air density	3.2586e-12	$kg\ m^3$
Aerodynamic coefficient	2.5	-
Center of aerodynamic pressure arm	0.02	m

Table 3.1: Atmospheric Drag parameters

Solar Radiation Pressure Torque
Among the contributions to consider, the one from the Sun is not negligible at all: a spacecraft in LEO receives electromagnetic radiation and the Sun is the hugest source. These radiations exert normal forces on space objects, known as solar radiation pressure: this pressure has its origin in photonic momentum exchange and causes a cyclic disturbance torque which may be modeled in a similar form as it has been done for the aerodynamic pressure torque:

$$T_{sp} = \frac{S_0}{c}(1+r) \cdot \sum_{k=1}^{n} \vec{r_{s,k}} \times (n_k^T \cdot S) \cdot S \cdot A_k \tag{3.6}$$

in which again the body has been divided into six normalized surfaces.
The solar constant S_0 is defined as the normal energy flux onto a unit area per unit time, outside of the atmosphere, at 1 AU distance to the sun; although it could be considered as a constant, it varies by approximately 3.4 % during a year, due to the eccentricity of the Earth's orbit about the Sun, from 1316 to 1428 $\frac{W}{m^2}$, with the nominal value at 1371 $\frac{W}{m^2}$.
As it is possible to see from the $(1+r)$ term, the values are strongly dependent on

the type of surface illuminated; typical values for r are between 0.4 and 0.7, with 0 representing a perfect absorbing surface and 1 a perfect reflecting one.
The other term's items represent: S the normalized sun vector in the body frame, $A_k, n_k, r_{s,k}$, as for the previous disturbance, respectively the area, the normal vector and the vector from CG to the area center of the k-th surface.
Therefore, the only external inputs to provide in that case are again the area of the surfaces illuminated, the actual orientation of the spacecraft and the Sun vector, already given normalized from the Space Environment block, properly connected from ECI frame to the Body one.
It could be interesting also to note that the solar pressure torque is typically a non-continuous source of disturbance, since the spacecraft may not be exposed to direct sunlight at all times of the orbit. For that consideration, a reasonable simplification for LEO satellites is to set the contributions to zero if the spacecraft is in umbra and consider this disturbance only otherwise.
In the table 3.2 the constant parameters used are reported:

Parameter	Value	Unit
Reflectance Factor	0.263	-
Solar radiation pressure arm	0.02	m

Table 3.2: Solar radiation parameters

Residual Magnetic Dipole Torque

The last torque taken into account is the one generated by the interaction with the magnetic field of the Earth. Any residual magnetic field inherent to the spacecraft, whether generated by a magnetized material or by an electrical current system, will interact with the geomagnetic field to produce a mechanical torque T_m. Magnetic fields may be represented by equipotential field lines of a magnetic potential field; local magnetic flux density vectors are oriented tangentially to these field lines. The strength of a magnetic field is governed by the magnetic dipole of the body generating the field.
The simplest model of the geomagnetic field is that of an ideal dipole and is the one usually used for these applications.
Expressing the net residual magnetic field in terms of residual dipole D, the disturbance torque imposed on the spacecraft's structure can be conservatively and simply written as

$$T_m = D \cdot B \tag{3.7}$$

with B representing the geomagnetic flux density.
In that case, the inputs to provide are both the estimation of the residual dipole of the spacecraft, in $[Am^2]$ and the Earth Magnetic field vector in body frame,

determined in the Space Environment block.
In the table 3.3 the constant parameter used is reported:

Parameter	Value	Unit
SC residual dipole moment (D)	0.08	Am^2

Table 3.3: Residual magnetic dipole parameters

SATELLITE STATE

The Satellite State block is the core of the simulator, as it takes as input all the information previously calculated in the other blocks, Space environment and Disturbances, to provide as an output the evolution of the satellite's state, only in terms of attitude, which includes orientation, current value of the attitude, and angular velocity, how the attitude is varying in time.
Therefore, in the application presented, only Attitude Dynamics has been considered, as the focus of the project is the Attitude Determination and Control System, without focusing on the orbit itself.
For studying the dynamics, as also seen in the other blocks explained, there could be chosen both the analytical approach, implementing "manually" the equations of orbital dynamics, and the embedded approach, using directly the block available in Simulink, which models the rotational dynamics of spacecraft using numerical integration. Again, it has been preferred to use the embedded block, as it allows to get results with higher accuracy.

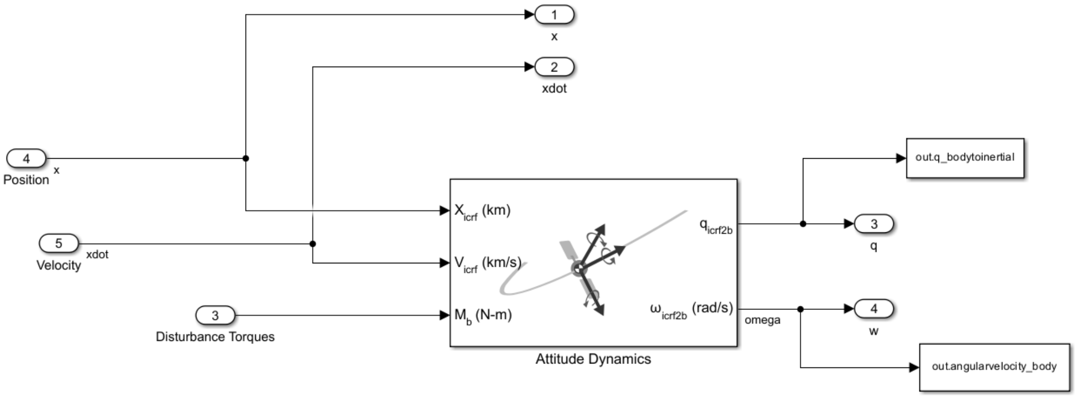


Figure 3.8: Satellite State model

As already mentioned, this block computes the attitude and angular velocity of the spacecraft introduced over time, using the provided position and velocity states. Therefore, first of all, it has been necessary to introduce the inertial vectors of the current position and velocity of the satellite, calculated for each time frame of the simulation and then also the torque disturbances in body frame have to be introduced externally. After that, inside the block, it is requested to introduce all the relevant data regarding the reference time and the satellite itself: from the mass type and value, the inertia tensor, the attitude representation and relative starting values.

The outputs of the block, according to the choices taken, are the quaternions body to inertial frame and the angular velocity vector body to inertial frame, in $\frac{rad}{s}$. These values are fundamental, for the controller algorithms, to understand whether the actual state is close or not to the desired value and from that information, to elaborate the control law to apply, in order to modify and correct them.

In the table 3.4 the satellite's data used are reported:

Parameter	Value	Unit
Mass	150	kg
Dimensions	$0.6 \times 0.6 \times 0.6$	m^3
Inertia	$\begin{bmatrix} 70.8919 & -0.1589 & -0.2727 \\ -0.1589 & 70.8590 & -0.3060 \\ -0.2727 & -0.3060 & 25.6712 \end{bmatrix}$	$kg \cdot m^2$
Initial quaternion	$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$	-
Initial angular velocity	$\begin{bmatrix} 0.01 & 0.01 & 0.01 & 0.01 \end{bmatrix}$	rad/s

Table 3.4: Satellites Data

3.2.2 Simulated components

As discussed in the introduction of the section, models to simulate actuators and sensors physically missing in the bench have to be implemented.

Simulated Sensors

Starting with the sensor ones, for the particular simulation chosen (refer to section 5.2.5), only models for fine and analog sun sensors have to be developed, as the other attitude and rate sensors needed are available on the bench and the magnetic field sensor is not required for the specific scenario simulated.

The models used for the two typologies of sun sensors are almost the same: they both take as input the sun position in the body frame, computed in the Space Environment block, but they provide different outputs. The fine sun sensors provide both the 3D measurements for sun position, in the body frame, and the intensity of the light detected; the analog sun sensors, instead, as photodiodes, provide only the intensity measurements of the sunlight.

The process used to elaborate the input position is based on the following steps:

- First of all, a conversion from body to sensor frame is needed, via matrix multiplication between the sun vector and proper rotation matrix, in order to manage the measurement in the device relative system of reference.
- Then, with measurements in the sensor frame, it is possible, providing in input also the field of view of the sensors used, to check if the sunbeam detected is inside the FoV of the sensor.
- Once checked the measurement, noise and resolution factor need to be added.
- At the end, corrected measurements are converted again in the body frame, before being sent to the hardware.

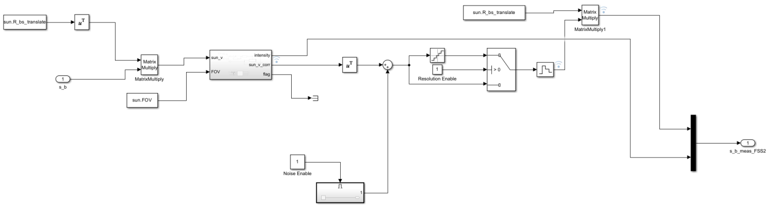


Figure 3.9: Fine Sun Sensor Model

Parameter	Value	Unit
FOV	120	deg
R_{bsX}	$\begin{bmatrix}0 & 0 & -1\\0 & 1 & 0\\1 & 0 & 0\end{bmatrix}$	-
R_{bsY}	$\begin{bmatrix}1 & 0 & 0\\0 & 0 & -1\\0 & 1 & 0\end{bmatrix}$	-
R_{bsZ}	$\begin{bmatrix}0 & 0 & -1\\0 & 1 & 0\\1 & 0 & 0\end{bmatrix}$	-
Bias	[0.05 0.05 0.05]	deg
Resolution	0	-
Noise Standard Deviation	$\frac{1}{\sqrt{3}} \cdot [0.1\ 0.1\ 0.1]$	deg

Table 3.5: FSS parameters

For the analog sun sensor, the difference is that only the intensity is computed and processed, from the starting sun position.

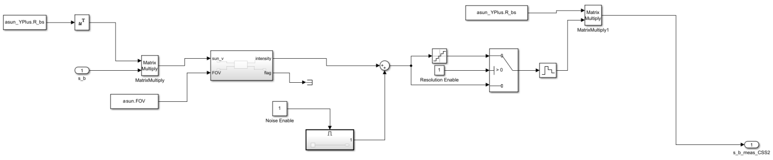


Figure 3.10: Coarse Sun Sensor Model

Parameter	Value	Unit
FOV	180	deg
R_{bsX}	$\begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix}$	-
R_{bsY}	$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}$	-
R_{bsZ}	$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$	-
Bias	[0.05 0.05 0.05]	μA
Resolution	$3.7876e^{-7}$	-
Noise Standard Deviation	$\frac{1}{\sqrt{3}} \cdot [0.10.10.1]$	μA

Table 3.6: FSS parameters

Simulated Actuators

For the same reason already explained for sensors, even models to simulate attitude controllers have been implemented and in particular the Reaction Control System (RCS) is needed for the scenario chosen 5.2.5.

Two main steps are necessary to convert the commanded forces, received from the software, into control torques to be applied to the satellite to control its attitude over time:

- a block, shown in figure 3.11 which implements a Pulsed Width Pulsed Frequency (PWPF) Modulator, to convert the floating point values output by the controller to the pulses that the thruster can realize.

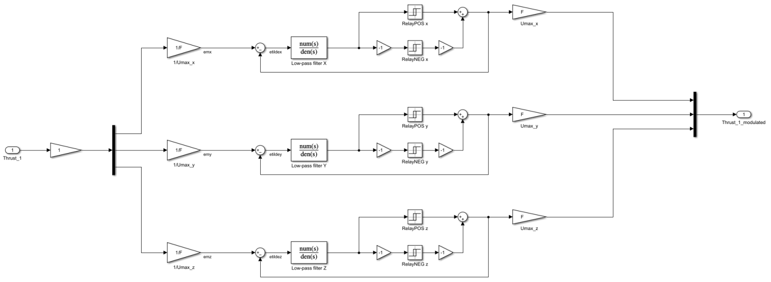


Figure 3.11: PWPF Modulator model

- the actual thruster model, referred to the discussion presented in [32], which converts the input forces provided into the equivalent torques generated on the satellite, using an Allocation Matrix, properly defined according to the position of the thrusters.

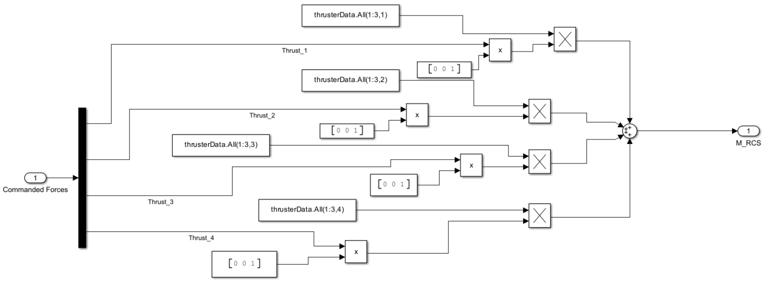


Figure 3.12: RCS Model

The distribution matrix used for the torque allocation is the following:

$$DM = \begin{bmatrix} -0.0885 & 0.0068 & 0.1062 \\ 0.0085 & 0.1062 & 0.0068 \\ -0.0085 & -0.0068 & -0.1062 \\ 0.0085 & -0.1062 & -0.0068 \end{bmatrix} \tag{3.8}$$

3.2.3 Communication

The last remaining blocks to be discussed are the data transmission ones, which allow to send and receive, each second of the simulation, the requested data to the hardware, from the Simulator, and vice-versa.

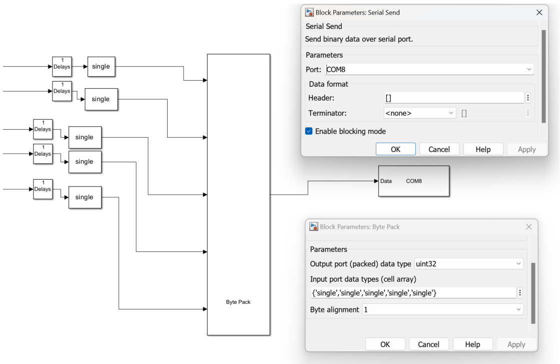
Serial Send

Starting from the sending process, the need is to provide data from the simulated sensor models to the OBC hardware, where they will be processed in the ADCS software.

