POLITECNICO DI TORINO

Master Degree in Aerospace Engineering

Ground Control Software for Small Satellites Missions

Tutors
Prof. Sabrina Corpino
Eng. Fabrizio Stesina

Candidate
Michele Foggetta 230881

A.A. 2017/2018

"E' così bello fissare il cielo e accorgersi
di come non sia altro che un vero e proprio
immenso laboratorio di fisica
che si srotola sulle nostre teste"
Margherita Hack

Ringraziamenti

Oggi si conclude una tappa fondamentale della mia vita, un viaggio lungo sei anni all'intero del Politecnico di Torino.

Vorrei ringraziare la Prof.ssa Corpino, mia relatrice, per avermi dato la possibilità di svolgere questa tesi magistrale.
In particolare, ringrazio l'Ing. Stesina, il quale mi ha guidato in questi mesi di lavoro. Grazie per avermi spronato ad andare avanti e per i mille consigli.
Un ringraziamento speciale va ai miei genitori per avermi sostenuto moralmente ed economicamente in questo percorso.
Ringrazio mia sorella, Marianna. Sei la sorella che tutti vorrebbero e dovrebbero avere. Senza di te non ce l'avrei mai fatta.
Ringrazio mio fratello, Antonio, mio padrino. Anche se lontano, sapevo di poter contare su di te in qualsiasi momento.
Ringrazio Mous per avermi sopportato in questi mesi di stress.
Ringrazio i miei amici torinesi, Antonio, Arianna, Lea, Sharon, per i caffè presi a tutte le ore del giorno, per le ore passate a giocare a briscola e per avermi sopportato in questi sei anni. Lo so, non ho un carattere semplice, ma voi mi avete accettato nel bene e nel male.
Ringrazio Cristina, senza di te non avrei mai superato i brutti momenti iniziali di questo viaggio.
Un ringraziamento speciale va ai miei amici di una vita, Giusy, Simone e Graziana. Anche se a km di distanza, mi siete stati sempre vicino.
Infine, vorrei ringraziare ME STESSO. Sono stati anni lunghi, tra momenti di gioia e di sconforto, ma abbiamo raggiunto il nostro obiettivo.

Ora inizia il vero viaggio della vita e spero di affrontarlo, come sempre, nel modo migliore.

Abstract

This thesis will cover the development, implementation and validation of the software of the Ground Control Station (GCS) for Nanosatellite missions, a Mission Control Software (MCS), that aims to perform the control and the management of spacecraft subsystems telemetry and commands.
Typically, ground operations during the mission are always complex and require a lot of efforts in terms of time, cost and needed resources.
A way to reduce time and costs is to use a specific software that allows user to manage all subsystems telemetry and commands at the same time in autonomy way.
To do this, we proceed searching existing software, and evaluating if, for our aim, it is required to employ one previously conceived and tested, choosing between "open" and "close" source, or, if it is necessary design one "in-house developed".
The final software set up and utilized is a combination of an open source existing software with a graphical layout that facilitates user to employ it, and an implementation of functions for the control and the management of telemetry data and commands.
To validate this software, we utilize telemetry data received by E-st@r 2, spacecraft developed by CubeSat Team of Politecnico di Torino, during a preliminary phase test, simulating the satellite that sends telemetry packets through an executable program.
Then, we manage, control and analyze telemetry packets received in real time with a Development Board on which are placed the radio and EPS of E-st@r 2 prototype.

Finally, we adapt the software to manage a spacecraft constellation composed by three Nano Satellites, one of which is E-st@r 2, and, through a string generator, we create telemetry packets for the other two spacecrafts.

Contents

List of Figures

List of Tables

Acronyms

A – ADCS	Active Attitude Determination and Control System
ALTEC	Aerospace Logistics Technology Engineering Company
API	Application Programming Interface
CFS	Core Flight System
C&DH	Command and Data Handling
COSMOS	Comprehensive Open-Architecture Space Mission Operation System/Support
COTS	Commercial Off-The-Shelf
ESA	European Space Agency
E – st@r	Educational SaTellite @ politecnico di toRino
EPS	Electrical Power System
FCS	Frame-Check Sequence
FM	Frequency Modulation
GCS	Ground Control Systems
GENSO	Global Educational Network for Satellite Operators
GNC	Guidance Navigation and Control
GS	Ground Station
HSFL	Hawaii Space Flight Laboratory
JSON	JavaScript Object Notation
KISS	Keep It Simple, Stupid
MCC	Mission Control Center
MCS	Mission Control Software
MGCS	Mobile Ground Control Station
NOS³	NASA Operational Simulator
OIPP	Orbit, Inview and Power Planning
Open MCT	Open Source Mission Control Technologies
POCC	Payload Operations Control Center
RF	Radio Frequency
SOCC	Spacecraft Operations Control Center
SOH	State-Of-Health
TNC	Terminal Node Controller
TT&C	Telemetry, Tracking and Control

Introduction

Space field was characterized for years by big and expensive missions based on large satellites and probes. The space market was dominated by governments that imposed the roadmap for the future missions and systems development. The entire missions and the space products required great efforts in terms of involved resources. The entire product life cycle had long duration and the costs were very high, technology was expensive and completely designed to survive to the space environment.

In the last years, this context is changing. Small and standardized satellites are gaining the interest of governments, space agencies, private companies that recognize them as attractive space platforms for pursuing a broad set of mission goals, including science and Earth observation, technology demonstration, communication. Significant cost reduction, also due to components of the shelf use, and a relatively faster development time along the product life cycle compared with the traditional larger satellite missions constitute the main reasons of this success. Moreover, constellations of nanosatellite/CubeSats in LEO are becoming a reality and possible applications for interplanetary exploration missions are deeply investigated and planned.

The increasing capabilities of the small satellites and the tremendous increment of the number of these platforms in orbit impose that the ground segment follows the new trend and is able to support the operations of missions based on small satellites. In this sense, a major autonomy from the operators and capabilities to manage large number of data from different sources at the same time are the focal issues that the developers and operators of the ground stations and mission control centers shall address.

This thesis is focused to improve software section of ground segment, researching and developing a software able to manage, control and validates telemetry data received by satellite and to send it telecommands.
This work is based on a suggestion of ALTEC (Aerospace Logistics Technology Engineering Company) S.P.A, whose interest, nowadays, like many space industries and agencies, is the management and control of Nano Satellite. This concept is also shared by Politecnico di Torino, in particular DIMEAS (Dipartimento di Ingegneria Meccanica e Aerospaziale), which has made available all objects and devices needed to reach the purpose of this thesis.

Structure of The Thesis

The thesis is divided in five chapters.
The first chapter presents a description of Ground Control Stations (GCS) architectures, highlighting its main features and a bibliographic analysis of software, currently existing, utilized to control satellites.
Advantages and disadvantages of use closed or open source software are discussed.
Furthermore, we present the chosen software and how install and use it.

In chapter two, we describe how we modify the software and all functions developed and included in source code are presented in detail. We simulate the reception of telemetry data by one satellite through a second computer, which simulates the spacecraft that sends telemetry packets to MCC, on first computer, through serial communication.
To validate the software, we will simulate the reception of telemetry data of E-st@r 2.

In chapter three, we test the software created, receiving, validating and managing telemetry packets in real time, utilizing a Development Board on which are set radio and EPS of satellite prototype.

Chapter four presents similar functions as in chapter two, but modified to manage, control and validate telemetry packets received by a spacecraft constellation composed by three satellites.

In chapter five, we introduce the use of a plugin available on chosen software that allows user to create and manage warnings that inform and warn him about the state of subsystems.

In conclusion section, the results obtained are highlight, providing future advices to improve the software.

Chapter 1

Ground Control Stations Architectures

A basic element in space missions is the GCS, that consists of fixed and mobile ground stations, which acquire mission data from a spacecraft and its instruments and transfer it to the data users, supplying any telemetry and tracking information that users may need, and control center, which takes the main decisions for the mission.
The main features of GCS are:

- to support spacecraft and payload:
  - command and control them;
  - monitor their health;
  - track it to determine orbital position;
  - determine spacecraft attitude from sensor information;
  - process telemetry.
- to interact with the users:
  - to provide data,
  - to receive and control requests.

As described in Figure 1.1, users send command to the Space Segment through the Spacecraft and Payload Support, which receives telemetry and data about its overall health, which depends heavily on the state of individual instruments and systems.
The mission data sent by spacecraft are handled by Data Relay, which deals with distribute data to user community. Its handling capabilities are to multiplex and demultiplex, encode and decode, compress, store and file data, monitoring the quality.

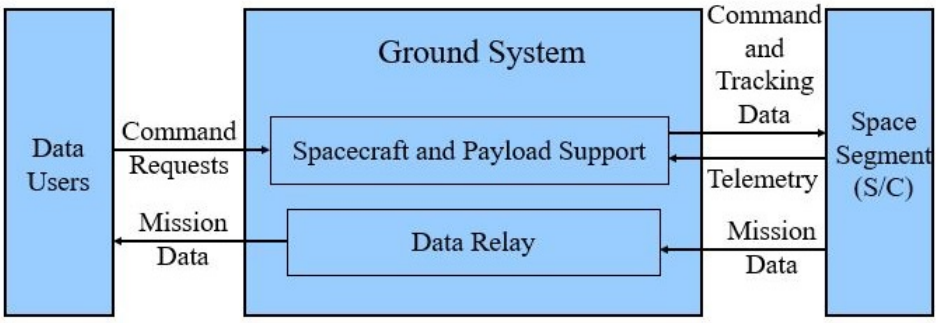


Figure 1.1: GCS Data Flow

1.1 Mission and Facility Elements for Ground System

The Ground System is characterized by two types of elements: mission and facility elements.

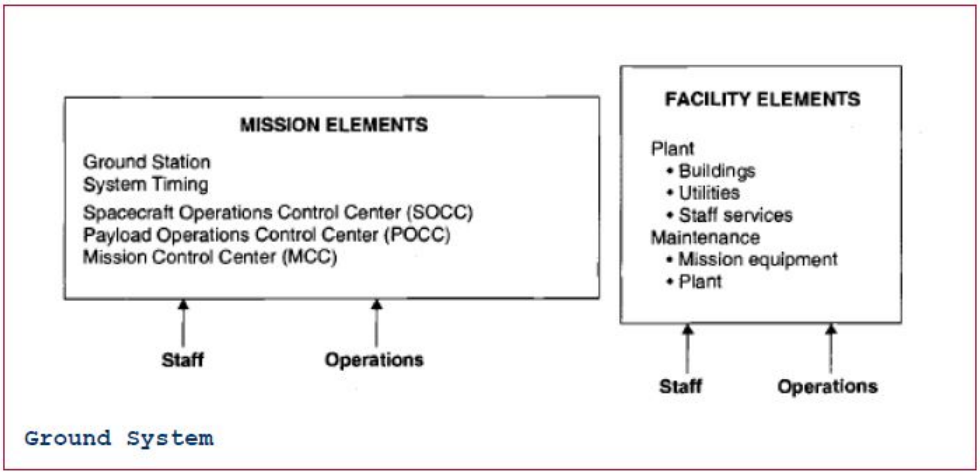


Figure 1.2: GCS Layout

The facility identifies components useful to support infrastructure, such as physical plant (buildings, utilities and staff services) and maintenance (mission equipment).
The mission elements consist of the following components:

- Spacecraft Operations Control Center (SOCC): this center monitors and commands the spacecraft bus and common systems, analyzing its telemetry and mission data. Its hardware includes data monitoring equipment and consoles, commanding facilities, and associated communications. This system is usually computer automated for quick response, but humans may intervene at any time to control the satellite.

- Payload Operations Control Center (POCC): it analyzes telemetry and mission data from onboard payload instruments and sends commands to them, with depend on approval by the mission control center.
- Mission Control Center (MCC): it plans and operates the entire space mission, including the configuration and scheduling of resources for both space and ground system. It, also, computes and issues information needed by ground system elements and data users.
- System Timing: it distributes clock time and frequency to the other system station elements and its accuracy is within millisecond or better.
- Ground Station (GS): it is a radio station designed for telecommunication with spacecraft.

While determining where to locate the ground system's components, it is important evaluate the communications options, because the GS require a long-distance communications links of sufficient bandwidth between their distributed elements and between them and the data users.

1.2 Typical Ground Control Stations

The Ground Control Station is the Earth based point of communications with the space segment for control and user data.

