UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

A Ground Station Software Framework and Telemetry Management System for Nano-satellite Missions

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Christian Ballesteros Sánchez

In partial fulfilment of the requirements for the degree in Telecommunication Systems Engineering

Advisor: Carles Araguz López
Adriano Camps Carmona

Barcelona, June 2016

Abstract

This thesis addresses the development of a telecommands and telemetry management system for nano-satellite Ground Stations, including its graphical control environment and an interface to communicate with other existing modules. A telemetry, tracking and command (TT&C) system must be able to control the satellite from an Earth operator, transmit uplink commands to be processed by its subsystems, receive downlink packets with the status information or the required scientific data of the mission and report permanently the satellite position, computing future visibility intervals as well.

The final objective is the implementation of the designed framework on the S-band/UHF/VHF Ground Station located on the B3 rooftop of UPC Campus Nord and should be design modular enough to be used on ^{3}Cat-1, ^{3}Cat-2 and future missions.

The development of this Ground Station framework has been done with: web technologies (for the website-based GUI) running in a LAMP platform, a Raspberry Pi SBC for the communications module, a GNURadio software for the Morse beacon reception and decoding and, finally, an orbit propagation library based on SGP4 algorithm to compute the satellite passes over the Ground Station field of view.

Resum

Aquesta tesi tracta sobre el desenvolupament d'un sistema de gestió de les telecomandes i de la telemetria de l'estació terrestre de nanosatèl·lits, on s'hi inclou un entorn gràfic de control i les interfícies necessàries per comunicar-se amb altres mòduls existents. Un sistema de telemetria, rastreig i comandes (TT&C) ha de ser capaç de controlar el satèl·lit des d'un operador terrestre, transmetre comandes ascendents per ser processades pels seus subsistemes, rebre paquets descendents amb la informació sobre l'estat o dades científiques de la missió i informar permanentment sobre la posició del satèl·lit, calculant a més els propers intervals de visibilitat.

L'objectiu final és la seva implementació a l'estació terrestre situada al terrat de l'edifici B3 del Campus Nord de la UPC i ha de ser suficientment modular per a ser usat tant als projectes ^{3}Cat-1, ^{3}Cat-2 com a futures missions.

Seguidament, aquestes són algunes de les eines emprades en el seu desenvolupament: llenguatges de programació web (per la GUI basada en un lloc web) sobre una plataforma LAMP, un ordinador de placa única Raspberry Pi pel mòdul de comunicacions, un software GNURadio per la recepció i la descodificació de la balisa (*beacon*) Morse i, finalment, una llibreria per la propagació d'òrbites basada en l'algoritme SGP4 que permetrà calcular les passades del satèl·lit sobre el camp de visió de l'estació terrestre.

Resumen

Esta tesis trata del desarrollo de un sistema de gestión de telecomandos y de la telemetría de la estación terrena de nanosatélites, el cual incluye un entorno gráfico de control y las interfaces necesarias para comunicarse con otros módulos existentes. Un sistema de telemetría, rastreo y comandos (TT&C) debe ser capaz de controlar el satélite desde un operador terrestre, transmitir comandos ascendentes para ser procesados por los subsistemas del mismo, recibir paquetes descendentes con la información sobre el estado o datos científicos de la misión e informar permanentemente sobre la posición del satélite, calculando además los próximos intervalos de visibilidad.

El objetivo final es su implementación en la estación situada en la azotea del edifico B3 del Campus Nord y debe ser suficientemente modular como para ser usada en los proyectos ^{3}Cat-1, ^{3}Cat-2 así como en otras misiones futuras.

Seguidamente, éstas son algunas de las herramientas usadas en su desarrollo: lenguajes de programación web (para la GUI basada en una página web) sobre plataforma LAMP, un ordenador de placa única Raspberry Pi para el módulo de comunicaciones, un software GNURadio para la recepción y decodificación de la baliza (*beacon*) Morse y, finalmente, una librería para la propagación de órbitas basada en el algoritmo SGP4 que permita el cálculo de los pases del satélite sobre el campo de visión de la estación terrena.

Dedication

To everyone who pushed on the ^{3}Cat team to achieve our goals, students, professors or external helpers. To all those people who believed that one day our little satellite could reach the space. And, especially, to my family and my friends, who give me breath to face any challenge.

Acknowledgements

Before deepening into the theory, I think it is mandatory to name everyone that contributed on the realization of this thesis.

To begin, I want to express my gratitude to Prof. Adriano Camps, the person who suggested me to enroll in this project and my principal advisor.

Special thanks also to Carles Araguz, who has suffered me from the first day (since PAE), your bright ideas and your work are priceless on this thesis and the whole project.

Finally, I would like to thank their help to all my mates from the NanoSat Lab: Pol Via, Joan Frances Muñoz, Alexandre Cortiella, Marc Marí, Arnau Solanellas and Jaume Jané.

Revision history and approval record

Revision	Date	Purpose
0	09/05/2016	Document creation
1	18/05/2016	First chapters revision
2	13/06/2016	Technical chapters revision
3	17/06/2016	Document revision
4	25/06/2016	Final revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Christian Ballesteros Sánchez	christian.bs.8@gmail.com
Adriano Camps Carmona	camps@tsc.upc.edu
Carles Araguz López	carles.araguz@upc.edu

Written by:		Reviewed and approved by:	
Date	09/05/2016	Date	24/06/2016
Name	Christian Ballesteros Sánchez	Name	Adriano Camps Carmona
Position	Project Author	Position	Project Supervisor
		Name	Carles Araguz López
		Position	Project Supervisor

Table of contents

List of Figures

List of Tables

Glossary

ACK Acknowledgement (as an answer on a communications protocol)

ACS Attitude Control System

ADCS Attitude Determination and Control System

AJAX Asynchronous JavaScript and XML

AMSAT Amateur Satellites (organization)

AOS Acquisition of Signal

CDS CubeSat Design Specification

COMMS Communications (module)

CSV Comma-Separated Values

ECI Earth-Centered Inertial

EPS Electrical Power System

FIFO First In, First Out

FOV Field of View

GPS Global Positioning System

GS Ground Station

GST Greenwich Sidereal Time

GUI Graphical User Interface

HWMod Hardware Module

IGRF International Geomagnetic Reference Field

ITU International Telecommunication Union

JSON JavaScript Object Notation

LAMP Linux, Apache, MySQL and PHP (archetypal model of web service stack)

LEO Low Earth Orbit

LONESTAR Low-earth Orbiting Navigation Experiment for Spacecraft Testing Autonomous Rendezvous and docking

LOS Loss of Signal

NORAD North American Aerospace Defense Command

OP Orbit Propagator

OS Operating System

PAE *Projecte Avançat d'Enginyeria* (Advanced Engineering Project, in Catalan)

PDR Preliminary Design Review

PNG Portable Network Graphics

RTL Realtek chipset

SBC Single Board Computer

SGP4 Simplified General Perturbations model (1988 FORTRAN IV publication)

SPI Serial Peripheral Interface

SQL Structured Query Language

SSH Secure Shell

TLE Two-Line Element set

TT&C Telemetry, Tracking and Commands

UHF Ultra High Frequency

UPC *Universitat Politècnica de Catalunya*

USB Universal Serial Bus

USRP Universal Software Radio Peripheral

VHF Very High Frequency

1. Introduction

During the last years, some CubeSat projects have been developed at the UPC Laboratory of Small Satellites and Payloads (or UPC NanoSat Lab), located at the A3 building of the UPC Campus Nord in Barcelona. In particular, the ^{3}Cat project encompasses several generations of nano-satellites, including ^{3}Cat-1 and ^{3}Cat-2, fully designed and manufactured at this university, mostly by graduate and undergraduate students.

The essential parts of satellite missions are:

- the space segment, i.e. the satellite
- the ground segment or Ground Station; and
- the launcher, only used on at the beginning of the mission

This thesis is focused on the ground segment and all the technologies involved on the Telemetry, Tracking and Commands (TT&C) subsystem and the control of the communications from the Earth to the Satellite. In ^{3}Cat generation satellites, as in other CubeSat missions, the frequency bands used on the communications are VHF and UHF for telemetry and commands, which involve few data and need less bandwidth and S band, higher on the spectrum and used for large amounts of scientific data.

Therefore, the software framework that will be designed must include a user-friendly interface to manage and visualize the TTC subsystem information and other modules to automate multiple processes like computing the next visibility intervals, controlling the antennas hardware or decoding some beacon signals. Then, on this document, three main parts can be differentiated, with one more that includes the rest of them: a Graphical User Interface (GUI) development, an Orbit Propagator design, the Ground Station (GS) software structure design, including a Morse beacon decoder, and the fourth part is dedicated to all the elementary and full tests.

1.1. Graphical User Interface

A graphical interface is basically an intermediary between the user and the machine, a GS in this case. It is responsible of showing clearly some information and it must read and correctly interpret the user inputs.

At first, it is necessary to mention that this part is a continuation of the work previously done on the Advanced Engineering Project course (PAE), where a preliminary design of the web-based interface to control the GS software was proposed by the same author of this thesis. It was divided in three parts: the last state information, a telecommands queue and a telemetry register. While the first two have been modified, improved and debugged over the last months from the previous design, the last one, the telemetry part, has been addressed solely in this thesis. All of them include the management of a MySQL database, which is used by the GS software to pack, send and receive the commands and it includes the log files information of sensors, an orbit register, some important data of the satellite, etc. Besides the implementation of these parts, some other features have been designed and introduced to the interface in order to meet multiple the needs of the mission or improve the user experience.

The GUI requirements are detailed here:

- the ability to program commands and update their status once introduced on the queue,
- a telemetry visualization framework and the ability to monitor the status of the spacecraft,
- the satellite last state details display,
- data filtering and visualization options to help users on the data analysis, and
- an access control with user authentication protocols.

1.2. Orbit propagator

Once the GS software framework and the communications protocol are defined, there is still another question to solve: when is the satellite going to pass over the GS field of view (FOV), so there is a guarantee that the commands will be sent and received properly? This task is assigned to the orbit propagator. It is, actually, a program able to predict the satellite position at a given time; therefore determining the satellite "passes" over the GS (i.e., when the GS has to start and finish the communication process).

The program has to accomplish the following goals:

- calculate the visibility time intervals,
- save them on a table in the database,
- provide other essential parameters, if required (such as velocity, GST time…) and
- manage time divergences on the visibility intervals update

1.3. Morse beacon and Ground Station software structure

A last part related to the GS software is the decoding of a beacon signal. This is something that was tackled in other students' thesis from the NanoSat laboratory, but was never fully integrated to the GS environment. A first approach is found on a Matlab program designed by Ms. Elisabet Tremps on her thesis *Design and implementation of the ground station for* ^{3}*Cat satellites* [1]. Her design was based on a Matlab program, but has been finally discarded due to the incompatibility with the Linux platform used in the GS and its high computational cost.

Another objective, besides the beacon decoder inclusion, is the software framework design with all the required modules integrated on the same system. This design should be used on both missions adapting them for the particular specifications of each one. Therefore, the most important features to focus on are the reusability and the modularity.

The Morse beacon software must fulfil these requirements:

- beacon signal reception and processing to obtain readable data,
- message decoding (from Morse code to text) and
- storage of the message on a database

On the other hand, these are the Ground Station software main parts that have to be designed, implemented or tested:

- A web server to host the user interface
- A back-end structure for the command and telemetry management and the required communication protocols in order to send or receive data packets
- The beacon receiver program, including data decoding
- The GS hardware drivers and controllers (e.g. rotors control, polarization switches, etc.)

1.4. Testing

The test campaign includes all the validation methods developed during the integration of the previously described software modules. Moreover, the execution of the command instructions in the satellite subsystems (ADS, EPS, COMMS…) must be verified on a real communications environment. The methodology includes the link establishment, the validation of the commands protocol for each case and the data analysis with the obtained files or the satellite state (e.g. power consumption or sensor values).

Detailed test procedures, values and extended methodology, as well as a thorough break down of the actions and expected results designed for the test campaign, are presented in the Validation Plan reports [2][3], also available at the appendix C.

1.5. Work Plan

Table 1.1 shows a list of the tasks performed on the project, with their start and end dates and an estimated duration.

Table 1.1 Work Plan tasks

Task	Start	End	Duration*
1. GS-1 GUI development	15/02/2016	22/03/2016	27 days
2. GS-2 GUI development	02/05/2016	06/06/2016	26 days
3. GS software restructuring	05/05/2016	10/06/2016	27 days
4.1.1 GS-1 SW testing	25/02/2016	29/03/2016	24 days
4.1.2 GS-1 end-to-end testing	22/03/2016	08/04/2016	14 days
4.2.1 GS-2 SW testing	09/05/2016	10/06/2016	25 days
4.2.2 GS-2 end-to-end testing	13/06/2016	24/06/2016	10 days
5. Orbit propagator design	18/04/2016	18/05/2016	23 days
6. Morse beacon decoder	25/04/2016	20/05/2016	20 days
7. Security protocols implementation	20/04/2016	06/05/2016	13 days

The figure 1.1 shows the Gantt diagram including the previous tasks.

During the project development, some tasks have been modified, basically their milestones dates, due to multiple decisions associated to an immediate launch of ^{3}Cat-1 satellite, although it was later postponed. This first mission acquired a critical priority at testing time, and all other tasks were postponed.

*Weekends are not considered on the duration

The first set of tests detected some bugs in the software that had to be fixed and this was another reason why the following steps were delayed.

The GS backend responsible of the packet processing was not still fully debugged at the time of the tests and the process became harder. The GUI had some errors too, that were solved on those weeks as well.

Otherwise, [3]Cat-1 tests were foreseen for the lasts weeks of April in the preliminary review document. The high priority that they took (the satellite should be delivered by mid-April) forced to advanced them almost a month. In conclusion, despite all the extra difficulties found, the time plan was accomplished with a couple of weeks of margin and the new features that must be included or the oldest ones to be reviewed had the expected time of dedication or even more.

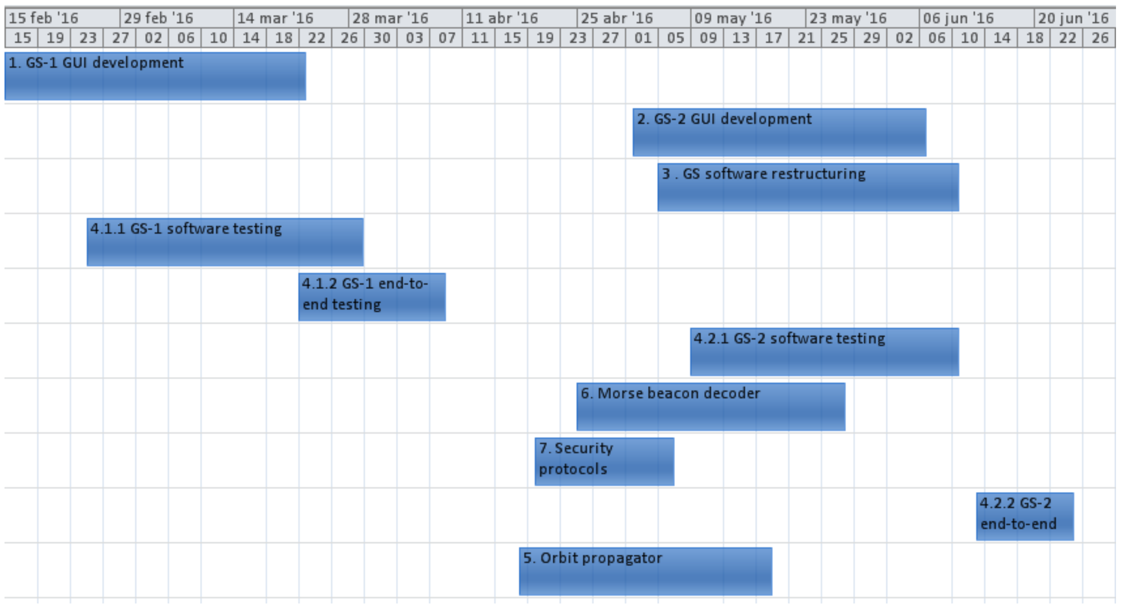


Figure 1.1 Gantt diagram

2. State of the art

The adoption of CubeSat platforms in satellite systems, specially for Earth Observation and technology demonstration missions, has grown over the past years and is expected to increase even more, with the adoption of newer and larger form factors. But, what exactly is a CubeSat? It is a small satellite (a nano-satellite, actually) based on a cube-shaped basic structure. These "cubes" can be combined to form larger satellites with more units. The most common structures are made of one (1U), three (3U) six (6U) or twelve units (12U), but almost any design is possible.

In the following figure, there are the expectations of the nano-satellites and micro-satellites launches for the next years.

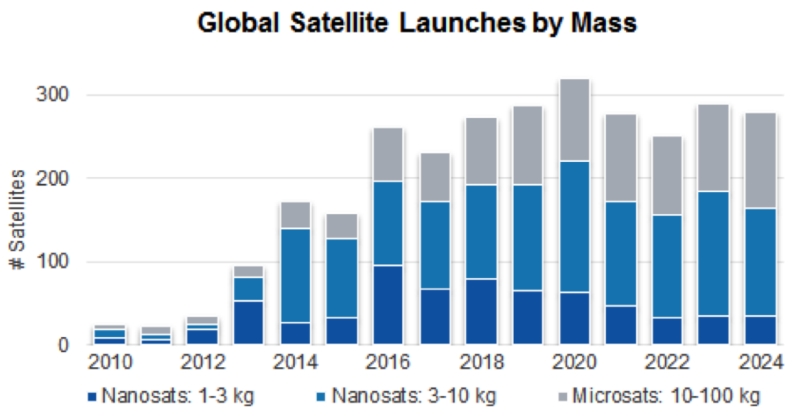
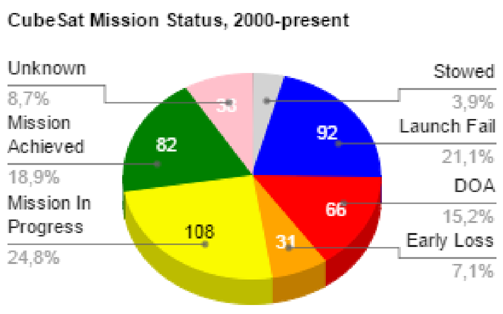


Figure 2.1 Nanosat and microsat launches (source: NSR[4])

[3]Cat-1 is included on the 1-3 kg group, while [3]Cat-2 is in the 3-10 kg group, as they are 1U and 6U CubeSats (10x10x10 and 10x20x30 cm). They follow the California Polytechnic State University standard defined on their document *CubeSat Design Specification (CDS)* from The CubeSat Program[5].

A list of all projects developed to the date can be found in a database published by the Saint Louis University (Missouri, USA), which is divided by launch year, mission type or current status[4].



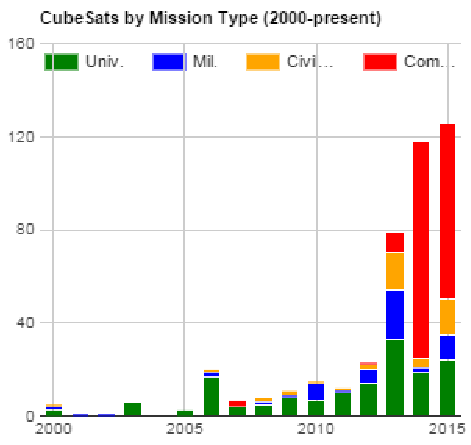


Figure 2.2 CubeSat missions' status (left) and type (right) by SLU, MO, USA [6]

The work in this thesis is focused on Ground Station systems. It includes all the communications software and hardware and a tracking system, always necessary to establish a reliable link. Many universities are currently developing their own systems because they are a rather affordable solution to develop space missions with educational purposes, although the commercial, industrial and scientific use is clearly growing as figure 2.2 shows on the right chart.

It is important to know how the GS software works to determine whether the developed program meets the requirements and execute the tests successfully. Anyway, it could be interesting to mention some other projects that are being or have been performed related to CubeSats.

An important feature to take into account is the frequency band on which they are operating. Most of these projects, as they are developed by universities for educational and research purposes, operate in amateur bands, mainly on the 70 cm band located from 420 to 450 MHz (UHF) or the 2 m band from 144 to 148 MHz, in ITU Regions 2 and 3, or 146, for Region 1 (VHF)[7]. For example, the Radio Amateur Satellite Corporation (AMSAT) and the Innovative Solutions In Space (ISIS) company launched in June 2014 the EO-79 satellite [8], also known as QB50p1 and FUNcube-3, a 2U CubeSat operating in the 435.035 – 435.065 MHz range for the uplink and the 145.935 – 145.965 MHz range for the downlink.

Even the National Aeronautics and Space Administration (NASA) has its own CubeSat launch program. The Educational Launch of Nanosatellites (ELaNa)[9] initiative includes some experimental nano-satellite projects destined to science and engineering students. ELaNa-XII is their most recent mission (October 2015, although ELaNa-VII launch was finally one month later), which includes four different CubeSats: ARC, BisonSat, LMRST-Sat and Fox-1. As an example, the first one, the Alaska Research CubeSat (ARC)[10], is a 1U CubeSat with a telemetry downlink and CW beacon. In this case, there is also a payload downlink at 2440.5MHz, equivalent to the S band, as in ^{3}Cat-2.

In terms of the Ground Station segment, a project that has some similarities with the one being explained on this document is FUNcube-1 (the precursor of the EO-79), an educational 1U CubeSat developed by experts from AMSAT-UK and AMSAT-NL. The ground segment is based on a USB receiver (to make data reachable to everyone who could be interested in it) and a computer program which uses the Internet to communicate with a remote database. In figure 2.3, its basic structure is presented. On the telemetry warehouse webpage [11], the user can see real time data without installing any program or using any device.

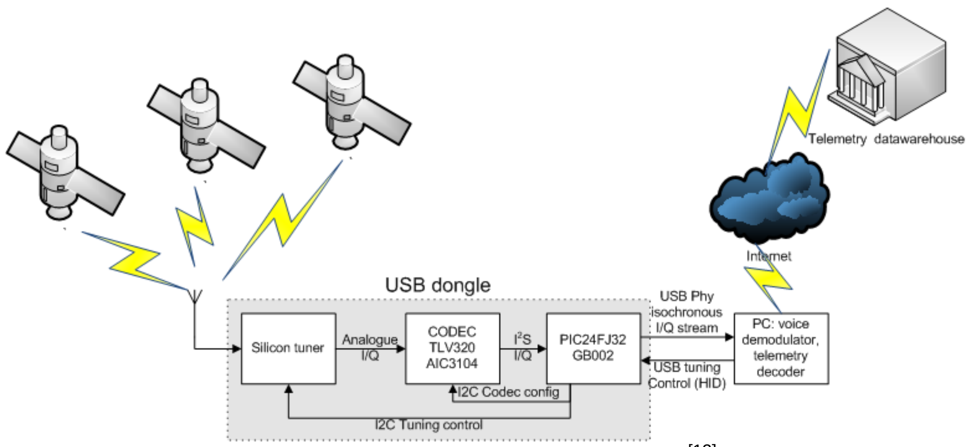


Figure 2.3 FUNcube-1 Ground Segment [12]

Another critical part is the command and data handling, which allows both the user and the satellite to communicate and correctly interpret the given information. One of the latest research projects in this direction is presented in *A Reusable Command and Data Handling System for University CubeSats* [13], where some members of the Texas Spacecraft Laboratory from the University of Texas at Austin (UT-Austin) have designed a Command and Data Handling (C&DH) system based on a centralized system architecture, which has been improved and is being implemented for their current missions. The first one is RACE (standing for Radiometer Atmospheric CubeSat Experiment), which was on board the Antares during the launch accident in October 2014. Bevo-2 is part of the second mission of the LONESTAR program, after Bevo-1, that implemented for the first time a main flight processor and a C&DH software in C++ and Linux environment. And the last one is ARMADILLO (standing for Atmosphere Related Measurements And Detection of submILLimeter Objects). The Flight Software (FSW) is quite alike to ^{3}Cat-2, including a communications (COMMS) module, an Attitude Determination and Control System (ADCS), the Electrical Power System (EPS), the Star Tracker software with the associated camera and the C&DH module itself. The only part that has been added and it does not exists on ^{3}Cat-2 satellite is the GPS receiver, which has a direct relationship to the mission scope, different on both cases.

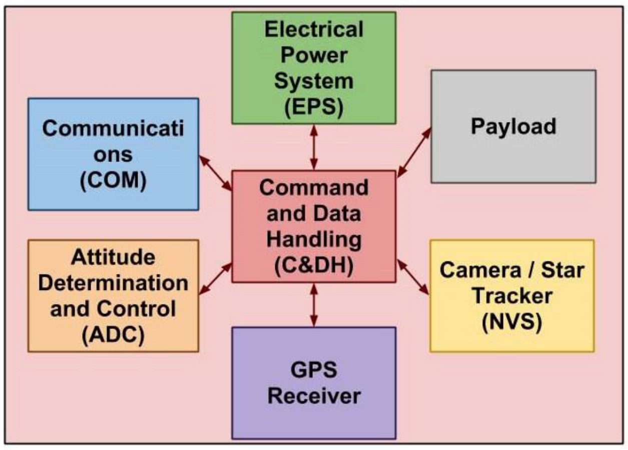

Electrical Power System (EPS)
Communications (COM)
Payload
Command and Data Handling (C&DH)
Attitude Determination and Control (ADC)
Camera / Star Tracker (NVS)
GPS Receiver

Figure 2.4 FSW modules of TSL missions[13]

Finally, the most important feature that should be emphasized is its reusability. On a large project, such as ^{3}Cat or LONESTAR, where multiple satellite generations are developed, it is a key point that could save a lot of time if it is considered from the beginning. Every software part has to be generalized to a large range of cases and developers must disengage it as much as possible from the existing hardware, because it could be very different on future missions, but not the final operation. And, probably, this is the most important challenge to achieve in the end.

3. Ground Station software architecture

There are some basic requirements that must be fulfilled on any satellite mission. On the GS segment, these are mainly four:

- transmission of telecommands,
- reception of telemetry,
- reception of scientific data, and
- satellite tracking

For each one, there are multiple modules responsible of every individual task and, in addition, other features can be implemented to make them easier or more robust.

3.1. Scope and main objectives

As it was previously commented on the introduction, in the framework design for this thesis, the main user interface is a website-based GUI. It includes all the necessary scripts and functions to store or display multiple telemetry data types or update a telecommands list. However, this is not enough for the data handling. A GS back-end software module (not the same as the website back-end) is responsible of the data packing and sending to the communications hardware module.

The tracking module has to be divided into two parts: the orbit propagator, to compute the future passes, and the rotors control, to point on the right direction once the GS is certain of the satellite visibility. Moreover, other features such as the transmission power can be controlled, as well as the antennas polarization or periodicity of the next passes update.