For this purpose, the outputs of the Sensor models block, in float format, have to first be properly delayed, then converted into a compatible data format, packed and sent:

- Tapped Delay block is used to delay each signal a certain amount of sample periods and provide as output the delayed version of the signal. The most important of this block to be set is the Sample Time, which allows to choose the entity of the delay and the number of delays.
- For the conversion action the Data Type Conversion block is efficient and it requires to set only the output data type, which is *single* for the particular case analyzed.
- Once delayed and converted, data need to be stored using Byte Pack block, which allows to pack easily input data to a single output vector of the required type. Therefore, the correct input and output port data types have to be configured.

The output data packet can then be provided to the Serial Send block, in which, for the specific application, only the correct communication port needs to be specified, as no headers or terminators are used.



Block Parameters: Serial Send

Serial Send

Send binary data over serial port.

Parameters

Port: COM8

Data format

Header: []

Terminator: <none> []

☑ Enable blocking mode

Block Parameters: Byte Pack

Parameters

Output port (packed) data type uint32

Input port data types (cell array)

{'single','single','single','single','single'}

Byte alignment 1

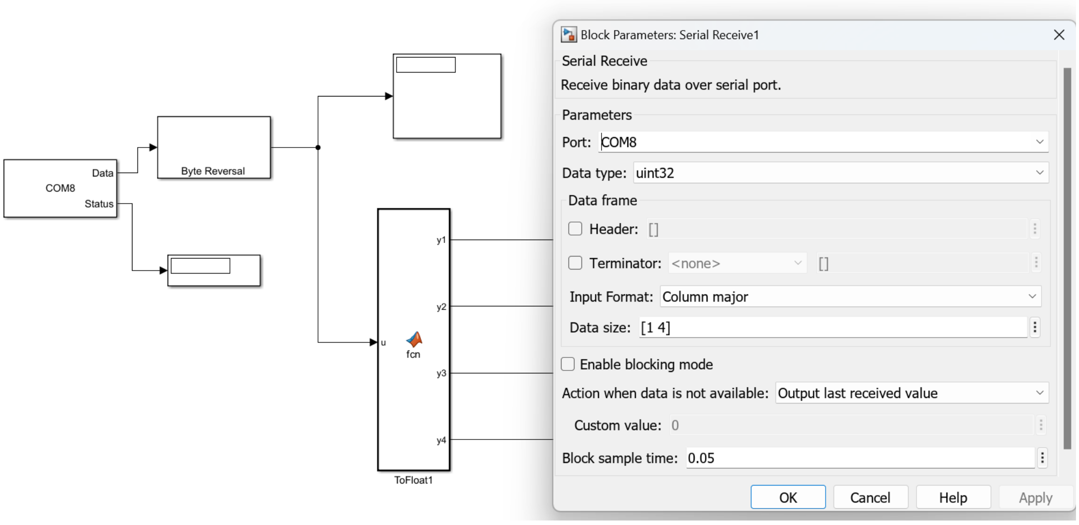
Figure 3.13: Send Data

Serial Receive

To receive the controlled forces, computed by the ADCS software, embedded in the OBC hardware, which are required to properly control the satellite attitude, the Serial Receive block is used and the process to follow is the opposite with respect to the sending one.
To use correctly the Serial Receive block, the following parameters need to be configured:

- communication port (in the application presented, the same port is used to send and receive data);
- the data type of the packet received (uint32);
- the data size of the packet received, specifying rows and columns of the arrays ([1 4]);
- Block sample time, to set the frequency of the request of data from the port selected (0.05 s, according to the simulation sample time).

It could happen that data retrieved with this block have flipped bytes, therefore Byte Reversal block may be needed in order to receive the expected data values. The format of the packet received is not the one expected in the Actuator models as input, so a proper function to convert the data type is required: for this purpose, a Matlab function has been implemented to take as inputs the $uint32$ data and convert them into 'single' format, to be used then as floating values.



Block Parameters: Serial Receive1

Serial Receive

Receive binary data over serial port.

Parameters

Port: COM8

Data type: uint32

Data frame

□ Header: []

□ Terminator: <none> []

Input Format: Column major

Data size: [1 4]

□ Enable blocking mode

Action when data is not available: Output last received value

Custom value: 0

Block sample time: 0.05

Figure 3.14: Data Receive

Chapter 4

Modular Avionics Test Bench Design and Setup

The design of the test bench used for the work, as presented in [11], has been developed for the relative project, with the main aim to map all the interconnectivity technologies offered by European suppliers, design the Command and Data Handling (CDHS) hardware and software to match those technologies and test the design with the supplier's components, providing a final product that is 80 % compatible with the EU CubeSat subsystem suppliers.
In this chapter, it is going to be discussed briefly how the design of the bench has been built and how the components have been selected and integrated.

4.1 Requirements and design

Each FlatSat developed in the project has been designed to host the subsystems required for an operational satellite, which essentially are:

- attitude determination;
- attitude control;
- Telecommunication, with optional antenna;
- Power;
- Rate Determination;
- Optional Payload.

For the project presented, only the subsystems related to the attitude and rate determination and control have been kept to test the modular ADC system, with the addition of the Electrical Power System for creating a more realistic and complete setup. Furthermore, it is important to know that each FlatSat uses a PCB to connect all its subsystems together: this board is designed to be as modular as possible but there is a customization for each chain produced. The general design is well represented in 1.10: each bench has 9 PC104 form factor slots for subsystems, with PC104 connectors; it can be powered on externally, either with a power line which goes to the EPS on its input power lines, or with 5 V and 3.3 V power lines; it also has a JTAG connector to communicate directly with the MA61C board from a computer. JTAG, or Joint Test Action Group, is a standardized interface and protocol used in the electronics industry for testing and programming integrated circuits (ICs) on printed circuit boards (PCBs) and other electronic devices. For the case of interest, this kind of connector has been very worthwhile during the testing and debugging phases.

4.2 Selection of hardware components

The choice of the components to be connected to the bench has been made taking into consideration the ones present in the previous ADCS project: as the current work represents an extension of the SmallSat version already designed and tested, the components involved have to be as close as possible to the starting ones, in terms of performance.
According to that purpose, the main criteria of selection has been the performance that each component can ensure:

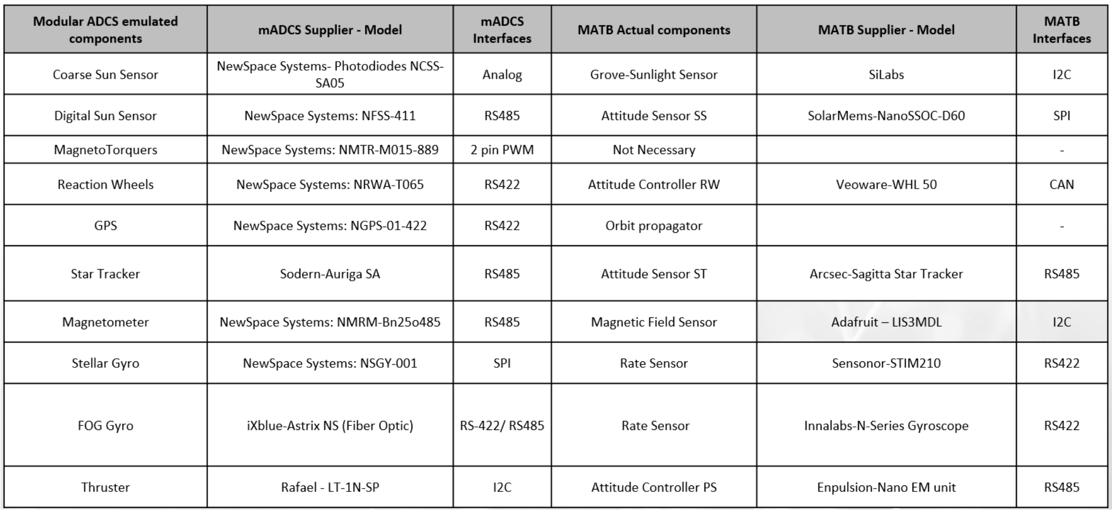
- For **Sensors**:
  - Sun sensor: the focus has been on the field of view, sensor accuracy and precision;
  - Star Tracker: the most important parameters considered are the cross-boresight and around-boresight, besides the update rate limitations;
  - Magnetometer: sensitivity and range of measurement have guided the choice;
  - Rate Sensor: the reference parameters have been rate estimation accuracy, bias knowledge and stability, and measurement range.
- For **Actuators**:
  - Reaction wheel: the selection has been made with respect to torque capability and momentum capacity;

 – Thruster: nominal thrust and specific impulse have led the choice criteria.

For two components in particular, involved in the starting setup, any correspondent device has been selected for the integration in the test bench:

- GPS among sensors, as it has been chosen to replace the relative measurements with the ones provided by the orbital propagator simulated.
- MagneTorquers among actuators, as it has been considered sufficient to provide the voltage outputs, computed by the software, without having proper hardware to manage.

In figure 4.1 the list of all the components selected is presented, coupled with the reference ones from the SmallSat project.



Modular ADCS emulated components	mADCS Supplier - Model	mADCS Interfaces	MATB Actual components	MATB Supplier - Model	MATB Interfaces
Coarse Sun Sensor	NewSpace Systems- Photodiodes NCSS-SA05	Analog	Grove-Sunlight Sensor	SiLabs	I2C
Digital Sun Sensor	NewSpace Systems: NFSS-411	RS485	Attitude Sensor SS	SolarMems-NanoSSOC-D60	SPI
MagnetoTorquers	NewSpace Systems: NMTR-M015-889	2 pin PWM	Not Necessary		-
Reaction Wheels	NewSpace Systems: NRWA-T065	RS422	Attitude Controller RW	Veoware-WHL 50	CAN
GPS	NewSpace Systems: NGPS-01-422	RS422	Orbit propagator		-
Star Tracker	Sodern-Auriga SA	RS485	Attitude Sensor ST	Arcsec-Sagitta Star Tracker	RS485
Magnetometer	NewSpace Systems: NMRM-Bn25o485	RS485	Magnetic Field Sensor	Adafruit – LIS3MDL	I2C
Stellar Gyro	NewSpace Systems: NSGY-001	SPI	Rate Sensor	Sensonor-STIM210	RS422
FOG Gyro	iXblue-Astrix NS (Fiber Optic)	RS-422/ RS485	Rate Sensor	Innalabs-N-Series Gyroscope	RS422
Thruster	Rafael - LT-1N-SP	I2C	Attitude Controller PS	Enpulsion-Nano EM unit	RS485

Figure 4.1: Components comparison between the projects

In the following sections, a brief description of each chosen subsystem is provided, to better understand the devices involved in the simulation.

Coarse Sun Sensor - Grove Sunlight Sensor Si1551

Figure 4.2: Coarse Sun Sensor [33]

The Si1551, from Grove data sheet [33], is an ambient light sensor, proximity, and gesture detector with I2C digital interface and programmable event interrupt output. The sensor could offer adequate performance under a wide dynamic range and a variety of light sources, including direct sunlight and is able to support multi-axis proximity motion detection.
The best resolution possible, as an ambient light sensor, is less than 100 mlx, allowing even operation under dark glass, and it ensures a dynamic range possible up to 128 klx, across two ADC range settings.
The supply voltage could vary between 1.62 and 3.6 V.

Fine Sun Sensor - NanoSSOC-D60

Figure 4.3: Fine Sun Sensor

Nano Sun Sensor, from SOLARMEMS Technology datasheet [34], is a two-axis low cost sun sensor for high accurate sun-tracking and attitude determination. The device measures the incident angle of a sun ray in both azimuth and elevation, providing an accurate angle reading.
It could communicate via UART, I2C or SPI but SPI interface version has been chosen for the application.
The power supply voltage to be provided is 3.3V and the main performances are the following:

- wide Field of View: ±60°;

- High accuracy in FoV: 0.5°;
- Precision: 0.1°.

Star Tracker - Sagitta Star Tracker

Figure 4.4: Star Tracker [35]

The Sagitta Star Tracker, from ArcSec datasheet [35], is a high accuracy, high robustness and easy interfacing device that offers arc second range pointing accuracy for CubeSats and large satellites.
It communicates via an RS485 interface and it needs a supply voltage of 5 V.
It provides the following main performances:

- cross-boresight: $2''(1\sigma)$;
- around-boresight: $10''(1\sigma)$;
- Tracks up to 32 stars;
- Update rate of 5 Hz.

Rate Sensor - STIM210

Figure 4.5: Rate Sensor - SAFRAN [36]

STIM210, from SAFRAN datasheet [36], is a small, tactical grade, affordable, robust and reliable, ultra-high performance MEMS gyro module with up to 3 axes. For the application presented, a 3-axis device is used.
It communicates via an RS422 interface and needs a supply voltage of 5 V. The main performances provided are:

- Range of measurement: $\pm\frac{400^\circ}{s}$;
- Bias instability: $\frac{0.3^\circ}{h}$;
- Sampling Frequency: 2000 samples per second.

Rate Sensor - N-Series Gyroscope

Figure 4.6: Rate Sensor - InnaLabs [37]

N-series Gyroscope, from InnaLabs datasheet [37], is an industrial and tactical grade device which delivers less than $\frac{10^\circ}{hr}$ over the full temperature range. For the application presented, a 2 axes device is used.
It communicates via an RS422 interface and needs a supply voltage of 12 V.

The main performances provided are:

- In-run Bias Stability (Room temperature, 1σ): less than 0.22 deg/hr;
- Bias stability (full temperature range, 1σ): less than 10 deg/hr;
- Angular Random Walk: 0.01 $deg/\sqrt{hr}$.

Reaction Wheel - WHL50

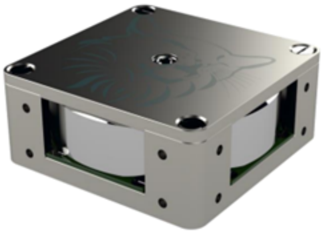


Figure 4.7: Reaction Wheel [38]

WHL-50, from Veoware datasheet [38], is a fully integrated reaction wheel unit for high-performance satellite attitude control for Micro and Nano-satellite missions. In particular, it is an integrated 3-phase out runner Permanent magnet synchronous motor (PMSM).
Each wheel has a CAN bus interface (or RS422) with CSP making them accessible to the satellite communication bus.
The power supply voltage required is 12-16 V unregulated DC.
The main performances are:

- max nominal RPM: 10000;
- Momentum of 42 mNms at maximum rate;
- Max Torque higher than 5 mNm.