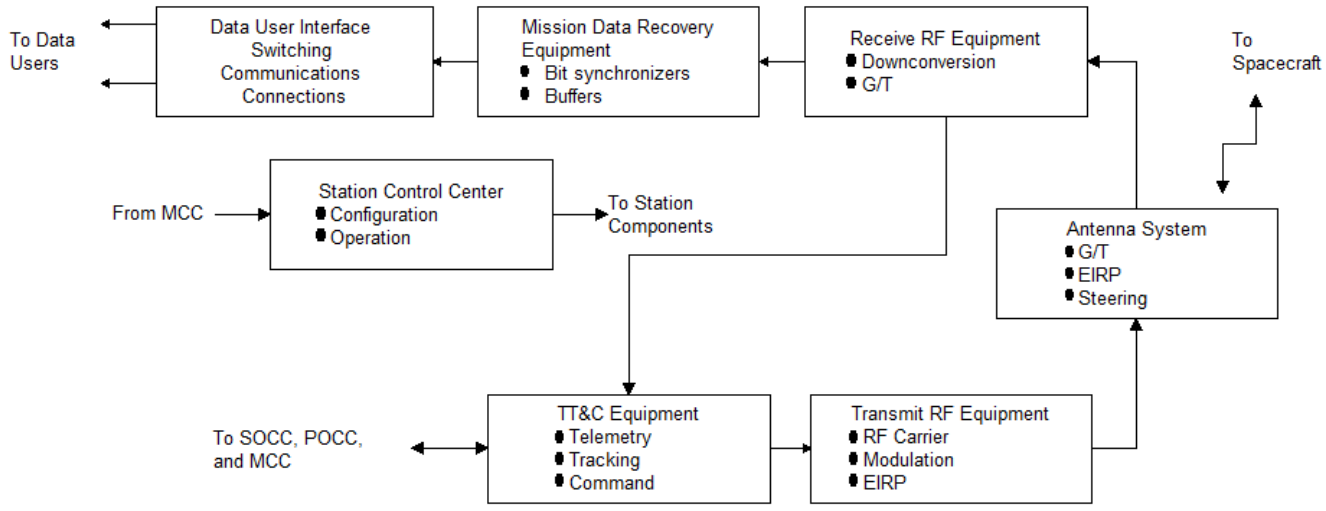


Figure 1.3: Ground Control Station Layout

Its components are:

- Antenna system, which includes: the antenna and mount, its associated electromechanical actuators, the consoles and servo circuitry which control the antenna, and the feed and transmission lines which carry RF signals to and from the RF equipment. The antenna should satisfy the following requirements:

  a) High gain for transmission and reception, requiring reflectors which are large in relation to the wavelength, and have high efficiency;

  b) Low level of interference (for transmission) and of sensibility to interference (for reception);

  c) Radiation with high polarization purity.

  d) a low sensibility to thermal noise due to ground radiation and various losses, for reception.

- Receive RF Equipment: it is located to minimize transmission-line losses to the antenna. This equipment accepts the downlink carrier frequency from the antenna system, downconverts it to intermediate frequencies, and demodulates it to baseband signals for the equipment devoted to mission data recovery and TT&C.

- Transmit RF Equipment: it accepts tracking and command signals from the ground systems TT&C component and modulates them onto the RF uplink.

- Mission Data Recovery Equipment: it handles the mission data before relaying it to data users and ground system components.

- Telemetry, Tracking and Control (TT&C) Equipment: it conditions and distributes received telemetry and tracking signals. It usually processes these tracking signals and data on the antenna-pointing angle to inform users about range, range rate, and spacecraft position. TT&C functions are usually highly automated because of the need for speed, timeliness, and accuracy.

- Data User Interface: it connects the data recovery equipment and the data user.

- Station Control Center: it controls the configuration of, and the interconnects between, the ground station components.

1.3 Dedicated vs Existing Ground System

Create a GS has the advantages that it is dedicated to a particular mission; it is compact and self-contained, allowing all communications between elements. To the other side, it does not allow to support more than one spacecraft link at a time because of the single antenna and ground station and it not allow to recovery data; in fact, if one item fails, we lose data. It has, also, the disadvantage that, to be secure, a completely collocated ground system should be on domestic soil.
To avoid these problems, it is possible to use existing ground support networks to supply part or all the elements needed.
This procedure allows to save and reuse resources, ensuring a certain level of reliability, because elements utilized are already tested.
Contrariwise, matching the user needs with the system may make the overall mission less effective. In fact, users must usually meet severe constraints to make their missions compatible with the host system and other user missions.
If we want to use existing systems, we must define the parameters mission requirements and then arrange them to the candidate host systems.
In evaluating service-provided ground systems, we must determine:

- how much equipment users must provide;
- how much access users have to ground stations or central distribution points;
- the costs host system compared to that of building and maintaining a dedicated system, both evaluated over the mission's lifetime.

1.3.1 Existing GCS

Nowadays, there are a lot of existing GCS. We decide to analyze GCS of student team around the world. The most important are described below.

Cal Poly

The PolySat is a student run, multidisciplinary independent research lab. Its primary objective is to give students hands-on experience designing, building, and operating satellites while promoting team-building across multiple engineering disciplines.

Hertz Station

This station is characterized by:

- A radio Yaesu 847 connected to a Yagi antenna of 2m and 70cm and an HF antenna;
- Antenna: it mounted on Rohn JRM23810 tower, which base is 5 ft per side and the center pole is 10 ft long. The rotor system is a Yaesu G-5500.

- Control Software: it consists of Raspberry Pi 2 running custom Satcom software and Windows 7 PC running SDR-Console/Server, remote accessible via VNC. Two Mac Mini's located in the main lab act as control terminals.

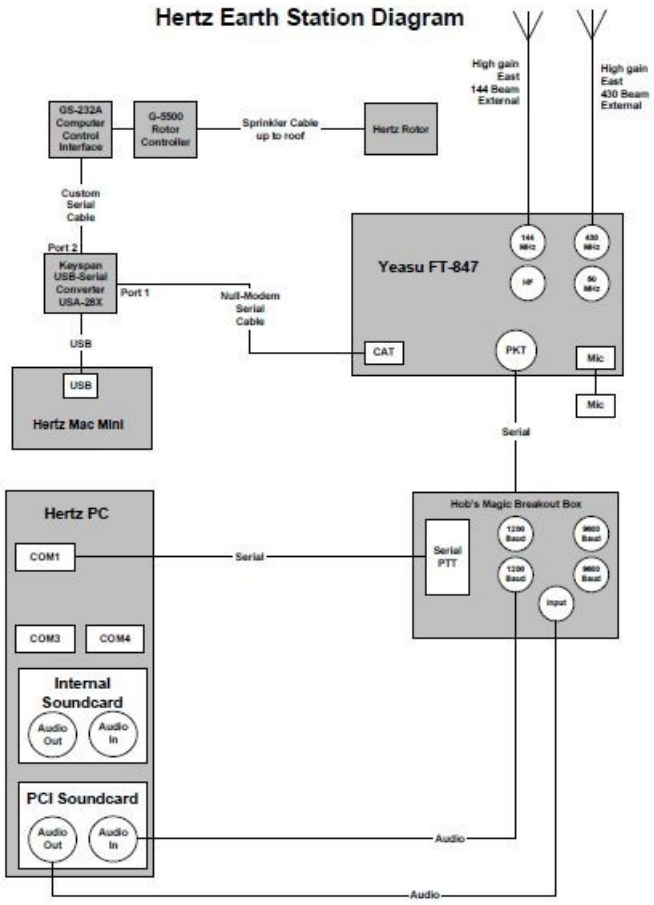


Figure 1.4: Hertz GCS Diagram

E-st@r GCS

The E-st@r program is an educational project under development at the Department of Aerospace Engineering of Politecnico di Torino. The program has been funded by the Italian Ministry of University and Research and by the Politecnico di Torino. E-st@r is a CubeSat, developed by students under the guide of researchers and professors, and it has been accepted by the European Space Agency (ESA) to be launched by the Vega LV during its maiden flight.
The project consists of a main GCS, located on radio amateur station, the ARI, section of Bra (Ham Radio Club IQ1RY),60 km south of Torino, which supplies all the elements needed to communicate with E-st@r satellites: it can send commands to the satellite and to receive the telemetry packets.
It consists of two stations includes an IC756-PROIII and a FT-1000 MARK-V transceivers connected to two amplifiers.
The antenna system is composed by a 16 elements Yagi for 10-15-20m (8el in 10m, 4el in 15 and 4el in 20m), two elements for 40m, 3 inverted V dipoles for 40m, 80m e 160m, one loop for 80m - all on the first tower - and by a 3 elements quad for 10-15-20m - and the second tower.

Figure 1.5: ARI HF Station [1]

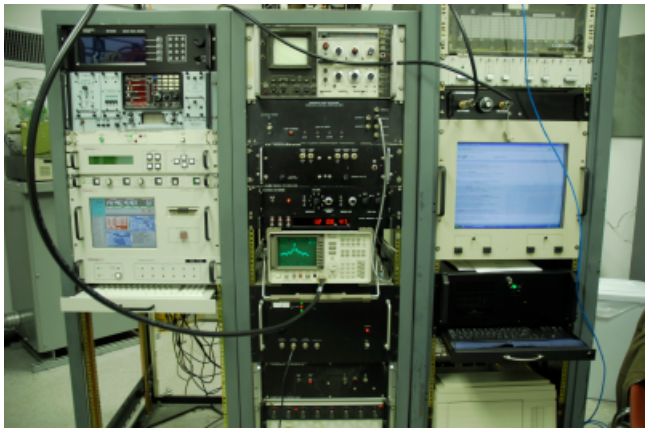
Morehead State University Ground Station

The Space Science Center (Department of Earth and Space Sciences) at Morehead State University (MSU), Morehead, KY (USA) operates a 21 meters diameter, full motion, research quality, parabolic dish antenna system, built under contract for MSU by Vertex-RSI, one of the premier fabrication corporations for high gain antennas. This system, referred to as the MSU 21 m Space Tracking Antenna, is engaged in ongoing research programs in radio astronomy and is also capable of operation in a satellite ground station mode, providing telemetry, tracking, and command (TT&C) services for a wide variety of satellite systems. This system has the capability of tracking fast moving, low transmitting power small satellites in low Earth orbit (LEO), as well as satellites at geostationary, lunar, and potentially Earth-Sun Lagrangian orbits.

The ground station provides a unique educational tool that serve as an active laboratory for students to have hands-on learning experiences with the intricacies of satellite telecommunications and radio astronomy.

The VHF/UHF satellite ground station is a stand-alone facility operated by MSU as the primary low bandwidth Earth station of the KySat project. This station is comprised of hardware and software that permits the auto-acquisition and tracking of low earth orbiting satellites for the purpose of command, communications and control. It is primarily built from commercial off the shelf hardware and uses readily available shareware for the tracking and communication portions of the software task. It uses a well-established architecture developed by the Amateur Radio Community for similar projects and makes use of amateur frequencies for the purposes of Earth-to-space and space-to-earth communications.

The basic 21 M performance characteristics are described in Table 1.1.



(a) The MSU 21 M Space Tracking Antenna Control Room

(b) The MSU Space Science Center 21 M Space Tracking Antenna, Morehead, KY

Table 1.1: 21 M Characteristics

Function	**Performance**
Antenna Diameter	21 Meter
Receive Polarization	RHCP, LHCP, VERT, HORZ
Travel Range	AZ +/- 275^o from South EL -1^o to 91^o; POL +/- 90^o
Velocity	AZ Axis = 3^o/sec EL Axis = 3^o/sec POL Axis = 1^o/sec
Display Resolution	AZ/EL = 0.001^o POL = 0.01 ^{o}
Tracking Accuracy	<= 5% Received 3dB Beamwidth (0.028^o RMS L-Band) (0.005^o RMS Ku-Band)
Pointing Accuracy	<= 0.01^o

ALMA MATER Ground Station

ALMASat (ALma Mater SATellite) is a student project of the School of Engineering and Architecture of the University of Bologna, that aims to develop compact space systems, characterized by high levels of autonomy and innovation, through the use of advanced technologies in the field of electronics, micromechanics and telecommunications.
The first satellite design, ALMA Sat-1, has been successfully launched in Low Earth Orbit (LEO) in February, 13 2012 onboard the VEGA Maiden Flight from Europe's Spaceport in Kourou, French Guiana. It is a cubical prism, 300mm side, with mass about 12.5 kg. The main objective of this first mission was to test the key performance of all microsatellite technologies developed in the lab to prepare for future missions dedicated to Earth observation which requires a high 3-axis pointing accuracy.
To control this satellite, a VHF/UHF Yagi amateur radio, fully automated ground station for communication with LEO-satellites has been installed.
This station consists of:

Figure 1.6: ALMA Ground Station Antenna

- VHF antenna: it is a 2x9 elements with a gain of 13.20 dB;
- UHF antenna: it is a 2x19 elements, with a gain of 16.20 dB;
- two VHF and UHF preamplifiers are placed along the transmission line, between the antennas and the transceiver;
- Two AlfaSpid rotors to control azimuth and elevation;
- ICOM IC-910H transceiver, a full-duplex system for data and voice transmission at VHF and UHF bands;
- a DC Regulated Power Supply, which power Most components of the ground station.