In conclusion, the main goal of a GS software structure like this one is a reliable communications link between the user and the satellite, and it involves multiple protocols, interfaces or hardware control drivers. Each one can be found on a different system layer but this document will be focused mainly on the highest parts, including user interaction, data handling and a high abstraction intelligence, i.e. how the telemetry and commands information are managed from both user and GS sides tacking into account the subsequent processes.

3.2. Software modules

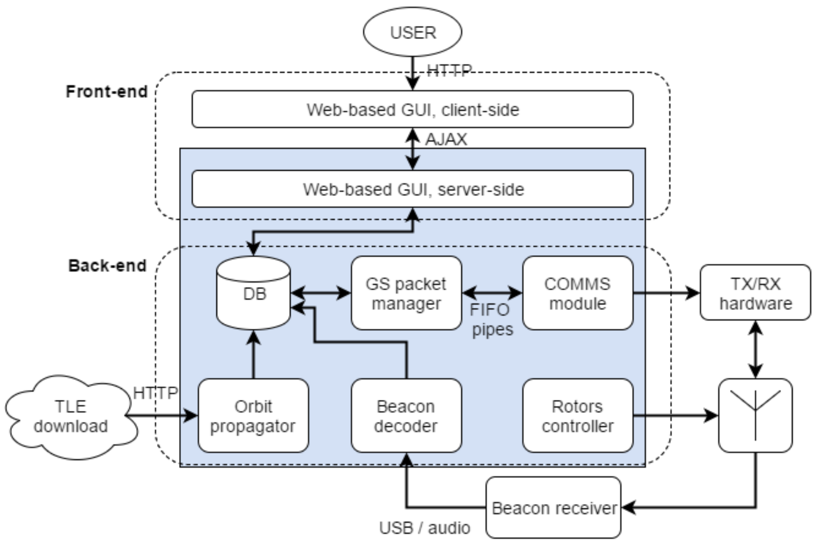


Figure 3.1 Software modules design

4. Framework design

4.1. Introduction

The main part of this thesis is focused on the development of a dynamic command and telemetry management system which automatically schedules commands, plots telemetry data, and allows the user to interact with the satellite through the GS and communications subsystem. A basic feature that it must have is the ability of being run on any platform (computer), with independence of its O.S. or the hardware device. Some alternatives are:

- a GUI written in C/C++ using portable graphical libraries,
- a Java interface, portable by definition (it is executed on a virtual machine), and
- web platforms and technologies

Finally, the GUI was decided to be based on a web platform because it does not require any type of previous installation, there is a huge amount of available resources for the graphical development and the communication with the server side is quite simple. In fact, it only needs a web browser to be executed. The "intelligence" to manage commands, telemetry or any information is located on the server and it can be controlled directly from the hosting device or remotely, for example, with an SSH connection.

In addition to the interface, a packet manager was required as well. The telecommands and telemetry data shown on the GUI needs to be received or sent with some communication modules coordinated by a main application. It is also the same module responsible of the database update, processing the information sent from the satellite or changing the state of the queue, for example. It will be continuously running and managing the whole system, giving an autonomous operation to the GS.

4.2. User interface

4.2.1. Telecommand queue

As many of other satellites, ^{3}Cat-1 and ^{3}Cat-2 base their communication modules on some instructions known as telecommands (or simply "commands"). Once sent to the spacecraft, the commands trigger different system actions (e.g. enable a subsystem, change a given configuration parameter, set the power mode, etc.) or request mission/housekeeping data (i.e. payload data, sensor data, system logs, etc.), but all the commands follow the same structure and protocol, although these are usually mission-specific. The GS command queue keeps the list of commands programmed by the user on a table from a database and, following a protocol which will be explained in future sections, they are sent to the satellite in packets by the communications module.

The commands list is based on a FIFO ordered queue, so the first command introduced is the first to be sent and the sending process of the next one will not start until a valid response is received or the GS segment is certain of the command reception on the satellite. This is, in fact, the simplest way of scheduling but the most robust too.

At any time, the queue can be modified changing the order or deleting a single command, but the interface software must be "intelligent" enough to prevent undesirable modifications that could affect the communications protocol. In addition, an extra feature has been introduced to allow a more complex command scheduling. Even there is only one queue table in the database, each command has an identifier associated to a future visibility period (also called orbit or satellite pass). It permits the user to program a queue that can be sent on a concrete time, not just the first time a link is established on the future.

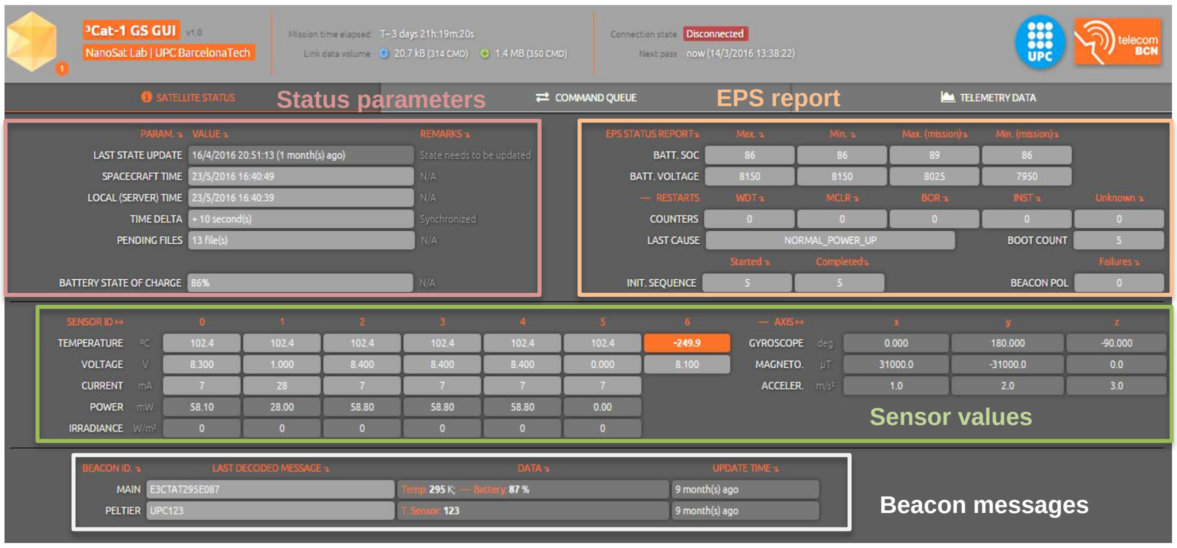
4.2.2. User-side software development

The front-end architecture includes all the interfaces, functions or scripts that are directly executed by GUI but two sides can be differentiated: the user and the server side. The modules located on the first one are basically programmed using two languages, HTML and JavaScript, apart from the CSS files for the style, at the user side; and . The webpage is based on three different layouts, namely: satellite status (default view), the commands queue, and telemetry data. These views are accessed through a navigation bar placed below the state information header. Their implementation is detailed in appendix A.1.

On the following chapters, the main functions and files used on the GUI development will be explained.

4.2.3. Layout and design of the GUI views

The satellite status page includes the main information that has to be displayed to know the state of the satellite vitals, and the last updates from the sensors, EPS, etc. These tables are based on `div` tags (http://www.w3schools.com/tags) managed by the Cascade Style Sheets (CSS), where the elements properties have been defined for the different type of cells, rows or the table itself. Full page screenshots of the interface design can be consulted in the appendix A.2, but the following images show some examples of the layout.



Mission time elapsed T–3 days 21h:19m:20s
Link data volume 20.7 kB (314 CMD) 1.4 MB (350 CMD)
Connection state Disconnected
Next pass now (14/3/2016 13:38:22)

PARAM.	VALUE	REMARKS
LAST STATE UPDATE	16/4/2016 20:51:13 (1 month(s) ago)	State needs to be updated
SPACECRAFT TIME	23/5/2016 16:40:49	N/A
LOCAL (SERVER) TIME	23/5/2016 16:40:39	N/A
TIME DELTA	+ 10 second(s)	Synchronized
PENDING FILES	13 file(s)	N/A
BATTERY STATE OF CHARGE	86%	N/A

EPS STATUS REPORT	Max.	Min.	Max. (mission)	Min. (mission)	
BATT. SOC	86	86	89	86	
BATT. VOLTAGE	8150	8150	8025	7950	
— RESTARTS	WDT	MCLR	BOR	INST	Unknown
COUNTERS	0	0	0	0	0
LAST CAUSE	NORMAL_POWER_UP			BOOT COUNT	5
	Started	Completed			Failures
INIT. SEQUENCE	5	5		BEACON POL	0

SENSOR ID		0	1	2	3	4	5	6
TEMPERATURE	ºC	102.4	102.4	102.4	102.4	102.4	102.4	-249.9
VOLTAGE	V	8.300	1.000	8.400	8.400	8.400	0.000	8.100
CURRENT	mA	7	28	7	7	7	7	
POWER	mW	58.10	28.00	58.80	58.80	58.80	0.00	
IRRADIANCE	W/m²	0	0	0	0	0	0	

— AXIS		x	y	z
GYROSCOPE	deg	0.000	180.000	-90.000
MAGNETO.	µT	31000.0	-31000.0	0.0
ACCELER.	m/s²	1.0	2.0	3.0

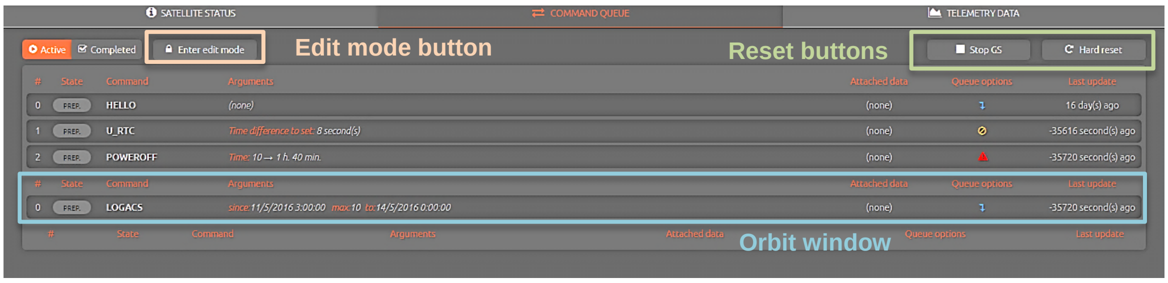
BEACON ID.	LAST DECODED MESSAGE	DATA	UPDATE TIME
MAIN	E3CTAT295E087	Temp: 295 K; — Battery: 87 %	9 month(s) ago
PELTIER	UPC123	T. Sensor: 123	9 month(s) ago

Figure 4.1 Index page (satellite status)

The telecommands queue has been divided in two lists: active and completed. It is useful to separate them to have a clearer vision of both types. The first one is a dynamic list based on multiple queues, one for each visibility interval stored on the database. The list is blocked during the communication process by the packet manager, but can be edited by the user if the back-end module has released it (the “edit mode” state is available). Active commands have multiple editable options. Their position on the queue can be modified (up or down) or they can be deleted from the list as well. Each command can be inserted in a given satellite pass (i.e. GS visibility window determined by the satellite orbit). In addition, the “queue option”, which is referred to the action to perform when there is an erroneous answer or the communication is failed, can be chosen between three options: move the command to the next window in first position, discard it and continue or stop all queues. Command parameters cannot be changed. Instead, the command has to be deleted and replaced by a new one.

For the completed commands, different information is displayed: given that they cannot be modified (because they have already been processed by the system), there are some buttons to download, or display in the case of sensor logs, the received attached files data.

In the queue page, there are two important buttons that have been added too: one for a soft reset, to tell the back-end software not to process any command until a user instruction, which is replaced by a restart button when the previous is triggered, and another to execute a hard reset, which consists on killing the process, restarting the whole communication module and start it again.



#	State	Command	Arguments	Attached data	Queue options	Last update
0	PREP.	HELLO	(none)	(none)	↴	16 day(s) ago
1	PREP.	U_RTC	Time difference to set: 8 second(s)	(none)	⊘	-35616 second(s) ago
2	PREP.	POWEROFF	Time: 10 → 1 h. 40 min.	(none)	▲	-35720 second(s) ago

#	State	Command	Arguments	Attached data	Queue options	Last update
0	PREP.	LOGACS	since: 11/5/2016 3:00:00 max: 10 to: 14/5/2016 0:00:00	(none)	↴	-35720 second(s) ago

#	State	Command	Arguments	Attached data	Queue options	Last update

Figure 4.2 Commands queue page

Finally, the telemetry page includes tables and areas that plot sensor values in the time axis and some registers of beacon, system (satellite) and backend errors messages. Some filters have been added for a proper visualization and the charts have some zoom and slide features, besides an ID selector for each data set. All the graphs can be also downloaded as an image, on a PNG file, or CSV.

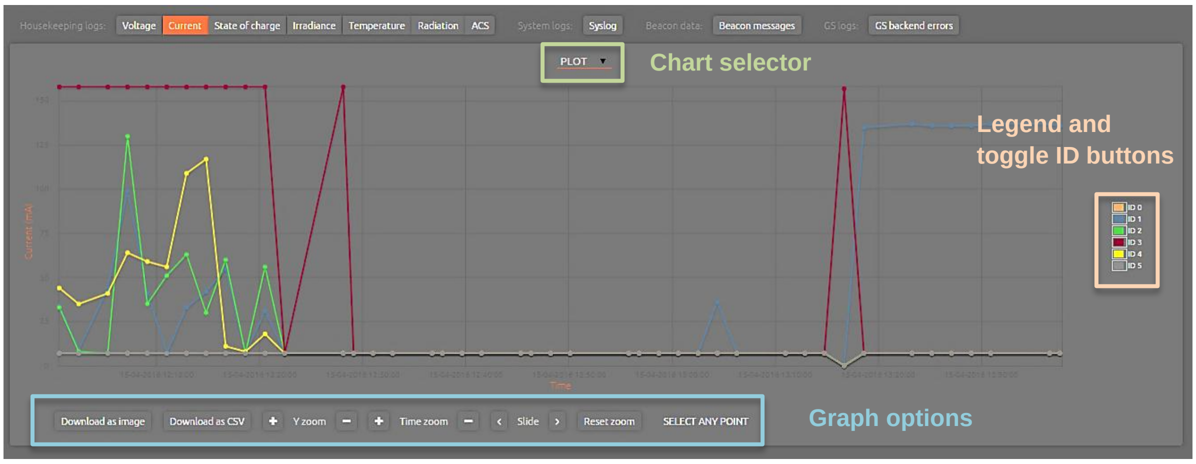


Figure 4.3 Telemetry page

4.2.4. Front operation

In order to control this interface, there are multiple functions that are required. They require synchronization between them and have been designed minimize to load and maximize browser performance, therefore maintaining user experience regardless of the number of displayed features.

The status page does not require interaction from the user. A periodic function call updates the whole data and the clock times.

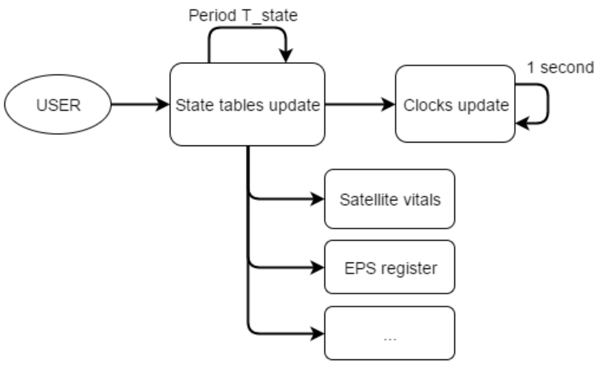


Figure 4.4 State diagram

The queue page, on the other hand, has multiple options, a dynamic layout and allows the user to interact with the commands list, besides some critical buttons. It controls the edit mode for active commands, checking whether the queue is blocked by the GS back-end or not, and displays all the information for each one. For the completed commands queue, the most important part is related to the attached data that is received from the satellite. The buttons are created every time the queue is updated and they must have a suitable reference parameter to download the appropriate file or redirect the interface to the telemetry page to display the correct data. But there is an especially critical function that is related to the GS reset. When the user clicks on the “soft-reset” button, the only change is made on a parameter on the configuration table from the server database, but the “hard-reset” button implies to kill the communication process and it is recommended to avoid it unless it is strictly necessary. In any case, the back-end software is responsible of the secure reset.

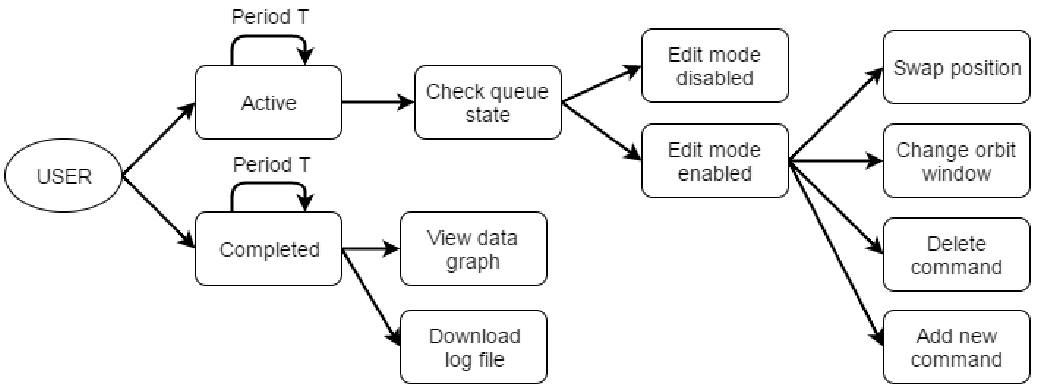


Figure 4.5 Queue diagram

The last part, and maybe the most complex one, is the telemetry page. On the graphical design chapter, the general layout has been shown (figure 4.3) but there are multiple functions that have been implemented for its proper performance and increase its complexity larger than the previous pages. The graphic charts can be modified by the user, with options that include zooming for both axis, a data slide on time, a zoom option with mouse selection and data download buttons, as a PNG image or a CSV file. Another interesting feature on the graphs is the option of choosing what data sets the user wants to view, helpful when there are many lines on the chart. The system, beacons and GS errors logs, as well as the rest of types, include tables where the downloaded data is represented in time descending order, with filters by start and end time (or data type in case of the satellite system logs).

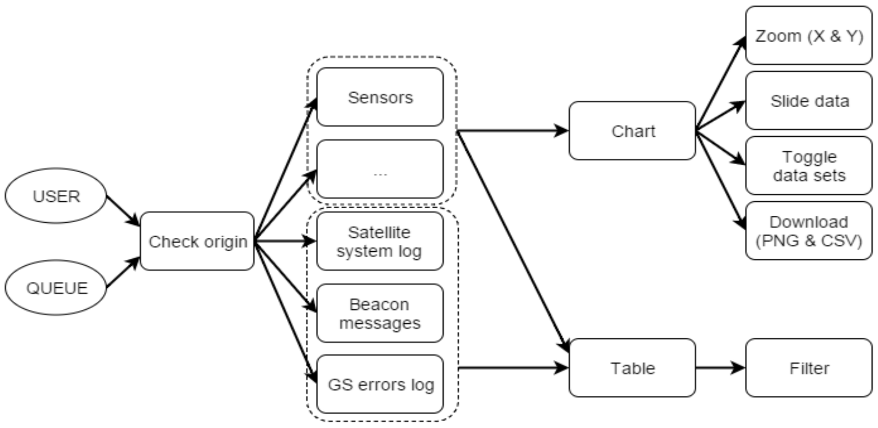


Figure 4.6 Telemetry diagram

4.3. Databases and server side

This part is referred to all the code that is executed by the GS. On the previous chapter, the user side was described, but every action that is carried out has necessarily a reaction on the server. For example, when the user adds a command to the list, the server has to include it on the database and update its state.

4.3.1. Database structure

All the information about sensors, telecommands, orbit passes or the satellite status is stored in a MySQL database, which allows an easy data download of ordered or filtered data. This is an important feature for the project purpose. Any information received from the satellite is stored in a dedicated table and, in addition, others are created to manage various aspects from the interface, the GS itself or other modules.

Table 4.1 Data model design

Type of table	Columns
Telecommands queue	- packID: command unique identifier - orbitID: orbit window identifier - cmd: command name - state: state of the command (only in GS-1) - n/t_parts: received/total files (only in GS-1) - argX: arguments (only in GS-1) - attached_xxx: RX/TX data path (only in GS-1) - xxx_path: satellite/GS path (only in GS-2) - type: Uplink/Downlink (only in GS-2) - time: timestamp of last update
Sensor values	- time: measure time - sensorID: sensor identifier - value: measure value
Satellite and GS parameters	- param: name of the parameter - value: value of the parameter

Reports and messages	- time: message generation time - message: information
Users	- userID: unique identifier - login: user name - pass: encrypted password - cookie: encrypted cookie - valid: validity period

As a convention, every time field introduced on the database must be on UNIX format or, in case it is a timestamp, referred to the UTC standard. At the user side, this has to be taken into account to display them with the local time offset if it is wanted.

4.3.2. Server performance

The website can be run using an open-source HTTP web server called Apache (on its 2.0 release). Moreover, the 5.5.9 version of the PHP interpreter will be used to handle databases and dynamic content. The GS domain is hosted on the UPC servers and, nowadays, its access is only available from the university network for security reasons. In future revisions, a VPN remote access or other technologies can be considered.

PHP functions and scripts are organised into multiple files, which can be divided in some groups: queue management, telemetry and information download, back-end management and user login (functionality is described on the next chapter). There is a strong relationship between these files and the front-end pages because, thanks to the AJAX calls, the information can be uploaded for each single purpose using a comprehensible structure named JSON. From it, the front-end functions can extract and display data on a user readable way, method that makes the program easy to understand.

4.3.3. GS back-end software

The GS back-end is composed of a set of programs and scripts which run in the GS server machine and are decoupled from the GUI. These programs handle packet generation and decoding and interface the hardware components of the GS (comms. module driver, rotors, etc.). Because of that, this set of programs is hardware-specific and its implementation will differ depending on the mission and communications protocol definition. While the protocols and data structure definition are out of the scope of this thesis, the treatment of this data and the implementation of ^{3}Cat-2's packet manager will be addressed in the following chapters. For example, the data obtained from the sensors has to be stored in a different way than the scientific data files, as they will need other structure to be properly displayed in the GUI. Even there is a strong dependence with the data it is working with, the proposed modular design decouples this from the rest of the GS architecture.

The back-end structure includes the following modules:

- a packet manager and data handler,
- the communications software module,
- an orbit propagator, and
- a beacon signal decoder

Only the last two modules have been designed and implemented during this thesis and will be described in future chapters.

4.4. User authentication and security

For a website-based GUI, security is an essential aspect to prevent the server from undesired intrusions or attacks, even more when the managed data is relative to a satellite mission and its success could be truncated because of it. That is an important reason why the interface access has been strongly restricted (only from the university network) despite the used technologies could allow a remote access. On the other hand, the remote use is a clear advantage that has been considered and, maybe in further revisions, will be added using a VPN access or other methods, always without compromising its security.

In any case, only known users have to be able to access the interface and, for this purpose, a user authentication method is required, even inside a secure network. To accomplish it, some extra functions have been added to both front and back files, which allow the interface to identify the user with a simple login method based on encrypted passwords. To prevent a constant validation, cookies are used to save the session for a defined time interval (currently set in 30 minutes).

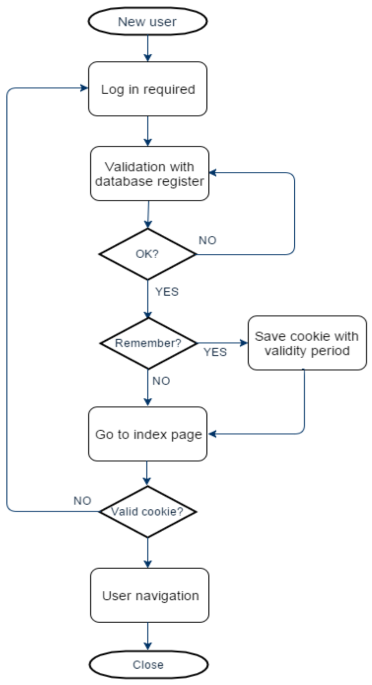


Figure 4.7 User authentication flowchart

5. Orbit propagator and satellite tracking

Once the communication protocols are defined, it is necessary to know when the GS must switch to transmission and reception mode. While there is no satellite visibility, the GS back-end process remains slept and only awakes to check the next visibility interval. The program knows it by looking to the *orbit* table from the database, where there are stored the intervals for the following passes. To update it, a second program has been developed during this thesis and the most important features are described on the next chapters.

5.1. The SGP4 model

In order to compute an interval of visibility, the program predicts the satellite position (i.e. "propagates" its orbit trajectory) until a number of passes is detected. The given number of passes to detect is left as a configurable parameter. At every propagated position, a function determines whether the satellite is in the Field Of View (FOV) of the GS antenna. Its implementation is explained on the next chapter.

The orbit propagation is implemented using a publicly available C library [14] based on the Simplified General Perturbations (SGP) and Simplified Deep Space Perturbations (SDP) models. The SDP is not used as ^{3}Cat satellites are on LEO orbit, which means that the orbit height is under 2000 km over the Earth's surface, but it is also considered in the library. Both models were developed by the NORAD in the 1970s and first programmed in FORTAN in the 1980s. SGP4 and SDP4, the real models used in the program, are a modification from the original implemented in 1988 to handle the larger number of objects in orbit and the error is ~1 km at the Two-line elements set (TLE) epoch. The TLE is a data format that encodes a list of orbital elements of an Earth-orbiting object for a given time, the epoch (https://www.celestrak.com/NORAD/documentation/tle-fmt.asp). The library theoretical bases and the used codes can be found in [15].

To obtain a prediction, the program initializes a structure with all the orbit parameters extracted from the satellite TLE (downloaded from CelesTrack webpage[16]). Then, a function named *satpos_xyz* is called with the orbit "struct" and the time to compute the position in Julian date as arguments. It can give as outputs the position and the velocity at this time.

The accuracy of the predictions has been measured comparing multiple results with industry-renowned mission analysis tools (i.e. Agi's Systems Tool Kit, STK[17]), based on the same SGP4 model. These results are presented in section 8.2.

5.2. Ground Station field of view

When the satellite position is computed, the program must check if it is on the GS FOV or not, i. e. if there is visibility on the specified time. On figure 4.1, there is a simple sketch that illustrates how this is computed. Basically, it is considered that the satellite is on FOV if the angle between the satellite and the normal vector on the Earth surface at the GS position, θ, is less than 90º (under 85º considered on the real function to ensure visibility, that could be affected by multiple causes).