Thruster - IFM Nano

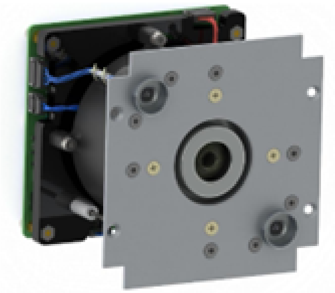


Figure 4.8: Reaction Control System [39]

IFM NANO thruster, by Enpulsion datasheet [39], belongs to the electrical propulsion system class, for micro and nanosatellite and provides a wide range of thrust, excellent throttability, and a high specific impulse, allowing to significantly increase the mission range of such satellites in low orbits.
For the application analyzed, the configuration chosen is based on an RS485 communication protocol and 12 V of supply voltage.
The core specifications are:

- Thrust level between 1 μN to 1 mN;
- Specific Impulse up to 5000 s;
- Capacity of 10 kNs.

Magnetometer - LIS3MDL

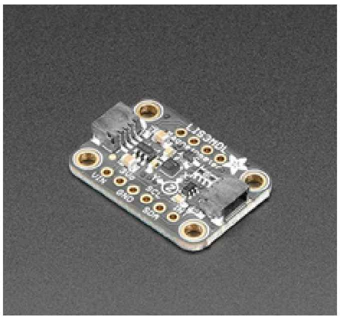


Figure 4.9: Magnetometer [40]

The LIS3MDL, from Adafruit datasheet [40], is an ultra-low-power high-performance three-axis magnetic sensor. It is characterized by a wide supply voltage, between

1.9 V and 3.6 V and by user-selectable magnetic full scales, between ±4 and ±16 gauss (the equivalent of hundreds of $\mu Tesla$).
The sensor includes an I2C serial bus interface that supports standard and fast modes.

EPS - PicoEPS

Figure 4.10: Electrical Power Supply [41]

PicoEPS, from DHV datasheet [41], is designed to be integrated into different CubeSats platforms and represents the interface that manages the power of the solar panel inputs and the battery charge. This module is also responsible for the generation of 3.3V,5V,12V and VBAT power buses including the switched power lines.
The unit chosen unifies a Power Module and a Battery Module in one single board, optimizing mass and volume for large payloads integrated in the platform.
It operates via an I2C interface and in the particular case of exam, it is powered via an external power supply with 12 V as voltage.

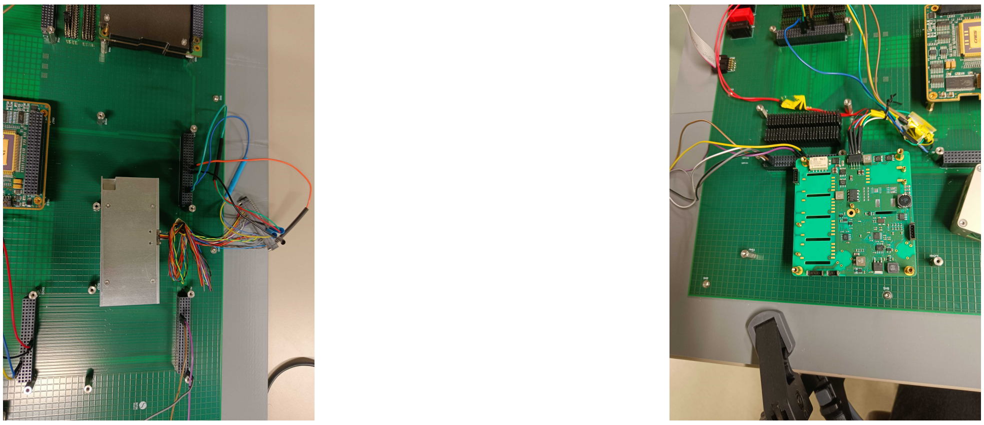
4.3 Integration and Assembly

Once the components have been properly selected, the next step is to integrate them into the test bench presented.
For each device, it is possible to use one of the available spots on the bench, which has a PC104 form factor, to set up a proper connection with the On Board Computer, located in the middle spot of the test bench.
Some of the components are designed with PC104 form factor, therefore they do not require a dedicated harness but they can be simply plugged in using the connector integrated on the bench (figure 4.12).
For the other components, which are not designed with that form factor, a dedicated harness has to be prepared, according to the interface used, to connect pins on the device itself with specific pins on the connector available (figure 4.11).

To understand how to build the connection, two reference documents have to be followed:

- *Interface Control Document* for each device to understand the location of the pinouts to be connected;
- *MA61C Cubesat Hardware Design document*, to identify which pins of the board, and consequently on each PC104 connector of the bench, are involved for the particular interface used for the communication.

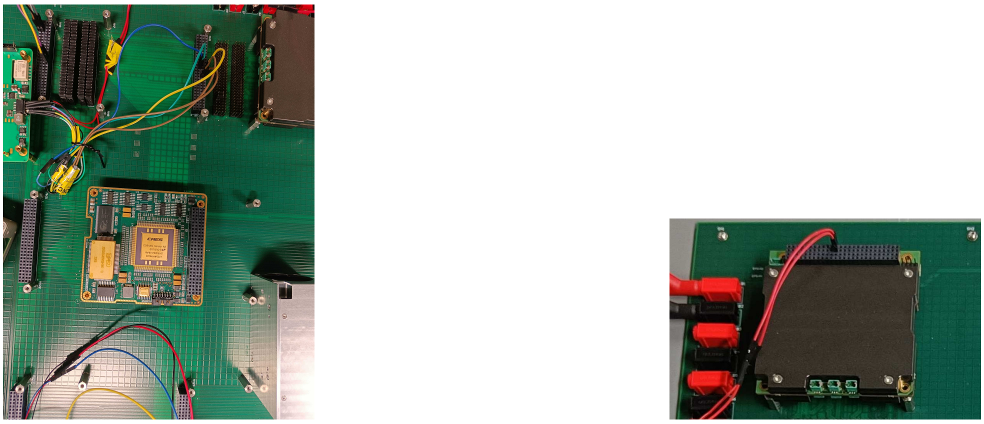
Following that procedure, the connections with all the devices selected could be installed and the test bench setup can be fully integrated.
In the pictures below, some examples of connections are displayed.



(a) Sagitta Star Tracker connection

(b) IFM Nano connection

Figure 4.11: Example of dedicated Connections



(a) MA61C PC104 Form Factor

(b) PicoEPS connection

Figure 4.12: Example of PC104 form factor Connections

Besides the connections between the bench and devices, two other external connections have to be provided and are present in the schematics:

- connection with an external power supply, in figure 4.13, which delivers the correct amount of power requested for the full operation of the bench. This connection can be made in two different ways:
  - by connecting the power supply directly with the bench, via its dedicated ports;
  - by connecting the power supply to the EPS, via its dedicated power lines, which then delivers by itself the requested power to all the other devices.

  In the picture, the first method is presented.

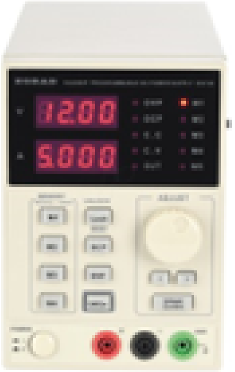


Figure 4.13: Power Supply [42]

- the second connection displayed is the one with the external PC, via a proper JTAG connector (in figure 4.14), which enables a direct communication channel with the laptop, for debugging and controlling operations via graphical user interfaces or terminals.

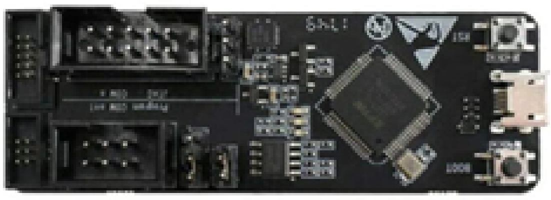


Figure 4.14: Jtag connector

Besides these two external connections, a third one is present and essential, even if not shown explicitly in the diagram: it is the one that allows the exchange of data between the Simulink Simulator model, in the computer, and the OBC hardware, based on a USB to UART TTL interface, displayed in figure 4.15. This device has one terminal (USB) connected to the driver manager unit of the computer and the other terminal connected directly to the bench, via dedicated pinouts of UART TTL interface on one of the PC104 form factor connectors.

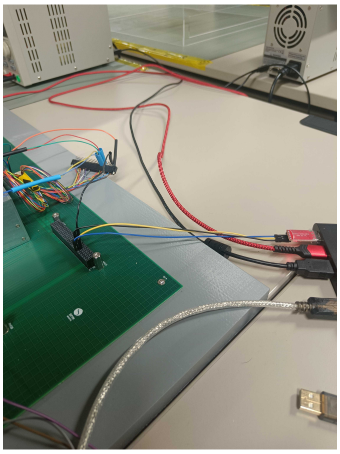
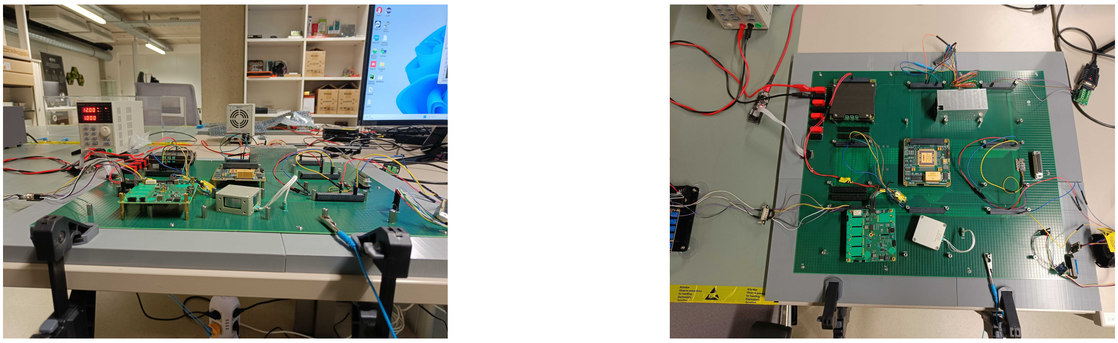


Figure 4.15: UART connection between Simulator model and Hardware

The physical hardware setup of the test bench is finally reported in 4.16.



(a) Test bench setup front view

(b) Test bench setup top view

Figure 4.16: Test bench Setup

Chapter 5

Testing Process Description

In this chapter, the focus is on the testing campaign preparation, the description of all the tools involved, and the explanation of the reasons that have carried to their choice.
A section will be dedicated also to the description of the assumptions taken into consideration for making the testing phase easier to execute and the related limitations that have been introduced with this approach.
The final discussion proposed is to explain how the final and complete test on the setup prepared will be executed, before presenting the results in the dedicated section of the next chapter.

5.1 Test Scenarios

First of all, it is important to define which are mission scenarios taken into consideration for the study and which is the one chosen for the testing phase performed.
The set of starting scenarios has been inherited, as done for the components, from the one used in the reference project. For that work, there have been considered three different scenarios:

- Telecommunication scenario;
- Earth Observation scenario;
- Space Tugs scenario.

5.1.1 TeleCommunication Scenario

In the TC scenario, the main target is to point the spacecraft, which has a relatively higher altitude compared to the LEO (around 1200 km), and the mission payload,

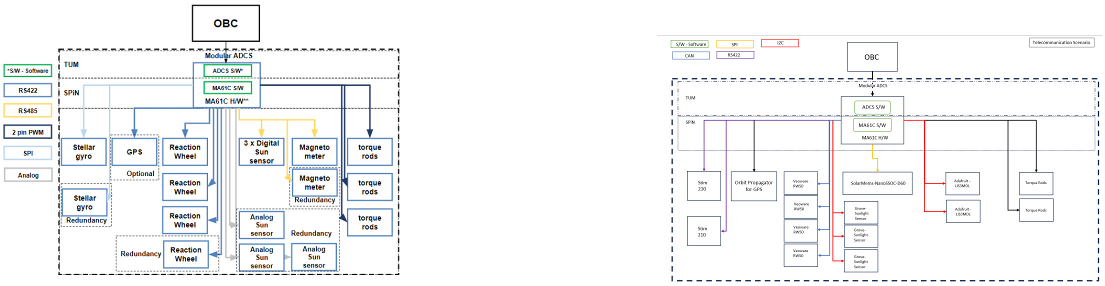
which is the antenna in this application, constantly points at the Earth, similar to the missions in communication satellites.
The main sensors, whose telemetry is required for the attitude determination and control system are:

- Stellar Gyroscope;
- Magnetometer;
- Sun Sensors.

The main actuators needed are:

- Reaction Wheels;
- Magnetorquers.

The required pointing performance for this mission is less than 1 degree.
The mass of the satellite involved is expected to be 150 kg, with roughly 1 $m \times 1$ $m \times 1.3$ m in size (ONEWEB 0102 satellite is taken as a model).
In figure 5.3 there is displayed the relationship between ADCS software, OBC software and attitude determination and control sensors and actuators.



(a) TC scenario original version [9]

(b) TC scenario adapted

Figure 5.1: TC scenario Block Diagram

5.1.2 Earth Observation Scenario

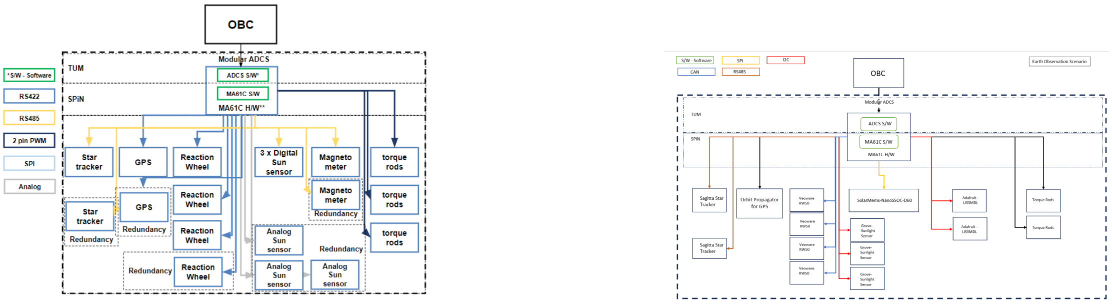
In the Earth Observation scenario, the main task is to perform imaging in a certain region of the World and download the imaging data to the ground station. Since the target is much more sensitive, the pointing performance for this application is set to 0.05 degrees: this requirement affects the choice of the components involved. The main sensors, whose telemetry is required for the attitude determination and control system are:

- Star Tracker;
- Magnetometer;
- Sun Sensors;
- GPS.

The main actuators needed are:

- Reaction Wheels;
- Magnetorquers.

The mass of the satellite involved is expected to be 60 kg, with roughly 40 cm × 46 cm × 84 cm in size (BlackSky-1 pathfinder satellite is taken as a model).
In figure 5.3 there is displayed the relationship between ADCS software, OBC software and attitude determination and control sensors and actuators.