AAU CubeSat Ground Station

AAU-CubeSat(Aalborg University Student Satellite) project has the aim to let students build and launch a nano-satellite with the purpose to provide for "hands-on" education and gain experience with nano-satellite technology. The satellite was launched on the 30th of June 2003 from Plesetsk in Russia on top of the Rocket Launcher.
The AAU-CubeSat ground station consists of:

- Two Antennas WX-706 from RF-connections are on top of University. They are a 2x18 element crossed Yagi directional antennas, with a gain of 14 dB.
- Double Yaesu G-5500 motors, to allow the antenna array to follow the satellite across the sky.
- ICOM IC-910H radio transceiver, which can work in the 144 MHz band as well as the 430 440 MHz band.
- A level converter design by student to connect radio and RS-232 serial port on computer.
- 15 meter of 1mm 50 Ω coaxial cable between the antenna and the radio, with a dampening of 8 dB/100m.
- SP-7000 signal amplifier with signal gain of 20 dB when receiving, and can withstand transmitting up to 300 Watts in the FM mode.
- MX909 modem and to control is a C167 MCU from Infineon mounted on a MCB167 evaluation board from Keil. The data comes from the radio and is demodulated in the modem and then the MCB167 receives the data from the modem over a parallel connection. It them relays the data to the ground station pc via a serial connection.

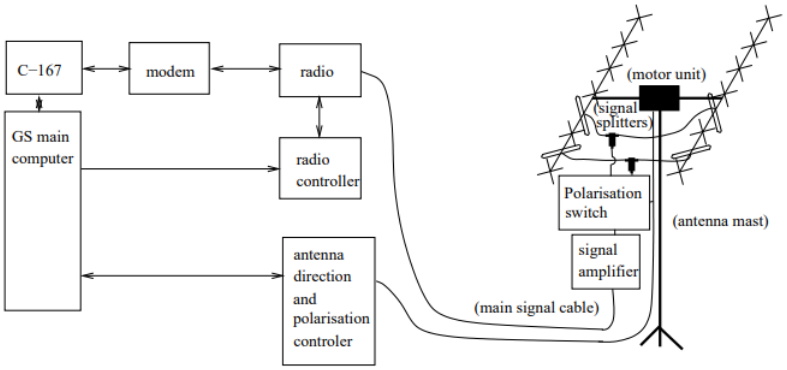


Figure 1.7: Aalborg University's GCS hardware configuration

GENSO Ground Station Network

The Global Educational Network for Satellite Operators (GENSO) is a software package for ground station computers which aims to drastically increase a satellites availability to control operators.
Its first goal is increasing the amount of data that can be downlinked from a particular satellite by increasing the available pass times. This aim can be achieved by complete automation of existing ground stations to allow unattended operation, and by building new ground stations in desirable locations around the world. Another main goal of the project is the continuing education of students in the radio amateur and the satellite construction realm. The GENSO network has three distinct components:

- The Authentication Server, which authenticates nodes on the network and distributes spacecraft and ground station lists.
- The ground station Server: it is the piece of software that moves the rotors and tunes the radio and it is located at the amateur radio station. When a GSS wants to get onto the network, it contacts the Central Server, from which gets a Participating Spacecraft List (PSL), a list of spacecrafts on the network, their status, and the frequencies, modes, and Keplerian elements. Also included on this list, is the encryption keys of the spacecrafts MCCs, so a GSS can contact MCC directly to transfer downlinked data.
- The Mission Control Client (MCC): it is the application where the satellite operator can control the network's handling of a satellite. It also has a prediction section where the satellite operator can predict which ground stations their spacecraft will pass over, and contact the appropriate GSS for booking passes.

GENSO will handle two different types of satellite passes. Passive passes involve only the downlink of telemetry (RX) and can be scheduled autonomously by any ground station. The downloaded data is cached in the ground station and forwarded to the mission control client on request. Active passes involve both RX ant the uplink of telemetry (TX) and have to be requested by mission controls and negotiated between mission controls and ground stations.

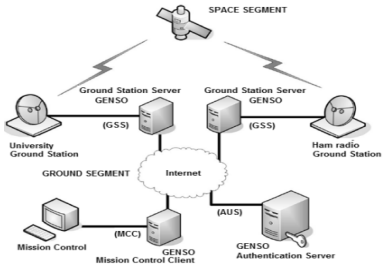


Figure 1.8: Architecture of GENSO Network

1.4 Software for Operation Tracking

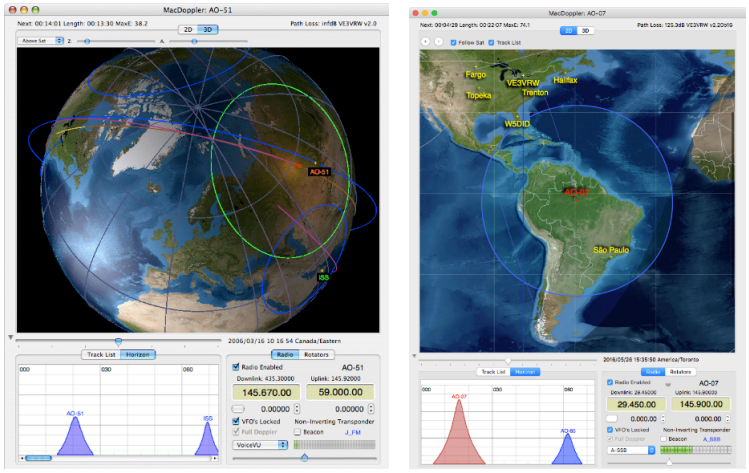
Typically, the ground station integrates tracking , professional software, such as *MacDoppler, Nova, InstantTrack* or open source software, such as *WxTrack, Orbitron.*
These software allow users to have information about satellite position at any time.

1.4.1 MacDoppler Ham Radio Satellite Tracking

MacDoppler is a software that provides any level of station automation user needs from assisted Doppler Tuning and Antenna Pointing right on up to fully automated Satellite Gateway operation.
It is used around the world by Amateur Radio operators, satellite spotters, educators and commercial customers from CBS News to the International Space Station Amateur Radio Hardware Management program, Delta Telemetry Tracking and Control at Boeing Integrated Defense Systems, Florida State University, and the CalPoly CubeSat Project.
It supports the following transceivers:

- YAESU FT-847, FT-736R;
- ICOM IC-970, IC-820, IC-821H, IC-910, IC-9100;
- Kenwood TS-2000, TS-790A/E;
- FlexRadio Signature 6000 Series;
- UDP Broadcast.

This software allows user to track satellite in two ways: through a 2-D or 3-D maps (Figures 1.9 (a) and (b)), providing a list of all tracked satellites, with passes time and Keplerian elements. The display can be filled with Horizon Panel, which plots the tracked satellites upcoming elevation in the Y axis (0-90) against time in minutes in the X axis, and Radio Panel which controls the connection to the radio interface.



(a) 3-D Map (b) 2-D Map

Figure 1.9: MacDoppler Software

1.4.2 WxTrack

For E-st@r mission, WxTrack is used; it is an open source software designed to predict the tracks of satellites both as paths above the Earth, and as images produced by these satellites when scanning the ground.

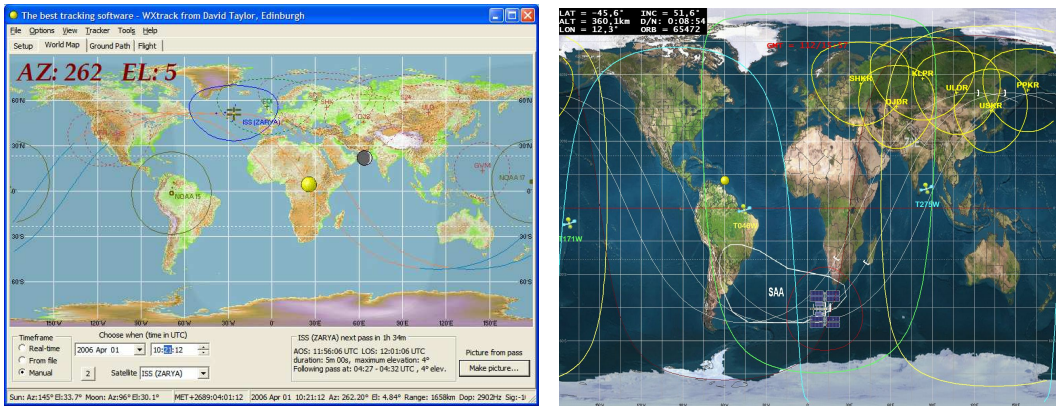


Figure 1.10: WxTrack Examples

The Figure 1.10 shows WxTrack displaying the positions of more satellites. One satellite has been selected as primary - the International Space Station (ISS) - and this is highlighted with its visible footprint being a dark-blue colour, showing its ground tracks.

1.5 Software for Operation Management

With the rapid growth in the number of space actors, there has been a marked increase in the complexity and diversity of software systems utilized by big and small companies to control Nano Satellites.
The software of our interest are that preform a MCS (Mission Control Software), that executes the following main functions:

- Reception, acquisition, packet extraction, and validation of telemetry data;
- Telemetry analysis for status checking, limit transgression, and visualization of data received;
- Process telecommand to control the spacecraft.

Many software of this kind have been created and commercialized with "closed" code, which limits interoperability, inhibits the code transparency that some developers need to modify software as they want.
These software have a cost related to license, that, in many cases, can be:

- Complete version: this license allows user to utilize the software in any kind of domain, but its cost is higher;
- Educational version: the license's cost is lower than the previous one, sometimes free, but user can use all functions of software and it can't be used in industry and commercial projects.

For these reasons, customers are induced to switch their attention to open source software. Open source aerospace software is based on developers collaboration, which has the potential to bring greater transparency, interoperability, flexibility, and reduced costs. It is easily adaptable, a feature useful for the rapidly changing mission needs, and can generally be delivered at lower costs to meet mission requirements.

For these reasons, bibliographic analysis is conducted on open source software.
Only three software are interesting and present all the base requirement exposed before.

1.5.1 NOS^3:NASA Operation Simulator

NOS^3 is a software developed by Simulation-to-Flight 1 (STF-1) team, which was selected for participation in NASA's CubeSat Launch Initiative, in February 2015. It aims to demonstrate the technologies' flexibility and effectiveness on mission using CubeSat Platform. This software aims to simulate a complete mission, through mission operation/training, verification and validation, test procedure development. It is composed by other software that develop every single part of simulation, such as:

- NASA Operational Simulator (NOS): it simulates hardware busses to connect the flight software and the simulated hardware components.

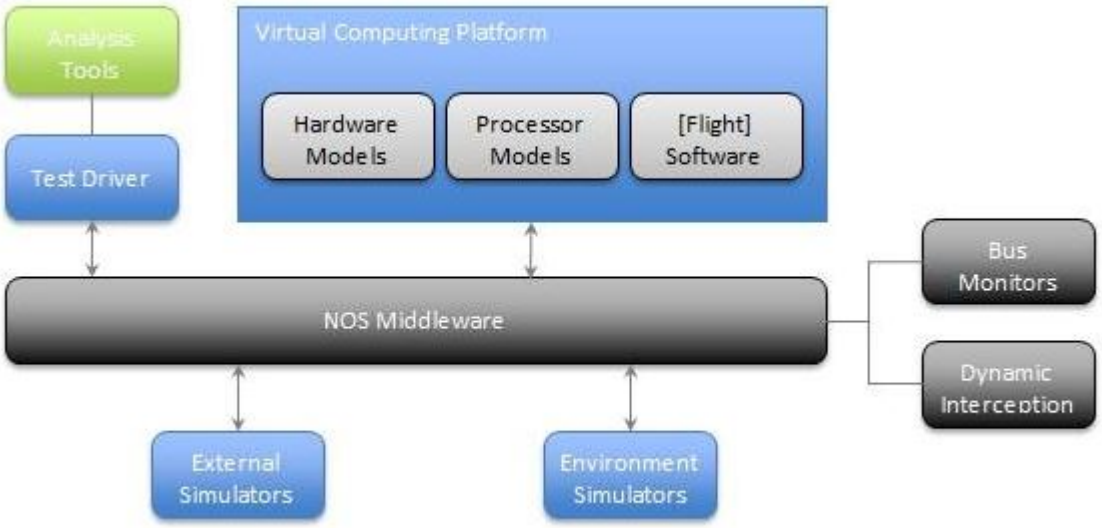


Figure 1.11: NOS Architecture

NOS provides the capability to test the flight software without the presence of actual spacecraft sensors, actuators, and instruments. The simulation environment runs the flight software in a CPU emulator alongside custom hardware models. This allows the simulation to run in real-time or faster than real-time environments.