This angle is computed as it follows:

$$\theta = arcos\left(\hat{n} \cdot \frac{\overrightarrow{GS-Sat}}{\|\overrightarrow{GS-Sat}\|}\right) \tag{1}$$

where the vectors are computed:

$$\hat{n} = \frac{\overrightarrow{GS}}{\|\overrightarrow{GS}\|} \quad , \tag{2}$$

$$\overrightarrow{Sat - GS} = \overrightarrow{Sat} - \overrightarrow{GS} \, . \tag{3}$$

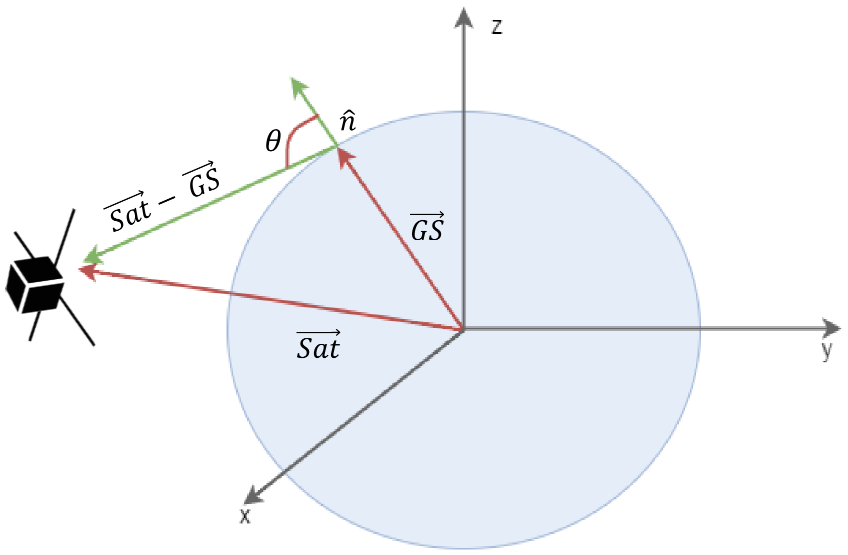


Figure 5.1 Field of view sketch

There is another important fact that has to be taken into account. All the vectors involved in the previous equations must be defined on the same coordinate system. In this case, a Cartesian coordinate system has been used, which considers the Earth rotation: the Earth-centered inertial (ECI) system. ECI has been chosen given that it is the same coordinate system used to represent satellite position in the SGDP4 libraries. On the other hand, it is easier to know the GS position with another coordinates system, based on its latitude, longitude and height, the geodetic coordinates or LLH. Equations (4) and (5), extracted from the CelesTrack columns called Satellite Times [14], relate both systems and they have been used to convert the GS coordinates to the ECI system before computing the θ angle.

Latitude (φ) to ECI:

$$z = R_e \cdot sin\varphi \quad , \tag{4}$$

$$R = R_e \cdot cos\varphi \, , \tag{5}$$

where R_e is the Earth radius on the GS position (~6371 km).

Longitude (λ_e) to ECI:

$$\theta(t) = \theta_g(t) + \lambda_e \quad , \tag{6}$$

$$x(t) = R \cdot cos\theta(t) \quad , \tag{7}$$

$$y(t) = R \cdot sin\theta(t) \quad , \tag{8}$$

where θ_g(t) is the Greenwich Meridian rotation depending on the time, as the Earth is rotating on the XY plane.

The next equations are used to obtain the previous value, extracted from the page 50 of the Explanatory Supplement to the Astronomical Almanac [18]:

$$\theta_g(\Delta t) = \theta_g(0^h) + \omega_e \cdot \Delta t \quad , \tag{9}$$

$$\theta_g(0^h) = 24110.54841 + 8640184.812866 \cdot T_u + 0.093104 \cdot {T_u}^2 - 6.2 \cdot 10^{-6} \cdot {T_u}^3 \; [sec], \quad (10)$$

$$T_u = d_u/36525 \;, \quad (11)$$

where d_u is the (integer) number of days elapsed since JD 2451545.0 (2000 January 1, 12h UT1), $\theta_g(0^h)$ is the Greenwich Meridian rotation at midnight, Δt is the amount of seconds elapsed since 12 PM and ω_e is the Earth's rotation rate (7.29211510 × 10-5 rad/s).

5.3. Sunset and sunrise algorithm

This is the last part and it is not strictly necessary on the orbit propagation process. The team decided to establish communications just during the day due to power limitations. The solar panels can only charge the batteries when the Sun is lighting them and, during the communication process, a lot of power is consumed. To prevent from spending lots of energy from the batteries, which could cause a lower lifetime, any link is avoided at night. Then, another function was added to the program to discard those visibility intervals that are occurring after the sunset. Only if the start time is before the sunset, the communication process will be initiated.

The algorithm to compute the sunset and sunrise time can be found, for example, on the Ed Williams' Aviation page [19][20] and its description is in the appendix B.1.2. In order to measure the predictions accuracy, many calculus have been made and compared with the data found on some Internet pages like Time and Date sunrise and sunset calculator page [20]. The accuracy of the algorithm has been considered with no error, as the results are exactly the same as the found in the mentioned page.

6. Morse beacon

A beacon signal is a periodic message with general information about critical state values, for example, temperature or the batteries state of charge. On ^{3}Cat-1, this is just the information included in the beacon. ^{3}Cat-2 has also his own beacon signal, but this chapter is only focused on the first one, because it is the one implemented in the frame of this project and they are essentially different (modulation, coding and information contained on them).

6.1. Beacon signal

^{3}Cat-1 beacon is Morse-coded, a simple and widely known code. The main reason to use it is the amateur band license. In Spain, the spectrum from 435 to 438 MHz is free to the satellites communications with high restrictions on their use. One of such restriction is that any user should be able to receive and decode the signal easily. This beacon signal is modulated in Amplitude Shift Keying (ASK) modulation and the carrier frequency is 437.25 MHz.

The basic unit of a Morse code is a dot or "dit" but there are other units like the dash (or "dah") and silences, long and short. These are the basic rules:

- a dash is three units (dots) ,
- the space between parts of the same letter is one unit ,
- the space between letters is three units and
- the space between words is seven units

Table 6.1 shows the message sent in the beacon signal. There are two distinguishable parts: the fix structure (header, 'T' and 'E' letters) and the variable numbers, depending on the temperature in Kelvin degrees and state of charge in %.

Table 6.1 Morse beacon message

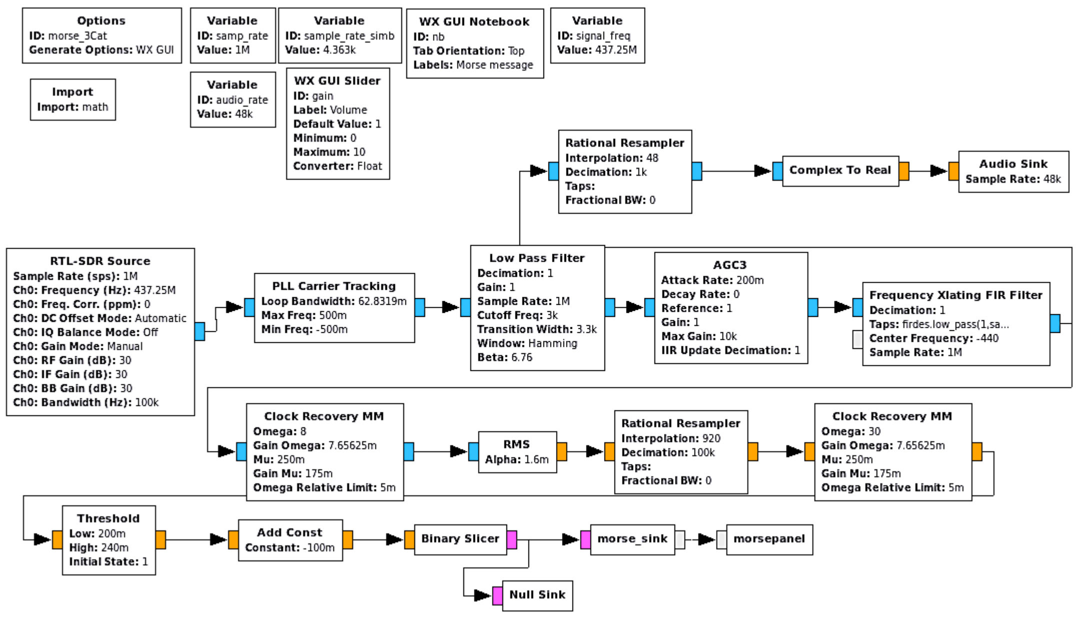
Message	Temperature	State of charge
EC3CTAT###E###	Any value	000-100, 255

For the temperature values, any case is considered as possible, even knowing that the most common will be between 250 to 350 ºK. The state of charge can only have values in the 0-100 range because it is represented in %, but there is another possibility: when the batteries are disconnected and all the consumption is supplied by another source (solar panels on the space or a power supply at the laboratory), this parameter takes the maximum value of an integer byte (255).

6.2. Reception and decoding

The first part to design is the signal reception. Depending on the input, a different signal processing method is needed. At the beginning, the beacon signal was decided to be entered as an audio signal, once it has been filtered and converted with a radio reception device. Some tests revealed a low SNR signal, which decreases the success rate on the message decoding. In order to improve it, the existing receiver, with an audio output, was replaced by a software defined radio (SDR) device (based on the RTL2832U chipset) directly connected to the antenna and to use SDR tools as well for the signal processing.

GNURadio is an open-source program that allows the user to process any digital signal with pre-defined modules or custom processing blocks. For this part, a previously designed program, developed at the NanoSat Lab, has been reused with some changes to filter the input signal and decode the Morse message. In addition, a new module has been developed to save the message into the database, in a MySQL table specifically prepared for that purpose. Figure 6.1 shows the program structure.


Options ID: morse_3Cat Generate Options: WX GUI

Variable ID: samp_rate Value: 1M

Variable ID: sample_rate_simb Value: 4.363k

WX GUI Notebook ID: nb Tab Orientation: Top Labels: Morse message

Variable ID: signal_freq Value: 437.25M

Import Import: math

Variable ID: audio_rate Value: 48k

WX GUI Slider ID: gain Label: Volume Default Value: 1 Minimum: 0 Maximum: 10 Converter: Float

Rational Resampler Interpolation: 48 Decimation: 1k Taps: Fractional BW: 0

Complex To Real

Audio Sink Sample Rate: 48k

RTL-SDR Source Sample Rate (sps): 1M Ch0: Frequency (Hz): 437.25M Ch0: Freq. Corr. (ppm): 0 Ch0: DC Offset Mode: Automatic Ch0: IQ Balance Mode: Off Ch0: Gain Mode: Manual Ch0: RF Gain (dB): 30 Ch0: IF Gain (dB): 30 Ch0: BB Gain (dB): 30 Ch0: Bandwidth (Hz): 100k

PLL Carrier Tracking Loop Bandwidth: 62.8319m Max Freq: 500m Min Freq: -500m

Low Pass Filter Decimation: 1 Gain: 1 Sample Rate: 1M Cutoff Freq: 3k Transition Width: 3.3k Window: Hamming Beta: 6.76

AGC3 Attack Rate: 200m Decay Rate: 0 Reference: 1 Gain: 1 Max Gain: 10k IIR Update Decimation: 1

Frequency Xlating FIR Filter Decimation: 1 Taps: firdes.low_pass(1,sa... Center Frequency: -440 Sample Rate: 1M

Clock Recovery MM Omega: 8 Gain Omega: 7.65625m Mu: 250m Gain Mu: 175m Omega Relative Limit: 5m

RMS Alpha: 1.6m

Rational Resampler Interpolation: 920 Decimation: 100k Taps: Fractional BW: 0

Clock Recovery MM Omega: 30 Gain Omega: 7.65625m Mu: 250m Gain Mu: 175m Omega Relative Limit: 5m

Threshold Low: 200m High: 240m Initial State: 1

Add Const Constant: -100m

Binary Slicer

morse_sink

morsepanel

Null Sink

Figure 6.1 Morse receiver and decoder program

The input signal is filtered and its power is increased thanks to an auto-gain module (AGC3). Then, the root mean square (RMS) is computed and a resampler reduces the number of samples per second, from 10^6 samp/s, corresponding to the RTL sampling rate, to 9200 samp/s. The threshold module converts the samples under a certain value to 0 and, over it, to 1. When the signal is binary, the slicer module passes each bit to the *morse_sink*, which detects dots and dashes and decodes them to obtain readable characters. After decoding each letter, it is passed to the *morse_mysql* module that is responsible to save them into the database once a complete message is received. As the message structure is previously known, some "intelligence" has been added to detect and solve some errors.

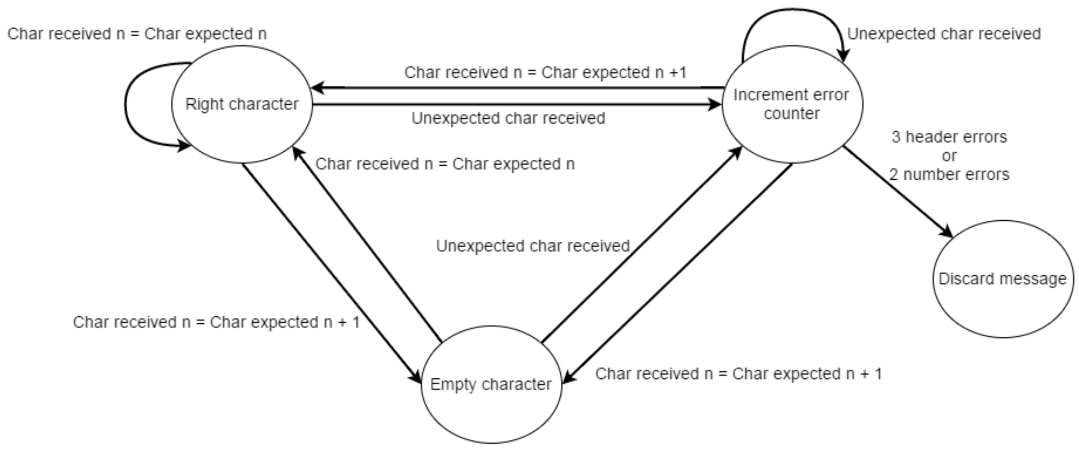

Char received n = Char expected n
Right character
Char received n = Char expected n +1
Unexpected char received
Unexpected char received
Increment error counter
3 header errors
or
2 number errors
Char received n = Char expected n
Unexpected char received
Discard message
Char received n = Char expected n + 1
Empty character
Char received n = Char expected n + 1

Figure 6.2 State diagram of the *morse_mysql* module

A character is considered as empty when the received letter does not correspond to the expected one, but the next. An error is expressed as the '#' character on a decoded message. This is the criterion:

- for the header, the character is different than the expected ,
- it is different to the next character in the structure ,
- a number from the temperature value is a letter ,
- a number from the state of charge value is a letter and
- the first number of the state of charge is different from 0, 1 or 2

To detect the message start, there is a timer set on the previous character reception. This time is compared to the current date and, after a certain delay, the received character is considered as the first of a new message. Given that the variable part is made of numbers, the longest message is "EC3CTAT000E000" ('0' is equivalent to "-----", five dashes, in Morse code). The time of a basic unit, a dot, is set as 100 ms in [3]Cat-1, so the longest message duration, counting silences, is 20.1 seconds. The sending period is 1 or 3 minutes, depending on the state of charge of the batteries. After doing these calculations, the threshold time has been set in 20 seconds, which has enough time margin between the longest message end and the next message start.

7. Framework implementation

Having presented the generic software framework design, this chapter describes the implementation of this framework both for the 3Cat-1 and the 3Cat-2 Ground Stations. Each one has its own features, but their architecture and functionality are similar due to the modular and reusable design. Then, after talking about the common parts in a first section, there are two dedicated to each particular implementation, focused on the most important differences

7.1. Common features

7.1.1. Client-side front-end

On the next chapter, the main functions used on each page will be described. For more information, a larger and detailed list of them is in the appendix X, which includes their input and output parameters and a brief explanation of their performance.

7.1.1.1. State

The satellite status functions are included on the *state.js* file. State tables are divided depending on their content, but the whole information is periodically updated with the same function: *state_update*. Beacon messages are updated through a secondary call inside this function to *beacons_update* and, moreover, the server and satellite times are updated simulating a real-time clock with another call every second to *times_periodic*.

7.1.1.2. Queue

All the functions used in the queue page are defined in the queue.js JavaScript file. There are three main functions that manage the behaviour of the active queue, which show the pending commands to be sent: *load_active*, *add* and *queue_op*. The rest are related to one (or more) of them or they are used to improve the user experience implementing extra functionalities. With *load_active*, the webpage shows the stored list of active commands. It includes the commands name, position on the queue, arguments, size of the sent data, and the time of the last update. On the same list, when the “edit mode” is enabled, two extra columns are displayed: “Move position”, to change the commands order in a queue or delete it, and “Orbit window”, which allows the user to change the window where this command will be sent.

For the completed list, *load_completed* does the same as *load_active* with a significant change: the user can download or view the information with different buttons on the attached data column. On the other hand, *add* and *queue_op* functions call a PHP file used to move, add or delete commands from the list and change the orbit window for a single command too.

7.1.1.3. Telemetry

The third part, telemetry, can be split in two: tables and charts, regardless of all the functions being defined in telemetry.js. Users can choose the type of information to be presented, but never both at the same time. In any case, the process behind them is very similar. One function, *table_download* or *chart_download*, is called when a telemetry option is chosen and it calls, using AJAX, a PHP file to download the required information. Data will be shown differently depending on the chosen option. There are two display functions: *chart_plot* to draw a graph and *tabs_update* to create a new table.

Finally, there is some extra information that is updated on all pages, but has not been mentioned. A file called *common.js* includes the header information update and, depending on the satellite visibility, it modifies the update period of the previous data (commands queue, state tables or telemetry data).

7.1.2. Server-side files

This part includes the PHP files used for the front-end data management and the rest of the software used in this GS framework. As the hardware modules and the GS performance are quite different, the only module shared by them is the GUI server-side, i.e. the PHP files. In table 7.1, there is a list of all them with a brief description. Their content will change on each implementation but the structure and performance are the same.

Table 7.1 PHP files

PHP file	Description
backend_reset	Executes a hard reset of the *gs_backend*
chart_download	Downloads telemetry data using a suitable structure for chart drawing
connection_state	Data for the header information update
login_functions	Functions set for user login
manage_command	Adds, moves or deletes commands from the queue
manage_queue	Sets or gets queue parameters
queue_list	Downloads the commands queue
state_update	Downloads the status tables information
telemetry_download	Downloads telemetry data
user_login	Session check and user register or login

7.2. ^{3}Cat-1 Ground Station

7.2.1. Architecture

The GS-1 design is shown in figure 3.1, with blue blocks for the software modules and green blocks for hardware devices. The first idea was to allocate all the software in the Rapsberry Pi SBC and use the computer, with Windows O.S., just to run the mission control and operations software (i.e. Orbitron suite), which also controls the antenna rotors. This level of integration has been unreachable due to the Raspberry Pi limitations and a second solution has been decided. The main software will be located in a computer with Linux O.S. (Lubuntu 16.04) and only the modules required for the communications drivers module (COMMS HWmod or just HWmod) will be kept in the SBC. This decision is based on the high computational requirements of the web server. It was really hard for the Raspberry Pi to manage all the back-end processes and this caused multiple server failures. With this split design, the communication between modules is more complex but allows a faster and a robust execution. On the other hand, it allows the use of open-source satellite tracker, like Gpredict.

It is important to mention that the lowest part of the GS-1 software, corresponding to the communications module and the back-end had previously been developed by members of the software team at the NanoSat Lab (Carles Araguz and Marc Marí) who, given their involvement in the development fo ^{3}Cat-1's on-board software, also contributed during the verification and test campaign of the GS framework.

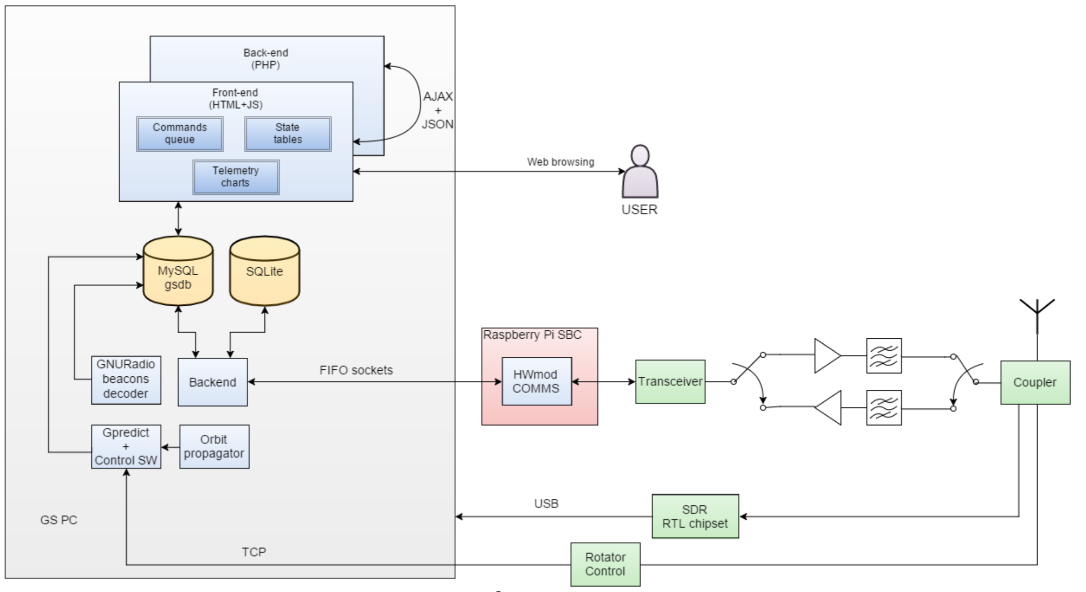


Figure 7.1 ^{3}Cat-1 GS diagram

An SPI connector is used to send and receive the data signals in the Raspberry Pi and an RTL device is connected to the antennas to receive beacon data. The SPI is directly controlled by the COMMS HWmod, which exchanges information with the *gs_backend* software through some FIFO pipes. Then, the GUI back-end is responsible of managing the "gsdb" database where the information is stored in once it is unpacked. On the user side, the front-end takes it to be properly shown.

7.2.2. Client-side front-end

This part is very similar in both cases because its operation does not depend on the managed data. It will only affect in minor changes, mostly graphical like titles or input data on the forms.

Table 7.2 Telecommands states

State	Description	GUI view
Preparing	Value by default. In this state, the packet can be modified.	PREP.
Ready	The command has been packed.	READY
Sending	Trying to send it to the satellite.	SENDING
Sent	There is confirmation that the packet has been sent.	WAITING
Answered	The packet answer has been received.	UNPACK.
Completed	The answer has been processed and the next command can be send.	No image

However, there is an important feature only included on the ^{3}Cat-1 GUI: the state of the active command, with a colour code detailed in table 7.2. It is included as a column in the queue and is updated by the *gs_backend* when a new stage has been completed during the transmission process. It helps the user to understand what is currently happening during the satellite pass.

7.2.3. Server-side and back-end

All the sensors and other data shown in the client-side are received, processed and stored by three modules, namely:

1. the COMMS HWmod, which is responsible of the packet sending and reception;
2. the *gs_backend,* which transforms the commands into packets (or the reverse) and stores the obtained data properly in the database; and
3. the website processes at the server side, which send the required information from the to the client.

The specific tables used on the GS-1 database, called *gsdb*, are in table 7.3. Most of them will not be used for the second implementation, because the stored data is completely different and new tables will be created.

Table 7.3 ^{3}Cat-1 database (gsdb)

Table	**Content**
ads	Magnetometer, accelerometer, gyroscope and IGRF values.
beacon_morse	Morse beacon messages
beacon_peltier*	Peltier beacon messages
current	Current sensors values
eps_status_report	EPS reports (batteries consumption, resets and *initseq*)
error	Server side errors
irradiance	Irradiance sensors values
orbit	Time intervals with FOV
queue	Commands list
radiation	Radiation sensors values
sat_state	Times and pending files
soc	Batteries SOC register
state	GS status (visibility, queue param., etc.)
syslog	Syslog messages register
temperature	Temperature sensors values
users	Users register
voltage	Voltage sensors values

Finally, the last part to mention is related to the orbit propagator and the beacon decoder. The first one must be periodically called by the *gs_backend* to update the orbit table and the second one is constantly running and uses the RTL's data to obtain possible Morse messages from the satellite.

7.3. ^{3}Cat-2 Ground Station

7.3.1. Architecture

The entire software of GS-2 is implemented in a computer, with a Linux O.S. (Ubuntu 14.04). While parts of the GS backend are still under development, the main structure of the proposed design has been implemented successfully. A *daemon* process acts as the GS back-end, performing packet management and has been developed by Joan Frances Muñoz, another member of the NanoSat Lab. To interact with it, there are two functions for transmission and reception. They need the command to be sent or received as the input and some other extra parameters. Figure 7.2 shows the complete diagram of the ^{3}Cat-2 GS. The most important difference with GS-1 is the two SDR's devices (namely Universal Software Radio Peripherals or USRPs) used as communications hardware, one for the UHF uplink and VHF downlink and the other only for downlink, at S band.

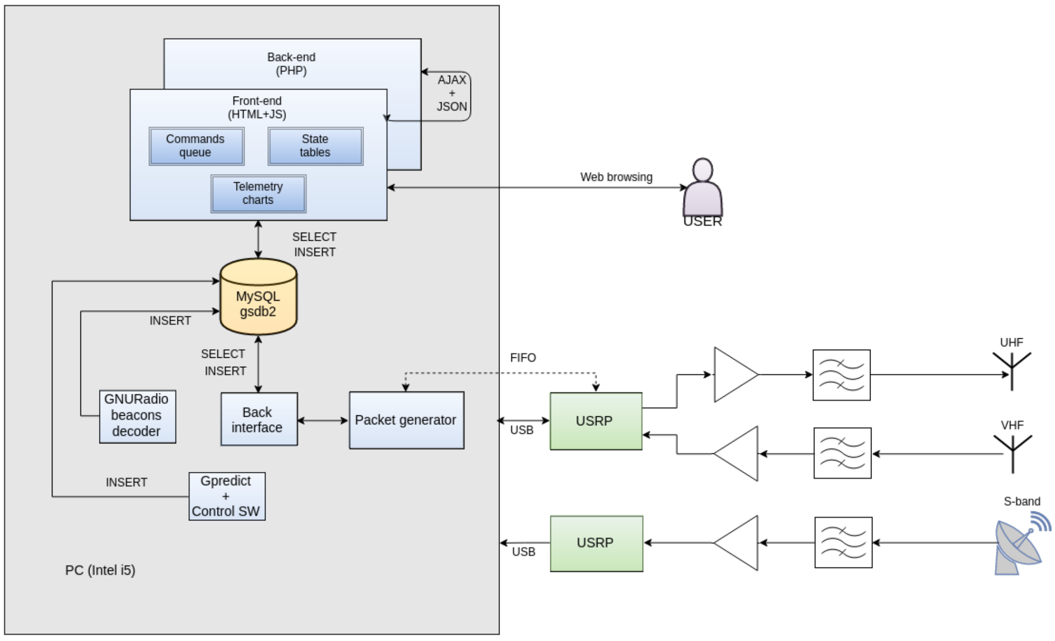


Figure 7.2 ^{3}Cat-2 GS diagram

It is important to emphasize that, since the project started, the team had in mind the target of making a modular web design to use it on both GS-1 and GS-2. Even they were initially based on dissimilar technologies, a website-based interface allows its reusability on multiple platforms. Then, apart from the GS back-end, the rest of software files are exactly the same.