(a) EO scenario original version [9]

(b) EO scenario adapted

Figure 5.2: EO scenario Block Diagram

5.1.3 Space Tug Scenario

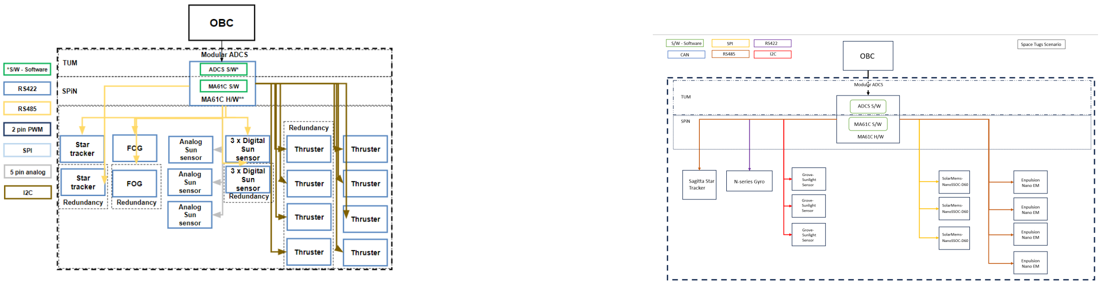
The goal of this reference mission is to transfer the target spacecraft of tug mission to another desired altitude or orbit. Since this mission includes both in plane and out of plane transfers, a reaction control system with thrusters is necessary. The main sensors, whose telemetry is required for the attitude determination and control system are:

- Star Tracker;
- Sun Sensors;
- Fiber-Optic Gyroscope.

The main actuators needed are:

- Thrusters, at least 4 to ensure 3-axis body control.

The pointing performance for this mission is not as strict as for the others, as there is no imaging activities: therefore 0.1 degrees, for nadir pointing, and 1/2 degrees, for deployment, are taken as reference values.
The mass of the satellite involved is expected to be 150 kg, with roughly 60 $cm \times$ 60 $cm \times$ 60 cm in size (COMET spacecraft is taken as a model).
In figure 5.3 there is displayed the relationship between ADCS software, OBC software, and attitude determination and control sensors and actuators.

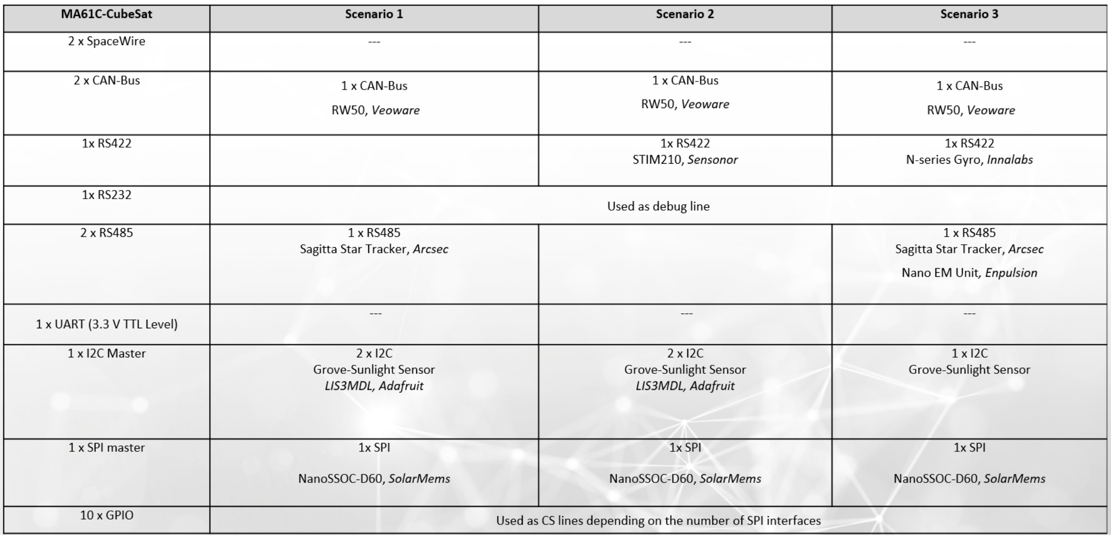


(a) ST scenario original version [9]

(b) ST scenario adapted

Figure 5.3: ST scenario Block Diagram

A complete overview of the three scenarios presented, with the components selected for the current project, is displayed in figure 5.4, with a particular focus on the different interfaces used.



MA61C-CubeSat	**Scenario 1**	**Scenario 2**	**Scenario 3**
2 x SpaceWire	---	---	---
2 x CAN-Bus	1 x CAN-Bus RW50, *Veoware*	1 x CAN-Bus RW50, *Veoware*	1 x CAN-Bus RW50, *Veoware*
1x RS422		1x RS422 STIM210, *Sensonor*	1x RS422 N-series Gyro, *Innalabs*
1x RS232	Used as debug line		
2 x RS485	1 x RS485 Sagitta Star Tracker, *Arcsec*		1 x RS485 Sagitta Star Tracker, *Arcsec* Nano EM Unit, *Enpulsion*
1 x UART (3.3 V TTL Level)	---	---	---
1 x I2C Master	2 x I2C Grove-Sunlight Sensor *LIS3MDL, Adafruit*	2 x I2C Grove-Sunlight Sensor *LIS3MDL, Adafruit*	1 x I2C Grove-Sunlight Sensor
1 x SPI master	1x SPI NanoSSOC-D60, *SolarMems*	1x SPI NanoSSOC-D60, *SolarMems*	1x SPI NanoSSOC-D60, *SolarMems*
10 x GPIO	Used as CS lines depending on the number of SPI interfaces		

Figure 5.4: Interfaces Scheme over the three scenarios

5.2 Simulation methodologies

This section is dedicated to presenting the methods chosen to recreate the correct conditions for performing the tests, in particular for the space environment simulation.
This step is really important as some real space components, such as gyroscope, sun sensor, star tracker and magnetometer, need external inputs, i.e. rotation, light, sun, magnetic field, to provide valid data that could be used in the ADCS software.
For that purpose, it has been necessary to find an efficient way to provide these external sources, taking into consideration budget and time constraints.

5.2.1 Micro-gravity Simulation - Robotic Arm

As well and clearly discussed in [15], there are several methods to recreate micro-gravity, such as

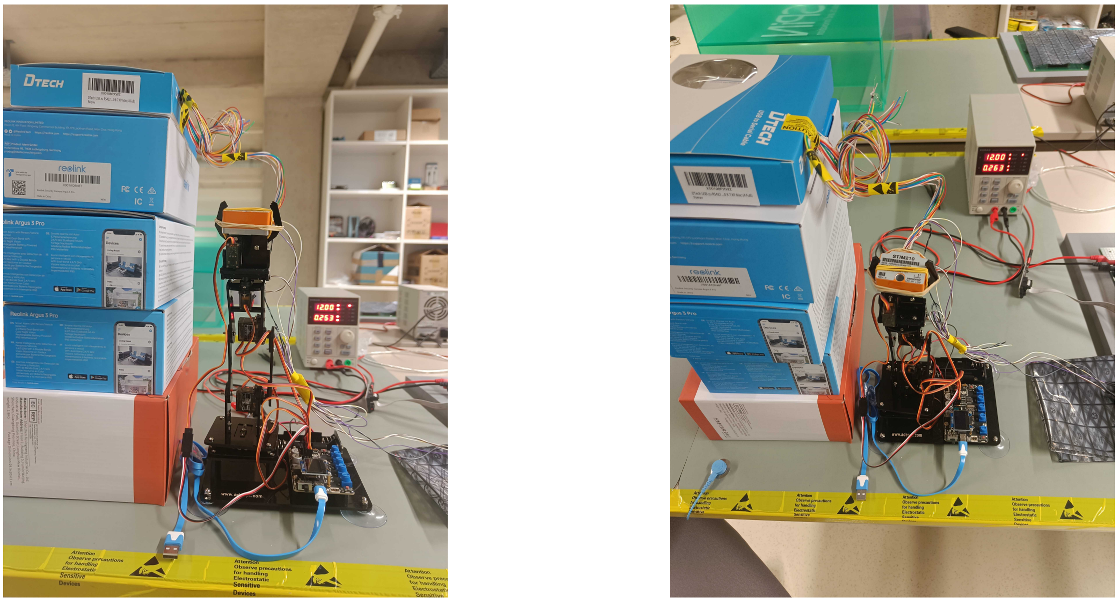
- cable suspension
- robotic arms
- air bearings

One of the most used and efficient solutions is based on the use of an air-bearing type facility which works by exploiting a pressurized airflow to generate a film

between two corresponding surfaces, acting as a lubricant between them, therefore reducing friction.
In the case object of study, there has been no availability in space, time and cost to develop a similar solution, so an easier, cheaper and less efficient one has been chosen: a robotic arm.
The setup prepared for the microgravity simulation is really simple: the gyroscope is held and fixed on the top of the arm, in a vertical configuration, as displayed in figure 5.5. Since the arm it is not designed to hold objects as heavy as the rate sensor, this configuration has been evaluated as the most stable one to use during the simulation.
From this position, the rotations needed for the tests are performed, respecting certain constraints, discussed in section 5.3, again due to stability issues. Any movement of the arm can be easily programmed and executed by following these steps:

- First of all, it is necessary to connect the Arduino board integrated at the basis of the arm, via a USB cable, to a computer. Once connected, open Arduino IDE and start building the code to rotate the different degrees of freedom available, according to the desired movement. When the code is ready, compile and upload it to the board connected in order to be stored. Then remove the USB connection.

- To see the actual movement coded, it is enough to provide power, via an external power supply or via batteries, to the arm, by using the power pinouts available on the support base.



(a) Robotic arm Setup - Front view

(b) Robotic arm Setup - Top view

Figure 5.5: Micro-gravity Simulation

5.2.2 Sun Simulation - Torch Light

To replicate, even if only partially, the solar radiation characterizing the satellite's orbit, an adequate and specific source of light is necessary.

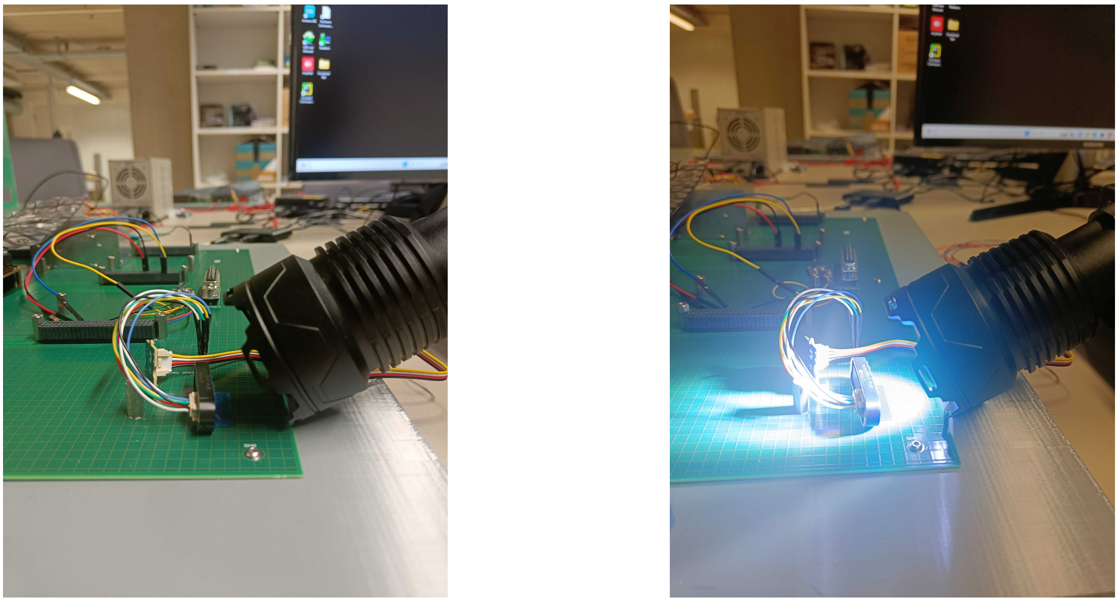
In some applications, as the one presented in [15], proper Studio Light, i.e. COST LED Studio Light, with specific features, is used as the light source. Following that example, in the case under exam, a particular torch light has been procured, with intensity capability up to the average Sun intensity, corresponding to about 120000 lumens.

As for the choice of the microgravity simulation device, even in that case the solution proposed is far easier and less precise than the one presented in the reference work but it has been led by the same constraints mentioned above.

To simulate as loyally as possible the Sunlight, besides intensity, the following aspects should be taken into consideration:

- spectral matching;
- homogeneity of the space;
- steadiness over time;
- collimation of the light beam over a broad area.

Since the torch light selected uses an LED source, some features are correctly matched, such as the flicker-free emission, high efficiency and good match in the visible part of the solar radiation spectrum; nonetheless, the near absence of emission in the IR and UV bands represents a limitation related to that choice. However, this limitation could be considered negligible as most of the COTS sun sensors are built on CMOS,CCD, PSD whose response is maximum in the visible and falls rapidly in the IR and UV wavelengths. Instead, photocells can be more affected by the absence of these band ranges.
The configuration chosen, in figure 5.6, aims to reduce, in the absence of proper collimation devices, as much as allowed the distance between the source of light and both the sun sensors, in order to receive and detect the maximum amount of intensity possible. On the other hand, this condition, for long exposure time, can induce also overheating on the surfaces of the sensors exposed: for this reason, a temperature sensor, in figure 5.7, has to be properly integrated close to the light beam to constantly monitor the air temperature, ensuring that it remains below the maximum values allowed for the correct operating conditions of the devices.



(a) Torch Light Setup - OFF

(b) Torch Light Setup - ON

Figure 5.6: Sun Simulation

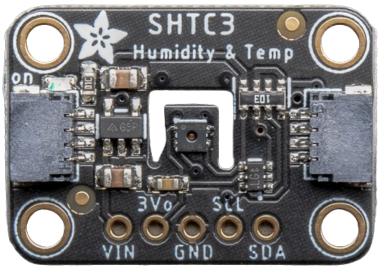


Figure 5.7: Temperature Sensor

5.2.3 Star Simulation - Simulator Mode

The simulation of stars' environment is the most complex one to take into consideration for its high level of complexity and high cost required, both for hardware and software involved.
Furthermore, in the application presented, one more and bigger limitation is represented by the Star Tracker available: the version of the device purchased is not equipped with optics, as it has been procured for ground testing activities, therefore, any external inputs could be provided.
The only alternative solution possible for using it in the simulation is by exploiting the *Simulator Mode*, available and embedded inside the firmware of the device. By using this mode, for testing purposes, the star tracker quaternion solution can be overwritten by a simulated quaternion: a quaternion simulator is properly turned on, generating a quaternion which overwrites the tracking solution computed; the simulated quaternion is generated by rotating a previously simulated quaternion each time interval by a given rotation quaternion. For that purpose the following parameters have to be set at the beginning of the simulation, to enable the Simulator mode:

- time interval between the rotation updates of the simulated quaternion;
- four elements (scalar and vectorial part) of the initial simulated quaternion which will be rotated each time interval;
- four elements (scalar and vectorial part) of the rotation quaternion by which the initial quaternion is rotated each time interval.