- Core Flight System (cFS): it is a platform and reusable software framework. It aims to reduce time to deploy high quality flight software, project schedule and cost.
- Hardware Simulators: they are software models of a specific piece of flight hardware to provide the flight software with an accurate representation of input/output data.
- Vagrant: it allows a computer to set up the Virtual Machine necessary to run the applications associated with the NOS.
- Orbit, Inview and Power Planning (OIPP): it is a planning tool developed to know when satellite will be in view and in direct sunlight to plan satellite power usage and communication times.
- 42: it is an open source simulator of spacecraft attitude, orbit dynamics and environmental models. The simulator is developed to support the GNC design cycle.

Figure 1.12: OIPP Timeline Example

This is an open source software, but, currently, it is in phase of upgrade and it is not available [2][3][4].

1.5.2 COSMOS: Comprehensive Open-Architecture Space Mission Operation System/Support

This software, developed by Hawaii Space Flight Laboratory (HSFL) at the University of Hawaii at Manoa, aims to support the development and operations of one or more small spacecraft. It merges software and hardware tools that enables the operations team to interface with the spacecraft, ground control network, payload and other customers to perform the mission operation
functions including mission planning and scheduling, data management and analysis, simulations and flight dynamics.
It is composed by a tool that simulates the ground station, providing information about spacecraft position, coverage angle and communications monitoring.
COSMOS will be particularly suited for small operations teams with a very limited development and operations budget, such as universities.
The central pieces of this architecture are the visualization tools, support tools, and underlying programs that produce and manipulate the data needed by the rest of the tool sets. COSMOS allows to perform the following functions: mission planning and scheduling; contact operations; data management, mission analysis; mission state projection; simulations (including the operational testbed); ground network monitoring and control; payload operations, flight dynamics (including orbital and attitude); support of system management and quality assurance.

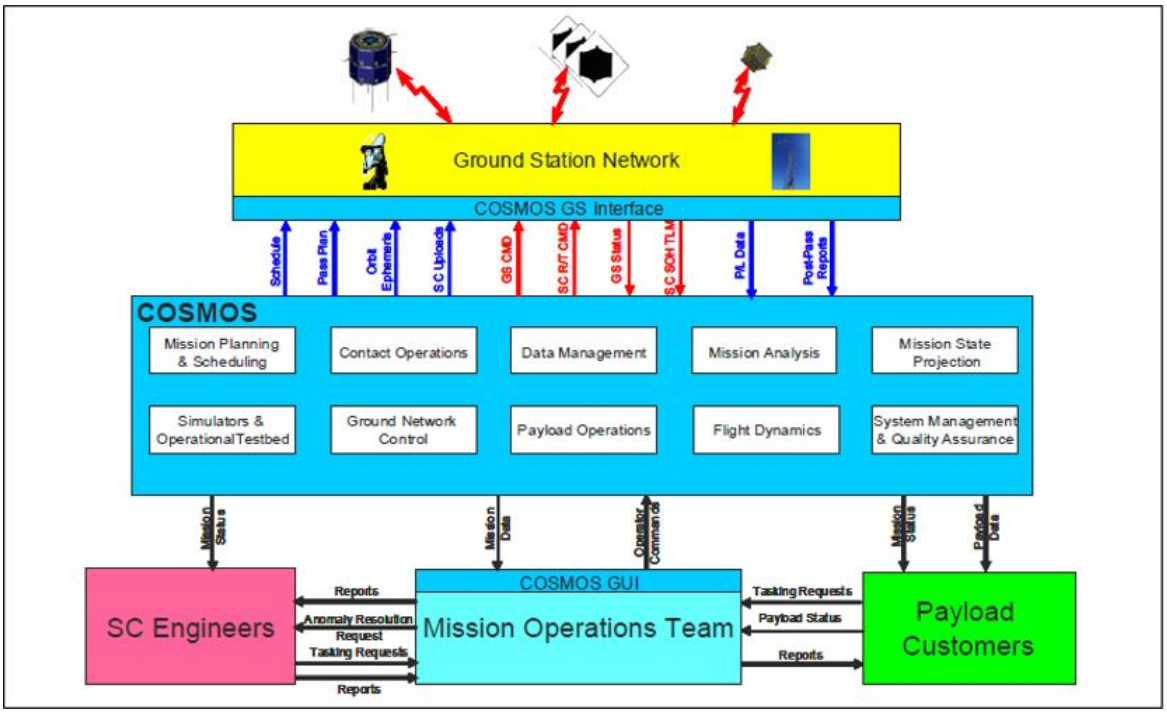


Figure 1.13: COSMOS Functional Architecture

COSMOS support the following major processes in mission operations:

- Mission Planning and Analysis which includes command sequencing and the simulators;
- Contact Operations which includes pre-contact, real-time and post-contact operations.
- Data Management which includes data processing, data archiving and data transfer through COSMOS and external users;
- Mission Analysis which analyze spacecraft and ground network state-of-health(SOH) data, orbital and trajectory data.

This software includes all requirements needed for this thesis work, but the source code is not easy to find and download. In addition, its graphics is bed and it isn't easy to use to not expert users [5][6][7].

1.5.3 Open MCT: Open Source Mission Control Technologies

Open MCT is a next-generation mission control framework developed at NASA's Ames Research Center in Silicon Valley, in collaboration with the Jet Propulsion Laboratory.
It is being used by NASA for data analysis of spacecraft missions, as well as planning and operation of experimental rover systems. As a generalizable and open source framework,

Open MCT could be used as the basis for building applications for planning, operation, and analysis of any systems producing telemetry data.
Open MCT is designed to meet the rapidly evolving needs of mission control systems. It is developed to support distributed operations, access to data anywhere, data visualization for spacecraft analysis and flexible reconfiguration to support multiple missions.

The core of Open MCT can be customized with plugins to support the specific needs of missions. It provides an extensible public API to enable the development of new visualizations, integration with telemetry sources, and other new features.
The source code is based on Node.js language and can be modified as the user wants.[5]

Node.js is a free open source server environment that runs on various platform, such as Windows, Linux, Unix, Mac OS X; it uses JavaScript on server.

Open MCT is client software: it runs in a web browser and provides a user interface, while communicating with various server-side resources through browser APIs.

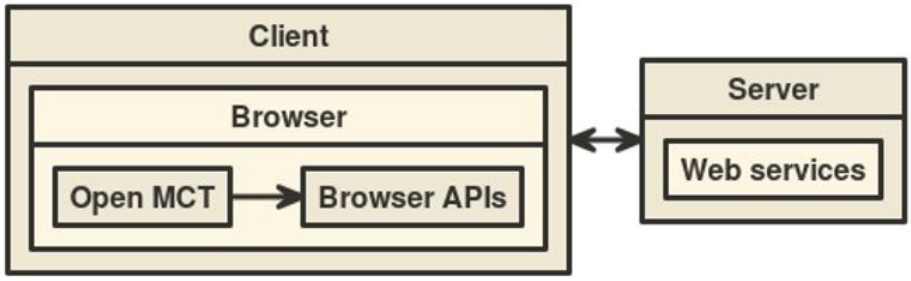


Figure 1.14: Open MCT Software Architecture

This software is composed by a "layered" architecture, where each layer specifies the behaviour of the software. These layers are:

- Framework: This layer is responsible for managing the interactions
  between application components. The software components it recognizes are:
  - Extensions: Individual units that can be added to or removed from Open MCT.
  - Bundles: A grouping of related extensions that may be added or removed as a group.
- Platform: The platform layer defines the general look, feel, and
  behaviour of Open MCT. This includes user-facing components like Browse mode and Edit mode.
  The Open MCT platform utilizes the framework layer to provide an extensible baseline for applications which includes:

  - A common user interface for dealing with domain objects of various sorts.
  - A variety of extension points for introducing new functionality of various kinds within the context of the common user interface.
  - A service infrastructure to support building additional components.

- Application: The application layer defines specific features of an application built on Open MCT. This includes adapters to specific back-ends, new types of things for users to create, and new ways of visualizing objects within the system.

The source code is available at the following link presents in bibliography [8].
There are a lot of tutorials that user can follow and do for move his first steps on Open MCT [9][10][11].

Figure 1.15: Open MCT Starting Window

1.5.4 Chosen Software: Open MCT Description

Once the bibliographic analysis has been carried out, it is necessary to establish guidelines for the selection of the software; firstly, it is necessary to understand if it is more convenient to use an existing software or to develop a new, own software that performs all requirements needed.
Both aspects have advantages and disadvantages.
For existing software, the advantages are the reliability of this software, because it is tested before in past missions, giving a high level of confidence; it reduces the working time and efforts of developers and therefore the reduction in costs; furthermore, it minimizes the time of learning and training of users.
The disadvantages are that software don't present a specific tool required for some project.
On the other hand, the choice to design a software "in-house developed" has the advantage of creating functions that allow to respect all the requirements and needs of the mission, without any restrictions, but this needs a lot of use of time for developers and a long process of validation of each single part.
The choice to use an existing open source software has the advantage of not starting from the base in software programming and to avoid validation times of basic functions.
The software chosen must contain primarily basic tools able to manage telemetry data and tools that provide a graphic view of data.
For these reasons, we have decided to use Open MCT, an open source software, which presents a graphical interface that allows the user to control subsystems; in fact, through a plugin, the user can place telemetry data as he wants in order to control and check information all together.

Installation

To build Open MCT from source code, the user must install prerequisites, such as GIT and Node.js.
Git is a free and open source distributed version control system designed to handle from small to large projects with speed and efficiency. It is the source control system of Open MCT and require a git user.
Node.js is a free and open-source, cross-platform JavaScript run-time environment that executes JavaScript code server-side. Its network model is an event-driven; this means that Node.js requires the operating system to receive notifications when certain events occur.
In Open MCT, Node.js is used to build the source code, and npm is used to manage build dependencies.

Open MCT code can be downloaded and built using the steps below, in a command window:

- Clone the source code: **git clone https://github.com/nasa/openmct.git**

- Install development Dependencies: **npm install**
- Run a local development server: **npm start**

Starting with an empty directory, these steps will check out Open MCT from GitHub, build it, and run a local web server.

How Use Open MCT

After installation, to use Open MCT, user must run the code always in command window, as shown in Figure 1.16:

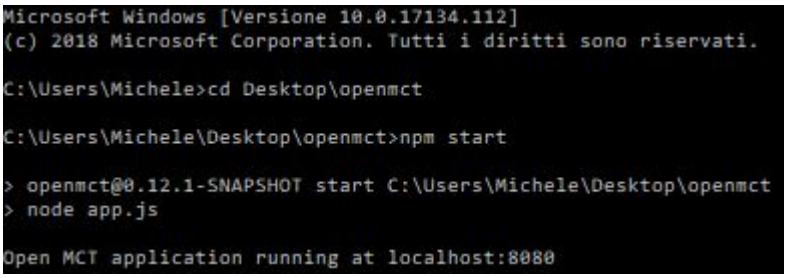


Figure 1.16: First Step to Run Open MCT

Going on folder where code has been installed, user inserts "npm start" command to run the software.
Opening the web site "localhost:8080" on browser, the software appears as shown in the figure below:

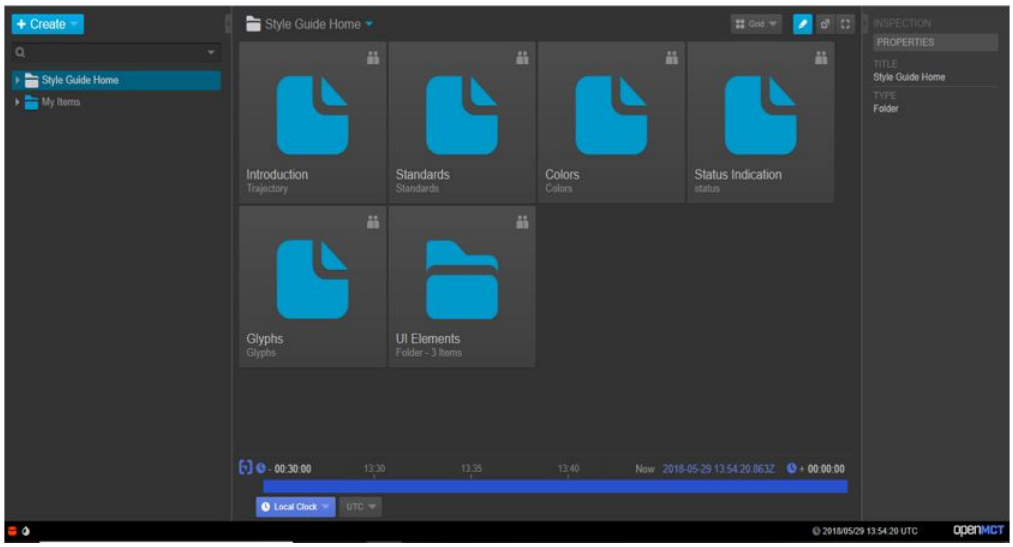


Figure 1.17: Open MCT Default Window

To the left side of the display, there is the "object tree", which contains all the objects user have access to, including telemetry objects, prebuilt displays and objects created by himself.
The center area, View Area, shows the contents of a selected item.
To the right side, there is a section that exposes useful information about selected item, such as title, type of object and its location in the software.
Furthermore, user can create an own object in two ways: clicking on button "Create" or writing an own code.
The first method is the faster and easier to do; in fact, user creates and modifies the object without open or write any line of code.
The second process requires user capabilities of write code lines and knowledge of JavaScript language. In addition, user must know how the core of Open MCT operates and where add or modify the code without compromising the correct functioning of the software.
The default objects that software allows to create are:

- Folder: user can create a default folder to conceptually organize objects;
- Telemetry Panel: creates plot with telemetry data;
- Telemetry Table: creates telemetry data ordered as a list;
- Display Layout: user can combine objects that display information as he wants;
- Timeline: user can create a timeline mission, itemizing phases mission and duration of each.