7.3.2. Server-side front-end

On the user side, there are not any relevant changes relative to the interface operation. The most important changes are related to the content that has to be displayed. Both satellites have completely different commands and modules, which generate new files that can be downloaded.

GS-2 has three communication links, one for each antenna: VHF and S band for downlink and UHF for uplink. Commands are not only differentiated by their content. When the user is going to introduce a new command to the list, the link direction is required, as well as the command name (they are not the same for uplink or downlink) and the execution time.

These are the different command types, according to the used link:

- Downlink VHF
- Downlink S-band
- Uplink UHF (with ACK)
- Uplink UHF (without ACK)

The status and telemetry information is different too. For ^{3}Cat-2 it is not differentiated by their sensors origin, but for the subsystem that generates the information. It implies another database structure too. In order to keep this distinction on the GUI, these are the new telemetry classes:

- ADCS sensors
- COMMS antennas
- COMMS TXS (flags and consumption)
- EPS housekeeping
- EPS Vboost (amplifiers)
- EPS current
- EPS temperature
- System logs
- Beacon messages
- GS logs

7.3.3. Server-side and back-end

The server-side files, despite having the same names for both interfaces, manage different information and they have been implemented from scratch. In any case, the GUI performance will be very similar and the basis of the code is exactly the same. There are new database tables, corresponding to the new information shown on the previous list, but the others that include common data have been kept (*error, orbit, state* and *users*).

This thesis is mainly focused on the user interface but some of the back-end functions (in C language) have been also developed, most of them to read the downloaded log files and save the extracted data into the database. Although this feature was readily available in GS-1, it had to be developed for the GS-2. These extra functions have already been designed, implemented and tested, but they are susceptible to any modification due to changes on the files content or the team criteria. The implemented functions are detailed in the appendix X.X.

Finally, it is worth mentioning that the other two modules defined in this thesis, the orbit propagator and the Morse beacon decoder, are not necessary in the GS-2. Previously, the team decided to use Gpredict for satellite tracking. On the other hand, the beacon message is completely different and it can include some payload data, so it is impossible to reuse the current GNURadio software and other solution will be investigated.

8. Tests and results

The complete document with all the tests and the obtained results can be consulted in the Validation Plan appendix, but on this chapter there is a brief summary of the most important information.

8.1. Graphical User Interface and end-to-end tests

To test the GUI performance, with all the required devices (GS and satellite, including the communications modules and others that could be involved), the team scheduled some lists of commands. Then, the response was analysed using the downloaded data, in the corresponding files and using the information stored into the database tables.

Appendix C includes all the end-to-end tests, besides the protocol followed in each case, i.e. the Software Validation Plan.

8.2. Orbit propagator error calibration

On the orbit propagator chapter, the validation methods were already commented. They are used to measure the error range in the prediction process. Figure 8.1 includes a graphical plot with the error measured on each axis in the ECI coordinates system and the error modulus, as well, compared to the STK predictions for the FUNcube-1. Otherwise, in figure 8.2, there is the ground track of both predictions, in latitude and longitude.

Test conditions:

- Start: 2016/06/08 00:00:00
- End: 2016/06/09 00:00:00
- Delta time (between predictions): 1 second

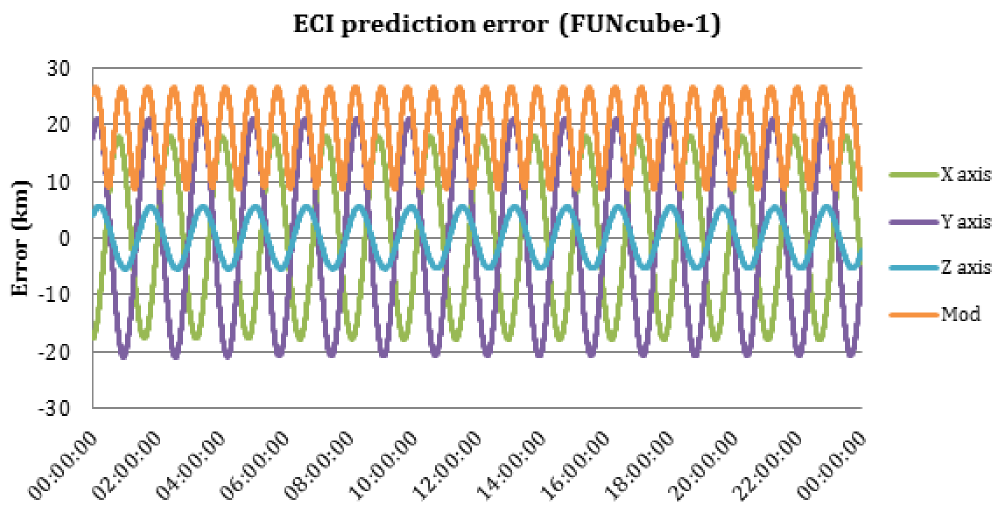


Figure 8.1 OP-STK error in ECI system

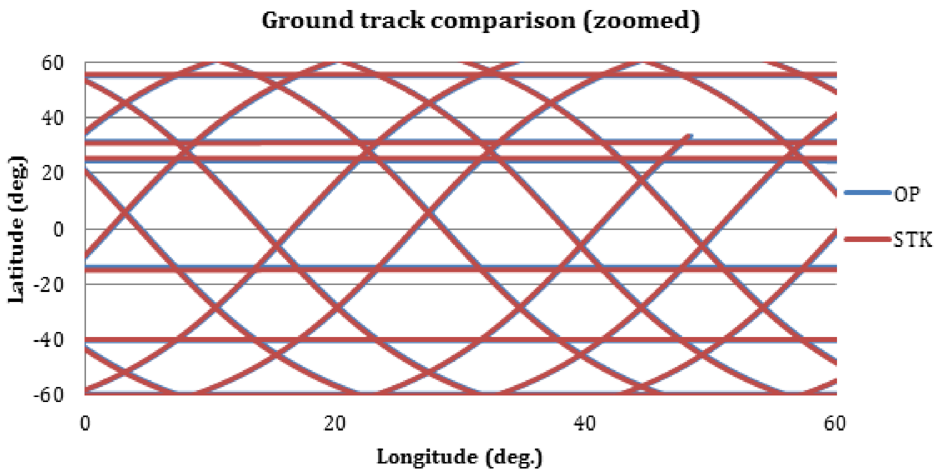


Figure 8.2 Ground track comparison (op-stk)

The function to compute visibility periods have been compared with Gpredict future passes using the ^{3}Cat-1 GS coordinates and the same satellite as before (FUNcube-1), with really encouraging results. The error between the orbit propagator and Gpredict for the AOS (Acquisition of Signal) and LOS (Loss of Signal) times is about 5-10 seconds for each case. These differences may be due to the 10-25 km errors in the position predictions made by the SGP4 library, irrelevant for this purpose because this program will not be related to the antennas' pointing and it will be only used to start the communications protocol when visibility is expected.

8.3. Morse beacon decoder tests

Given that the delivery of the ^{3}Cat-1 spacecraft was scheduled early during the development of this thesis, multiple files with raw and audio data were saved using an SDR device during the end-to-end tests and have been used to calibrate and test the Morse decoder.

As the tests conditions were not exactly the same as the real environment found during the mission, there is not absolute certainty that the used configuration will work then. In any case, the used parameters can be adjusted to the mission requirements. Tests and results can be consulted in the appendix B.2.

Table 8.1 Lab tests conditions

Parameter	Value
Dot time	100 ms
Message structure	EC3CTAT###E###
Before processing	
Signal power	[-20, -25] dBm
Noise level	[-65, -70] dBm
After processing	
Signal power	[-10, -15] dBm
Noise level	[-65, -70] dBm
Thresholds (bounded signal 0-1)	
Low	≤ 0.2 V
High	≥ 0.24 V

9. Budget

The approximate cost of the components and the personnel is shown in the following tables. There are more devices involved in the real design, but the hardware design is out of scope and most of the RF components have been omitted. Other theses more focused on this part have an accurate budget of these items. For example, it is included in *Design and implementation of the ground station for ^{3}Cat satellites* [1].

Table 9.1 Budget table

Concept	Units	Price (€/unit)	Total price (€)
Raspberry Pi B+	1	30.95	30.95
Computer (Intel i5, 500 GB HDD, 4GB RAM)	2	399	798
USRP B210	2	500	1,000
			1,828.95

Table 9.2 Personnel cost

Personnel	Weeks	Hours/week	€/hour	Total (€)
5	20	20	15	30,000

The personnel cost is computed as a junior engineer contract with part time conditions. Weeks are counted from the semester start (15/02/2016) to the thesis delivery deadline date (27/06/2016).

10. Conclusions and future development

The software framework detailed in this thesis has been used for its implementation on the two Ground Stations from the ^{3}Cat-1 and ^{3}Cat-2 project. However, this type of modular structure meets many specifications of other CubeSat projects and it is precise to remark that the main objective, the reusability, has been accomplished.

In any case, there is much more work to do now. The most interesting part, after both satellite launches, will show the real success of the mission and, more specifically to this thesis, of the GS design. In ^{3}Cat-2 mission, there are some tests already pending to be done, related to the module between the GUI and the GS back-end or the server implementation on the final computer.

Moreover, some applications are not coupled and they have only been tested without the rest of the software, for example the beacon decoder. The next step is to include them in the structure and make all work together. This is not critical because the remaining parts do not depend on any other, but they must be run with the rest without the user intervention. In addition, the software improvement is a continuous task and the final performance in real conditions will manifest some situations that maybe have not been considered and the team must solve them on time.

In conclusion, ^{3}Cat missions are far from their end, even they have overcome the hardest part. A new and exciting stage is coming and all the work done during the last months should be worth it.

Bibliography

[1] Elisabet Tremps Tor. "Design and implementation of the Ground Staiton for ^{3}Cat satellites". Degree thesis, Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona, Universitat Politècnica de Catalunya, Barcelona, Spain, 2015.

[2] J.F. Muñoz. “3Cat-2: Software Validation Plan”, September 2015. Unpublished manuscript.

[3] C. Ballesteros. “3Cat-1: Software Validation Plan”, June 2016. Unpublished manuscript.

[4] North Sky Research (NSR). Mass Challenge for CubeSats [Online] Available: https://sites.google.com/a/slu.edu/swartwout/home/cubesat-database. [Accessed: 15 May 2016].

[5] CubeSat Design Specification, revision 13. California Polytechnic State University, February 2014.

[6] M. Swartwout, PhD on Saint Louis University (Missouri, USA). CubeSat database [Online] Available: https://sites.google.com/a/slu.edu/swartwout/home/cubesat-database. [Accessed: 15 May 2016].

[7] Radio Amateur Satellite Corporation (AMSAT), EO-79 (QB50p1 and FUNcube-3). [Online] Available: http://www.amsat.org/?page_id=2944. [Accessed: 15 May 2016].

[8] NASA, Small Satellite Missions: ELaNa – Educational Launch of Nanosatellites. [Online] Available: http://www.nasa.gov/mission_pages/smallsats/elana/index.html. [Accessed: 15 May 2016].

[9] Alaska Space Grant Program (ASGP), Alaska Research CubeSat (ARC). [Online] Available: http://spacegrant.alaska.edu/content/alaska-research-cubesat-arc-selected-launch. [Accessed: 16 May 2016].

[10] International Amateur Radio Union (IARU) Regions. [Online] Available: http://www.iaru.org/regions.html. [Accessed: 16 May 2016].

[11] AMSAT-UK, FUNcube-1 Real Time Data. [Online] Available: http://warehouse.funcube.org.uk. [Accessed: 16 May 2016].

[12] AMSAT-UK, FUNcube Ground Segment. [Online] Available: https://funcube.org.uk/ground-segment/ground-station. [Accessed: 16 May 2016].

[13] Shaina A.M. Johl and E. Glenn Lightsey. "A Reusable Command and Data Handling System for University CubeSats ". Journal of Small Satellites (*JoSS), Vol. 4, No. 2, pp. 357–369*, October 2015.

[14] P. Crawford, The Dundee SGP4 code. [Online] Available: http://www.sat.dundee.ac.uk/psc/sgp4.html. [Accessed: 17 May 2016].

[15] D.A. Vallado, P. Crawford, R. Hujsak, and T.S. Kelso, "Revisiting Spacetrack Report #3," presented at the AIAA/AAS Astrodynamics Specialist Conference, Keystone, CO, 2006 August 21–24.

[16] CelesTrack, CubeSats TLE [Online] Available: http://www.celestrak.com/NORAD/elements/cubesat.txt. [Accessed: 3 June 2016].

[17] AGI, Systems Tool Kit . [Online] Available: http://www.agi.com/products/stk/. [Accessed: 3 June 2016].

[18] CelesTrack, Orbital Coordinate Systems, Parts I & II, from Satellite Times columns [Online] Available: http://www.celestrak.com/columns/. [Accessed: 3 June 2016].

[19] U.S. Naval Observaroty. *Explanatory Supplement to the Astronomical Almanac*, University Science Books, 1992.

[20] E. Williams, Sunrise/Sunset Algorithm [Online] Available: http://williams.best.vwh.net/sunrise_sunset_algorithm.htm/. [Accessed: 3 June 2016].

[21] Time and Date, Sunrise and sunset calculator, for the city of Barcelona [Online] Available: http://www.timeanddate.com/sun/spain/barcelona. [Accessed: 3 June 2016].

Appendices

A. Graphical User Interface

A.1 JavaScript functions

A.1.1. Common

Functions from common.js:

- connection_state
    - Arguments: none
    - Return value: none
    - Action:

      Downloads the satellite state information and updates the webpage header fields: mission time, link data volume, satellite visibility and next pass. It sets, as well, the update period of the page information (state, queue or telemetry). Called periodically with a setInterval function on document ready.

- sat_times_update
    - Arguments: none
    - Return value: none
    - Action:

      Updates periodically the mission and pass time, simulating a real time clock. Called by connection_state with setInterval.

A.1.2. State

Functions from state*.js:*

- state_update
    - Arguments: none
    - Return value: none
    - Action:
      Downloads the satellite state information, EPS last report, last sensor and beacons values and updates the status tables.

- server_difference
    - Arguments: none
    - Return value: none
    - Action:
      Downloads the server time and computes the difference with the local time.

- times_periodic
    - Arguments:
        - delta: Difference between satellite and server time
    - Return value: none
    - Action:

Updates periodically (due to *setInterval* call on *state_update*) the server and spacecraft time, simulating a real time clock.

.

- beacons_update
  - Arguments: none
  - Return value: none
  - Action:
    Downloads the last beacons messages from their respective tables and updates beacons tables. Called on *state_update*.

- enable_periodic_update
  - Arguments: none
  - Return value: none
  - Action:
    Calls *setInterval* to run periodically the *state_update* function every *T_state* seconds (a global parameter set on *common.js*).

A.1.3. Queue

Functions from *queue.js:*

- enable_periodic_update
  - Arguments: none
  - Return value: none
  - Action:
    Calls *setInterval* to run periodically the load*_active* or *load_completed* functions every *T* seconds (a global parameter set on *common.js*). A different function is chosen depending on the current list the user is watching to.

- load_active
  - Arguments: none
  - Return value: none
  - Action:
    Downloads the list of active commands (with a state different to completed; the whole list is on table 3.1) and displays them as a table on the active queue, including the position on the queue, the command state and name, its arguments and the size of the attached data. Depending on the back-end state, edit mode buttons are shown or the last modification time, instead.

- load_completed
  - Arguments: none
  - Return value: none
  - Action:
    Downloads the list of completed commands and displays them as a table on the completed queue. The displayed information is: time of completion, command ID, name, arguments, result of the communication procces (OK or error number) and, in case of received data, a button to download it, one for each file, with the file size. If this data is a log file, it can be displayed on the telemetry page by clicking the “view” button.

- filter
    - Arguments: none
    - Return value: none
    - Action:
      Sets the filter parameters according to user choose (type of command, interval “datepickers” or attached data only). Then, it calls the *load_completed* function, which takes into account these preferences.

- queue_op
    - Arguments:
        - op: operation to be executed (“up”, “down” or “del”)
        - cmdid: command ID
    - Return value: none
    - Action:
      This function modifies the position of a given command, moving it up or down on the list, or even deleting it. This is done by calling a PHP file responsible of managing commands on the queue and the instructions are passed by $_GET variables.

- add
    - Arguments: none
    - Return value: none
    - Action:
      The “new command form” is checked; depending on the chosen type different values are verified. If there is not any error, the form is submitted (which will do the same AJAX call as *queue_op*, changing the operation to “add”)

- backend_state
    - Arguments:
        - state: state the backend must be changed (“start” or “stop”)
    - Return value: none
    - Action:
      Changes the *stop_queue* parameter to “yes” or “no” according to the introduced argument. It is done again with an AJAX call to a PHP file, to manage the queue state on this case.

A.1.4. Telemetry

Functions from telemetry.*js:*

- enable_periodic_update
    - Arguments: none
    - Return value: none
    - Action:
      Calls *setInterval* to run periodically the *chart_download* or *table_download* functions every *T_tel* seconds (a global parameter set on *common.js*). A different function is chosen depending on the current view.

- chart_download
    - Arguments:

    - tel: type of telemetry log to display
  - Return value: none
  - Action:
    Downloads data for the chosen telemetry and calls *chart_plot* to draw it.

- chart_plot
  - Arguments:
    - tel: type of telemetry log to display
    - tel_data: data structure with the pairs of points to draw
  - Return value: none
  - Action:
    This function draws a chart using the Flot library (release 0.8.3) [8]. In addition, it sets the image download button URL and calls *table_download* for the CSV download button data.

- table_download
  - Arguments:
    - tel: type of telemetry log to display
    - tabs: option to choose whether the table has to be drawn or not
    - syslog_type: defines the type of syslog messages (if *tel* is “syslog” and optional in any case”
    - from: start date to download data
    - to: end date to download data
  - Return value: none
  - Action:
    Downloads data for the chosen telemetry and calls *tabs_update*, when the user wants to display a table, or creates the CSV file to be downloaded on the chart page.

- tabs_update
  - Arguments:
    - tel: type of telemetry log to display
    - tel_time: time set of the downloaded data
    - tel_data: values set of the downloaded data
  - Return value: none
  - Action:
    Creates a table for the chosen telemetry type.

- filter_table
  - Arguments:
    - tel: type of telemetry log to display
  - Return value: none
  - Action:

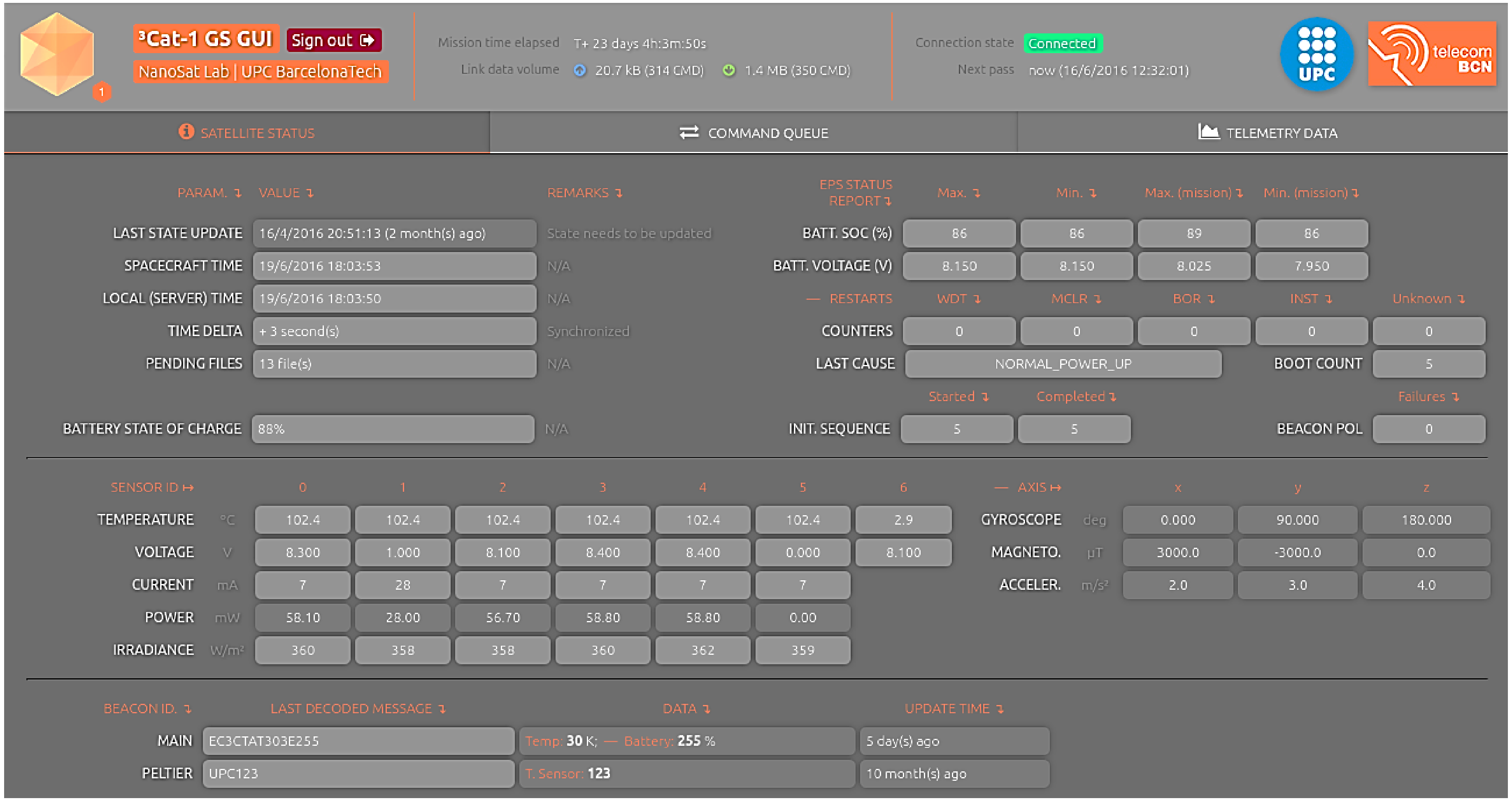
    Calls *table_download* with suitable parameters according to user choice. The table periodic update is disabled until the filter is cleared or the page is refreshed.

A.2 Operation screenshots

This appendix includes some images extracted from the GUI and represent different execution modes and options. Due to the huge number of options available, only a few will been shown (the most representative) because most of them have a similar behaviour but only changing the context: sensor types, commands on the queue…

All the screenshots that are represented in this chapter have been obtained from the ^{3}Cat-1 user interface, given that the procedures and graphical displays are exactly the same for ^{3}Cat-2. The only major change is on the content (commands, tables, etc.).

The state view has not any editable option but sensor values can be highlighted when they are abnormally high or low. The first image corresponds to a normal satellite state and the second one has some sensor alerts.