In this way, it has been possible to retrieve from the device, every second of the simulation, different values of quaternion, evolving in time, and simulating the motion of the satellite in space.

5.2.4 Earth Magnetic Field Simulation - Helmholtz Cage

The last environment to be simulated is the Earth Magnetic field, for magnetometers measurements and magnetorquers control actions. For this aim, as reported in [15], generally the most used solution is the Helmholtz Cage: this tool is based on a circular coil of radius R and N windings, with a current i running through it, at a distance x from the center, which generates a magnetic field B determined in the following way:

$$B = \frac{\mu_0 \; i \; N \; R^2}{2 \; (x^2 \; + \; R^2)^{\frac{3}{2}}} \tag{5.1}$$

This structure represents a way to create a uniform and controllable magnetic field in a certain volume.
The inspiring project in [43] has been taken as a reference for the design of that tool: the Helmholtz Cage designed, in figure 5.8 is a small 1-axis cage 3D printed, with the coils made out of two 3D printed parts that are screwed and possibly glued, together to reduce overhangs.

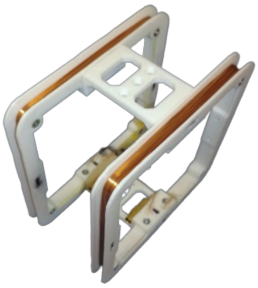


Figure 5.8: Single-axis Helmholtz Cage Reference Project

To simulate and control fully the Magnetic field needed, the complete setup to be prepared is composed of the following components:

- 3 orthogonal single-axis cages, represented in the figure, which corresponds to the x, y and z axes of the reference system, respectively;
- a dedicated magnetometer to control the magnetic field generated;
- a programmable power supply to provide energy to the cage;
- a Coil Drive unit to control properly the waveforms and current provided to the coils.

According to the setup proposed in [43] the maximum field reached with the cage should be above 2.1 Gauss with a drive current of 450 mA, which is far enough for

the simulation needed.
Although the solution proposed would perfectly fit the purpose of the work, the time required for the structures 3D-printing, the relative assembly and the manufacturing process of the coil drive units needed has not allowed to realize it early enough to be implemented and used in the current project.
This missing tool has affected also the scenario that could be chosen for the testing, presented in the next section.

5.2.5 Scenario Choice

Since all the components needed for the simulation of the three scenarios are available, the main factor that has limited the choice to only one possibility has been the availability of the simulation devices. As a consequence, the absence of a device to simulate the Earth's magnetic field has excluded the use of magnetometers in the simulation, therefore both the first, TeleCommunication, and the second, Earth Observation, scenarios have to be left out and the choice fell on the Space Tug scenario.
According to the scenario selected, the updated diagram with the relationship between ADCS software, OBC software and attitude determination and control sensors and actuators physically integrated in the bench is presented in figure 5.9. The missing components, with respect to the complete diagram shown in figure 5.3b, which are not present in the physical setup displayed, are properly simulated via the Simulink model discussed in chapter 3.2.
These are in particular:

- two analog sun sensors;
- two digital sun sensors;
- three thrusters to complete the reaction control system.

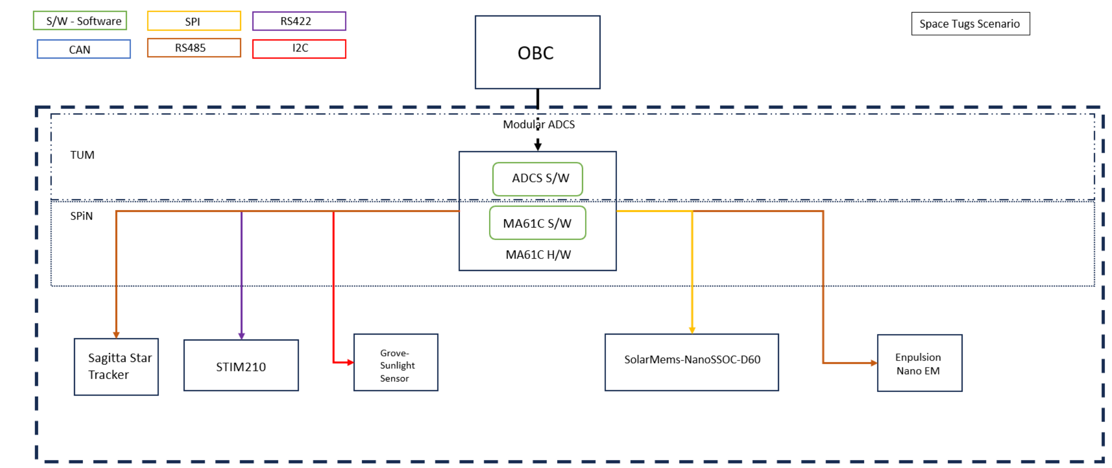


Figure 5.9: Software-Hardware interface of the chosen scenario

5.3 Assumptions and Limitations

Once having described the final scenario and setup chosen for the simulation, it is important to provide an overview of the assumptions and limitations introduced with these choices.
The most relevant are listed below:

- The inputs of the simulation, provided by the Simulator explained in chapter 3.2, are not exactly the same as the ones introduced in the previous version of the project. This starting discrepancy could affect the accuracy of the comparison of the results. For that reason, an offset of plus/minus 5 % with respect to the nominal input values is considered acceptable and allows to still have good outcomes;

- The strategies chosen to simulate the Sun Light, microgravity and star pattern for the satellite have some limitations to be taken into account:

  - for the **Sunlight simulation**, using the Torch light with 120000 Lumen intensity capability above mentioned, represents a good way to provide a huge light intensity but could be not fully reliable as in some of the cases tested the amount of light provided, at full intensity, has been just the 95 % of the desired value. This could depend on different factors, properly related to the tool and the setup used, but it could be considered acceptable for the level of accuracy that it is aimed.
    The other limitation is related to the difficulty in the simulation of the different phases of the Sun illumination along the orbit, as the instrument

used has only three relevant modes that could be exploited (high, medium, low) but these do not provide the same light intensity correspond for each phase. Medium and low modes are not accurate in simulating the penumbra, or antumbra, phases of the Eclipse Shadow Model, for each orbit.
Therefore, this could be considered a valid method to change the illumination simulated along the orbit but not an accurate one.
The last consideration about the use of that source is related to the temperature limitation: for the setup chosen, the distance between the source of light and the sensors is small enough to make necessary the use of a temperature sensor to monitor the value achieved and avoid to exceed the design limits guaranteed. In case of temperature limit excess, the source of light must be turned off and this could affect the measurements taken during the simulation, in contrast with the data provided by the simulator.

- for the **microgravity simulation**, the instrument used and the setup prepared present more than one limitation to be discussed:
  - The first one that is important to point out is the limitation in the sensitivity of rotation achievable with the tool available: by design, the robotic arm could provide a minimum angle of rotation of plus/minus 1°, that is almost one order of magnitude higher than the one expected for the satellite in exam. Nevertheless, this consideration has been considered acceptable for the application presented, as the rotation provided is small enough to simulate the space environment desired;
  - The second consideration is about the limit on the number of stable configurations, due to a weight capability limitation of the tool, and degrees of freedom, usable for the setup of the simulation. This affects the variety of cases and scenarios that is possible to test and also the completeness of the work but it is a point that could be for sure improved in further development.
- for the **Star simulation**, even if the solution implemented was the only one possible with the device available, adopting a simulated operating mode does not ensure a realistic evolution of the attitude in time; in addition, this is not controllable during the simulation: therefore, after having set the starting parameters at the beginning of the simulation, the evolution of the quaternions is established and independent from anything external.
- The last limitation is related to the fact that all these simulation strategies are not part of the closed-loop, or automated, but manual, therefore user actions are requested every time to activate or disable them during the

simulation. This aspect may introduce some time-related inaccuracies that propagate until the final results, but also not coherent state simulated, with respect to the one computed in the orbital simulator.

5.4 Tests Execution

Once all the devices involved in the simulation setup and the reference assumptions have been clarified, it is time to discuss how practically the tests will be executed and which schemes to follow.
The tests involve basically the Hardware in the Loop simulation, shown in the block diagram in figure 5.10, which consists mainly of three parts:

- a simulated environment modelized in Simulink, whose aim is to provide the measurements from the simulated components, the updated values of attitude, rate and position of the satellite to the OBC hardware, in which is embedded the ADCS software. It takes as inputs the commanded forces, computed by the control algorithms implemented in the ADCS software, needed to properly control the satellite attitude each second of the simulation.
- the Cubesat mADCS board, which includes the OBC hardware and software and the integrated ADCS software that, using the input provided, computes all the information required by the user according to the mode simulated. This module is the core of the tests as it is connected, via the bench, to all the devices involved, and allows to close the loop properly by providing data back to the Simulator.
- the external components and devices connected to the core board via the bench: these include both sensors and actuators but also simulation devices needed to provide external inputs to the sensors.

The overall physical test setup, including all the main actors described, is represented in figure 5.11.

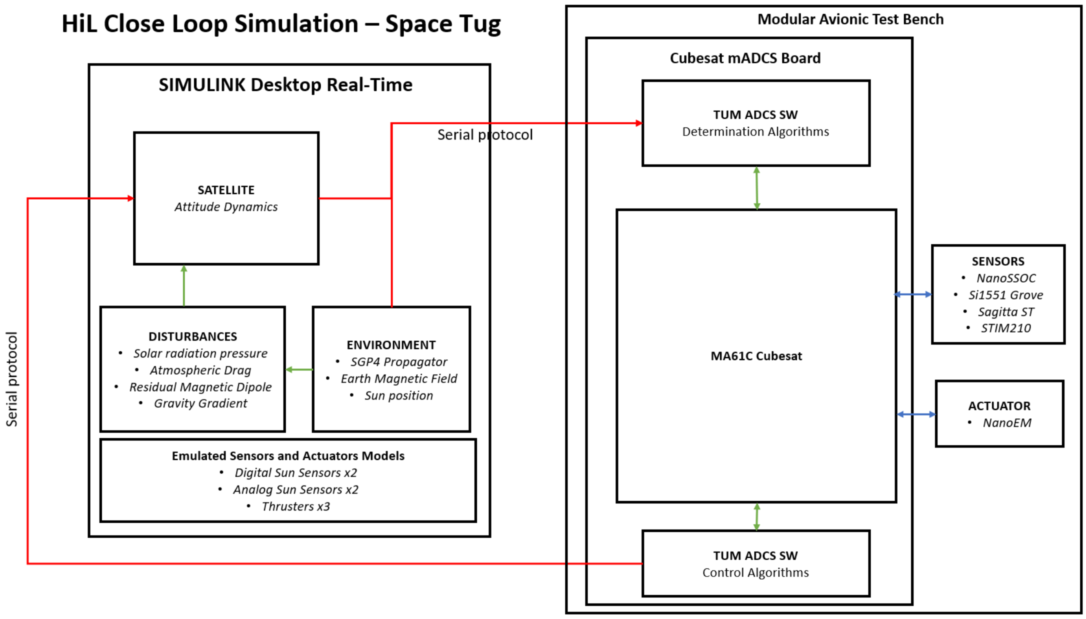


Figure 5.10: Hardware in the Loop Simulation Scheme

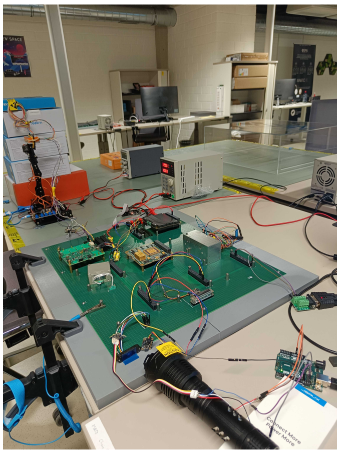


Figure 5.11: Final Setup of MATB for the Test

The execution of the tests is based on the following actions:

- First of all, to switch on the simulation devices, i.e. robotic arm and torch light, connected to the relative components
- Accessing the computer command prompt, entering the GRMON environment (via the command $grmon\ -ftdi\ -u$), loading the software version to be tested (using command *load* followed by the name of the compiled version of the

software saved in the proper folder, and run it (using simply *run* command).

- After the software starts, even the inputs from the Simulator have to be provided; therefore, via the Simulink model window, it is possible to run the simulation, starting the closed loop of data between the simulator and the hardware.
- Once all these steps are performed, in the prompt all the information requested in the software are displayed and the operation of the mode selected can be evaluated.

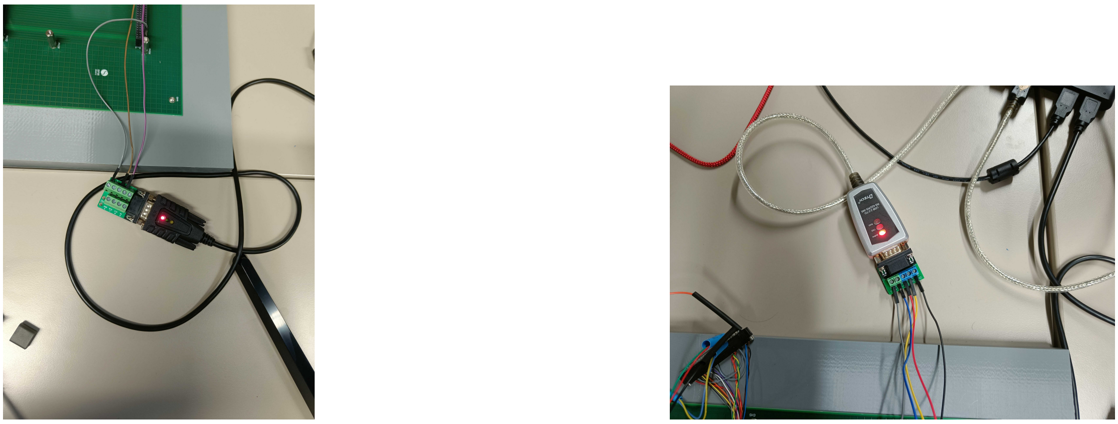
5.5 Debug tools and interfaces terminals

In the final section, an overview of the main instruments and tools used for debugging purposes during the testing phase is provided. First of all, it is important to point out that debugging tools play a crucial role, particularly in the implementation and testing phases of space components in a space project, critical to ensuring that spacecraft systems and components operate reliably and meet their intended objectives. The question is how debugging tools are used during these phases:

- Identifying and Resolving Software issues: in these phases, software is developed to control and manage various spacecraft components, so these tools are used to identify and rectify software bugs and issues, with the involvement of step-by-step execution of the software code, setting breakpoints, and inspecting variables and memory states to pinpoint the root causes of errors.
- Testing Communication Link and Protocols: space components often communicate with each other and with ground stations using specific communication protocols, so these tools help verify that these protocols are correctly implemented.
- Verifying Hardware Behavior and Data Analysis: In addition to software, they could be used also to test and verify the behavior of hardware components. This includes checking sensor readings, instrument responses, and the performance of onboard processors and electronics.