Through these plugins, user can start his first step to use Open MCT.
A detailed tutorial is available at the link in bibliography [12].
For this thesis work, the second method has been preferred, because it make available all changes, objects and sections that user creates on any browser he decides to use, not only on which he utilizes with the first methodology.

Chapter 2

Software Setting for Communication with One Satellite

2.1 Workflow Logic

In the preliminary phase of the thesis, we decide a guide line of all work to do to reach the final goal.

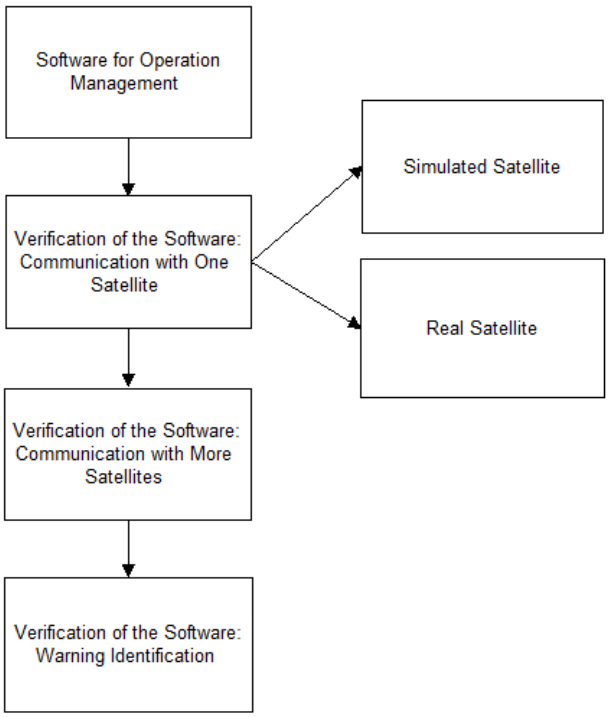


Figure 2.1: Thesis Work Flowchart

As shown in Figure 2.1, we start choosing an existing software for operation management; it has been modified on code level to respect the following requirements.

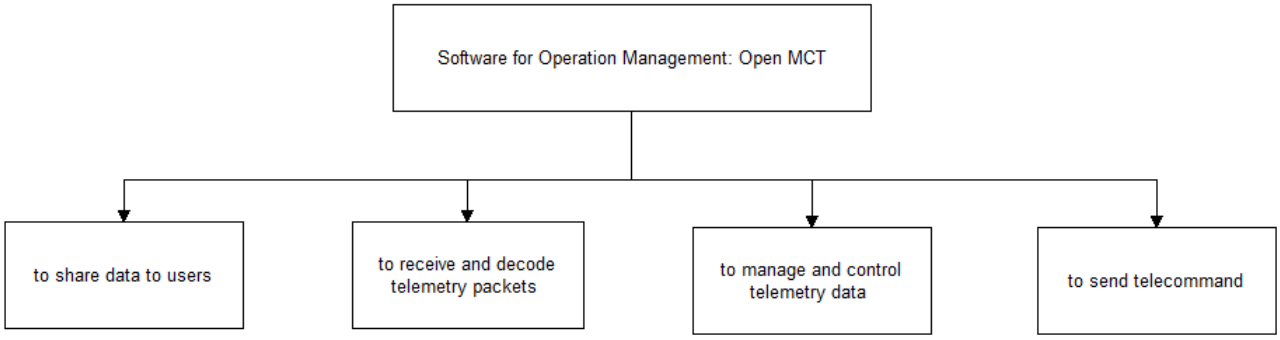


Figure 2.2: Open MCT Code Changes

- **to share data to users**: this requirement is respected creating a specific section for each subsystem that produce telemetry; decoded data fill these sections and user can visualize them through telemetry plot and table, plugin available on Open MCT;
- **to receive and decode telemetry packets**: the reception of data is possible through a developed C executable program, which reads and saves data received via serial communication in "data.kss" file.
  An encoding algorithm decodes data saved in this file, validates and processes them to guarantee that are visualized in correct format;
- **to manage and control telemetry data**: additional functions developed allow user to choose if he wants analyze telemetry packet that respects a determinate length inserted by himself, or all packets received; on Open MCT interface, user can check the status of health of each subsystem through warning plugin;
- **to send telecommand**: we modify and improve C program to allow user to send telecommand to spacecraft.

We proceed to test changes applied through the reception, control, validation and management of telemetry data sent by satellite.
Telemetry is a term that identifies the housekeeping data collection that provide the health status of the spacecraft, in the form of measurements such as electrical data (voltage and current) or physical data (temperature or pressure). These information are gathered, processed and formatted and then sent to MCC, that receives, validates and decodes packets.

This phase consists in elaborating and designing additional functions that allow to receive,

decode and visualize telemetry data. To verify them, we analyze telemetry packets received by E-st@r 2 spacecraft, checking the correct format of packets received, protocol and length, and then comparing with result analyzed by CubeSat Team.

The other parts of workflow are described in detail in next chapters.

2.2 Development and Preliminary Test of Additional Function

Typically, satellite elaborates, manages and packets subsystems telemetry data and, when it is in line of sight with the GS, sends telemetry through packets.
For this reason, it is necessary to know how satellite packets are composed and which telemetry data are included.
To do this, it is necessary introduce additional functions on code level that must respect the following requirements:

- High Flexibility: each function must have the ability to adapt to possible changes to verify different configurations.
- Easy to Manage Data: the functions must be simple and clear because user must understand what he does without any problems or errors; furthermore, those functions that manage data, must make view data clear and place data correctly.
- Effectiveness of Data: it is a fundamental feature to identify input data (which and when spacecraft sends packets). The outputs must be also clear and in correct format.
- Real Time Execution: it means that function for this type of analysis must guarantee a certain precision in the execution of code. For telecommands, the code must process telemetry data, identify spacecraft which sends packets and asks user if he wants to send command and, if yes, sends telecommand, all in 6 seconds (see Section 2.4).

This additional function must be validated, to guarantee the integrity and correct operation of the software.
The first function we add is "updatespacecraft1", whose basic line is explained in the flowchart in Figure 2.3.
This function allows to received telemetry packets, validate and, through an encoding algorithm, obtains decodified data to visualize on Open MCT, filling subsystems subsections.
Each part is detailed in the next paragraphs.
To verify this test, it is necessary that pass/fail criteria are respected. These are:

- the software receives at least 50 of 445 packets that we analyze;
- packets received are correctly decoded;
- telemetry data are visualized on the software in correct format.

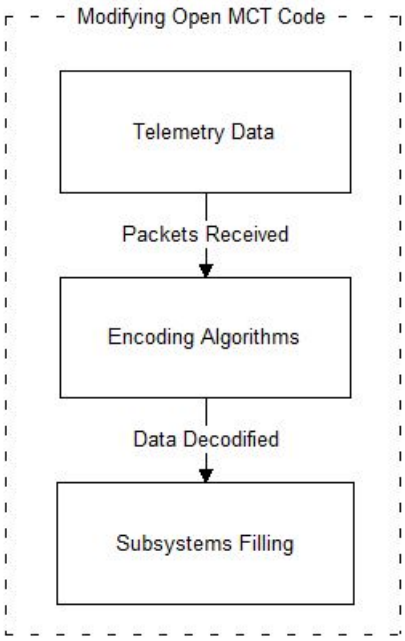


Figure 2.3: First Function "updatespacecraft1" Flowchart

2.2.1 Telemetry Data

E-st@r 2 sends telemetry data in packets, a string of characters long 111 chars. Each char identifies a single value that must be decoded to extract and visualize data in format required. To do this, it is necessary primarily identify which subsystems produce data and, then, proceed to create a section on the software for each of them.

Subsystems Identification

On E-st@r 2, subsystems that generate telemetry are: Batteries 1 and 2, Solar Panels, Bus 5 V, Bus 3.3 V and Bus Batteries. Each subsystem generates three type of data, such as: voltage, current and temperature.
To introduce these subsections on Open MCT, we create a JSON code "dictionary. json" (Appendix A), in which we design object tree.

In the Figure 2.4, there is an abstract and Figure 2.5 shows the object tree on Open MCT.

```
{
    "name": "Telemetry",
    "identifier": "sc",
    "subsystems": [
        {
            "name": "General Satellite 1",
            "identifier": "gen",
            "measurements": [

                {
                    "name": "Destination Address",
                    "identifier": "gen.da",
                    "units": "None",
                    "type": "None"
                },
```

Figure 2.4: Dictionary.json Abstract

Figure 2.5: Telemetry Folder on Open MCT

"Telemetry" identifies the folder where user can see telemetry data, which is subdivided into macro sections:

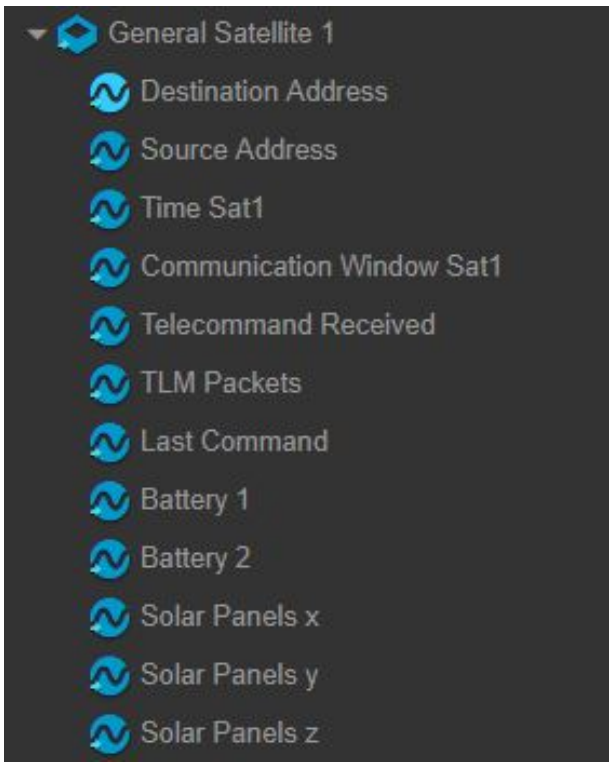
- "General Satellite 1": it identifies generic satellite data, such as:
  - "Destination Address": it represents the first part of telemetry packet which identifies to who the satellite sends telemetry;
  - "Source Address": it identifies satellite address that sends data;
  - "Time": it shows mission duration from the first communication;
  - "Communication Window": it shows when satellite sends data to the ground station;
  - "TLM PCKs": it is the telemetry packets' number sent;
  - "Last Commands": it identifies the last command received;
  - "Battery 1 & 2": it represents battery voltage values;

  - "Solar Panels x,y,z": it shows solar panels voltage values in the three-direction x, y and z;

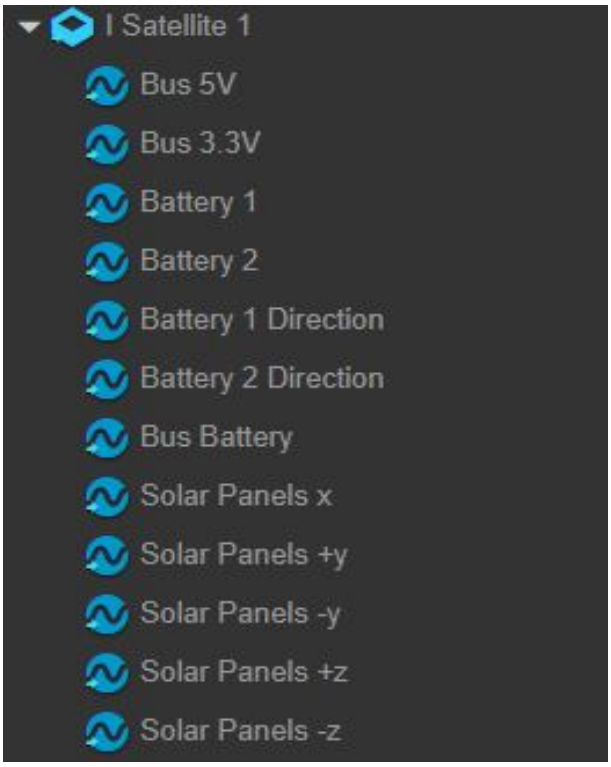
- "I Satellite 1": it shows subsystems electrical current values in the section below:
  - "I Bus 5V";
  - "I Bus 3.3V";
  - "I Batteries 1 & 2";
  - "Batteries 1 & 2 Direction": which shows if butteries are charging or discharging;
  - "I Bus Battery";
  - "I Solar Panels x,+y,-y,+z,-z".