³Cat-1 GS GUI Sign out
NanoSat Lab | UPC BarcelonaTech

Mission time elapsed T+ 23 days 4h:3m:50s
Link data volume 20.7 kB (314 CMD) 1.4 MB (350 CMD)

Connection state Connected
Next pass now (16/6/2016 12:32:01)

SATELLITE STATUS | COMMAND QUEUE | TELEMETRY DATA

PARAM.	VALUE	REMARKS
LAST STATE UPDATE	16/4/2016 20:51:13 (2 month(s) ago)	State needs to be updated
SPACECRAFT TIME	19/6/2016 18:03:53	N/A
LOCAL (SERVER) TIME	19/6/2016 18:03:50	N/A
TIME DELTA	+ 3 second(s)	Synchronized
PENDING FILES	13 file(s)	N/A
BATTERY STATE OF CHARGE	88%	N/A

EPS STATUS REPORT	Max.	Min.	Max. (mission)	Min. (mission)
BATT. SOC (%)	86	86	89	86
BATT. VOLTAGE (V)	8.150	8.150	8.025	7.950

— RESTARTS	WDT	MCLR	BOR	INST	Unknown
COUNTERS	0	0	0	0	0

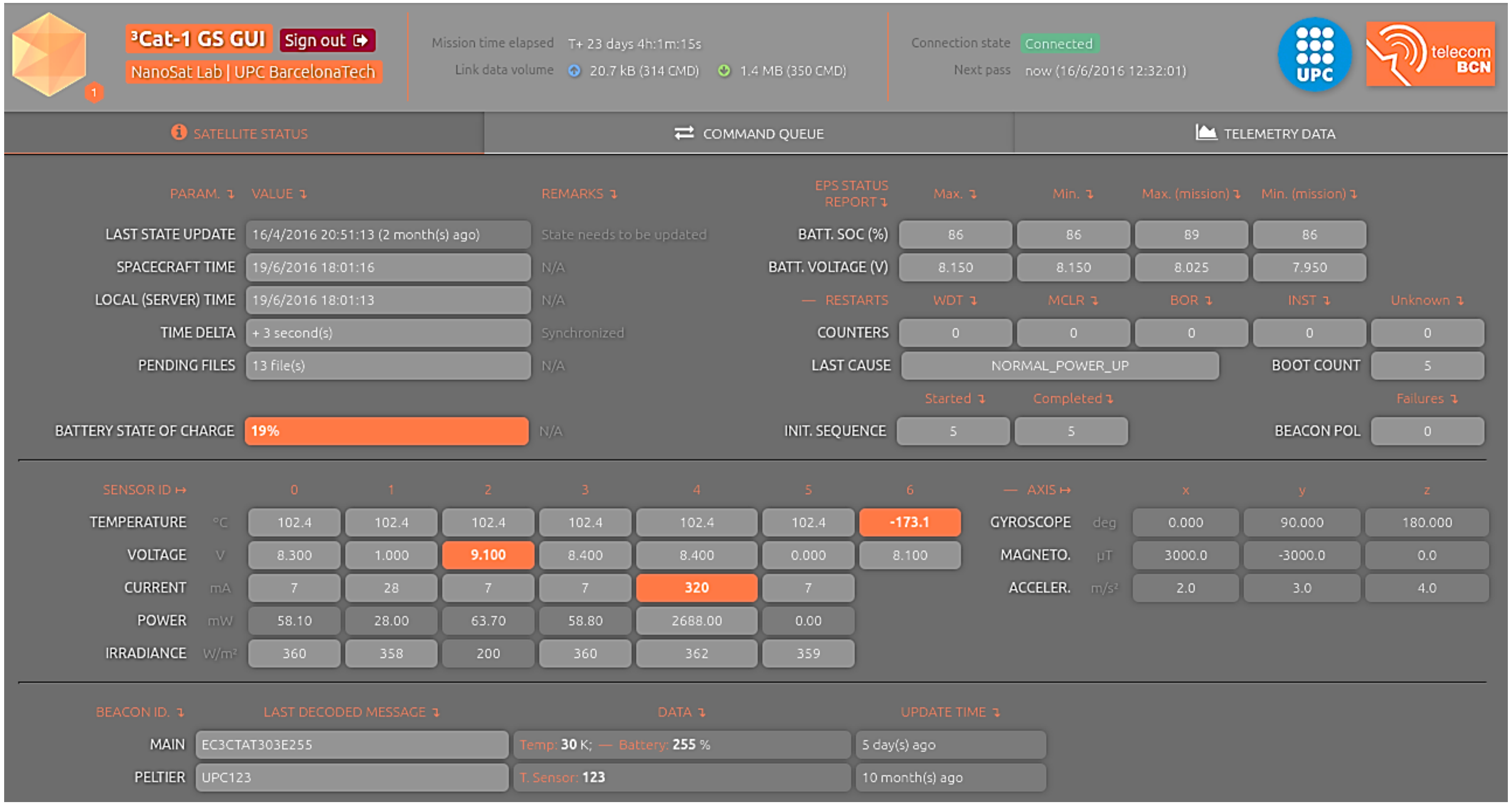
LAST CAUSE NORMAL_POWER_UP BOOT COUNT 5

	Started	Completed	Failures
INIT. SEQUENCE	5	5	
BEACON POL			0

SENSOR ID		0	1	2	3	4	5	6
TEMPERATURE	°C	102.4	102.4	102.4	102.4	102.4	102.4	2.9
VOLTAGE	V	8.300	1.000	8.100	8.400	8.400	0.000	8.100
CURRENT	mA	7	28	7	7	7	7	
POWER	mW	58.10	28.00	56.70	58.80	58.80	0.00	
IRRADIANCE	W/m²	360	358	358	360	362	359	

— AXIS		x	y	z
GYROSCOPE	deg	0.000	90.000	180.000
MAGNETO.	µT	3000.0	-3000.0	0.0
ACCELER.	m/s²	2.0	3.0	4.0

BEACON ID.	LAST DECODED MESSAGE	DATA	UPDATE TIME
MAIN	EC3CTAT303E255	Temp: **30** K; — Battery: **255** %	5 day(s) ago
PELTIER	UPC123	T. Sensor: **123**	10 month(s) ago



Mission time elapsed T+ 23 days 4h:1m:15s
Link data volume 20.7 kB (314 CMD) 1.4 MB (350 CMD)
Connection state Connected
Next pass now (16/6/2016 12:32:01)

PARAM.	VALUE	REMARKS
LAST STATE UPDATE	16/4/2016 20:51:13 (2 month(s) ago)	State needs to be updated
SPACECRAFT TIME	19/6/2016 18:01:16	N/A
LOCAL (SERVER) TIME	19/6/2016 18:01:13	N/A
TIME DELTA	+ 3 second(s)	Synchronized
PENDING FILES	13 file(s)	N/A
BATTERY STATE OF CHARGE	19%	N/A

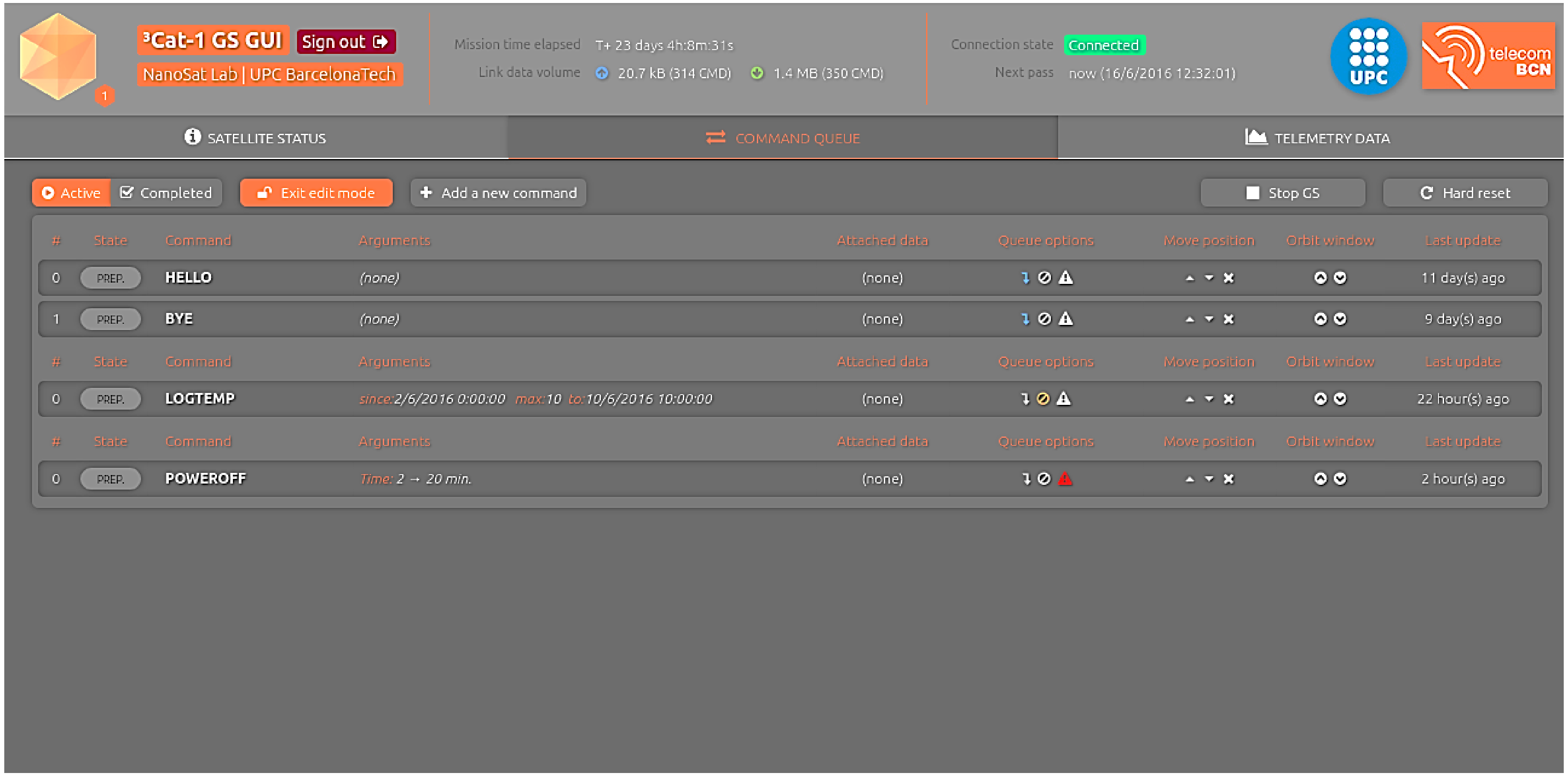
EPS STATUS REPORT	Max.	Min.	Max. (mission)	Min. (mission)	
BATT. SOC (%)	86	86	89	86	
BATT. VOLTAGE (V)	8.150	8.150	8.025	7.950	
— RESTARTS	WDT	MCLR	BOR	INST	Unknown
COUNTERS	0	0	0	0	0
LAST CAUSE	NORMAL_POWER_UP			BOOT COUNT	5
	Started	Completed			Failures
INIT. SEQUENCE	5	5		BEACON POL	0

SENSOR ID		0	1	2	3	4	5	6
TEMPERATURE	°C	102.4	102.4	102.4	102.4	102.4	102.4	-173.1
VOLTAGE	V	8.300	1.000	9.100	8.400	8.400	0.000	8.100
CURRENT	mA	7	28	7	7	320	7	
POWER	mW	58.10	28.00	63.70	58.80	2688.00	0.00	
IRRADIANCE	W/m²	360	358	200	360	362	359	

— AXIS		x	y	z
GYROSCOPE	deg	0.000	90.000	180.000
MAGNETO.	µT	3000.0	-3000.0	0.0
ACCELER.	m/s²	2.0	3.0	4.0

BEACON ID.	LAST DECODED MESSAGE	DATA	UPDATE TIME
MAIN	EC3CTAT303E255	Temp: **30** K; — Battery: **255** %	5 day(s) ago
PELTIER	UPC123	T. Sensor: **123**	10 month(s) ago

On the queue page, multiple options can be displayed. The next figures show its operation when the user is editing its content or just visualizing it. The first one presents a random active list with three different queues, on for each orbit window. The second image corresponds to the same queue but with a blocked edit mode.

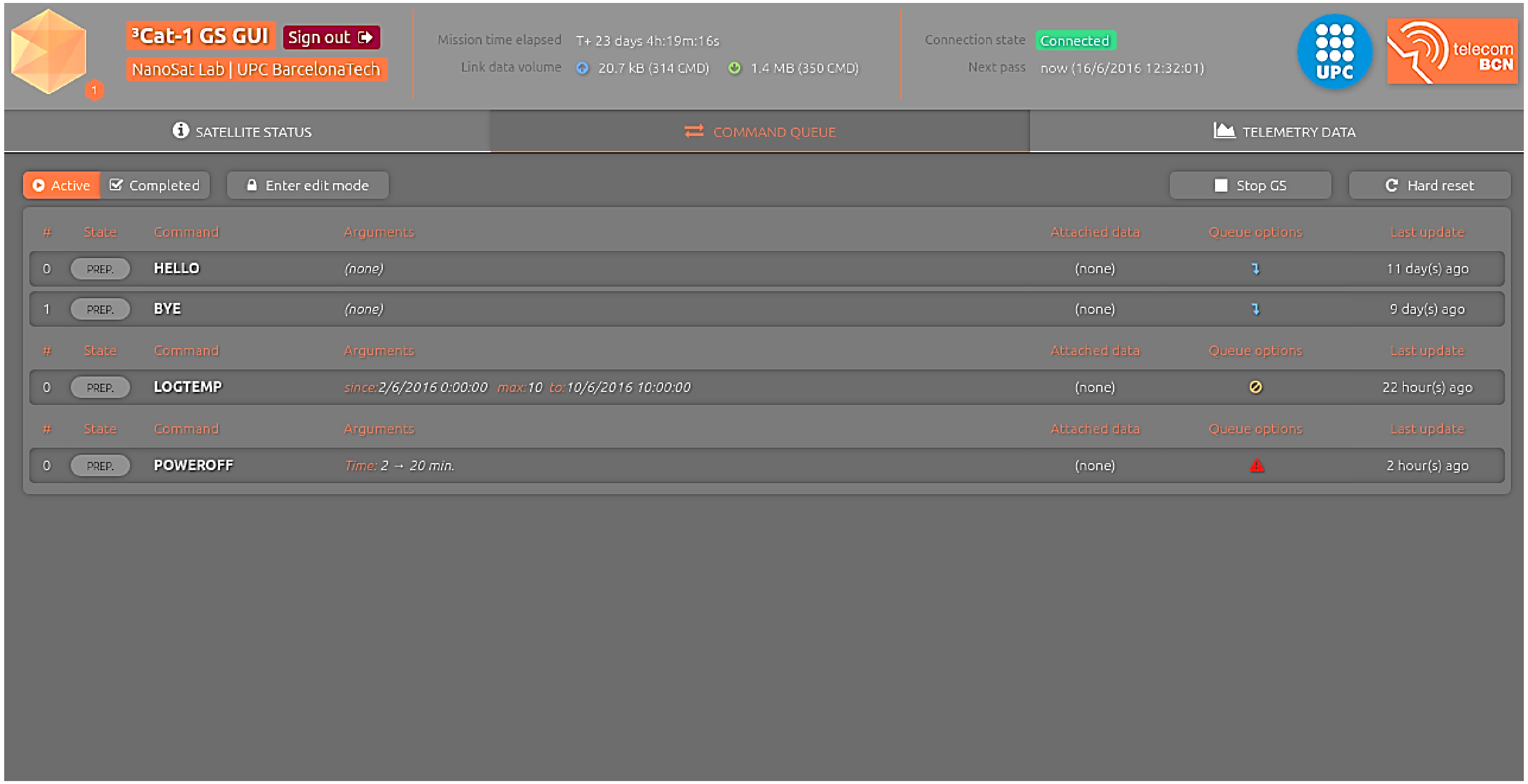


Mission time elapsed T+ 23 days 4h:8m:31s
Link data volume 20.7 kB (314 CMD) 1.4 MB (350 CMD)
Connection state Connected
Next pass now (16/6/2016 12:32:01)

#	State	Command	Arguments	Attached data	Queue options	Move position	Orbit window	Last update
0	PREP.	HELLO	*(none)*	(none)				11 day(s) ago
1	PREP.	BYE	*(none)*	(none)				9 day(s) ago

#	State	Command	Arguments	Attached data	Queue options	Move position	Orbit window	Last update
0	PREP.	LOGTEMP	*since:2/6/2016 0:00:00 max:10 to:10/6/2016 10:00:00*	(none)				22 hour(s) ago

#	State	Command	Arguments	Attached data	Queue options	Move position	Orbit window	Last update
0	PREP.	POWEROFF	*Time: 2 → 20 min.*	(none)				2 hour(s) ago



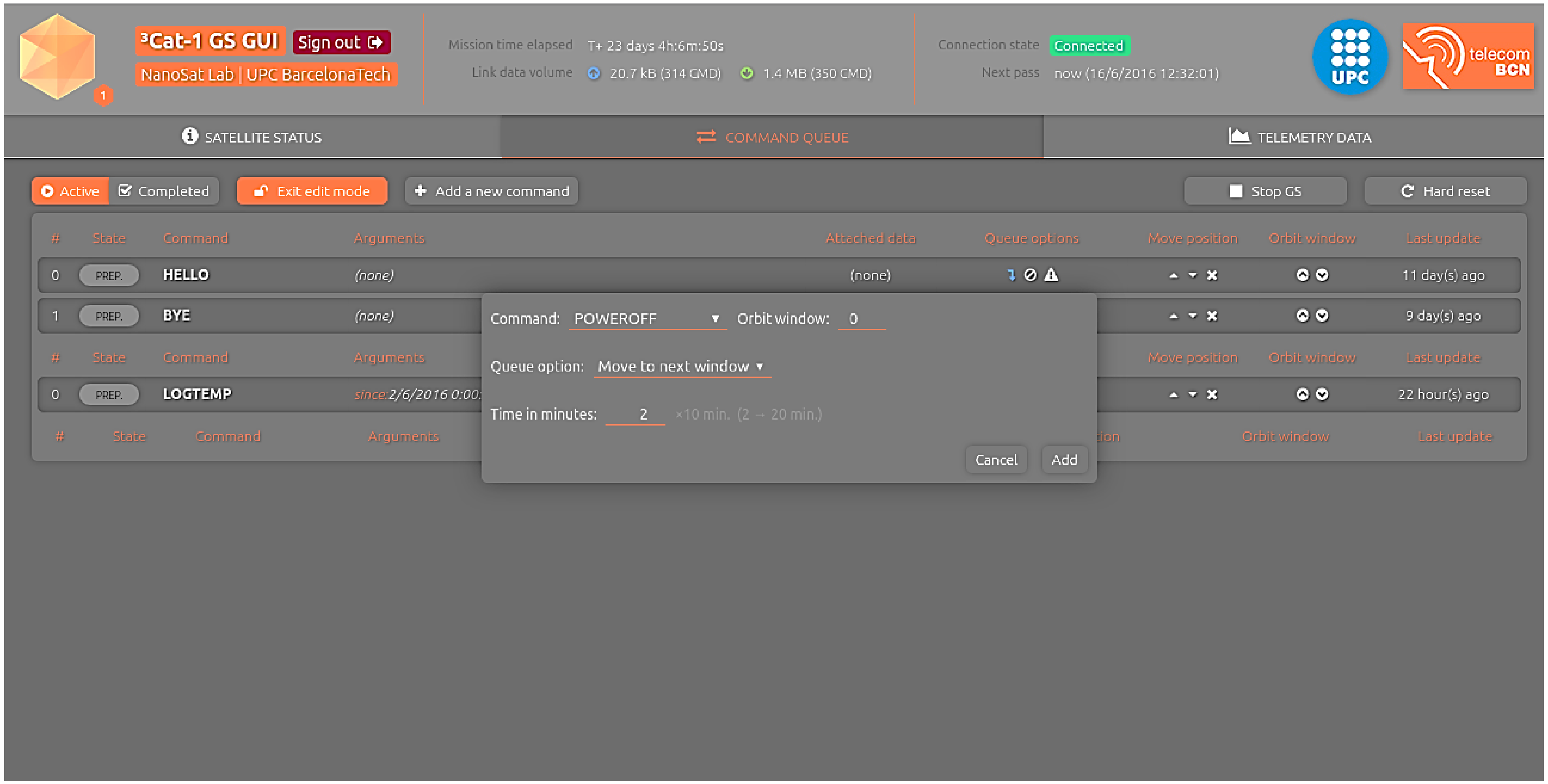
Mission time elapsed T+ 23 days 4h:19m:16s
Link data volume 20.7 kB (314 CMD) 1.4 MB (350 CMD)
Connection state Connected
Next pass now (16/6/2016 12:32:01)

#	State	Command	Arguments	Attached data	Queue options	Last update
0	PREP.	HELLO	*(none)*	(none)	↴	11 day(s) ago
1	PREP.	BYE	*(none)*	(none)	↴	9 day(s) ago

#	State	Command	Arguments	Attached data	Queue options	Last update
0	PREP.	LOGTEMP	*since:2/6/2016 0:00:00 max:10 to:10/6/2016 10:00:00*	(none)	⊘	22 hour(s) ago

#	State	Command	Arguments	Attached data	Queue options	Last update
0	PREP.	POWEROFF	*Time: 2 → 20 min.*	(none)	⚠	2 hour(s) ago

Next figure includes the new command form before adding, in this case, a “POWEROFF” telecommand (the last one in the previous image).



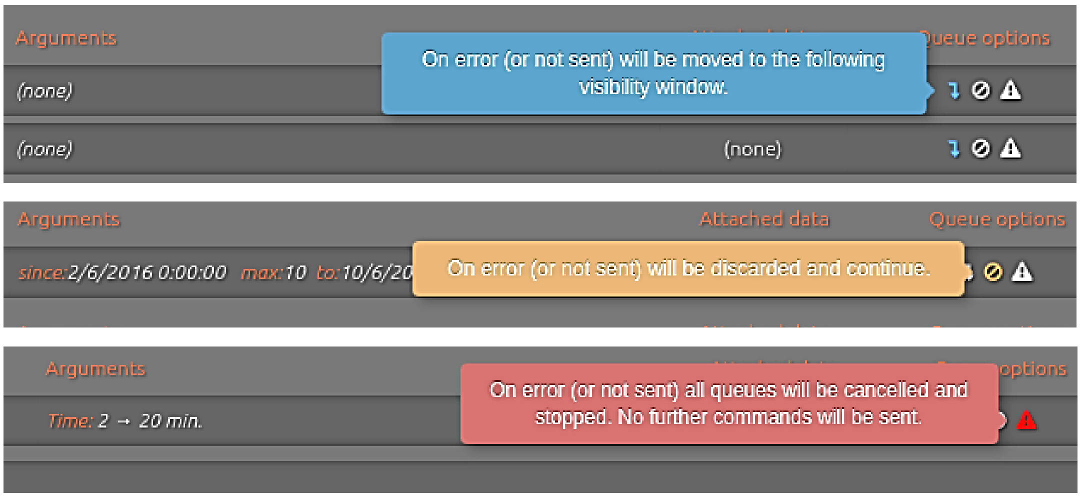
Mission time elapsed T+ 23 days 4h:6m:50s
Link data volume 20.7 kB (314 CMD) 1.4 MB (350 CMD)
Connection state Connected
Next pass now (16/6/2016 12:32:01)

#	State	Command	Arguments	Attached data	Queue options	Move position	Orbit window	Last update
0	PREP.	HELLO	*(none)*	(none)				11 day(s) ago
1	PREP.	BYE	*(none)*					9 day(s) ago

#	State	Command	Arguments	Move position	Orbit window	Last update
0	PREP.	LOGTEMP	*since 2/6/2016 0:00:*			22 hour(s) ago

Command: POWEROFF Orbit window: 0
Queue option: Move to next window
Time in minutes: 2 ×10 min. (2 → 20 min.)

The three “queue options” (actions to execute when a command returns an error code or its answer has not been received) display a brief description when the mouse hovers them, as can be seen in the next images.



On error (or not sent) will be moved to the following visibility window.

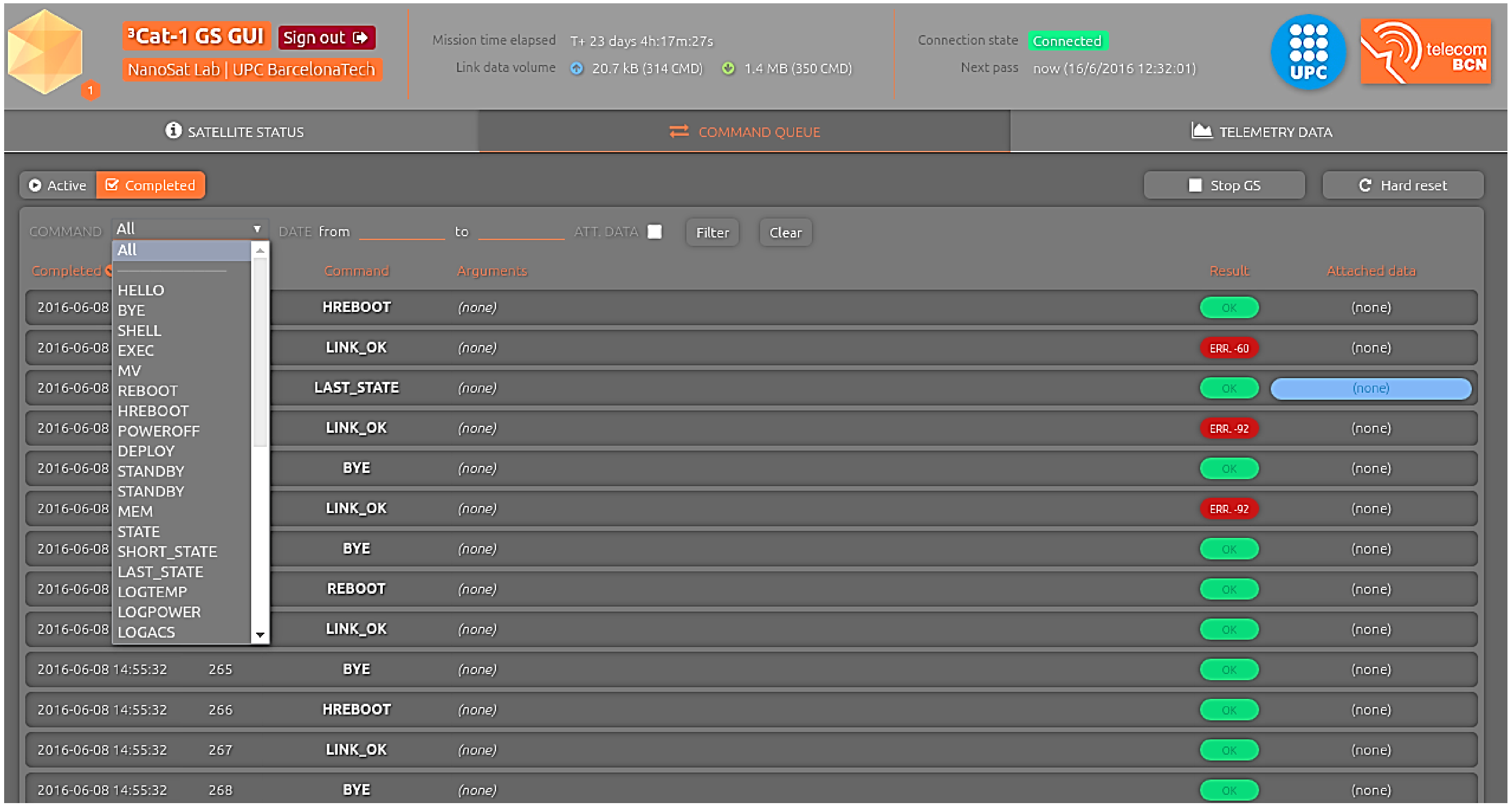
*since:*2/6/2016 0:00:00 *max:*10 *to:*10/6/20

On error (or not sent) will be discarded and continue.

Time: 2 → 20 min.

On error (or not sent) all queues will be cancelled and stopped. No further commands will be sent.

The completed queue includes a command filter, which appears on the following figure.