Furthermore, these tools provide, during testing, the capability to continuously monitor the health and status of spacecraft systems. This helps engineers detect and respond to anomalies, errors, and unexpected behavior as they occur, allowing for quick intervention and problem resolution.
It could be useful to give a quick overview of the main terminal used for the testing:

- *Termite* and *RealTerm* for all UART interfaces, e.g. RS-232, RS-422, RS-485, via the connectors shown in figure 5.12.



(a) USB to RS232 device

(b) USB to RS422/RS485 device

Figure 5.12: UART to Computer communication tools

- *PCAN view* for CAN interfaces;
- *Total Phase Control Center* or Arduino IDE for I2C or SPI interface;
- JTAG for Computer communication.

Chapter 6

Analysis and Results

This chapter aims to finally present the outcomes of all the tests conducted, verifying and validating both all the changes discussed in chapter 3 and the physical simulation setup, presented in 4 and 5.

6.1 Software performance analysis

The analysis of the performance of the software developed has to be done on different and subsequent layers and levels, as also the development itself of the software has been done in that way.
The various test cases carried out during the work could be presented and highlighted, keeping coherence with the software development chapter, in the following order:

- **Ma61c SmallSat version** with the original setup first, to give an idea of the structure of the output data that has to be achieved;
- **Ma61c CubeSat version stand alone**: testing the code with the CubeSat board without anything connected and without sending anything;
- **Ma61c CubeSat version communication test**: testing the code with the CubeSat board without anything connected but sending random data via a chosen interface available;
- **Ma61c CubeSat version functional test**: testing the code with the CubeSat board with the components connected, all mounted on the modular avionics test bench.
- **Ma61c CubeSat version Mission Scenario**: testing fully the final version of the software, within the complete setup of the simulation, including all the actors involved.

In addition, before the Communication test, it is useful to present also some insights into the testing of the Interface Test code developed; moreover, before the Functional Test with the actual components, the outcomes of the tests on new protocols are added.

6.1.1 Test 1: MA61C SmallSat Version

The first tested model is the SmallSat one, for which the following procedure has been followed: thanks to the fact that this version has been already developed and prepared before, the only thing to do has been to properly compile the main *.cpp* file: ADCS_performance.cpp.
Once that file is compiled, the procedure can start by first accessing remotely the SPiN test desktop, to load and run the compiled file that allows to start the hardware-in-the-loop test, *Equipment_emulator*, by sending inputs data to Arduino units and the MA61C EGSE emulator. After that step, the data flow begins and the SmallSat mADCS Node starts to provide outputs that could be obtained by loading and running the compiled file previously computed (ADCS_performance). The outputs have the following structure:

```
1725.50000000,0.00950963,0.00969349,-0.00906610,0.00585429,0.00000002,
-0.00000002,0.00000000,1.00000000
1725.50000058,0.00988900,0.00935927,-0.00904474,0.00589270,0.00000005,
-0.00000004, -0.00000001,1.00000000
1725.50000116,0.00988904,0.00935921,-0.00904471,0.00589270,0.00000007,
-0.00000007, -0.00000001,1.00000000
1725.50000174,0.00988908,0.00935915,-0.00904469,0.00589270,0.00000009,
-0.00000009, -0.00000002,1.00000000
1725.50000231,0.00988912,0.00935909,-0.00904467,0.00589270,0.00000012,
-0.00000012, -0.00000002,1.00000000
```

Focusing on every single couple of lines, it is possible to identify the following information provided by the model tested: in this case, in the reading output section of the main file, only the fields related to the torques measurements of the Reaction Wheel and the computed quaternions are selected, therefore, after the first value that regards the Julian date calculated, different for each time step, for each line it is displayed the evolution of the data relative to the torque in each direction and to the estimated value of quaternions.
According to the user's necessity and desire, it is possible to choose to obtain the time evolution of even the variables related to the attitude sensors emulated, such as gyros for angular velocity measures or sun sensors for attitude.

6.1.2 Test 2: MA61C CubeSat Version Stand Alone

For the CubeSat version, the test results cannot have immediately the same structure as the ones above, because of the differences between the models explained in the previous section: without the presence of the EGSE as an emulator of the components, it is not possible to follow the same test procedure as for the SmallSat version but, in the beginning, it will only be possible to compile the new version of the main file, which includes the CubeSat model's relative commands, and after that, loading and running the compiled file in the computer connected to the OBC Hardware, to see and understand if there is a data flow between the two devices (sent and received), in open loop.
Following that simplified procedure, the outputs obtained are there reported:

```
FOG Device not responding
FOG Device not responding
FOG Device not responding
FSS Device not responding
FSS Device not responding
FSS Device not responding
1725.50000000,-nan,-nan,-nan,-nan,nan,nan,nan,nan
FOG Device not responding
FOG Device not responding
FOG Device not responding
FSS Device not responding
FSS Parameter out of limit, value is: -129024.00000000, limit is:-1.10000000
Sent commands to restart, boot and initialize the FSS
FSS Device not responding
FSS Parameter out of limit, value is: -137280.00000000, limit is:-1.10000000
Sent commands to restart, boot and initialize the FSS
FSS Device not responding
FSS Parameter out of limit, value is: -138288.00000000, limit is:-1.10000000
Sent commands to restart, boot and initialize the FSS
```

The lines presented are relative to the first time frames returned, and the same structure is repeated for each following frame, in an open loop.
Concerning what has been obtained, it is necessary to highlight that, as expected from this version tested, no device has been found (that explains the first 6 lines shown), because of the lack of an EGSE emulator or the presence of physical space components connected to the Cubesat Hardware. Therefore, as a consequence, when asked to return the data measured by the sensors or actuators, the response is -nan (not a number), as displayed in line 7 of the outputs. It has been found

also in the next lines, as a response, parameters out of the limit set in the code, an index of random results obtained due to the lack of connection with components.

6.1.3 Interfaces Configuration and Test

After having validated that the updated version of the software, for Cubesat applications, is working, even without any external inputs, it comes time to verify if all the interfaces needed for the simulation can exchange information in both directions: receiving and transmitting.
According to the scenario and relative components chosen, the main interfaces to be tested are:

- I2C for EPS and Coarse/Analog Sun Sensor (CSS);
- SPI for Fine Sun Sensor;
- RS422 (UART 1) for Rate Sensor;
- RS485 (UART 2 and 3) for Attitude Control and Attitude sensor.

To perform functional tests on these, there are essentially three steps for each interface: first connecting to the bench both the OBC hardware and the relative interface to USB tool, connected to the computer, to check if there is a valid exchange of data between the two devices, properly configured; then the same process between the debug tool and the space component, to ensure that the device answers correctly to the commands sent and if it can provide valid data to the computer. The last one is to perform communication tests by removing the debug tool and sending commands directly from the OBC, as the master device, to each different space component, as the slave device.
Among these three tests, only the second one is reported below to show the correct interface connections and configuration for each device. Starting from I2C, as described in 2.2, the connection involves only two pins except for the ground one, SCL and SDA. The application used for the communication, $TotalPhase\ Control\ Center$, has to be properly set up in Master configuration, with the correct bit rate (100 kHz for EPS and 400 kHz for CSS) and the slave address specific for the device to be tested (0x51 for EPS and 0x53 for CSS), then the message to send could be written in the dedicated window and sent via the Master Write button; to read the eventual response from the device, the number of expected bytes has to be set before selecting the Master Read option. Both the messages, sent and received, are displayed and stored in the transaction log, allowing to have easier control of the device's operation.

I2C settings	Electrical Power System	Coarse Sun Sensor
Slave address	0x51	0x53
Bit rate	100 kHz	400 kHz

Table 6.1: I2C settings

For the SPI interface, one of the differences is the number of pins to be connected, as in this case there are MISO, MOSI, CLK, CS and also the settings to be defined, in the same application used before: besides the bit rate (set to 125 kHz), the polarity, phase and bit order are important parameters which define the configuration of the communication and have to be the same for the two devices that are interacting; the common process is to understand the ones of the slave, with which it is desired to communicate, and then as a consequence set the Master ones, which could be either the debug tool or the OBC. In the exam case, the bit rate is 125 kHz, the polarity is Falling/Raising, the phase is Setup/Sample, the bit order is MSB first and the CS polarity should be low.

SPI settings	Attitude Sensor
Bit rate	125 kHz
Polarity	Falling/Raising
Phase	Setup/Sample
Bit Order	MSB

Table 6.2: SPI settings

For UART interfaces the procedure to follow and the tools to use are different: starting with RS422, the connection is made of 4 pins (two lines for the transmitter, TX_P and TX_N, and the same for the receiver, RX_P and RX_N, and the application used is *RealTerm*. The configuration is defined once the following parameters are set: the port of the computer used for the connection, the data bits, usually set to 8, the stop bits, to 1, the Parity, the Flow control and Forward, to None and the baud rate, which changes according to the device connected (for UART 1 is 460800). For RS485 instead, the connection is simpler, as only two lines are needed, one for the transmitter and the other for the receiver and the application is the same. The main settings keep the previous values but the baud rate changes: 115200 for the Attitude Controller and 921600 for the Attitude Sensor. The other difference is also in the format of the data displayed: ASCII for the RS422 device and Hexadecimal for the RS485 devices.

For sending commands via the *RealTerm* application, the message has to be written in the correct format and with the correct encoding, in the SEND window, and then the response from the device is shown in the user interface.

UART settings	UART 1	UART 2	UART 3
Data Bits	8	8	8
Stop Bits	1	1	1
Parity	None	None	None
Flow Control	None	None	None
Forward	None	None	None
Baud Rate	460800	115200	921600

Table 6.3: UART settings

6.1.4 Test 3: MA61C CubeSat Version Communication Test

The following step, before introducing the physical sensors, has been to capture and send some data, via one of the interfaces available, to the MA61C board, to get back a response from the software, checking the correct working of the data flow process.
For doing that, it has been chosen to use as a debug terminal *Termite*, connected to the RS485 interface. After having loaded and run the compiled file, in the tool it has been possible to see first the request data sequence arrived from the software and after that, according to the auto-reply mode set, the encoded data sent back to the board.
In the meanwhile, in the prompt, the following lines of output are displayed:

```
1725.50000000,0.00953226,0.00967707,-0.00906957,0.00585587,0.00000348,
-0.00000002, 0.00000004,1.00000000
1725.50000058,0.00991161,0.00934285,-0.00904821,0.00589427,0.00000696,
-0.00000004, 0.00000009,1.00000000
1725.50000116,0.00991165,0.00934279,-0.00904819,0.00589427,0.00001044,
-0.00000007, 0.00000013,1.00000000
1725.50000174,0.00991169,0.00934273,-0.00904816,0.00589427,0.00001393,
-0.00000009, 0.00000017,1.00000000
1725.50000231,0.00991173,0.00934267,-0.00904814,0.00589427,0.00001741,
-0.00000012, 0.00000022,1.0000000
```

The output in this last test case has the same structure as the one obtained in the Ma61c SmallSat original version, of course with different results for each time frame considered, due to the varying inputs introduced.

This is justified by the fact that, unlike the previous CubeSat test case, now the code is recognizing some information coming from outside and for that reason, it is responding back in the same way that it did when the EGSE was connected and was emulating the data of the components.
This testifies that the communication line works and that when the components are connected to the Cubesat Hardware, there will be a correct exchange of data between them. It is also important to point out that to allow proper communication with all the components present in the test bench, it is required to update the software with a proper communication protocol for each device.
For this reason, the next section will be completely dedicated to protocol testing.

6.1.5 Encoding process and Output Management

As it has been told in the software development section, to verify if a protocol is good or not, it has to be tested properly, to make the on-board computer and the device speak the same language.
To test it, the process basically includes first eventually encoding the data to send, usually a command, send it, and then eventually decoding the data retrieved and sent back from the device, usually the telemetry. Sometimes, such as in the case of the coarse sun sensor, specific steps have to be followed every time it is necessary to send a command or to read a telemetry. Also for the action of writing or reading itself, according to the interface used, there could be some measures to keep in mind to build the data structure properly.
Below are listed all the devices with the relative protocols adopted:
Digital Sun Sensor Protocol
In the Interface Control Document of the device, it is explained how to build the frame format both for the request message, from the OBC (master) to the device (slave), and vice versa, for the response message.
The command message is composed of the command code (0x01 and 0x03 for requesting voltage values of the photocells or 0x04 for requesting angular positions of incident light), which is the code of the incoming command, the length, which for commands is fixed to 1 (0x01 in hexadecimal) and the checksum byte, different for each command. For the response message, the structure is the same except for the additional part related to the data sent from the device, according to the command selected.
Every request and response message is preceded by a synchronization word that indicates the start of a new frame, which is

$$1ACFFC1D$$

For the simulation, the command useful to get the measurements of the Sun Position is the $0x04$ and below is shown an example for this case: The expected data from

	Sync word	Command Code	Length	Data	Checksum
Request	0x1ACFFC1D	0x04	0x01	-	0x05
Response	0x1ACFFC1D	0x04	0x0E	$\alpha(4), \beta(4)$ and SI (4)	0x63

Table 6.4: Example of FSS messages

this command, in hexadecimal format, consists of 4 bytes for the floating value of Angle X (in degree), other 4 bytes for the second angle, Angle Y, 4 bytes for representing the floating value of Sun detection percentage and 1 last byte for the error code. The error code gives information about the angle calculation operation: if it is valid, affected by Earth Albedo, by Sun albedo or if the light is detected out of the FoV of the device.

To be able to retrieve the final values, useful for the simulation, it is necessary to convert from hexadecimal to the correct float codification (IEEE 754-1985) for single-precision floating 32 bits, by implementing a specific algorithm, reported in the Annexes.