- "T Satellite 1": this section shows subsystems temperature:
  - "T Batteries 1 & 2";
  - "T Solar Panels x,+y,-y,+z,-z".

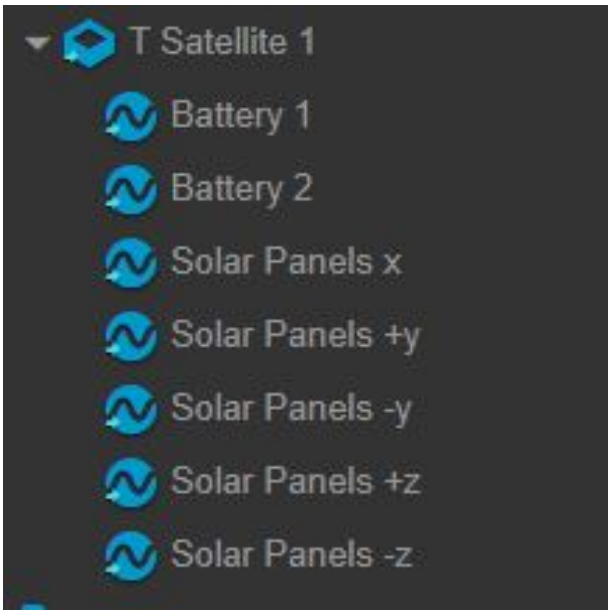
For each data, we must indicate the name, the code identifier, the unit (V, mA, None for string type) and the type, the format of value (string, float, integer, etc).
Each telemetry data is visualized through a basic plugin of Open MCT, a telemetry panel that collects each telemetry data and visualizes it, through a plot or table.
Figures 2.6 (a) shows how telemetry panels of each section are visible on Open MCT.



(a) General Data Section



(b) Current Data Section



(c) Temperature Data Section

Figure 2.6: Telemetry Data Sections

Data.kss and Test 445 Packets

During the first test, we try out the software with a list of telemetry packets, simulating the spacecraft that sends them, on the same computer of the software. We create a file "data.kss" that, second by second, contains each packet decodified through an encoding algorithm.
To start the work, we decide to analyze telemetry packets received by E-st@r 2 during a preliminary phase test, continued for a day. The packets are collected every two minutes. For the first analysis we decide to include packets from 333 to 778, in total 445.
Each packet consists in a string of characters, long 111 chars. An example string is below:

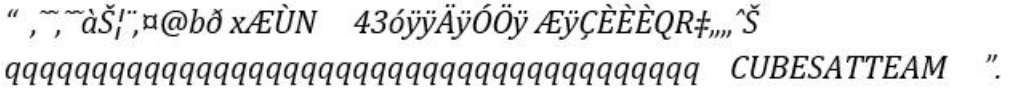


Figure 2.7: Example String

For this thesis work, we simulate that satellite sends telemetry packets approximately each 24 seconds.
We simulate satellite on the same computer to test firstly the software. To do this, we develop a C executable program that reads each packet, decodes it by chars to decimal representation and puts modified string in "data.kss"; it is a dynamic file that is overwritten whenever is received a new packet.

2.2.2 Encoding Algorithm

To decode packets, it is necessary an encoding algorithm that extracts each char by the packet, decodes it and sends data to the software.
To aim this scope, it is necessary understand the protocol through which E-st@r 2 sends data.
A protocol is a convention or standard that controls or enables the connection, communications and data transfer between two computing endpoints. It is the rules governing the syntax, semantics, and synchronization of communication.
E-st@r 2 protocol is AX.25 UI-FR AX.25 UI-PACKET FORMAT AME FORMAT.
This protocol consists to generate telemetry packets as described in Table 2.1.

Table 2.1: Protocol AX.25 Rules

Destination Address	Source Address	Control Field	Protocol ID	Info Field	FCS
7 bytes	7 bytes	1 byte	1 byte	1-255 bytes	2 bytes

The first 7 bytes represent the destination address, the identification to who telemetry packets are sent; when decoded, they are:

{ ,˜˜,˜˜à } => { ALLALLp }.

The following 7 bytes, source address, represent which satellite sends data;

{ Š¦¨,¤@b } => { ESTAR 1 }.

The Control field identifies the type of frame being passed;
Protocol ID identifies which protocol is used.
Info Field bytes represent information, data collected, and could be from 1 to 255 bytes.
The Frame-Check Sequence (FCS) is calculated both sender and receiver to ensure that the packet is not corrupted.
The end of the packet is identified by the following string:

{ CUBESATTEAM }.

Known the protocol, we can proceed to create the encoding algorithm.
This process is exposed in “app.js”, where we create a JavaScript code that for each packet, reads char in decimal representation, and decodes telemetry data.
In the figure below, it is shown an example of how to decode telemetry data that identifies the time when satellite sends packet from the start of simulation.
Time is identified by four characters that must be decoded in decimal representation, from which we obtain hours, minutes and seconds.

```
if(i>15 && i<21){//TIME

        t[k]=d;
        k++;
    if(k==4 ){
        var tempo,h,m,ss;

        tempo=(t[0]*Math.pow(256,3)+t[1]*Math.pow(256,2)+t[2]*256+t[3])/100;

        h=Math.floor(tempo/3600);
        m=Math.floor((tempo-h*3600)/60);
        ss=Math.floor(tempo-h*3600-m*60);
        console.log("Tempo: " + h+":"+m+":"+ss);
        spacecraft["gen.tem"]=tempo+ "  "+h+":"+m+":"+ss;

        k=0;
    }
}
```

Figure 2.8: Time Decoding Example

To execute "app.js", user must follow these instructions:

- Go to tutorial-server folder in Open MCT location on command window;
- Digit **node app.js**.

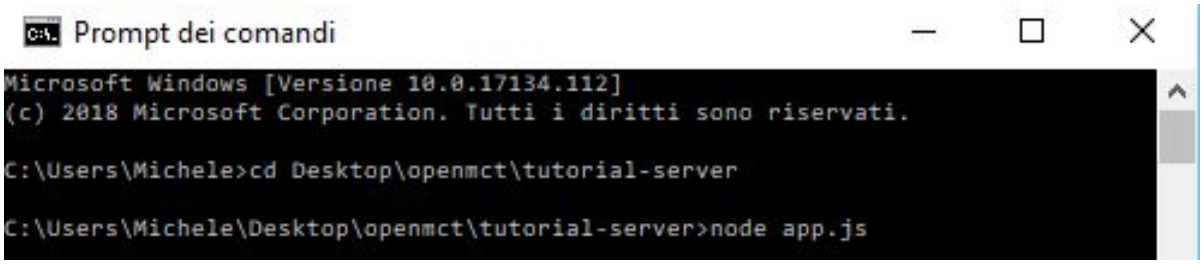


Figure 2.9: "App.js" Instructions

In Figure 2.10, a complete packet is decodified correctly. In fact, on top there are Destination Address and Source Address and packets ends with final string.
ADCS data are set on "q", because, during the test, it is not active.