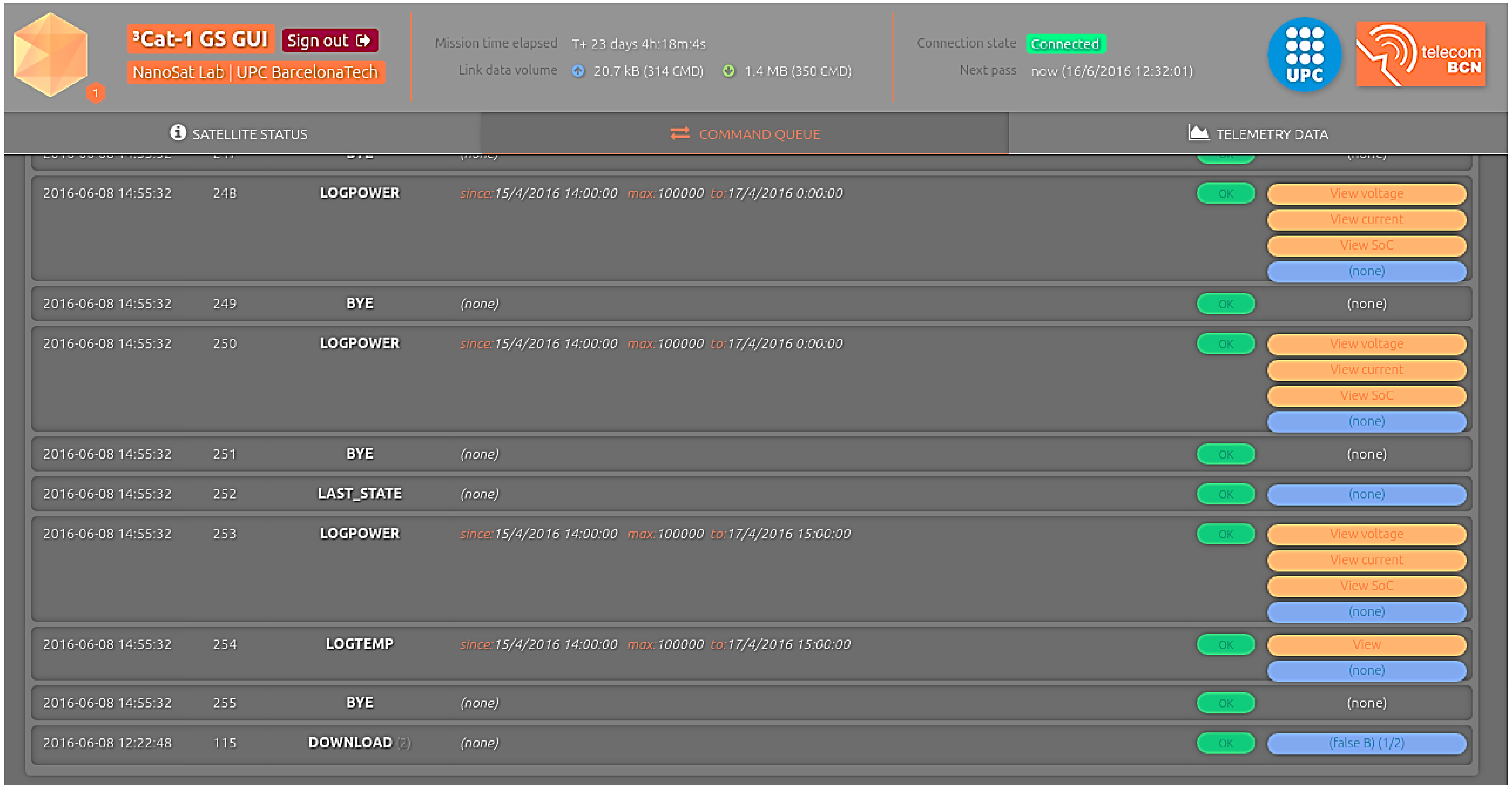


Mission time elapsed T+ 23 days 4h:17m:27s
Link data volume 20.7 kB (314 CMD) 1.4 MB (350 CMD)
Connection state Connected
Next pass now (16/6/2016 12:32:01)

COMMAND: All
- All
- HELLO
- BYE
- SHELL
- EXEC
- MV
- REBOOT
- HREBOOT
- POWEROFF
- DEPLOY
- STANDBY
- STANDBY
- MEM
- STATE
- SHORT_STATE
- LAST_STATE
- LOGTEMP
- LOGPOWER
- LOGACS

Completed		Command	Arguments	Result	Attached data
2016-06-08		HREBOOT	*(none)*	OK	(none)
2016-06-08		LINK_OK	*(none)*	ERR. -60	(none)
2016-06-08		LAST_STATE	*(none)*	OK	(none)
2016-06-08		LINK_OK	*(none)*	ERR. -92	(none)
2016-06-08		BYE	*(none)*	OK	(none)
2016-06-08		LINK_OK	*(none)*	ERR. -92	(none)
2016-06-08		BYE	*(none)*	OK	(none)
2016-06-08		REBOOT	*(none)*	OK	(none)
2016-06-08		LINK_OK	*(none)*	OK	(none)
2016-06-08 14:55:32	265	BYE	*(none)*	OK	(none)
2016-06-08 14:55:32	266	HREBOOT	*(none)*	OK	(none)
2016-06-08 14:55:32	267	LINK_OK	*(none)*	OK	(none)
2016-06-08 14:55:32	268	BYE	*(none)*	OK	(none)

Telecommands related with sensor logs have their own buttons to view (yellow) or download (blue) their content. There is one yellow button for each sensor type and a blue button for each received file, with the file size displayed inside it.



³Cat-1 GS GUI Sign out
NanoSat Lab | UPC BarcelonaTech

Mission time elapsed T+ 23 days 4h:18m:4s
Link data volume 20.7 kB (314 CMD) 1.4 MB (350 CMD)

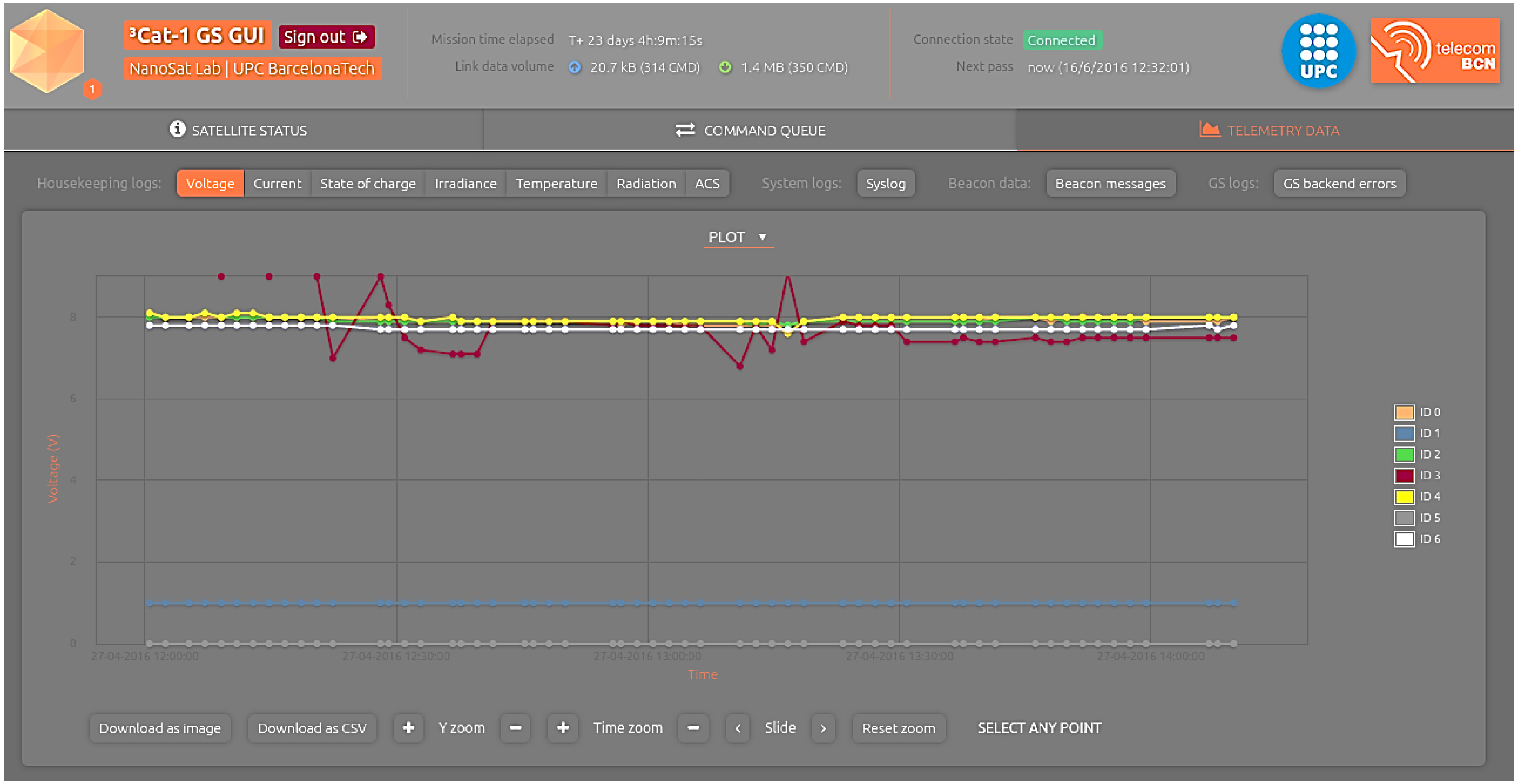
Connection state Connected
Next pass now (16/6/2016 12:32:01)

SATELLITE STATUS | COMMAND QUEUE | TELEMETRY DATA

2016-06-08 14:55:32	248	LOGPOWER	since:15/4/2016 14:00:00 max:100000 to:17/4/2016 0:00:00	OK	View voltage View current View SoC (none)
2016-06-08 14:55:32	249	BYE	(none)	OK	(none)
2016-06-08 14:55:32	250	LOGPOWER	since:15/4/2016 14:00:00 max:100000 to:17/4/2016 0:00:00	OK	View voltage View current View SoC (none)
2016-06-08 14:55:32	251	BYE	(none)	OK	(none)
2016-06-08 14:55:32	252	LAST_STATE	(none)	OK	(none)
2016-06-08 14:55:32	253	LOGPOWER	since:15/4/2016 14:00:00 max:100000 to:17/4/2016 15:00:00	OK	View voltage View current View SoC (none)
2016-06-08 14:55:32	254	LOGTEMP	since:15/4/2016 14:00:00 max:100000 to:17/4/2016 15:00:00	OK	View (none)
2016-06-08 14:55:32	255	BYE	(none)	OK	(none)
2016-06-08 12:22:48	115	DOWNLOAD (2)	(none)	OK	(false B) (1/2)

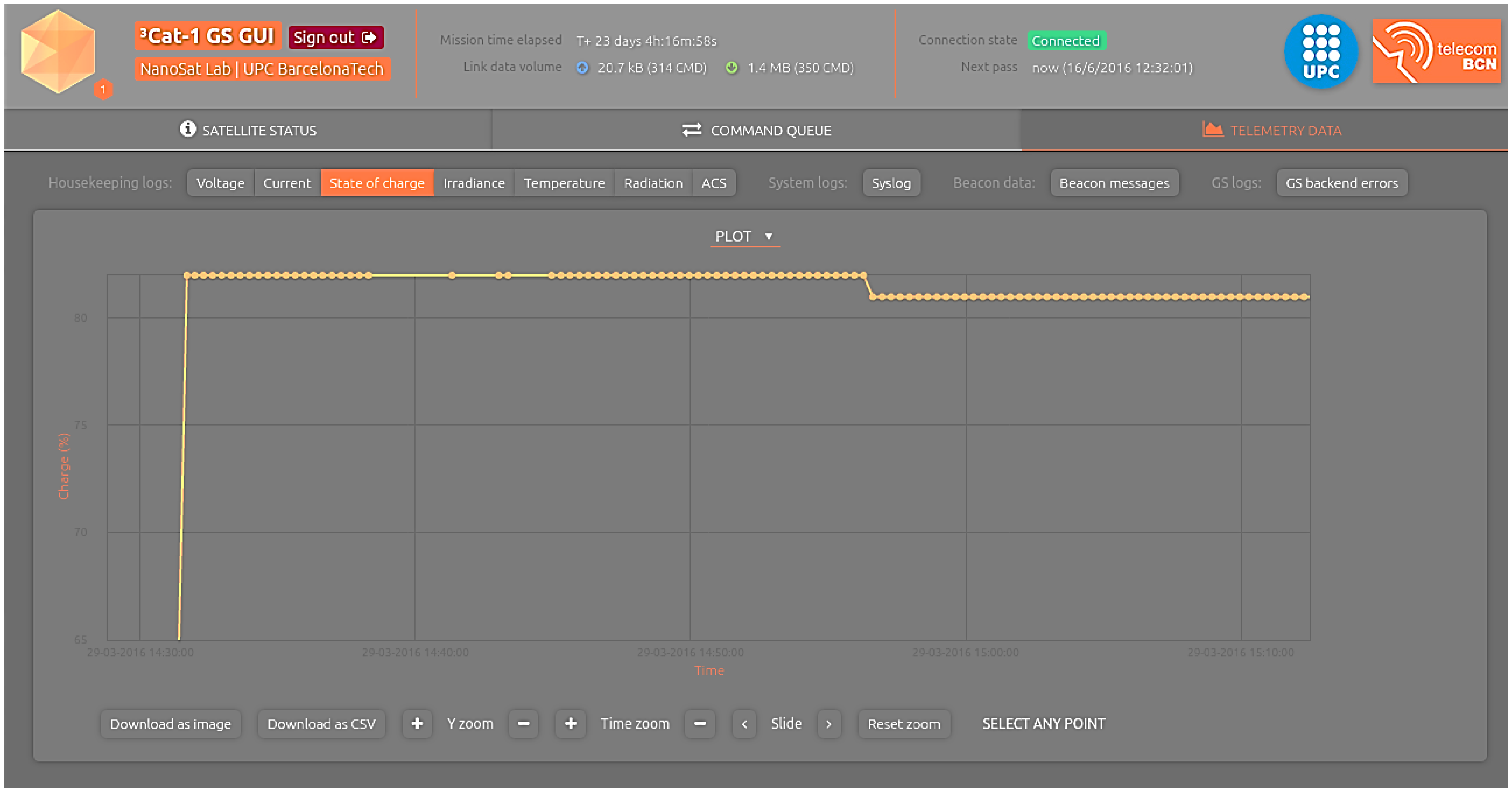
Finally, the telemetry page has two types of visualizations: graphical representations and register tables. They are divided by the information shown on them. The following figures present a qualitative representation of the telemetry layout.

This is a multi-sensor values graphical display (voltage).



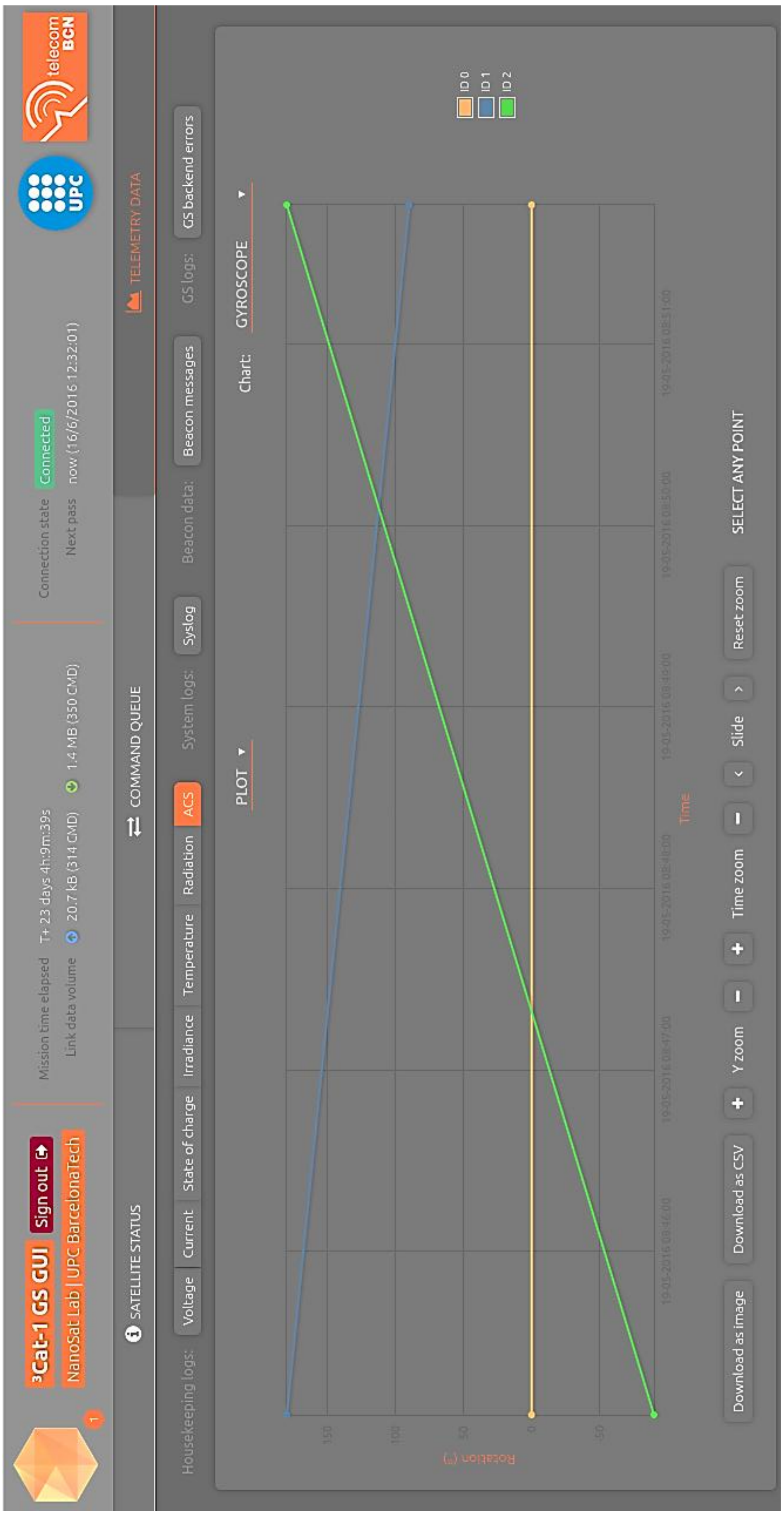
Mission time elapsed T+ 23 days 4h:9m:15s
Link data volume 20.7 kB (314 CMD) 1.4 MB (350 CMD)
Connection state Connected
Next pass now (16/6/2016 12:32:01)

And a single sensor graphical plot (state of charge).

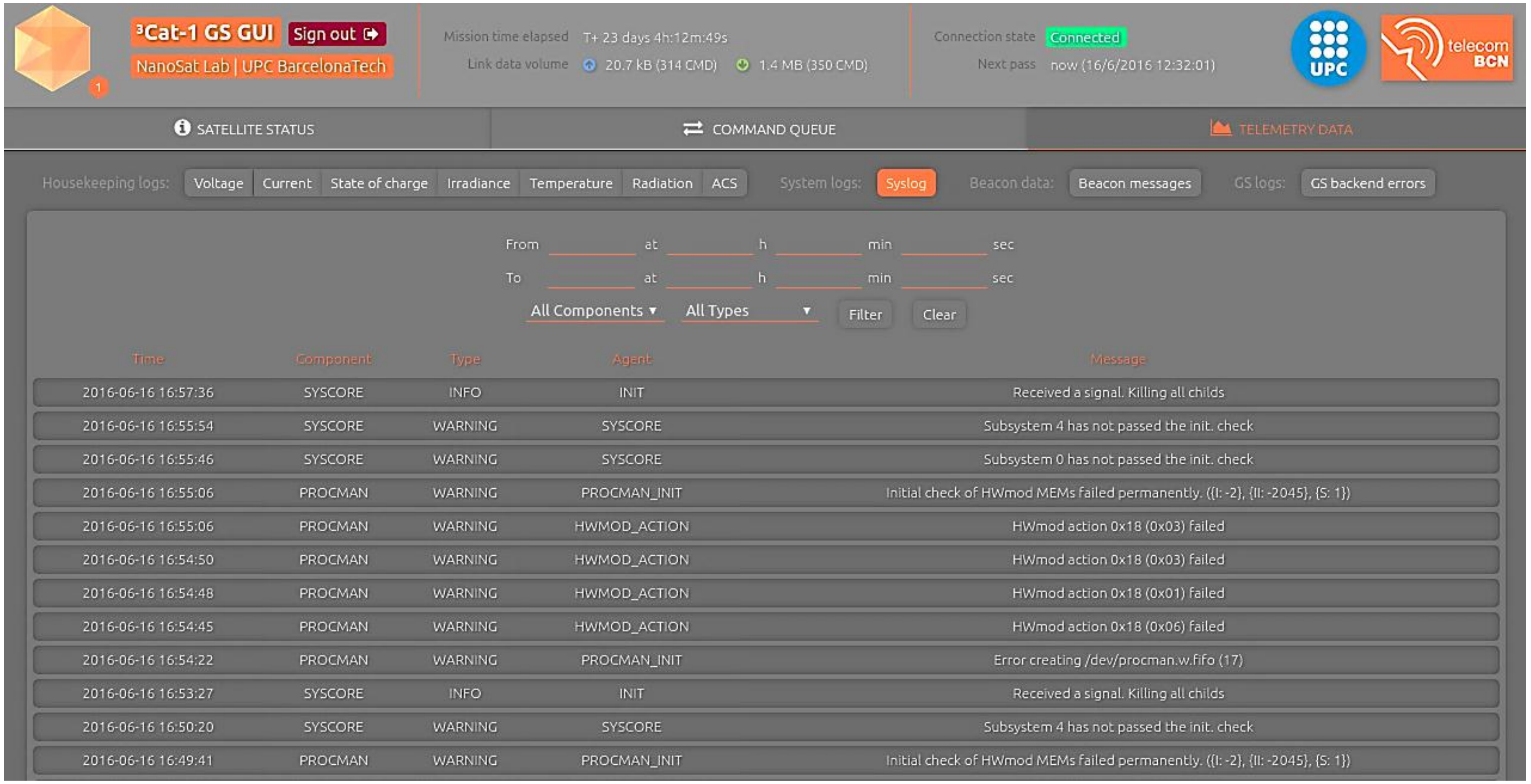


Mission time elapsed T+ 23 days 4h:16m:58s
Link data volume 20.7 kB (314 CMD) 1.4 MB (350 CMD)
Connection state Connected
Next pass now (16/6/2016 12:32:01)

The ADS section includes a chart selector as well in order to choose between three sensor types: magnetometer, accelerometer, gyroscope and IGRF.


3Cat-1 GS GUI
Sign out
NanoSat Lab | UPC BarcelonaTech
Mission time elapsed T+ 23 days 4h:9m:39s
Link data volume 20.7 kB (314 CMD) 1.4 MB (350 CMD)
Connection state Connected
Next pass now (16/6/2016 12:32:01)
SATELLITE STATUS
COMMAND QUEUE
TELEMETRY DATA
Housekeeping logs: Voltage Current State of charge Irradiance Temperature Radiation ACS
System logs: Syslog
Beacon data: Beacon messages
GS logs: GS backend errors
PLOT
Chart: GYROSCOPE
ID 0
ID 1
ID 2
Rotation (°)
150
100
50
0
-50
Time
19-05-2016 08:46:00
19-05-2016 08:47:00
19-05-2016 08:48:00
19-05-2016 08:49:00
19-05-2016 08:50:00
19-05-2016 08:51:00
Download as image
Download as CSV
Y zoom
Time zoom
Slide
Reset zoom
SELECT ANY POINT

Tables can be displayed by the user changing the “plot” option for every sensor type. System logs, beacon messages and back-end errors are shown as tables too. The following figure is an example of it. Tables also have time and type filters that are executed as the completed queue filter (with dropdown selectors and date-pickers).


³Cat-1 GS GUI
Sign out
NanoSat Lab | UPC BarcelonaTech
Mission time elapsed T+ 23 days 4h:12m:49s
Link data volume 20.7 kB (314 CMD) 1.4 MB (350 CMD)
Connection state Connected
Next pass now (16/6/2016 12:32:01)
SATELLITE STATUS
COMMAND QUEUE
TELEMETRY DATA
Housekeeping logs: Voltage Current State of charge Irradiance Temperature Radiation ACS
System logs: Syslog
Beacon data: Beacon messages
GS logs: GS backend errors
From at h min sec
To at h min sec
All Components
All Types
Filter
Clear

Time	Component	Type	Agent	Message
2016-06-16 16:57:36	SYSCORE	INFO	INIT	Received a signal. Killing all childs
2016-06-16 16:55:54	SYSCORE	WARNING	SYSCORE	Subsystem 4 has not passed the init. check
2016-06-16 16:55:46	SYSCORE	WARNING	SYSCORE	Subsystem 0 has not passed the init. check
2016-06-16 16:55:06	PROCMAN	WARNING	PROCMAN_INIT	Initial check of HWmod MEMs failed permanently. ({I: -2}, {II: -2045}, {S: 1})
2016-06-16 16:55:06	PROCMAN	WARNING	HWMOD_ACTION	HWmod action 0x18 (0x03) failed
2016-06-16 16:54:50	PROCMAN	WARNING	HWMOD_ACTION	HWmod action 0x18 (0x03) failed
2016-06-16 16:54:48	PROCMAN	WARNING	HWMOD_ACTION	HWmod action 0x18 (0x01) failed
2016-06-16 16:54:45	PROCMAN	WARNING	HWMOD_ACTION	HWmod action 0x18 (0x06) failed
2016-06-16 16:54:22	PROCMAN	WARNING	PROCMAN_INIT	Error creating /dev/procman.w.fifo (17)
2016-06-16 16:53:27	SYSCORE	INFO	INIT	Received a signal. Killing all childs
2016-06-16 16:50:20	SYSCORE	WARNING	SYSCORE	Subsystem 4 has not passed the init. check
2016-06-16 16:49:41	PROCMAN	WARNING	PROCMAN_INIT	Initial check of HWmod MEMs failed permanently. ({I: -2}, {II: -2045}, {S: 1})

A.3 Server-side implementation

GS software operation includes three different parts: the web-site server-side, the back-end packet manager and the communications module. A brief description of each one is detailed in the next list:

a) Web-site server-side files
   - Technology: AJAX interfacing and PHP
   - Operation:
     This module is responsible of the user-side communication with the GS. It includes the required algorithms and features to execute the requests to the databases and download the proper data for its visualization or update.

b) Back-end packet manager
   - ^{3}Cat-1 equivalent module: *gs_backend*
   - ^{3}Cat-2 equivalent module: *daemon*
   - Technology: C-programming executable, connections based on hardware
   - Operation:
     The back-end software is responsible for the generation of data packets and the databases update. It also coordinates all the GS modules, such as the orbit propagator, the beacon decoder, the communications software, etc.

c) Communications module
   - 3Cat-1 equivalent module: COMMS HWmod
   - 3Cat-2 equivalent module: included in SDR device
   - Technology: C-programming executable, connections based on hardware
   - Operation:
     It includes all the necessary functions for the packets transmission and reception. They process the data to adjust the structure depending on the final hardware and control the synchronization of the uplink and downlink channel.

Table A.1 shows the comparison between GS-1 and GS-2 implementations.

Table A.1 Ground Stations comparison

Function	GS-1	GS-2
Web-site server-side	PHP files	PHP files
Packet manager	*gs_backend*	*daemon*
COMMS software	COMMS HWmod	SDR
FOV passes	Orbit propagator	Gpredict
Satellite tracking	Gpredict	Gpredict
Beacon decoder	GNURadio	Not defined

B. Test results

B.1 Orbit propagator tests

B.1.1 Satellite position prediction

In order to measure the error on the prediction of the satellite position, the SGP4 algorithm used on the orbit propagator has been compared with the AGI's STK tool.