The response provides the direction and the intensity of the light detected with respect to the following system of reference:

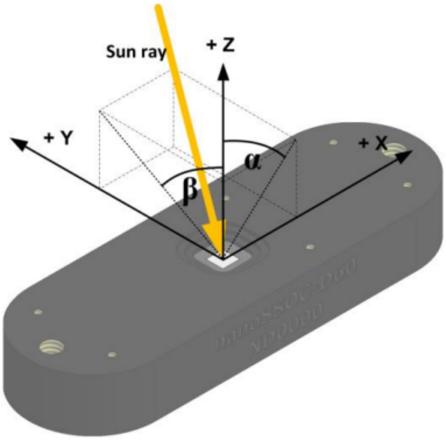


Figure 6.1: Sensor System of Reference [34]

Angle α and angle β in the picture, represent Angle X and Angle Y of the response. For the ADCS software, the measurements that need to be provided for the sun sensors concern the three body axes of the satellite, so two steps are required to convert the measurement provided and the one needed: first converting the direction vector, defined by the two angles on the XZ plane and YZ plane, into its three components with respect to the Sensor Frame; then, via a proper rotation matrix, considering the position of the sensor in the spacecraft, the vector must be rotated to the Body Frame.

Analog Sun Sensor Protocol

For the coarse sun sensor, a particular encoding format to send and receive messages is not requested: as the device works with an I2C interface, in the software there are already embedded functions to write and read properly a certain amount of data from a master unit to a slave device and vice-versa.
Nonetheless, a specific set of actions has to be executed according to the command chosen:

- to start the device, it is necessary to write in the COMMAND register, whose address is 0x0B, the code of the START command, which is 0x13;
- to set a determined configuration of parameters, for each of them more than one command is needed: first writing the value to assign to the parameter in the HOSTIN0 register (0x0A); then writing in the COMMAND register the location of the parameter, in binary, inside the parameters table, calculated by doing the logic operation OR between 0x80 and the code of the parameter (e.g. for changing the number of channels involved in the measurements the parameter to be changed is the Channel List (code 0x01), the location is 0x80 | 0x01 = 0x81, in binary 10000001, so 0b10000001 should be written in the COMMAND register);
- to read the value of a precise parameter it is required first to write in the COMMAND register the location of the parameter, like done before, but instead of 0x80 using 0x40 for the calculation, and then to read from the RESPONSE1 register, the data retrieved.

To read the output of the measurements needed for the simulation, it is enough to read from the HOSTOUTx registers, whose addresses vary from 0x13 to 0x2C, according to the channel selected. In the exam case, every channel result is represented with two bytes, one for each register; in the situation in which only the first channel is used for the measurements, the output data should be read from HOSTOUT0 and HOSTOUT1 and the value to use is calculated in the following way for IR band:

$$Measurement_{IR} = HOSTOUT0 \cdot 256 + HOSTOUT1$$

Star Tracker Protocol

For the Star Tracker the encoding format, defined as SLIP framing, is essential for the correct communication: every frame, sent or received, must start and end with the character 0xc0; this starting frame is followed by a fixed SLIP ID byte

(0x21), and for most of the messages by one byte defining the type of the message, one defining the ID of the parameter involved and one checksum at the end. For some messages, e.g. set parameter request, between the parameter ID and the checksum, there could be one additional field with the value to assign to the specific parameter. For the responses, usually this field is replaced with the Result one, showing if the command has been executed without errors, or with the parameter value itself if the type of the message is to request telemetry.
The types of messages used for the simulation are:

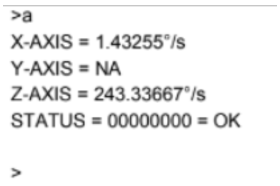
- set parameter (0x00 for request and 0x80 for response) to configure the parameters for the Simulator mode (parameter code: 0x18);
- request telemetry (0x02 for request and 0x82 for response) to retrieve the calibrated quaternions (telemetry code: 0x18);
- action (0x03 for request and 0x83 for response) to boot the device at the beginning (action code: 0x01).

An example of command and response is reported below:

Action	SLIP	Type	Action ID	Field Set / Result	Checksum
Request	0x21	0x03	0x01	0x01	0x9c349a61
Response	0x21	0x83	0x01	0x00	0xd802d483

Table 6.5: Boot action

For the simulation purpose, as the calibrated quaternions retrieved from the Solution telemetry response are in hexadecimal format, it is necessary to convert each set of 4 bytes into the correspondent floating representation, as already done for the measurements of the digital sun sensor and reported in the Appendix.
Gyroscope Protocol
For the gyroscope chosen, the format of the messages is ASCII and no particular encoding format is needed except for <CR> character at the end of the command. The device, by default, after powering on, starts directly to provide autonomous rate measurements, which are not easy to read or use, so two commands are needed: first setting the operational mode by sending the SERVICEMODE command, to get human-readable information; then performing a single-shot measurement, via the command *a*, to receive one measurement at a time for angular velocities detected. An example of the single-shot command is shown in the picture below:



```
>a
X-AXIS = 1.43255°/s
Y-AXIS = NA
Z-AXIS = 243.33667°/s
STATUS = 00000000 = OK

>
```

Figure 6.2: Single-shot measurement

The rate provided is in ASCII, therefore it needs to be converted to float before being sent to the simulation software. The code used is reported in the specific Appendix section.

Thruster Protocol

For the Attitude Controller the message, both sent and received, must be built with a certain structure: each data packet consists of a mandatory header and optional load (value), with variable length. The first byte of the header is the sender address (0x00 for OBC), the second byte is the receiver address (0xFF to send the information to all the thrusters at once), the third one is the message type, then one more byte for the checksum and the two last bytes for the payload length. The CRC8 checksum, in compliance with CRC-8-CCITT, polynomial 0x07, is generated over the whole packet.

The message types used for the simulation are:

- 0x01 (OK) type of message displayed in the response, after WRITE request is executed successfully.
- 0x03 (READ) for requests to read telemetry or parameters;
- 0x04 (WRITE) to set with proper values the parameters available;
- 0x05 (DATA), type of message displayed in the response, after READ request.

In the exam case, only the command for enabling and disabling the thrusters need to be sent. An example of communication is shown below:

	Sender	Receiver	Message Type	CRC	Payload Length	Value
Request	0x00	0x0C	0x04	0xC6	0x0200	0x0E01
Response	0x0C	0x00	0x01	0xFD	0x0000	-

Table 6.6: Enable command

In the request message, in the value field, the first byte is to define which parameter to change, via its address, and the second one is the new value to assign.

6.1.6 Test 4: MA61C CubeSat Version Functional Test

After having tested first all the interfaces and then all the encoding protocols needed to communicate with each component of the simulation, it has come time to replace partially the emulated devices present in the software with the ones tested and physically available. Unfortunately, the amount of devices available is not the one needed to make the software work properly, therefore some of them have to remain emulated or, in the exam case, simulated, as discussed in 3.2.
In the main file of the software, to simulate the most complete mode, e.g. sun-tracking mode, it is requested to provide at least the following measurements:

- one set of 3D measurements for angular velocities (Gyroscope);
- three set of 3D measurements for sun position (three digital sun sensors);
- three set of 1D measurement for sun intensity (three analog sun sensors);
- one set of 4D measurements for quaternions (Star Tracker).

With the devices available, it is possible to retrieve valid measurements for angular velocities and quaternions, but only one out of three sets for both the sun sensors. Therefore, for two digital and two analog sun sensors, sunlight measurements need to be simulated in the Simulator environment developed.
To perform this test, all the components are connected to the test bench, together with the OBC, and the software, properly updated with the code to retrieve all the telemetry needed, is compiled, loaded and run.
The outcome is displayed below:

```
SERVICE MODE command sent
BOOT command sent
SET PARAMETERS command for SAGITTA sent
Mode configuration set
Telemetry from STIM210 arrived
Response Length is:91
Angular velocities:3.47538,1.60106,6.50589
Sensor data (Gyroscope) updated
Telemetry requested from NanoSSOC
3D measurements for Sun Position are:0.0550653,-0.065731,0.996317
Intensity level of Sun Light is:33.1609
Sensor data (FSS1) updated
STR quaternions (x,y,z,w):0.250843,0.294029,0.92226,-0.00788785
Sensor data (Star Tracker) updated
Photodiode intensity measurements:0.424032
```

Sensor data (Photodiodes) updated

The first four rows represent initialization commands

- SERVICEMODE to set gyroscope in operational mode;
- BOOT action to start the star tracker;
- SET PARAMETER to configure the Simulation Mode of the star tracker;
- ADCS mode configuration, to set which mode to be tested.

After the initialization, measurements start to be requested: first angular velocities from the gyroscope, for which it is possible to see the length of the whole response of the single-shot measurement command and the three values in rad/s; then sunlight position and intensity from the sun sensor, displayed in the two rows after the confirmation of the request: the 3D measurements are adimensional and the intensity is in percentage with respect to the average Sunlight intensity. After the first digital sun sensor, the other two simulated ones are retrieved (FSS2 and FSS3) and the values are clearly displayed.
After the digital sun sensors, first the quaternions from the star tracker are requested and displayed, and then, at the end, the last measurement of the light intensity is provided.
This test has allowed to check that all the updates inside the software work and all the components connected on the bench are responding correctly to the commands and are providing valid telemetries.
The next and final step is to test and validate a complete mission scenario, with all the different parts involved simultaneously, including the simulated environment.

6.1.7 Test 5: MA61C CubeSat Version Mission Scenario

This final test includes all the elements developed and needed for complete hardware in the loop simulation:

- ADCS software to test;
- Simulator that provides simulated models for components missing;
- robotic arm to simulate microgravity environment and satellite rotation;
- torch light to simulate Sun brightness.

As discussed in the previous chapter concerning the testing methodologies 5.2, for running the simulation, first it is necessary to turn on the simulation devices, both the torch and the robotic arm, then the software can be loaded and run, and immediately after it comes to the simulator, to provide the needed simulated data. In this way, the software will have all the inputs required to run correctly the chosen operating mode designed.
For each scenario and mode simulated, it is possible to choose the outputs to be displayed and presented. Below an example of output for the Space Tug scenario selected, during Deploy mode, is reported:

```
SERVICE MODE command sent
BOOT command sent
SET PARAMETERS command for SAGITTA sent
Mode configuration set
Telemetry from STIM210 arrived
Sensor data (Gyroscope) updated
Sensor data (FSS1) updated
Simulated Components Telemetry arrived
Sensor data (FSS2) updated
Sensor data (FSS3) updated
Sensor data (Star Tracker) updated
Sensor data (Photodiode) updated
JD: 1725.60578704
Orbit Position is: 3655.58813477,1508.31762695,5654.20556641
Orbit Velocity is: -4.93920135,-3.91615415,4.24282837
Earth Magnetic Field is: -0.00003140,-0.00000953,-0.00002045
Sun Position is: -0.99999487,0.00293503,0.00127193
q1 desired: -0.29181179
q2 desired: 0.90806264
q3 desired: -0.02412785
q4 desired: -0.29947597
w1 desired: 0.00000000
w2 desired: -0.00110122
w3 desired: 0.00000000
```

As seen in the previous test case, the first twelve rows are for initialization and data retrieval from the components, therefore the mission scenario outputs start from the thirteenth row with the Julian Data (JD).
The time indication gives information about the progress of the simulation and it increases by 0.05 seconds each step. After the temporal reference, in the example,

data about the evolution of position and velocity are reported, and with them also vectors related to the Earth's Magnetic Field and Sun position at each specific time. Under the environmental data, there are reported the attitude and rate values determined by the guidance algorithms embedded in the software.
As already mentioned, the data displayed are just some of the possible outputs calculated by the software; the purpose of the test has been only to show the correct run of the software with the data provided and to validate the testing methodologies chosen for this particular scenario.

6.2 Output discussion

According to the tests conducted and the output achieved it is possible to state that the software version, together with the setup and the Simulator designed, has allowed to preliminarly test in all its levels the Attitude Determination and Control subsystem under exam.
The main features validated are highlighted and wrapped up in the following list:

- with the 6.1.1, as the original SmallSat version has been tested, the outcome represents a reference point and a guide for the results to be achieved with the final simulation of the updated code in the loop. It has been also useful to take confidence with the data provided by the ADCS software and with the relative structure.

- with the 6.1.2 the focus switches to the CubeSat version, first of all updated by replacing only the SmallSat software with the CubeSat one. This test allows to validate this first update and show how the ADCS software responds when it does not receive any kind of input from the devices. The outcome confirms that the software is running correctly but, as expected, without finding any devices and any data provided, so it is not able to calculate the results requested.

- the 6.1.3 role before the software communication test is to understand and validate the correct configuration of each of the interfaces used, before managing them directly inside the software.

- with the 6.1.4 the aim is to test if and how the software responds to artificial data provided by one of the interfaces available. In this way, the whole operation of the ADCS updated application is validated for the first time, although without useful and valid data. It is easy to highlight and identify a similar structure of the response as the one seen in the original SmallSat version.

- the 6.1.5 section is essential to understand how commands and responses are built and encoded, in order to manage correctly the outputs retrieved and correctly provide them to the ADCS software. This represents the first phase of the testing which involves also the physical components chosen for the simulation and it is useful to show how the different devices work and which kind of data and in which format are generated.

- with the 6.1.6 a step further toward the final simulation is achieved, as the communication between the OBC and all the components is fully tested, by updating the software with all the protocols and decoding procedures to read correctly the data provided.

- with the last section 6.1.7 The hardware-in-the-loop (HiL) setup has been tested fully, involving all the previous subjects tested and the Simulator to achieve a closed-loop environment to test the performance of the ADCS modular application. The outcomes of this process validate the setup chosen, the HiL logic designed and the exchange of data between the different integrated systems, part of the testing loop. Nevertheless, the real outputs, which have to be compared with the reference ones from the SmallSat project, have yet to be determined, because of mADCS software inconsistencies, that have to be investigated.

6.3 Anomalies and deficiencies

In this section it is interesting to collect and point out the most significant issues and anomalies encountered during the testing.
As most of the tests have been conducted step by step, validating each additional update or change separately before proceeding, there have not been any particular anomalies before the final and complete simulation.
Once everything had been merged to run the overall HiL simulation the first unexpected behaviors appeared:

- after the request for the calibrated quaternions is sent correctly to the Star Tracker, the software manages hardly to retrieve the full response from the device, which includes a total amount of 89 bytes, even if enough time passes before the reading procedure starts. Avoiding considering that the problem can be in the receiving line as the same software is capable of retrieving even longer telemetries from other components, e.g. gyroscope, it could be related either to a harness issue or to a deeper level software setting;

- during the validation of all the possible modes available for the scenario chosen, some of them did not work nominally, i.e. target tracking mode, probably due to the lack of some parameters previously present in the emulated version of the components but missing for the physical ones, or to the differences in the Simulator developed, far simpler and reduced; further investigations are needed to understand the specific points that are missing to make even these modes operate.