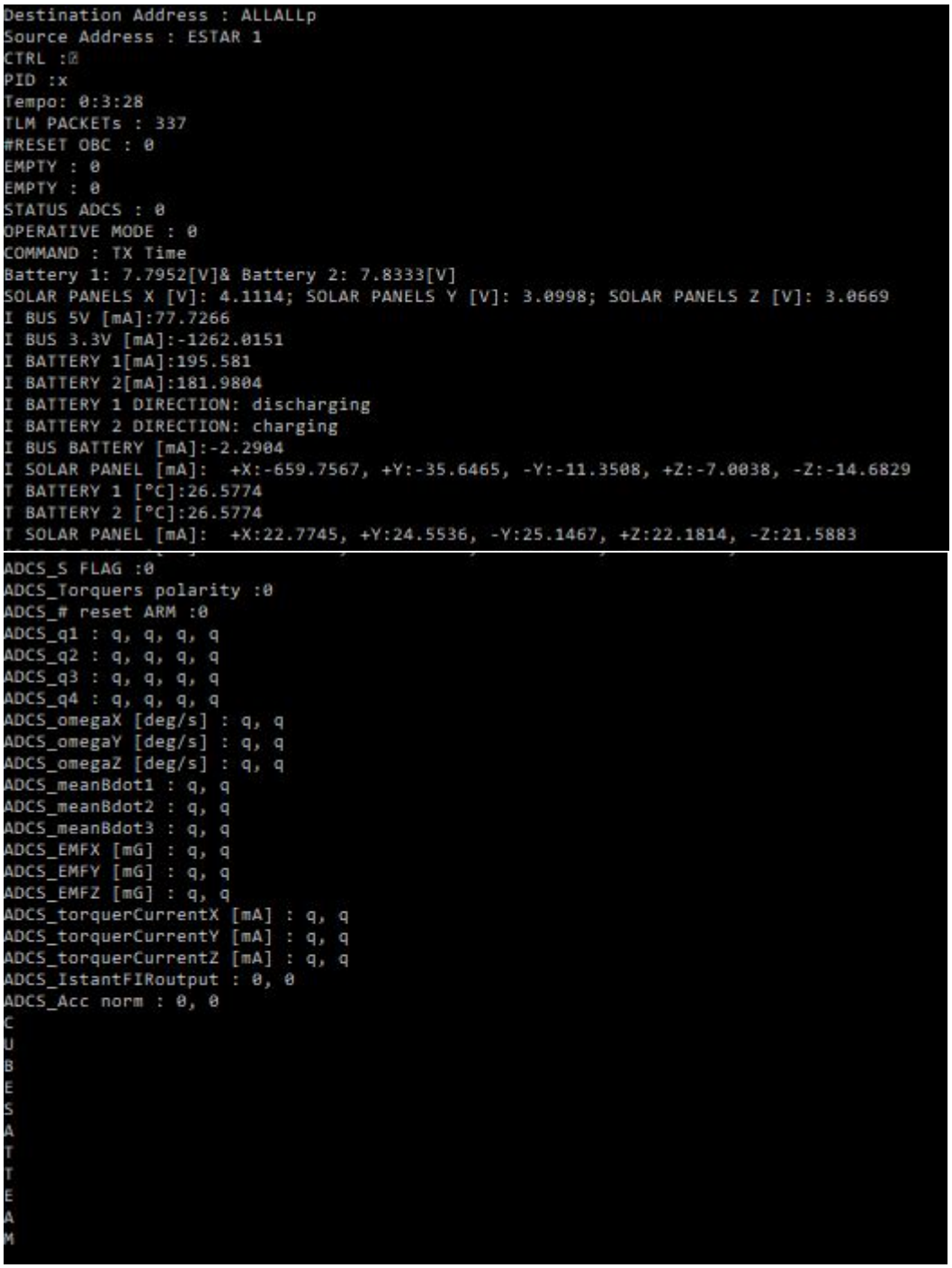


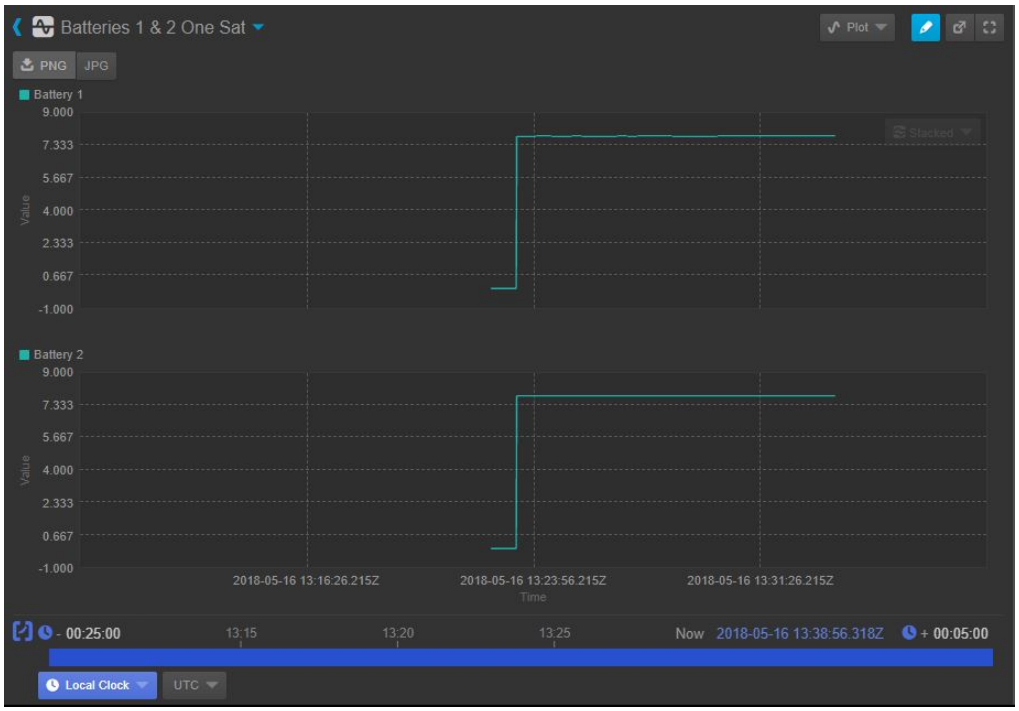
Figure 2.10: Packet Decoded Example

2.2.3 Subsystems Filling

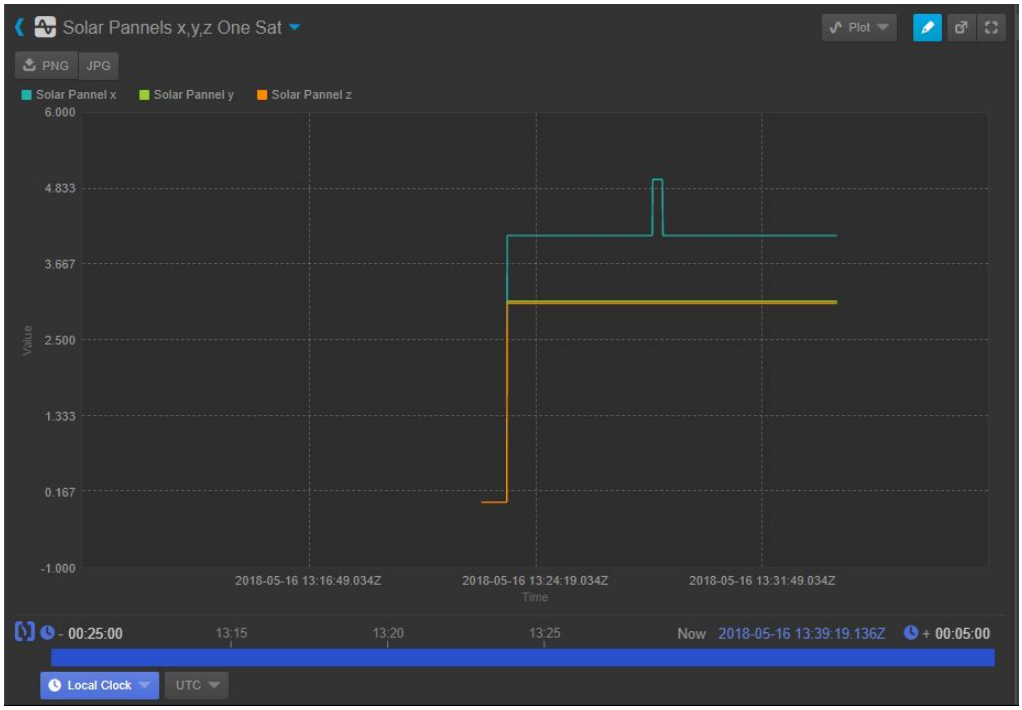
Data encoded are passed in a vector "spacecraft", where we saved all information received by satellite; Open MCT extracts these data and visualizes them in Telemetry section, where user can choose which information see.
The figures below show the first results obtained in this preliminary part.
Figure 2.11(a) shows data about Battery 1 voltage.
Open MCT allows to insert in the same plot more data, as shown in Figure 2.11(b), that can be split in different graphics, Figure 2.11(c).
Figures 2.11(d) and (f) show data, respectively, about voltage and current of Solar Panels.
Figure 2.11(h) and (h) shows data about temperature of Batteries 1 & 2 (approximately 27^{o}C) and Solar Panels(approximately 20^{o}C for -z and 25^{o}C for -y).

(a) Battery 1 Example

(b) Batteries 1 & 2



(c) Batteries 1 & 2 Split



(d) Solar Panels

(e) I Batteries 1 & 2

(f) I Solar Panels

(g) T Batteries 1 & 2

(h) T Solar Panels

Figure 2.11: Subsystems Filling

2.2.4 New Function

After the preliminary test, we proceed to introduce a new function that allows user to choose if he wants analyze telemetry packets inserting packets length or not ("updatespacecraft2" function). This is a preliminary selection of telemetry packets received by the spacecraft.

```
Microsoft Windows [Versione 10.0.17134.112]
(c) 2018 Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\Michele>cd Desktop\openmct\tutorial-server

C:\Users\Michele\Desktop\openmct\tutorial-server>node app.js

Choose if you want to analyse telemetry data by length (1) or not (2):
```

Figure 2.12: Preliminary Data Selection

In this way, user can see only packets which have the length insert, rejecting those that don't respect this restriction.

2.3 Simulated Satellite

2.3.1 Serial Communication

To simulate the act of receive data from a spacecraft, we have decided to use the serial communication. The test setup needs the following components:

- First Computer on which is run Open MCT;
- Second Computer, which simulates Satellite that sends data;
- Serial Communication Cable.

Figure 2.13: Serial Communication Setup

On software level, we proceed to design two interface C executable programs, one that simulates satellite and the other, run on main computer, that receives telemetry packets, as shown in Figure 2.14.

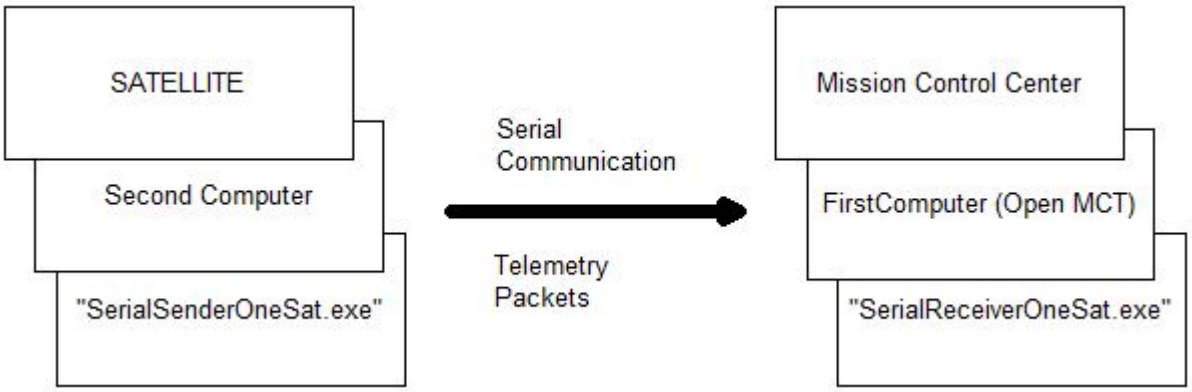


Figure 2.14: Serial Communication Diagram

Satellite (Second Computer)

To simulate the spacecraft that sends data, we create an executable C program "SerialSenderOneSat.exe" that sends packets to MCC, through serial communication.
We use RS-232 library that can be download at the link in bibliography [13], and then added in the code. This is ad hoc library required for serial communication.
To start the connection, we must set the following parameters:

- Baud rate: 9600; it represents the number of transitions per second occurring on the line.
- Databits:8; it represents bits used for one char;
- Parity: N; parity is an optional bit to verify if the received data is correct; in this case we use no parity;
- Stop bit: 1; it indicates for how many bits the communication is suspended.

The Communication Port number on second computer is COM1, identifies by cport_nr=0 in the code.

The program takes each telemetry from the file "dati.kss", where are listed the 445 packets used previously, and sends it, char by char, to MCC.
To simulate data reception in real time, we modify the packet chars that identify mission time. In fact, satellite simulated sends telemetry data approximately every 24 seconds.

Mission Control Center (First Computer)

To receive data communicated by satellite, we create an executable C program, "SerialReceiverOneSat.exe", where we set serial communication parameters:

- Baud rate: 9600;
- Databits:8;
- Parity: N;
- Stop bit: 1;
- ComPort: 2.

This code receives data char by char and, after modifying them in decimal representation, saves them in "data.kss", file read by Open MCT.

On Open MCT, all initial data values are set on 0, until MCC received first telemetry packet; then, it visualizes data received and, when satellite not communicate with the MCC,

Figure 2.15: I Batteries 1 & 2

shows the last values received.

To validate this test, it is necessary that pass/fail criteria are respected.
In this case, they are:

- the software receives at least 50 of 445 packets sent;
- packets received during serial communication must be necessarily long 110 chars;
- it must respect AX.25 protocol formatting.

Telemetry received respects pass/fail criteria. This is due to the fact that we utilize packets received during phase test of E-st@r 2.

2.4 Telecommand

The next step is to allow user to send telecommand.
Generally, user mission send command to satellite to control it, to active or disactive a subsystem, or to start a mission operative mode during spacecraft life.
Telecommand is received,decoded and handled by Command and Data Handling (C&DH) system and, then, directed to the appropriate subsystem.
When satellite receives it, it must send a packet string confirmation.

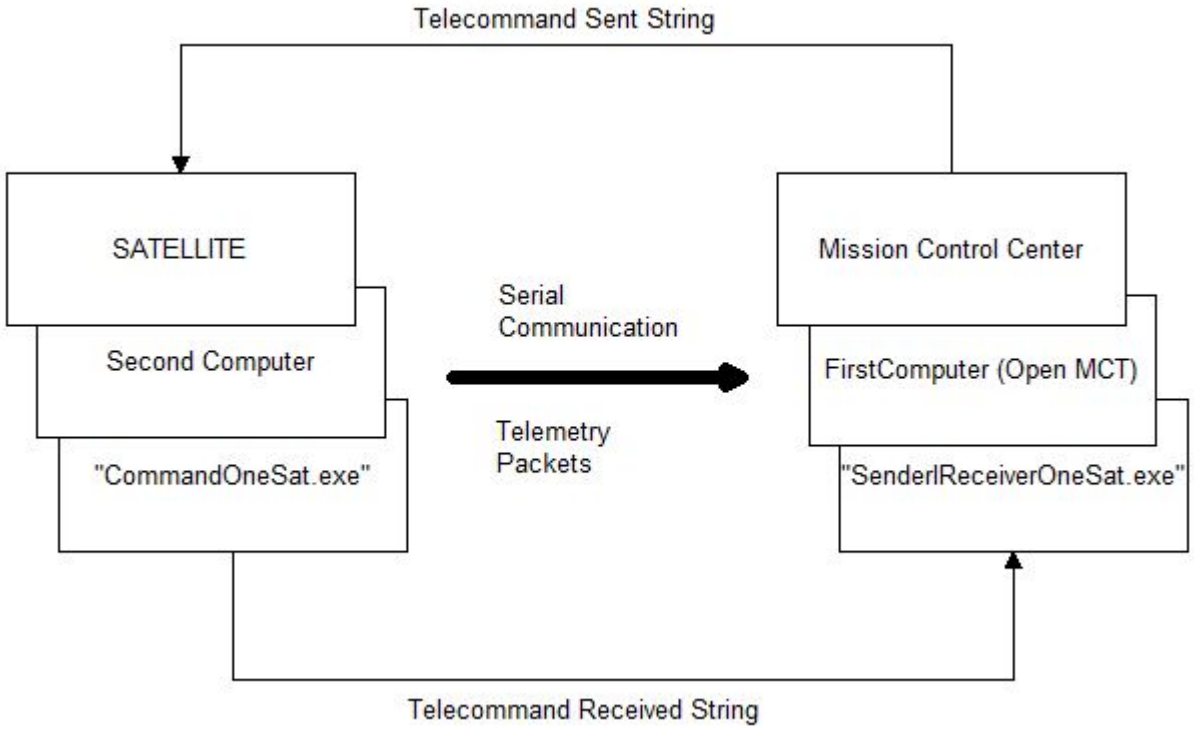


Figure 2.16: Telecommand Diagram

To do this, we proceed to modify both executable codes.