Test conditions:

- TLE set:
  FUNCUBE-1 (AO-73)
  1 39444U 13066AE 16166.18836951 .00000530 00000-0 72885-4 0 9993
  2 39444 97.6811 219.7932 0059655 32.1331 328.3495 14.81071453137137
- Start: 2016/06/08 00:00:00
- End: 2016/06/09 00:00:00
- Step between predictions: 1 second

The results obtained after the tests have been represented on the following graphs. Figure B.1 includes the error for each axis and its modulus, in ECI coordinates, computed as:

$$\overrightarrow{error} = \overrightarrow{OP} - \overrightarrow{STK} \tag{12}$$

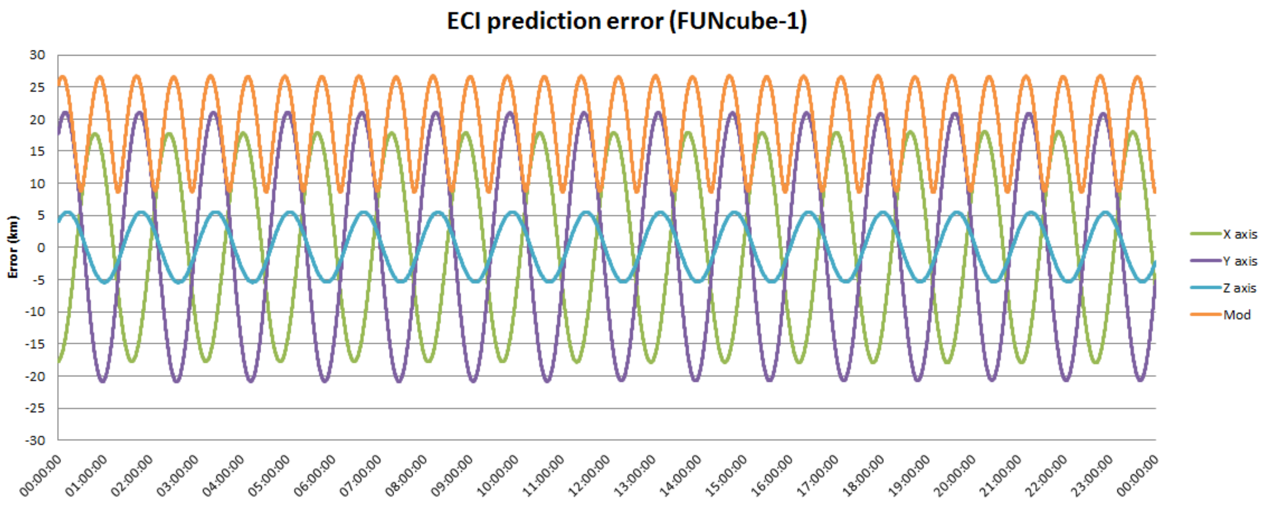


Figure B.1 ECI error op-stk

As can be seen, the error on each axis is based on a sine-wave form around zero, but the modulus is bounded around the 10 and 25 km of error. Due to the magnitude of the measures and the high satellite velocity on its orbit, these errors can be considered acceptable. Anyway, the ground track comparison between the two predictions, in figure B.2, stresses this conclusion. Both traces cannot be differentiated and the error in latitude or longitude is about the hundredth of a degree.

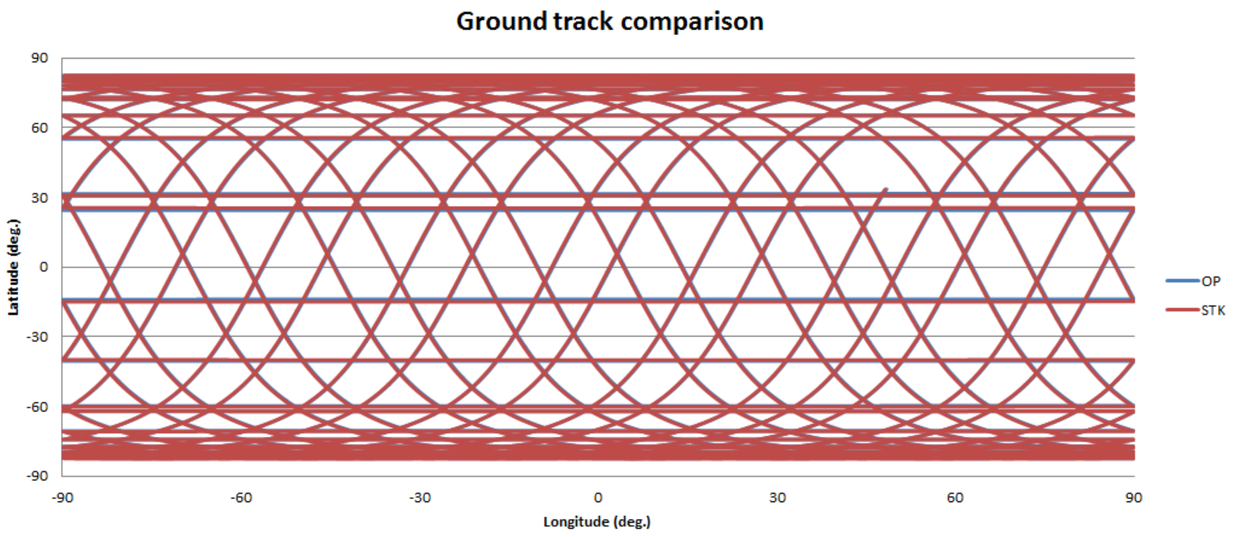


Figure B.2 Ground track comparison

B.1.2 Sunrise and sunset algorithm

The function used on the sunset and sunrise calculus has been developed from the Ed Williams' Aviation page [20] transcription of the algorithm found on the Almanac for Computers[19].

The sunrise and sunset algorithm:

```
Inputs:
        day, month, year: date of sunrise/sunset
        latitude, longitude: location for sunrise/sunset
        zenith: Sun's zenith for sunrise/sunset
                official = 90 degrees 50'
                civil = 96 degrees
                nautical = 102 degrees
                astronomical = 108 degrees

1. First calculate the day of the year
        N1 = floor(275 * month / 9)
        N2 = floor((month + 9) / 12)
        N3 = (1 + floor((year - 4 * floor(year / 4) + 2) / 3))
        N = N1 - (N2 * N3) + day - 30

2. Convert the longitude to hour value and calculate an approximate time
        lngHour = longitude / 15

        if rising time is desired:
          t = N + ((6 - lngHour) / 24)
        if setting time is desired:
          t = N + ((18 - lngHour) / 24)
```

3. Calculate the Sun's mean anomaly

```
M = (0.9856 * t) - 3.289
```

4. Calculate the Sun's true longitude

```
L = M + (1.916 * sin(M)) + (0.020 * sin(2 * M)) + 282.634
NOTE: L potentially needs to be adjusted into the range [0,360) by adding/subtracting 360
```

5a. Calculate the Sun's right ascension

```
RA = atan(0.91764 * tan(L))
NOTE: RA potentially needs to be adjusted into the range [0,360) by adding/subtracting 360
```

5b. Right ascension value needs to be in the same quadrant as L

```
Lquadrant  = (floor( L/90)) * 90
RAquadrant = (floor(RA/90)) * 90
RA = RA + (Lquadrant - RAquadrant)
```

5c. Right ascension value needs to be converted into hours

```
RA = RA / 15
```

6. Calculate the Sun's declination

```
sinDec = 0.39782 * sin(L)
cosDec = cos(asin(sinDec))
```

7a. Calculate the Sun's local hour angle

```
cosH = (cos(zenith) - (sinDec * sin(latitude))) / (cosDec * cos(latitude))

if (cosH >  1)
  the sun never rises on this location (on the specified date)
if (cosH < -1)
  the sun never sets on this location (on the specified date)
```

7b. Finish calculating H and convert into hours

```
if if rising time is desired:
  H = 360 - acos(cosH)
if setting time is desired:
  H = acos(cosH)

H = H / 15
```

8. Calculate local mean time of rising/setting

```
T = H + RA - (0.06571 * t) - 6.622
```

9. Adjust back to UTC

UT = T - lngHour

NOTE: UT potentially needs to be adjusted into the range [0,24) by adding/subtracting 24

10. Convert UT value to local time zone of latitude/longitude (not implemented, used times in UTC)

localT = UT + localOffset

B.1.2.1 Results comparison

Table B.1 shows the results computed with the previous algorithm and the official times found on http://www.timeanddate.com[21] for different days and locations. In order to adjust the UTC time to local time, an offset has been added depending on the city and the season of the year.

Table B.1 Sunrise and sunset results

Location	Date	OP sunrise	T&D sunrise	OP sunset	T&D sunset
Barcelona	20 August 2016	7:06	7:06	20:56	20:56
Barcelona	3 November 2018	7:25	7:25	17:44	17:44
New York	20 August 2016	5:25	5.25	20:31	20:31
New York	3 November 2018	7:29	7:29	17:50	17:50
Melbourne	20 August 2016	6:59	6:59	17:49	17:49
Melbourne	3 November 2018	6:12	6:12	19:56	19:56

All the computed times are exactly the same as the ones found on the webpage. Then, the algorithm accuracy is perfect and can be used on the orbit propagator.

B.1.3 Visibility intervals computation

This test is related to the FOV function because, once the position and sunset/sunrise predictions have been validated, the last feature that could affect the satellite passes calculation is the field of view. In chapter 5.2, the equations used on it were described and the next table, B.2, includes various results obtained during the test campaign.

The precision has been measured using the Gpredict prediction software. For a given satellite (FUNcube-1 in this case), multiple passes are measured and compared. On the test, night intervals have not been excluded.

Test conditions:

- Date: 2016/06/24 12:04:00
- Passes to predict: 10
- Time zone offset: UTC
- Location: 41.38º, 2.11º
- Twilight limit: 0º (over horizon)
- Minimum elevation: 0º (over horizon)

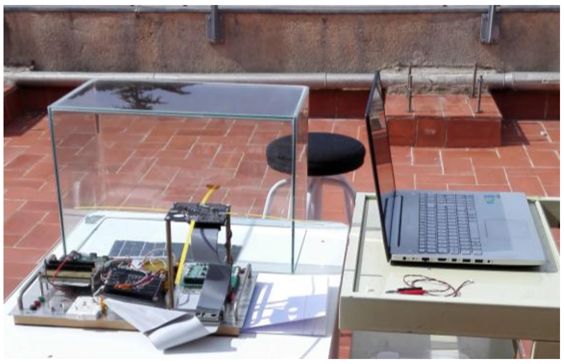
Table B.2 Visibility intervals comparison

#	OP AOS	Gpredict AOS	OP LOS	Gpredict LOS
1	2016/06/24 11:30:13	2016/06/24 11:30:05	2016/06/24 11:36:02	2016/06/24 11:36:16
2	2016/06/24 19:12:05	2016/06/24 19:12:10	2016/06/24 19:22:50	2016/06/24 19:22:50
3	2016/06/24 20:47:04	2016/06/24 20:47:04	2016/06/24 20:59:37	2016/06/24 20:59:39
4	2016/06/24 22:28:28	2016/06/24 22:28:08	2016/06/24 22:32:36	2016/06/24 22:32:54
5	2016/06/25 08:34:30	2016/06/24 08:34:32	2016/06/24 08:47:25	2016/06/24 08:47:25
6	2016/06/25 10:10:53	2016/06/24 10:10:52	2016/06/24 10:23:15	2016/06/24 10:23:20
7	2016/06/25 19:30:31	2016/06/24 19:30:35	2016/06/24 19:42:12	2016/06/24 19:42:13
8	2016/06/25 21:06:27	2016/06/24 21:06:25	2016/06/24 21:18:32	2016/06/24 21:18:36
9	2016/06/26 07:19:48	2016/06/24 07:19:58	2016/06/24 07:26:06	2016/06/24 07:25:54
10	2016/06/26 08:53:26	2016/06/24 08:53:27	2016/06/24 09:06:48	2016/06/24 09:06:49

The maximum error on the time calculus is around 20 seconds, but often it is less than 10 seconds. These errors may be due to the 10-25 km of difference on the position prediction compared to the STK and a lack of precision in the FOV function (Gpredict uses algorithms with higher accuracy). Otherwise, the only objective of the orbit propagator is the next visibility interval determination to start the communication protocol. It also has a configurable parameter to initiate the sending of the first HELLO command to establish the connection some minutes before. Therefore, a precision around seconds is enough for that purpose.

B.2 Morse beacon tests

On this chapter, the validation procedures of the Morse beacon decoder are detailed. In section 6, the basic concepts have been defined (message structure, decoding software design, etc.) and, in chapter 8.1, the test have been described as well. As was previously commented, the delivery of the ^{3}Cat-1 spacecraft was scheduled early during the development of this thesis and all the tests carried out on this part use multiple record files obtained during the last end-to-end verifications. Figure B.3 illustrates the environment where the data files have been recorded. In particular, the image corresponds to the batteries tests at the rooftop of the D4 building. While it was developed, the communications module was sending beacon messages, recorded on a computer with an RTL chipset connected to a dipole antenna.

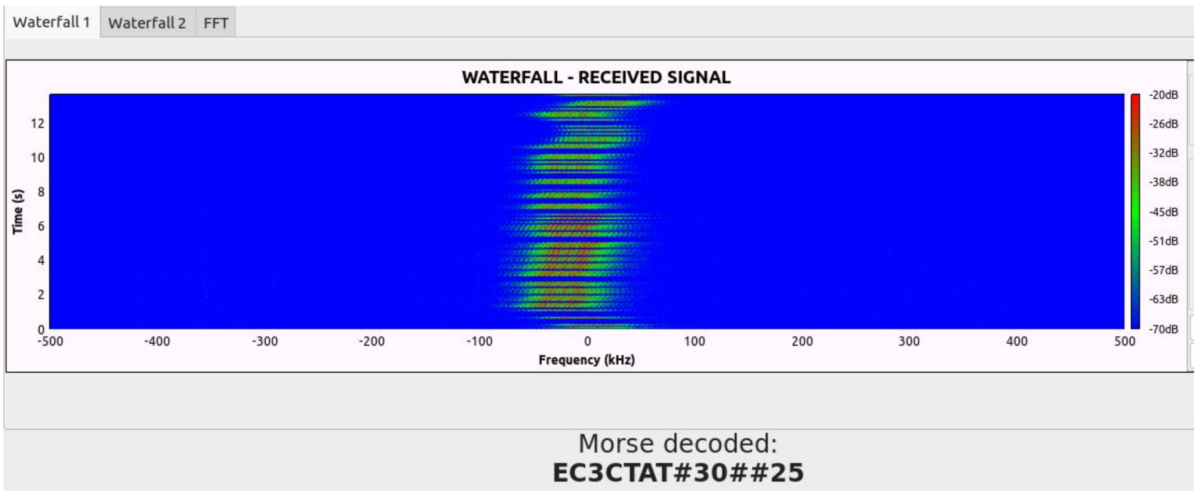


(a) Batteries test equipment

(b) RTL chipset

Figure B.3 Beacon recording environment

Figure B.4 shows the GNURadio interface used on the beacon decoder tests with a waterfall plot of the received signal in the RTL input. Under the graphical representation, there is the last decoded message, with '#' (hash) for those unknown characters.



Morse decoded:
EC3CTAT#30##25

Figure B.4 Beacon test received signal

Received signal:

- Center frequency: 437,25 MHz
- Signal bandwidth: around 100 kHz
- Dot time: 100 ms
- Signal power: around -25 dBm
- Noise level: [-65,-70] dBm

The tests also revealed a low frequency shift during brief shadowing periods up to 100 kHz

Figure B.5 shows the same signal after the GNURadio processing, just before the decoding modules. It reveals an SNR improvement and a bandwidth reduction.

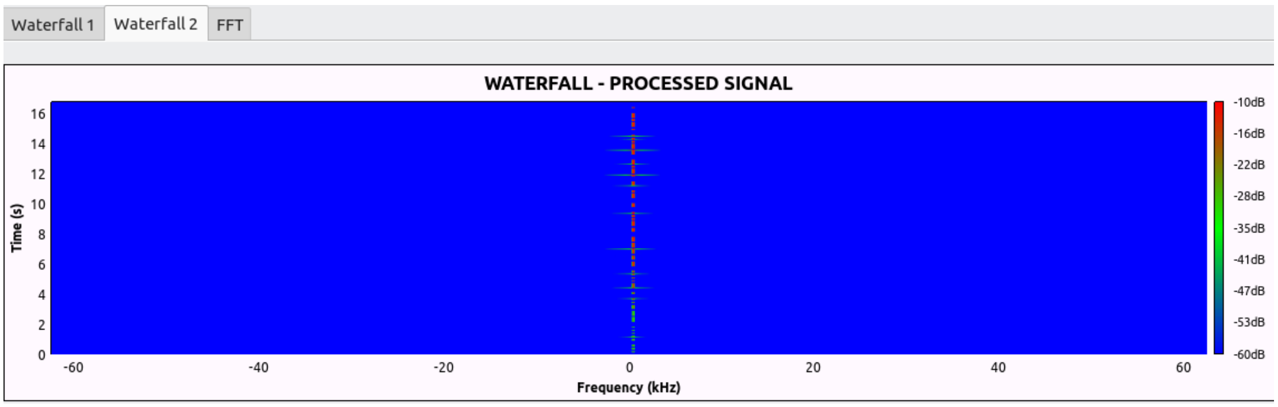


Figure B.5 Beacon test processed signal

Processed signal (base-band):

- Signal bandwidth: less than 1 kHz
- Signal power: [-10,-15] dBm
- Noise level: [-65,-70] dBm

C. ^{3}Cat-1 Software Validation Plan

This appendix corresponds to the 1.0 release of the ^{3}Cat-1 software Validation Plan. It has been elaborated by the author of this thesis and reviewed by the same advisors. All the elementary and end-to-end tests related to software development of the ^{3}Cat-1 project have been detailed, including the orbit propagator, the beacon decoder, multiple subsystems validation procedures and the GUI operation.

C.1 Methods and Test Types

There are, basically, three validations methods that have been carried out on the following verification procedures. These are analysis, inspection and test, which are described on the table below.

Table C.1 Verification methods

Verification Method	Description
Analysis	Theoretical or empirical evaluation of the verification test to obtain some expected results.
Inspection	Visual determination of physical characteristics.
Test	Performance of concrete procedures by using specific data. Used to evaluate and measure the requirements fulfilment, as well as the software robustness.

On the other hand, all tests can be differentiated according to three types: functional, interface or fault tests.

Table C.2 Test types

Test type	Description
Functional test	Test used to verify the software usage with nominal conditions, e.g. when the user introduces expected values without any error.
Interface test	This type of tests checks the communication between two or more units, verifying their dependences, triggers and data transfers.
Fault test	Reliability and robustness validation testing the items under unfavorable conditions, for example, errors introduced by the user or the communication channel.

C.2 Validation Tests Facilities

The software will be validated using the UPC NanoSat Lab facilities and, in case of the end-to-end tests, the satellite with the required integrated modules will be used as well.

All the tools used during the tests performance are detailed in the following list:

- Spectrum Analyzer, in order to ensure the correct transmitting signals shape and power.
- Oscilloscope, for PWM signals and voltage signals coming from the EPS board.
- Logic Analyzer, for low level interfaces debug, such as the connections between the different satellite modules.
- Voltmeter, to measure voltage levels on each module and ensure its correct performance.
- Power supply, which allows the batteries level regulation
- Raspberry Pi SBC, to manage the web server and the GS architecture

C.3 Units and Subsystems

Validation procedures can be divided in three stages:

1) Development and independent tests, which consist on validating each individual feature of any software part without the intervention of any other. Usually, it is done at the same time as the module development, when it is easier to fix any bug.
2) Communications tests, including data sending and reception. They are used to validate the protocols and data handlers during the communication process, from the command transmission to the answer reception.
3) End-to-end tests, with all the involved devices. Not only is the link tested, but the satellite response after a concrete instruction.

As any other mission, it can be divided into three segments: the satellite (space), the GS (Earth) and the launcher, only present at the beginning. The following figures show all the modules included in each segment (space and Earth), which are the sub-systems that will be tested.

C.4 Tests Specifications and Procedures

C.4.1 Validation Test Specifications

On the next table, all the tests that have been performed are listed with a brief description for each one, known as Validation Test Specifications (VTS). It includes a unique identifier, the aim of the test (what is going to be verified), the required inputs and the expected outputs, besides a success criterion.

Table C.3 Validation Tests Specifications

VTS ID	Tested functionality	Inputs	Expected output	Success criteria
1	Command upload	Command ID, arguments and attached file (last two, if needed)	Command stored in the queue and file uploaded to the right server path	The command appears correctly stored on the queue
2	Command packaging	Command in the queue	Temporal packet file created	".out" file created
3	Command completion procedure	Command packet	Satellite answer	Command processed at the right subsystem and answer file or ACK received
4	Queue edit mode	User interaction with edit mode buttons (GUI)	Change on queue page (changes applied and/or message)	Edit mode enabled/disabled or queue changes applied
5	GS back-end soft reset	User interaction with soft reset button (GUI)	"Stop queue" parameter modified	Back-end stops/restarts processing the queue
6	GS back-end hard reset	User interaction with hard reset button (GUI)	Back-end processes restarted	COMMS module and *gs_backend* restarted, "success" message received on the front
7	Next visibility period update	Next visibility period in orbit table	GUI header information updated	Remaining time and date coincide with Gpredict
8	Visibility intervals update	Satellite TLE file	Orbit table updated	Same intervals as Gpredict
9	Sensor values update	New log file received	New values stored in the right table	Values properly stored and same as log file, shown

				on the GUI too
10	Sensor values interface alerts	Sensor values over a pre-defined threshold	Highlighted cells	Cell is highlighted for those values over threshold
11	Telemetry charts update	New values in database	Tables and graphs updated	Last values included, time axis updated
12	Completed commands files download	Command with received files	File(s) received	File downloaded from GUI with “download” button in completed queue
13	Completed commands log files view	Command with log files	File(s) values visible in the proper graph	User redirected to graph with right values when “view” button is pressed
14	Morse beacon message decoding	Beacon signal	Message decoded	Right message structure
15	Satellite connection	HELLO command	ACK received	Connection established
16	Satellite disconnection	BYE command	ACK received	Connection released
17	Syscore reboot	REBOOT command	RESET_REQ sent to Syscore	Software reboot
18	Syscore hard reboot	HREBOOT command	HRESET_REQ sent to Syscore	Power-off software and EPS hard-reboot
19	System power off	POWEROFF command (N seconds)	POWER_OFF_OBC sent to SDB	Portux turned off during N seconds
20	System standby mode	STANDBY command (N seconds)	System in standby mode and TIME_RECAL_REQ to Syscore	System in standby mode for N seconds and time recal.
21	System mem mode	MEM command (N seconds)	System in mem mode and TIME_RECAL_REQ to Syscore	System in mem mode for N seconds and time recal.
22	Deployment	DEPLOY command	Run the deployment system	Deployment system enabled

23	Deployment stop	LINK_OK command	Stop the deployment system	Deployment system stopped
24	Log files download	LOGXXX command	Report generation and file received	File with last registers received and content stored in the database
25	Camera	CAMERA command	Camera HWmod started, run and stopped	Pictures downloaded and saved
26	Task enable	TASK_EN command	Selected task marked as enabled	Task enabled
27	Task disable	TASK_DIS command	Selected task marked as disabled	Task disabled
28	Task state report	TASK_STATE command	Task report generated and received	File with the state of the selected task downloaded
29	TLE update	TLE command	TLE file moved to TLE_PATH	TLE information updated
30	Configuration update	U_CONF or U_SCHEDOUT command	Configuration and/or scheduling file moved to right path	System configuration or scheduler updated
31	System time update	U_RTC command	TIME_RECAL_REQ to Syscore with given time	UNIX RTC time reconfigured
32	Mission time update	U_MISSIONTIME command	Updated files mtfbwy.r{1,2,3} with given time	Mission time reconfigured
33	Status report	STATE, SHORT_STATE or LAST_STATE command	System report generated and received from the satellite	Subsystems and sensors registers received
34	File execution	SHELL OR EXEC command	Execute the given file(s)	Instructions executed
35	Downloadable files	INFO_DOWNLOAD, DOWNLOAD command	File “infodownload.out” or downloadable files	Files downloaded
36	File upload	MV command	File moved to given path	Given file saved in the right path

C.4.2 Validation Test Procedures

On this chapter, all the VTPs performed for each previous specification will be detailed, including the procedure description, the tested functionality from the VTS list and the test environment.

VTP ID	1
Associated VTS ID	1
Functionality to be tested	Command without files upload
Required test environment	Software parts involved: GUI Required Hardware: —
Overview test procedure	Ensure that commands are properly stored in the queue table from the database, with the right argument values (if any)
Detailed description test procedure	The user adds a command with the GUI. Then, it is passed to the server using an AJAX request and POST variables. A PHP file (“manage_command.php”) stores the information in *queue* using an SQL query.
Validation	The test will be passed if the command parameters are stored properly in the corresponding columns of the table (name, arguments, initial state, chosen orbit window and current time) with a unique identifier.

VTP ID	2
Associated VTS ID	1
Functionality to be tested	Command with attached file upload
Required test environment	Software parts involved: GUI Required Hardware: —
Overview test procedure	Ensure that commands are properly stored in the queue table from the database, with the right argument values (if any) and the attached file is uploaded in the server.
Detailed description test procedure	The user adds a command with the GUI. Then, it is passed to the server using an AJAX request and POST variables. A PHP file (“manage_command.php”) stores the information in queue using an SQL query.
Validation	The test will be passed if the command parameters are stored properly in the corresponding columns of the table (name, arguments, initial state, chosen orbit window and current time) with a unique identifier and the “attached_sent” field includes a path to the uploaded file (with the same name given by the user).

VTP ID	3
Associated VTS ID	2
Functionality to be tested	Command packaging
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Validate the queue packaging process by the GS back-end software
Detailed description test procedure	The user adds a command with the GUI. Then, it is passed to the server using an AJAX request and POST variables. A PHP file (“manage_command.php”) stores the information in *queue* using an SQL query. Finally, the *gs_backend* processes the queue and generates a packet associated to the uploaded command (with “.out” extension)
Validation	A successful test involves the generation of a new packet on the server after the command upload.

VTP ID	4
Associated VTS ID	3
Functionality to be tested	Command without files completion procedure
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Verify the complete process of a command, from its transmission to the answer reception
Detailed description test procedure	The user adds a command with the GUI and it is stored on the database. The *gs_backend* generates the packet and tries to send it when visibility is guaranteed. An ACK is received when the packet has arrived and the back-end waits for the command answer. The satellite processes the command and returns an ACK or a file if it is required. When the communication process is finished and the answer has been processed, the command is sent to the “completed list” in the GUI.
Validation	To pass the test, a valid answer must be received for the given command and its state has to be updated at every stage (packed, sending, sent, answered, completed) until the process is ended.

VTP ID	5
Associated VTS ID	3
Functionality to be tested	Command with files completion procedure
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Verify the complete process of a command, from its transmission to the answer reception, and the upload of attached files.
Detailed description test procedure	The user adds a command with the GUI and it is stored on the database. The *gs_backend* generates the packet and tries to send it when visibility is guaranteed. An ACK is received when the packet has arrived and the back-end waits for the command answer. The satellite processes the command and returns an ACK or a file if it is required. When the communication process is finished and the answer has been processed, the command is sent to the “completed list” in the GUI.
Validation	To pass the test, a valid answer must be received for the given command and its state has to be updated at every stage (packed, sending, sent, answered, completed) until the process is ended. The attached file has to be located on a proper location in the satellite system as well.