- after the first time step of the simulation, sometimes it happens that, even if a new request for telemetry is sent to the different devices, the data stored in the sensor variables remain the same as the ones stored in the previous step, therefore the update process does not work correctly and this would affect also the outputs provided;
- for particular modes, e.g. sun tracking, there have been some issues in the calculation processes of the quaternions and as a consequence even in the processing of the control torques needed to control the attitude of the satellite; as for point 2, also in this case there could be some parameters to modify or to add in the ADCS software, to adapt it properly for the scenario and setup implemented.

For all these anomalies encountered, as already mentioned, further investigations are required and left for future development of the work, to understand and focus on the missing points noticed in the actual model.

Chapter 7

Conclusions

In the final chapter of the work the focus is first on collecting the most significant outcomes and implications of the project within the New Space environment, then on explaining the main limitations encountered and faced, and in the end, the open points left for further future development are discussed.

7.1 Key findings and implication of the research

It is interesting to highlight the key aspects that the solution developed brought and put into evidence and the benefits that the Space industry, particularly the New Space economy, could exploit from this work.
These could be summarised and expressed into three main concepts:

- **Intercompatibility**;
- **Modularity**;
- **Adaptability**.

All of these features are ensured by the use of **Avionic Test Bench**, a tool to revolutionize ADCS software validation and testing.

7.1.1 Intercompatibility

One of the highest achievements of this project is to have implemented a software application, for testing the Attitude Determination and Control subsystem for CubeSats application, which has the capability to communicate with space COTS components, from different suppliers and with different interfaces. This is a huge advantage in terms of flexibility as it allows every company that wants to test its own software to freely choose the best device for the simulation, in terms of

performance, time available and cost, avoiding that the decision is constrained to the tool or configuration available for the validation.
This key feature is due to the embedded software inside the On Board Computer unit, which includes several protocols and drivers needed to exchange messages with the devices provided by most of the European market realities. Some of these drivers are the ones shown and used in the simulation presented.

7.1.2 Modularity

The second important key feature is the modularity approach, adopted in both software development and hardware setup. On the software side, the modular application developed allows to make changes and updates only where needed, without affecting the whole model and even more significantly it provides a solution that could be tested module by module, before its completeness, reducing drastically the time required for debugging software issues. On the hardware side, this concept translates into the possibility of connecting one component at a time to test it, validate its performance, and evaluate if it fits with the setup or if it has to be replaced. This approach is surely performance-oriented while limiting at the same time costs and testing time.

7.1.3 Adaptability

The third key feature ensured by this solution is the possibility to test every subsystem needed, within each mission scenario and mode of the satellite. A flexible testing solution can accommodate changes in hardware and software configurations, enabling to test new iterations efficiently without significant redesign of testing setups; the setup can be customized to simulate mission, subsystem or mode-specific conditions, ensuring that the system tested meets the demands and the requirements imposed.
Two other reasons that highlight the importance of adaptability for this kind of application concern fault tolerance and optimization purposes. Since for Space missions more and more robust systems capable of handling unexpected failures or anomalies are required, such a solution, capable of simulating various failure scenarios, is essential to assess the system's fault tolerance and develop strategies for recovery or mitigation. Moreover, since usually spacecraft have to operate under strict power or weight constraints, flexibility helps engineers optimize the validating process to minimize resource usage while still obtaining adequate data on system performance.

All these features presented would have not been as well implemented as they are

without the availability of the avionic test bench. This tool has allowed to put into practice these three concepts in the most optimized way possible: by simply plugging in all the devices needed in the different spots available, it has ensured the opportunity to create customized hardware configuration for each scenario desired, easily adapting the setup to the needs of the user.
For the development of space missions, especially during the testing campaign, time management is essential and this bench has minimized the temporal effort needed for the validation processes.

7.2 Limitations and challenges

Despite all the achievements earned, there have been also several limitations in the simulation setup developed and used for the work, which could be improved in order to increase the validity and performance of the testing tool.
The main restrictions present in the actual model could be identified in the following points:

- The first limitation is related to the use of devices that simulate space environment: torch light and robotic arm. In particular for the torch light, in the current simulation, the only two (out of four available) states used are on, for Sunlight exposure time, and off during the eclipse but it is not the most accurate way to simulate the variation of light along the orbit; there are phases in which the satellite is partially enlightened by the Sun and even these should be properly simulated by the source of light available. Regarding the robotic arm, in the simulation presented, as explained in 5.3, only rotations mainly with respect to the Z-axis are available, and continuous ones, without being controlled in time; this condition is surely far from the real situation in space, for which the best simulation possible should be achieved by providing to the arm the current angular velocity of the satellite every time step, instead of setting up a random rotation. Furthermore, in an ideal simulation, the whole satellite, not only the rate sensor, should be rotated.

- The second limitation involves the choice of the mission scenario to be tested and simulated. Two out of three scenarios include the use of magnetometers, which need a specific device (Helmholtz Cage) to simulate the Earth's Magnetic Field along the orbit; due to time constraints, this device could not be procured soon enough to be implemented and integrated into the simulation setup, therefore only the third scenario has been available.

- The third limitation is about the space components available, in particular for the star tracker and for the coarse sun sensor. Starting from the star tracker, the version of the device procured is purely electronic-based, without optics, so it is not possible to use a star simulator to provide valid images for the measurements. For that reason, the only operational solution has been using the Simulator Mode, for which a set of starting quaternions and a defined evolution rate have been chosen as inputs, to receive back, as calibrated quaternions, the time-varying values. For the coarse sun sensor, instead, the main limitation has been on the device available: in this case, no space components were already available so it has been necessary to procure them specifically; for this purpose, as the budget was limited, cheap and simple devices have been selected and this choice affected the reliability and the performances, in terms of accuracy.

- The last limitation concerns the number of devices that could be connected simultaneously to the simulation bench. As the test bench used has a maximum amount of spots that could be used for the components, a setup that includes also the redundant devices could not be tested with all the physical devices plugged in, but simulated versions are still needed.

Most of the limitations and challenges highlighted in this section generate all the possible suggestions and improvements for future development of the work, presented in the next and final discussion.

7.3 Recommendations and Future Work

Time constraints and deadlines usually do not allow to develop the work as in-depth as planned or desired, therefore this section collects the main open points missing, to achieve the most complete version of the project proposed.
The recommendations for further development could be gathered in the following points:

- First of all, the Space Tug scenario, already started, has to be closed and completed fully, before advancing with the others: all the modes available need to work nominally, so a deeper investigation on the relative .cpp and header files is requested, to understand which parameters are missing or are not properly initialized. The focus should be on the files which involve the calculation of the quaternions and the control torques of the thrusters.

- together with the first point, also the simulation device operation has to be improved: in particular it is needed to find a way to transfer in real-time satellite rate information from the Simulator to the Arduino unit that controls the movement of the robotic arm; the second aspect to improve is related to the management of the sun exposure along the simulation: a temporal plan has to be defined to choose for each phase of the simulation which mode of the torchlight has to be used, to get enlightenment as close as possible to the one in the Simulator.

- after having completed fully the third scenario, and having optimized also the operation of the simulation devices, the next improvement concerns the realization of the solution, presented in 5.2, for the Helmholtz Cage. With this simulation device ready and the magnetometer already procured, even the other two scenarios could be tested, as all the other components needed are available from the previous projects. Even for this application, a way has to be found to provide from the simulator to the hardware which would control the magnetic field varying in time, the correct real-time values to simulate during the simulation.

For sure, these points are not immediate and easy to solve but with proper dedication and effort, exploiting the information given along the chapters, once solved, could make the project even closer to real space simulations and the optimized tool developed could be widely used in the space market to facilitate and give flexibility to the testing process of Attitude and Orbit determination and Control subsystem of small satellites.

Bibliography

[1] NASA CubeSat Launch Initiative. «CubeSat101: Basic Concepts and Processes for First-Time CubeSat Developers». In: (Oct. 2017) (cit. on pp. 1, 2).

[2] Divya Bathia. «Attitude Determination and Control System Design of Sub-Arcsecond Pointing Spacecraft». In: (October 2020) (cit. on p. 4).

[3] Small Spacecraft Systems Virtual Institute NASA. «State-of-the-Art Small Spacecraft Technology». In: (Jan. 2023) (cit. on p. 3).

[4] ESA. *Satellite Test Bench*. URL: `https://www.esa.int/ESA_Multimedia/Images/2016/04/Satellite_test_bench` (cit. on p. 4).

[5] Irina Gavrilovich et al. «Test Bench For Nanosatellite Attitude Determination And Control System Ground Tests». In: (May 2016) (cit. on p. 5).

[6] Ran Qedar Saish Sridharan. «Development of a Universal Plug-and-Play Adapter». In: (2016) (cit. on p. 6).

[7] Ran Qedar Saish Sridharan. «Plug and Fly (SPiN-1)». In: (2021) (cit. on p. 6).

[8] Ignaty Romanov-CHernigovsky Saish Sridharan Ran Qedar. «Early verification using innovative hardware-in-the-loop system». In: (2021) (cit. on p. 7).

[9] Ran Qedar Saish Sridharan. «Using MA61C-EGSE, A Swiss Knife Tool For System Digital Validation». In: (2023) (cit. on pp. 7, 14–16, 43, 87–89).

[10] SPiN. *Ma61c Cubesat*. URL: `https://www.spinintech.com/data-handling-and-avionics` (cit. on p. 7).

[11] Ran Qedar Saish Sridharan. «Modular Avionics Test Bench». In: (2023) (cit. on pp. 9, 10, 13, 73).

[12] COBHAM. «GR712RC Dual-Core LEON3FT SPARC V8 Processor». In: (2016) (cit. on pp. 10, 11).

[13] COBHAM. «GRMON2 User's Manual». In: (2016) (cit. on pp. 11, 12).

[14] F.Landis Markley et John L.Crassidis. «Fundamentals of Spacecraft Attitude Determination and Control». In: (2014) (cit. on pp. 21, 30, 61).

[15] Chiara Lughi. «Development and validation of a test bench for Attitude Determination and Control System for small satellite». In: (2023) (cit. on pp. 21, 90, 92, 95).

[16] Manuel Pecorilla. «Development of the Active Attitude Determination and Control System for a 3U Educational CubeSat». In: (2022) (cit. on pp. 21, 22, 25, 27, 29).

[17] Meysam Mahooti. *SGP4 MATLAB Central File Exchange*. URL: `https://www.mathworks.com/matlabcentral%20%5C%5C%20/fileexchange/62013-sgp4` (cit. on pp. 30, 56).

[18] Michael Lemmon 2009-02-01. *What is a serial interface?* URL: `https://www3.nd.edu/~lemmon/courses/ee224/web-manual/web-manual/lab9/node4.html` (cit. on p. 33).

[19] Grace Lau 01/10/2023. *Synchronous And Asynchronous Data Transmission: The Differences And How to Use Them*. URL: `https://www.computer.org/publications/tech-news/trends/synchronous-asynchronous-data-transmission` (cit. on p. 34).

[20] Admin AfterAcademy 19 Jan 2020. *What are the Data Transmission Modes in a network?* URL: `https://afteracademy.com/blog/what-are-the-data-transmission-modes-in-a-network/` (cit. on pp. 35, 36).

[21] Tsun-Kuang Chi Shih Lun Chen. «A Novel Low-Power Synchronous Preamble Data Line Chip Design for Oscillator Control Interface». In: (September 2020) (cit. on p. 37).

[22] Corrado Santoro. «The Serial Peripheral Interface (SPI)». In: (September 2020) (cit. on p. 38).

[23] Mary Grace Legaspi Eric Pena. «UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter». In: () (cit. on p. 39).

[24] Sparkfun. *I2C*. URL: `https://learn.sparkfun.com/tutorials/i2c/all` (cit. on p. 39).

[25] Scott Campbell. *BASICS OF THE I2C COMMUNICATION PROTOCOL*. URL: `https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/` (cit. on p. 40).

[26] Shreyas Sharma. *Understanding CAN Bus: A Comprehensive Guide*. URL: `https://www.wevolver.com/article/understanding-can-bus-a-comprehensive-guide` (cit. on p. 40).

[27] EADS Astrium - JF COLDEFY / C HONVAULT. «SpaceWire VHDL IP Presentation». In: (November 2003) (cit. on p. 41).

[28] MathWorks. *Hardware-in-the-Loop Simulation Workflow*. URL: `https://it.mathworks.com/help/simscape/ug/hardware-in-the-loop-simulation-workflow.html` (cit. on p. 44).

[29] Front Grade Gaisler. «RCC's User Manual». In: (December 2023) (cit. on p. 48).

[30] Felix R. Hoots and Ronald L. Roehrich. «Models for Propagation of NORAD Element». In: (1980) (cit. on p. 56).

[31] Vallado D. A. «Fundamentals of Astrodynamics and Applications». In: (2013) (cit. on p. 56).

[32] Ali Hassani et Mehrdad Ghorbani Milad Pasand. «A Study of Spacecraft Reaction Thruster Configurations for Attitude Control System». In: (2017) (cit. on p. 69).

[33] Silicon Labs. «Si115x datasheet». In: () (cit. on p. 76).

[34] SOLARMEMS. «nanoSSOC-D60 datasheet». In: () (cit. on pp. 76, 110).

[35] Arcsec. «SAGITTA STAR TRACKER datasheet». In: () (cit. on p. 77).

[36] SAFRAN. «STIM210 datasheet». In: () (cit. on p. 78).

[37] InnaLabs. «N-Series Gyroscope datasheet». In: () (cit. on p. 78).

[38] Veoware. «WHL-50 datasheet». In: () (cit. on p. 79).

[39] Enpulsion. «NanoEM datasheet». In: () (cit. on p. 80).

[40] Adafruit. «LIS3MDL datasheet». In: () (cit. on p. 80).

[41] DHV. «PicoEPS datasheet». In: () (cit. on p. 81).

[42] KORAD. *Digital Control DC Power Supplies - KD Series*. URL: `https://www.koradtechnology.com/product/84.html` (cit. on p. 83).

[43] Not Black Magic. *Helmholtz Cage Project*. URL: `https://notblackmagic.com/projects/helmholtz-cage/` (cit. on p. 95).