VTP ID	6
Associated VTS ID	4
Functionality to be tested	Queue edit mode (enable/disable)
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Validate the edit mode criterion
Detailed description test procedure	The user presses the edit mode button on the GUI. A PHP file (manage_queue.php) checks the state of the queue, constantly updated by the *gs_backend.* If the queue is being processed, it cannot be enabled. If not, edit mode is enabled and new buttons appear on the GUI to add new commands or edit them on the list.
Validation	Depending on the state of the queue, edit mode must be enabled or not. When the back-end is blocking it, a message has to appear.

VTP ID	7
Associated VTS ID	4
Functionality to be tested	Queue edit mode (up/down)
Required test environment	Software parts involved: GUI Required Hardware: —
Overview test procedure	Check the feature of moving commands in the queue
Detailed description test procedure	With the edit mode enabled, users can modify the position of a command on the list, moving it up or down. When a command is moved, the position of the previous (up) or the next one (down) must be changed as well.
Validation	A command is properly moved when its position and the one of the affected command are properly updated depending on their situation in the queue.

VTP ID	8
Associated VTS ID	4
Functionality to be tested	Queue edit mode (delete)
Required test environment	Software parts involved: GUI Required Hardware: —
Overview test procedure	Verify the command deletion option
Detailed description test procedure	With the edit mode enabled, users can delete a command from the queue. When a command is deleted, all the attached files and the generated packets must be removed from the server file system and it has to be deleted from the database table too.
Validation	A command is properly deleted if all the stored information is removed from the queue and the server.

VTP ID	9
Associated VTS ID	4
Functionality to be tested	Queue edit mode (window up/down)
Required test environment	Software parts involved: GUI Required Hardware: —
Overview test procedure	Check the command window change feature
Detailed description test procedure	With the edit mode enabled, users can change the orbit window where a command has to be sent, up or down. When a command is moved from a window to another, it must be added at the end of the new list and removed from the previous one.
Validation	To validate the orbit window change, the command must be shown on the new window list (in last position) and not on the oldest one. If the database is checked, the orbitID must correspond to the new one. In case the new window is not available, the GUI has to inform that the command cannot be moved and keep it in the same window at the same position.

VTP ID	10
Associated VTS ID	5
Functionality to be tested	GS back-end soft reset (stop)
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Validate the soft reset feature to stop/restart the queue processing
Detailed description test procedure	From the GUI, the user can press the “Stop GS” button. The “stop_queue” parameter is set to “yes” in the database and, as it is periodically checked by the *gs_backend*, it stops processing the queue and waits. On the user interface, the button changes to “Restart GS”.
Validation	When the button is pressed, the back-end has to complete the current process (send or receive) and, then, wait until the queue is released.

VTP ID	11
Associated VTS ID	5
Functionality to be tested	GS back-end soft reset (restart)
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Validate the soft reset feature to stop/restart the queue processing
Detailed description test procedure	From the GUI, the user can press the “Restart GS” button. The “stop_queue” parameter is set to “no” in the database and, as it is periodically checked by the *gs_backend*, it starts processing the queue again. If there is any command, it will generate the packet and try to send it in the next visibility period. On the user interface, the button changes to “Stop GS”.
Validation	When the button is pressed, the back-end has to start processing the queue again.

VTP ID	12
Associated VTS ID	6
Functionality to be tested	GS back-end hard reset
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: Raspberry Pi
Overview test procedure	Validate the hard reset feature
Detailed description test procedure	From the GUI, the user can press the “Hard reset” button. The server has to send a reboot signal to the Raspberry Pi SBC and reset the *gs_backend* process too.
Validation	The test will be passed if all the involved modules are properly stopped and restarted (COMMS HWmod on the SBC and *gs_backend* on the server).

VTP ID	13
Associated VTS ID	7
Functionality to be tested	Next visibility period update (satellite far)
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Check GS operation when satellite is not visible
Detailed description test procedure	The next visibility period is updated without visibility at the test time. The *gs_backend* has to set the visibility parameter from the *state* table to “far” and stop the communication process. On the GUI, the new interval start time is shown on the header and the connection state changes to “disconnected” (red-colored).
Validation	The test is considered successful if the back-end software updates the visibility state and stops the transmission of commands. The GUI must show the new values properly.

VTP ID	14
Associated VTS ID	7
Functionality to be tested	Next visibility period update (satellite visible)
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Check GS operation when satellite is visible
Detailed description test procedure	The next visibility period is updated with visibility at the test time. The *gs_backend* has to set the visibility parameter from the *state* table to “visible” and start the transmission of a HELLO command. On the GUI, the next pass value is changed to “now” on the header and the connection state changes to “visible” (orange-colored).
Validation	The test is considered successful if the back-end software updates the visibility state and tries to start a communication. The GUI shows the new values properly.

VTP ID	15
Associated VTS ID	7
Functionality to be tested	Next visibility period update (satellite connected)
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Check GS operation when satellite is visible
Detailed description test procedure	After entering into a period of visibility, the *gs_backend* sends the HELLO command. When the first answer is received, it updates the visibility parameter to “connected” and starts sending the first command of the queue, if any. On the GUI, the connection state changes to “connected” (green-colored and blinking).
Validation	The test is considered successful if the back-end software updates the visibility state and starts processing the queue. The GUI shows the new values properly.

VTP ID	16
Associated VTS ID	9
Functionality to be tested	Sensor values update
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Check GS operation when satellite is visible
Detailed description test procedure	When a new log file is received, the *gs_backend* processes the information depending on its type and stores the content in a table of the database. The last value of the sensors can be consulted on the status tables at the website index page, which has to be updated when the new data has been stored.
Validation	The test will be passed if the last sensor values are the same as the new log file ones.

VTP ID	17
Associated VTS ID	11
Functionality to be tested	Telemetry charts update (graphs)
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Check GS operation when satellite is visible
Detailed description test procedure	When a new log file is received, the *gs_backend* processes the information depending on its type and stores the content in a table of the database. The new sensors registers can be visualized on the GUI telemetry page as graphical charts with the latest values included at the end. The time axis must correspond to the time of the last introduced values.
Validation	The test will be passed if the last values are the same as the new log file ones.

VTP ID	18
Associated VTS ID	11
Functionality to be tested	Telemetry charts update (tables)
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Check GS operation when satellite is visible
Detailed description test procedure	When a new log file is received, the *gs_backend* processes the information depending on its type and stores the content in a table of the database. The new registers (sensor values or system messages) can be consulted on the GUI telemetry page as tables.
Validation	The test will be passed if the last registers are the same as the new log file ones.

VTP ID	19
Associated VTS ID	10
Functionality to be tested	Sensor values interface alerts
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Verify alerts for undesired temperature values
Detailed description test procedure	New temperature values are added to the database when the gs_backend processes a temperature file. If there is a value > 378.15 ºK (105ºC) on the status table, the corresponding cell alerts the user with an orange background.
Validation	A sensor cell has to change its state to alert (orange) when the contained value is over the threshold.

VTP ID	20
Associated VTS ID	10
Functionality to be tested	Sensor values interface alerts
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Verify alerts for undesired temperature values
Detailed description test procedure	New temperature values are added to the database when the gs_backend processes a temperature file. If there is a value < 275.15 ºK (2 ºC) on the status table, the corresponding cell alerts the user with an orange background.
Validation	A sensor cell has to change its state to alert (orange) when the contained value is under the threshold.

VTP ID	21
Associated VTS ID	10
Functionality to be tested	Sensor values interface alerts
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Verify alerts for undesired voltage values
Detailed description test procedure	New voltage values are added to the database when the gs_backend processes a voltage file. If there is a value > 9 V on the status table, the corresponding cell alerts the user with an orange background.
Validation	A sensor cell has to change its state to alert (orange) when the contained value is over the threshold.

VTP ID	22
Associated VTS ID	10
Functionality to be tested	Sensor values interface alerts
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Verify alerts for undesired current values
Detailed description test procedure	New current values are added to the database when the gs_backend processes a current file. If there is a value > 300 mA on the status table, the corresponding cell alerts the user with an orange background.
Validation	A sensor cell has to change its state to alert (orange) when the contained value is over the threshold.

VTP ID	23
Associated VTS ID	10
Functionality to be tested	Sensor values interface alerts
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Verify alerts for undesired irradiance values
Detailed description test procedure	New irradiance values are added to the database when the gs_backend processes a irradiance file. If there is a value > 354 W/m^2 on the status table, the corresponding cell alerts the user with an orange background.
Validation	A sensor cell has to change its state to alert (orange) when the contained value is over the threshold.

VTP ID	24
Associated VTS ID	10
Functionality to be tested	Sensor values interface alerts
Required test environment	Software parts involved: GUI, *gs_backend* Required Hardware: —
Overview test procedure	Verify alerts for critical state of charge values
Detailed description test procedure	New state of charge values are added to the database when the gs_backend processes a state of charge file. If the current value is < 30%, the cell alerts the user with an orange background.
Validation	The state of charge cell has to change its state to alert (orange) when the contained value is under the threshold.

VTP ID	25
Associated VTS ID	15
Functionality to be tested	Satellite connection
Required test environment	Software parts involved: *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Check the connection establishment
Detailed description test procedure	Before the upcoming visibility interval starts (with a pre-established margin), the *gs_backend* starts sending a first HELLO command. Once the satellite has answered it, the connection is established.
Validation	The test is passed when the back-end software automatically starts sending a HELLO command if a satellite pass is expected, the answer is properly received and the connection state is set to connected.

VTP ID	26
Associated VTS ID	16
Functionality to be tested	Satellite disconnection
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Check the connection end
Detailed description test procedure	Whenever the user wants to end the transmission, a BYE command can be sent. Once the satellite has answered it, the connection is considered to be concluded.
Validation	The test is passed when the command is sent, answered by the satellite and the connection state is set to disconnected.

VTP ID	27
Associated VTS ID	17
Functionality to be tested	Satellite reboot
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the satellite reboot process
Detailed description test procedure	The user sends a REBOOT command from the GUI. Then, it is sent by the back-end software and waits for an answer. When the satellite receives it, a RESET_REQ instruction is sent to the System Core (Syscore), which is responsible of the safely reboot of the entire satellite. Once the process is completed, an OK answer is received.
Validation	The satellite must reboot and answer to the GS.

VTP ID	28
Associated VTS ID	18
Functionality to be tested	Satellite hard reboot
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the satellite hard reboot process
Detailed description test procedure	The user sends a HREBOOT command from the GUI. Then, it is sent by the back-end software and waits for an answer. When the satellite receives it, a HRESET_REQ instruction is sent to the System Core (Syscore), which is responsible of the safely hard reboot of the entire satellite. Once the process is completed, an OK answer is received.
Validation	The satellite must do a hard reboot and answer to the GS.

VTP ID	29
Associated VTS ID	22
Functionality to be tested	Deployment
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Verify the deployment process
Detailed description test procedure	The user sends a DEPLOY command from the GUI. Then, it is sent by the back-end software and waits for an answer. When the satellite receives it, an ENABLE_POL instruction with POL number 10 is sent to the System Data Bus (SDB). The deployment system starts running. Finally, an OK answer is received.
Validation	The satellite must run the deployment system and answer to the GS.

VTP ID	30
Associated VTS ID	33
Functionality to be tested	Status report
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Ensure that the status reports are generated and received
Detailed description test procedure	The user sends a STATE command from the GUI. Then, it is sent by the back-end software and waits for an answer. When the satellite receives it, the following files are generated and sent: - syscore.log - Syscore log database (all registers except INFO up to 64 kB) - Procman log database (all registers except INFO up to 64 kB) - SDB log database (all registers except INFO up to 64 kB) - HWmod log database (all registers except INFO up to 64 kB) - Each of the sensors database (up to 64 kB) - mem.log - time.log
Validation	The satellite must generate the previous files with the detailed system report and the GS has to receive them.

VTP ID	31
Associated VTS ID	33
Functionality to be tested	Status report
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Ensure that the short status reports are generated and received
Detailed description test procedure	The user sends a SHORT_STATE command from the GUI. Then, it is sent by the back-end software and waits for an answer. When the satellite receives it, the following files are generated and sent: - syscore.log - Syscore log database (all FATAL registers in last 3h or up to 64 kB) - Procman log database (all FATAL registers in last 3h or up to 64 kB) - SDB log database (all FATAL registers in last 3h or up to 64 kB) - HWmod log database (all FATAL registers in last 3h or up to 64 kB) - Each of the sensors database (last 10 or up to 64 kB) - mem.log - time.log
Validation	The satellite must generate the previous files with the last 3h system report and the GS has to receive them.

VTP ID	32
Associated VTS ID	33
Functionality to be tested	Status report
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Ensure that the last state reports are generated and received
Detailed description test procedure	The user sends a LAST_STATE command from the GUI. Then, it is sent by the back-end software and waits for an answer. When the satellite receives it, the following files are generated and sent: - Each of the sensors database with last register - time.log
Validation	The satellite must generate the previous files with the minimum system report and the GS has to receive them.

VTP ID	33
Associated VTS ID	24
Functionality to be tested	Log files download
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Test the log files download
Detailed description test procedure	The user sends a LOGFILES command from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite generates a log database files with all registers in new_out, received or new_tarball. Once generated and transmitted, these files have to be received on the GS.
Validation	The satellite must generate the previous files and the GS has to receive them.

VTP ID	34
Associated VTS ID	24
Functionality to be tested	Log files download
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Test the kernel log files download
Detailed description test procedure	The user sends a LOGKERN command from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite generates the kern.log file cut to last 64 kB. Once generated and transmitted, this file has to be received on the GS.
Validation	The satellite must generate the previous files and the GS has to receive them.

VTP ID	35
Associated VTS ID	25
Functionality to be tested	Camera
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the camera operation
Detailed description test procedure	The user sends a CAMERA command from the GUI. Then, it is sent by the back-end software and waits for an answer. On the satellite, the camera is started, run and stopped. At the GS, the last pictures must be received.
Validation	The test will be passed if the last taken pictures are received.

VTP ID	36
Associated VTS ID	35
Functionality to be tested	Downloadable files
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Ensure that the report of downloadable files is properly generated
Detailed description test procedure	The user sends an INFO_DOWNLOAD command from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite generates the infodownload.out file with: - Number of valid files (existing and not empty) - Combined size - Biggest file size - Smallest file size Finally, it has to be received and the information at the state table must be updated.
Validation	The test will be passed if the previous file is received and the data information is updated.

VTP ID	37
Associated VTS ID	35
Functionality to be tested	Downloadable files
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Ensure that new files are downloaded
Detailed description test procedure	The user sends a DOWNLOAD command from the GUI (N or all files). Then, it is sent by the back-end software and waits for an answer. The satellite transmits N (or all) files of type “new_out”. Finally, they are received on the GS.
Validation	The test will be passed if the indicated number of files is received.

VTP ID	38
Associated VTS ID	23
Functionality to be tested	Deployment stop
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate that the deployment stop instruction is processed
Detailed description test procedure	The user sends a LINK_OK command from the GUI. Then, it is sent by the back-end software and waits for an answer. When it is received by the satellite, the EPS stops the deployment system. At the GS, an OK answers has to be received.
Validation	The EPS stops the deployment system and answers an ACK.

VTP ID	39
Associated VTS ID	19
Functionality to be tested	System power off
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Check the power off instruction
Detailed description test procedure	The user sends a POWEROFF command from the GUI (Nx10 minutes). Then, it is sent by the back-end software and waits for an answer. Once the command is processed at the satellite, a POWER_OFF_OBC instruction is sent to the SDB and all HWmod, except the EPS, are requested to halt. Before doing the power off, if there is not any error, an ACK is transmitted.
Validation	A successful test implies the satellite power off during the indicated time and a proper reboot.

VTP ID	40
Associated VTS ID	20
Functionality to be tested	System standby mode
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Check the standby instruction
Detailed description test procedure	The user sends a STANDBY command from the GUI (N seconds). Then, it is sent by the back-end software and waits for an answer. The system enters in standby mode for the given number of seconds and then sends TIME_RECAL_REQ to Syscore.
Validation	A successful test implies the satellite in standby mode during N seconds and a time recalculation, besides an ACK answer.

VTP ID	41
Associated VTS ID	21
Functionality to be tested	System mem mode
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Check the standby instruction
Detailed description test procedure	The user sends a MEM command from the GUI (N seconds). Then, it is sent by the back-end software and waits for an answer. The system enters in mem mode for the given number of seconds and then sends TIME_RECAL_REQ to Syscore.
Validation	A successful test implies the satellite in mem mode during N seconds and a time recalculation, besides an ACK answer.

VTP ID	42
Associated VTS ID	31
Functionality to be tested	System time update
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Check the standby instruction
Detailed description test procedure	The user sends an U_RTC command from the GUI (Unix seconds). Then, it is sent by the back-end software and waits for an answer. The system sends TIME_RECAL_REQ to Syscore with the given time. In order to check the time update, a RTCTIME command can be sent. A file containing the system time in Unix format will be received.
Validation	The test is passed if the updated time corresponds to the same as the given by the user.

VTP ID	43
Associated VTS ID	24
Functionality to be tested	Log files download
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the download of temperature log files
Detailed description test procedure	The user sends a LOGTEMP command (with start, max. number of registers and end date) from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite generates a temperature database file between start and end time with the maximum number of registers up to 64 kB. Once generated and transmitted, this file has to be received on the GS and the content is stored at the temperature table of the database.
Validation	The satellite must generate the previous file and the GS has to receive and process them.

VTP ID	44
Associated VTS ID	24
Functionality to be tested	Log files download
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the download of power log files
Detailed description test procedure	The user sends a LOGPOWER command (with start, max. number of registers and end date) from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite generates three database files between start and end time with the maximum number of registers up to 64 kB: voltage, current and state of charge. Once generated and transmitted, they have to be received on the GS and the content is stored at the corresponding tables of the database.
Validation	The satellite must generate the previous files and the GS has to receive and process them.

VTP ID	45
Associated VTS ID	24
Functionality to be tested	Log files download
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the download of ACS log files
Detailed description test procedure	The user sends a LOGACS command (with start, max. number of registers and end date) from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite generates an ACS database file between start and end time with the maximum number of registers up to 64 kB. Once generated and transmitted, this file has to be received on the GS and the content is stored at the ACS table of the database.
Validation	The satellite must generate the previous files and the GS has to receive and process them.

VTP ID	46
Associated VTS ID	24
Functionality to be tested	Log files download
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the download of Syscore log files
Detailed description test procedure	The user sends a LOGSYSCORE command (with start, end and debug depth) from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite generates a Syscore database file between start and end time including registers until the indicated debug depth up to 64 kB and a syscore.log file. Once generated and transmitted, they have to be received on the GS and the content is stored at the syslog table of the database.
Validation	The satellite must generate the previous files and the GS has to receive and process them.

VTP ID	47
Associated VTS ID	24
Functionality to be tested	Log files download
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the download of Procman log files
Detailed description test procedure	The user sends a LOGPROCMAN command (with start, end and debug depth) from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite generates a Procman database file between start and end time including registers until the indicated debug depth up to 64 kB. Once generated and transmitted, this file has to be received on the GS and the content is stored at the syslog table of the database.
Validation	The satellite must generate the previous files and the GS has to receive and process them.

VTP ID	48
Associated VTS ID	24
Functionality to be tested	Log files download
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the download of SDB log files
Detailed description test procedure	The user sends a LOGSDB command (with start, end and debug depth) from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite generates an SDB database file between start and end time including register until the indicated debug depth up to 64 kB. Once generated and transmitted, this file has to be received on the GS and the content is stored at the syslog table of the database.
Validation	The satellite must generate the previous files and the GS has to receive and process them.

VTP ID	49
Associated VTS ID	24
Functionality to be tested	Log files download
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the download of HWmod log files
Detailed description test procedure	The user sends a LOGHWMOD command (with start, end and debug depth) from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite generates an SDB database file between start and end time including register until the indicated debug depth up to 64 kB. Once generated and transmitted, this file has to be received on the GS and the content is stored at the syslog table of the database.
Validation	The satellite must generate the previous files and the GS has to receive and process them.

VTP ID	50
Associated VTS ID	36
Functionality to be tested	File upload
Required test environment	Requirement: Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Ensure that files are uploaded and moved properly
Detailed description test procedure	The user sends a MV command (with the file to move and the destination path) from the GUI. Then, it is sent by the back-end software and waits for an answer. An OK answer should be received after the file has been uploaded and moved to the right path.
Validation	The test will be passed if the file is in the chosen path

VTP ID	51
Associated VTS ID	26
Functionality to be tested	Task enable
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Check the task enabling instruction
Detailed description test procedure	The user sends a TASK_EN command (with the “taskID” to enable) from the GUI. Then, it is sent by the back-end software and waits for an answer. The task corresponding to the selected identifier is marked as enabled in the task database. Once it is done, the satellite answers with an ACK message.
Validation	The task has to be enabled in the database to pass the test

VTP ID	52
Associated VTS ID	27
Functionality to be tested	Task disable
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Check the task disabling instruction
Detailed description test procedure	The user sends a TASK_DIS command (with the “taskID” to disable) from the GUI. Then, it is sent by the back-end software and waits for an answer. The task corresponding to the selected identifier is marked as disabled in the task database. Once it is done, the satellite answers with an ACK message.
Validation	The task has to be disabled in the database to pass the test

VTP ID	53
Associated VTS ID	28
Functionality to be tested	Task state report
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the task report feature
Detailed description test procedure	The user sends a TASK_STATE command (with a "taskID") from the GUI. Then, it is sent by the back-end software and waits for an answer. At the satellite, a file containing the state of the specified task is generated and transmitted. Finally, it is received by the GS.
Validation	The test is passed if the file is received with the precise information

VTP ID	54
Associated VTS ID	34
Functionality to be tested	File execution
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Ensure the execution of shell script files
Detailed description test procedure	The user sends a SHELL command (with a ".sh" file) from the GUI. Then, it is sent by the back-end software and waits for an answer. When the file is processed by the satellite, it has to execute the instructions contained on it. Once it is done, the satellite answers with an ACK message.
Validation	The test will be passed if the file is executed on the satellite

VTP ID	55
Associated VTS ID	34
Functionality to be tested	File execution
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Ensure the execution of binary files
Detailed description test procedure	The user sends an EXEC command (with a binary file) from the GUI. Then, it is sent by the back-end software and waits for an answer. When the file is processed by the satellite, it has to execute the instructions contained on it. Once it is done, the satellite answers with an ACK message.
Validation	The test will be passed if the file is executed on the satellite

VTP ID	56
Associated VTS ID	30
Functionality to be tested	Configuration update
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the update of the satellite configuration
Detailed description test procedure	The user sends a U_CONF command from the GUI with one of these attached files: sycore.conf, procman.conf, sdb.conf, comms.conf, acs.conf, camera_noacs.conf, wpt.conf, gt.conf, mems.conf, mems_mode{N}.conf or geiger{N}.conf. Then, it is sent by the back-end software and waits for an answer. The satellite copies the configuration file to the proper location (depending on its name). If a file was syscore.conf or procman.conf, the change has to be notified. Once it is done, the satellite answers with an ACK message.
Validation	The test will be passed if all the sent files are copied on their corresponding locations

VTP ID	57
Associated VTS ID	29
Functionality to be tested	TLE update
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the update of the TLE information
Detailed description test procedure	The user sends a TLE command (with a text file containing the telemetry information) from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite moves the given file to the TLE_PATH. Once it is done, the satellite answers with an ACK message.
Validation	The test is considered as passed if the telemetry information is properly updated after copying the file

VTP ID	58
Associated VTS ID	29
Functionality to be tested	Configuration update
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Validate the update of the scheduler configuration
Detailed description test procedure	The user sends a U_SCHEDOUT command and two attached files (syscore.conf and task_planner.conf) from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite moves the scheduling file and the syscore configuration file to their respective locations and the Syscore is notified. Once it is done, the satellite answers with an ACK message.
Validation	The test is passed when the Syscore is notified and the information is updated

VTP ID	59
Associated VTS ID	32
Functionality to be tested	Mission time update
Required test environment	Software parts involved: GUI, *gs_backend,* COMMS HWmod Required Hardware: Raspberry Pi with COMMS module and the satellite
Overview test procedure	Check the mission time update instruction
Detailed description test procedure	The user sends a U_SCHEDOUT command (with the new mission start time in Unix format) from the GUI. Then, it is sent by the back-end software and waits for an answer. The satellite updates (or creates) the mtfbwy.r{1,2,3} files with the given time. Once it is done, the satellite answers with an ACK message. To check that the time has been properly updated, a MISSIONTIME command is sent. A missiontime.out file is received and the mission time information is updated in the database.
Validation	The test is passed if the new mission time is the same as the time introduced by the user

VTP ID	60
Associated VTS ID	14
Functionality to be tested	Morse beacon message decoding
Required test environment	Software parts involved: GUI, Morse beacon decoder (GNURadio) Required Hardware: RTL chipset
Overview test procedure	Validate the proper reception and decoding of beacon messages
Detailed description test procedure	While the satellite is operative, a Morse beacon message is transmitted every minute (or three minutes when the batteries are under the 20% of charge). The signal is received with the VHF antenna directly connected to an RTL chipset. The beacon decoder program is responsible for the signal processing and message decoding. Once decoded, the message is stored at the beacon table in the database.
Validation	A valid message (following the “EC3CTAT###E###”) stored in the corresponding table ensures the test success

VTP ID	61
Associated VTS ID	8
Functionality to be tested	Visibility intervals update (sunrise/sunset)
Required test environment	Software parts involved: Orbit Propagator Required Hardware: —
Overview test procedure	Verify the correct sunset and sunrise time calculation
Detailed description test procedure	For a given date (the same as the test date, for example) and a certain position (the GS coordinates), the sunset and sunrise times are calculated. Then, they are compared to the official ones that can be found in some websites (e.g. http://www.timeanddate.com/sun/spain/barcelona) The process is repeated for multiple dates and locations (10 days and 5 dispersed locations can be representative enough).
Validation	The test is passed if the error for all the predictions is less than 1 minute.

VTP ID	62
Associated VTS ID	8
Functionality to be tested	Visibility intervals update (position prediction)
Required test environment	Software parts involved: Orbit Propagator, AGI's STK Required Hardware: —
Overview test procedure	Verify the correct sunset and sunrise time calculation
Detailed description test procedure	For a given TLE file (FUNcube-1 orbital elements where used in this test), the satellite position is propagated between a start and end date with one second steps. The same prediction is calculated with AGI's STK. To obtain reliable results, the prediction should include more than one orbit. In this case, a 24h prediction is computed.
Validation	The test is passed if the maximum error is under 25 km.

VTP ID	63
Associated VTS ID	8
Functionality to be tested	Visibility intervals update (field of view)
Required test environment	Software parts involved: Orbit Propagator, Gpredict Required Hardware: —
Overview test procedure	Verify the correct sunset and sunrise time calculation
Detailed description test procedure	For a given TLE file (FUNcube-1 orbital elements where used in this test), the next visibility intervals are calculated, up to 5 by default. The same intervals are obtained with Gpredict software and compared.
Validation	The test is passed if the maximum error is less than one minute.