The telecommand test must respect previously fail/pass criteria and these new requirements:

- user must be able to send command each time that a packet is received;
- when a command is sent, software have to receive the command received string and visualize the Command ID in the specific section.

2.4.1 Telecommand Sender

We modify the telemetry receiver executable program to allow user to send telecommand. This section presents some restrictions:

- User must wait the reception of the first packets, before sending telecommand.
- Then, he can send command each 10 seconds.

When packet is received, the code asks user if he wants to send telecommand.

Figure 2.17: User Chooses to Send Telecommand

If he digits yes, the code shows the list of available telecommands.

Table 2.2: Telecommand Available

Telecommand ID	Telecommand
COM01	Update Mission Time
COM02	Update UTC
COM03	SMS
COM04	TX Time
COM05	TX Preamble
COM06	TX Stop
COM07	TX Resume
COM08	ADCS Mode
COM09	Reboot
COM10	Save Energy
COM11	Live Again

User sends these commands :

- Update Mission Time: to modify start time from which Satellite send telemetry, typing in days, hours, minutes and seconds;
- Update UTC: to modify UTC time;
- SMS: to send a short message, maximum length 20 chars;

- TX Time: entering ‘1’, he enables transponder to send data each 30s, ‘2’ for 60s and so on;
- TX Preamble: to precede data sent by a integer between ‘177’ and ‘255’;
- TX Stop: to stop transponder transmission;
- TX Resume: to start again transponder transmission;
- ADCS Mode: to shut off ADCS, entering ‘0’, or to active detumbling/save mode,‘1’, or to active determination mode,‘2’;
- Reboot: to restart satellite; this require a couple of minutes;
- Save Energy: to switch off the satellite;
- Live Again: to switch on the satellite from save energy state.

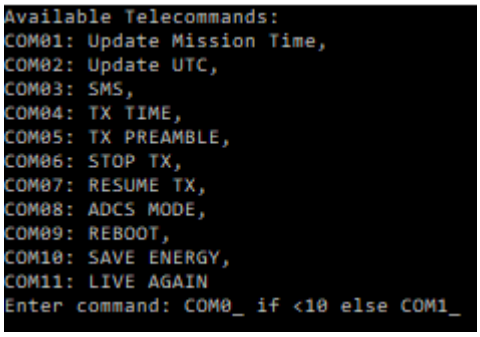


Figure 2.18: Telecommand List

Then, user inserts the Telecommand ID and, to control if satellite has received the right command, waits for the next packet sent by satellite. If telecommand format is wrong, the code generates an error message.
This process is executed in “SenderReceiverOneSat.exe”.

Telecommands have, also, an own protocol, a prefixed string; in fact, when user sends command, the code creates a new packet to send to spacecraft, char by char, in which the initial string is always the same, but the last part changes, according to the telecommand chosen, as list in Table 3.2.

Table 2.3: Modified Telecommand String

Telecommand ID	Telecommand String End
COM01	‘0x30’,‘0x41’
COM02	‘0x38’,‘0x49’
COM03	‘0x32’,‘0x43’
COM04	‘0x33’,‘0x44’
COM05	‘0x62’,‘0x4C’
COM06	‘0x63’,‘0x4D’
COM07	‘0x63’,‘0x4D’,‘0x47’,‘0x4F’
COM08	‘0x39’,‘0x4A’
COM09	‘0x31’,‘0x42’
COM10	‘0x61’,‘0x4B’
COM11	‘0x61’,‘0x4B’,‘0x47’,‘0x4F’

2.4.2 Telecommand Receiver

When satellite receives a telecommand by MCC, he must send a string of confirm reception.

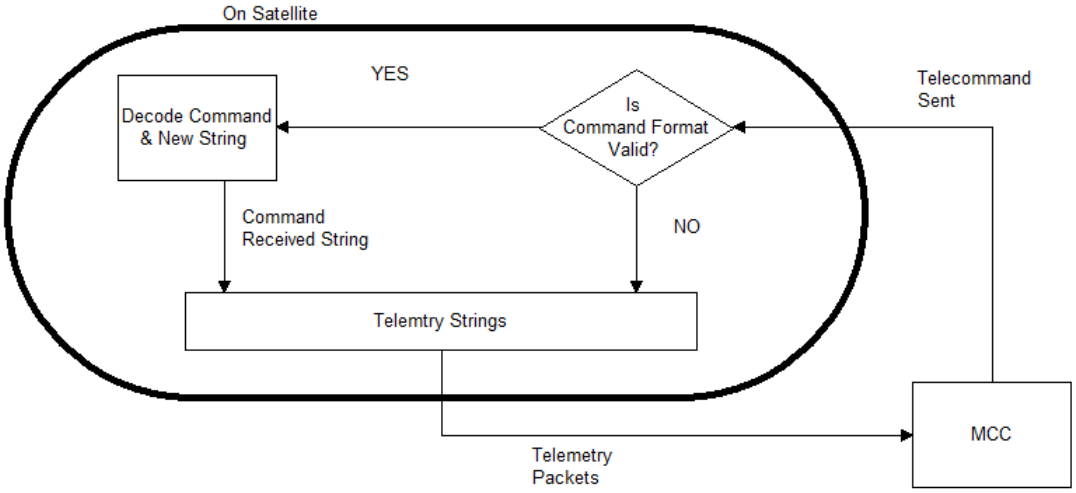


Figure 2.19: Telecommand Management Diagram

To do this, “SerialSenderOneSat.exe” has been modified; if simulated satellite receives a packet, it checks if the initial string is correct and then, it proceeds to decode the command and it creates a new telemetry packet to send to MCC.
If it doesn’t receive any string or format command received is wrong, satellite continues to

send telemetry data approximately every 24 seconds.
On the other hand, if user doesn't receive the command reception string, it means that he has not respect the timing provided by software to send telecommand.

This process is elaborated in "CommandOneSat.exe".

2.4.3 Open MCT with Telecommand

To visualize Telecommand received by satellite, we include a new subsection in "telemetry" section, "Telecommand Received".

```
{
    "name": "Telecommand Received",
    "identifier": "gen.tr",
    "units": "None",
    "type": "None"
}
```

Figure 2.20: Dictionary.json Modified Abstract

This part consists of a table that indicates the time when command received string is validate by MCC and what kind of telecommand is received.
Both functions, "updatespacecraft1" and "updatespaceraft2" are modified to validate this string and allow OpenMCT to visualize command received. In fact, if the string received contains COMxx, the code shows only the destination address, the source address, the control field, the protocol ID and by which satellite the command is received.

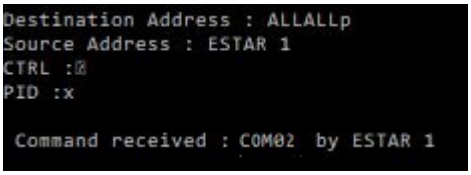


Figure 2.21: Telecommand Received Window

In addition, user can check, if command has been really received, validated and elaborated by satellite, opening last command subsection on Open MCT.

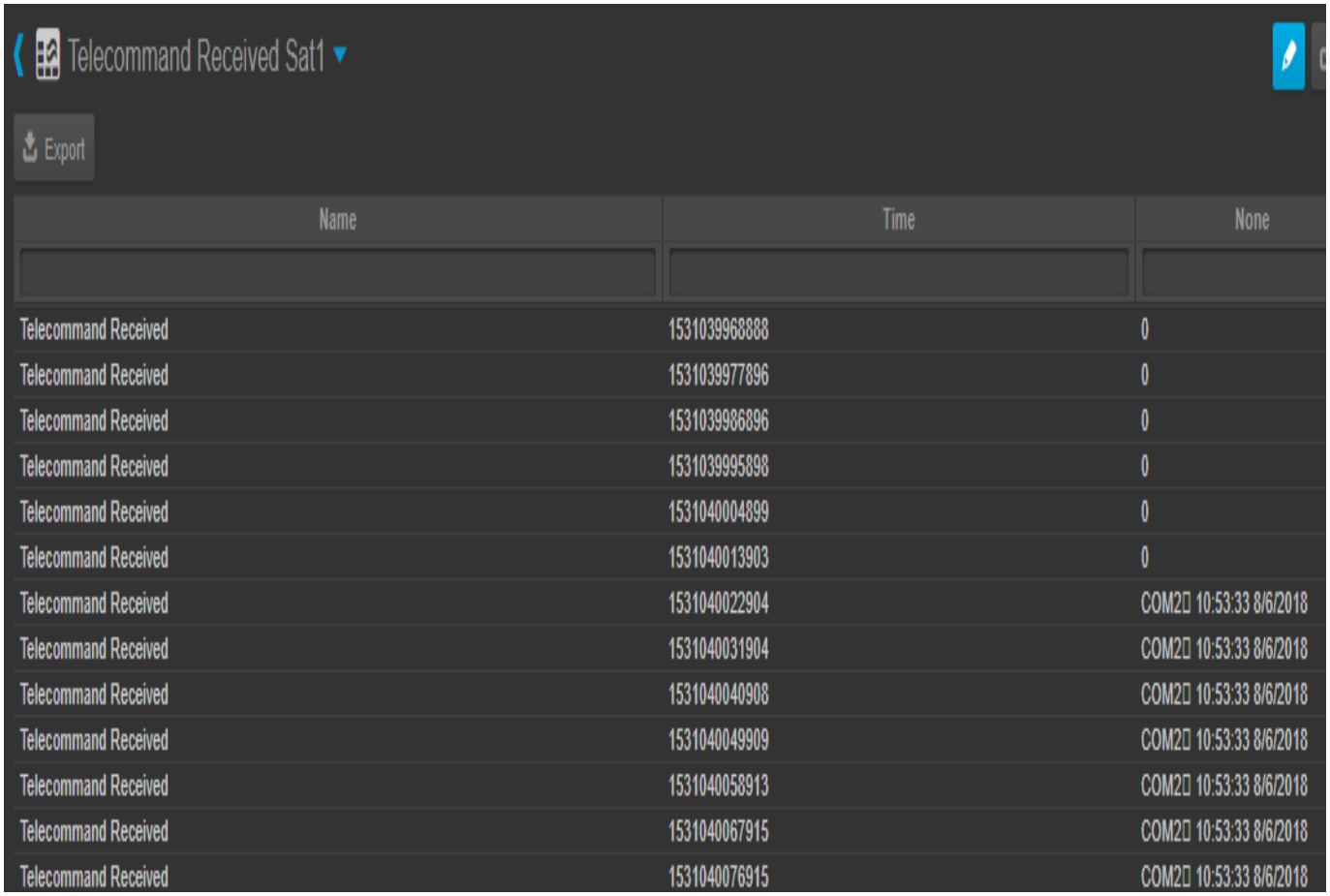


Figure 2.22: Telecommand Received Window

Figure 2.22 shows the Telecommand Received window on Open MCT. It consists of a table: the second column shows the UTC time in milliseconds and the third contains ID of telecommand sent with time [h:m:s] and date [day/month/year].

Chapter 3

Verification of the Software: Communication with Real Satellite

To validate the software, we proceed to test it, establishing a direct communication with the Development Board of E-st@r 2, the second CubeSat developed by CubeSat Team of Politecnico di Torino. It is a 1U CubeSat which aims to demonstrate the capability of autonomous determination, control and maneuver, through the development and test in obit of A-ADCS, entirely design and manufactured by students, and to test in orbit COTS technology and self-made hardware.

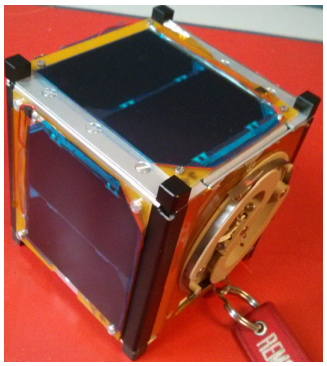


Figure 3.1: E-st@r 2 CubeSat

For this test, the pass/fail criteria are:

- software must receive at least consecutive 20 packets;
- it has to manage and control telemetry reception in real time;
- it has to visualize data in correct format.

Before proceeding to the test, it is necessary previously known how connect all components required. This is shown in Figure 3.2.

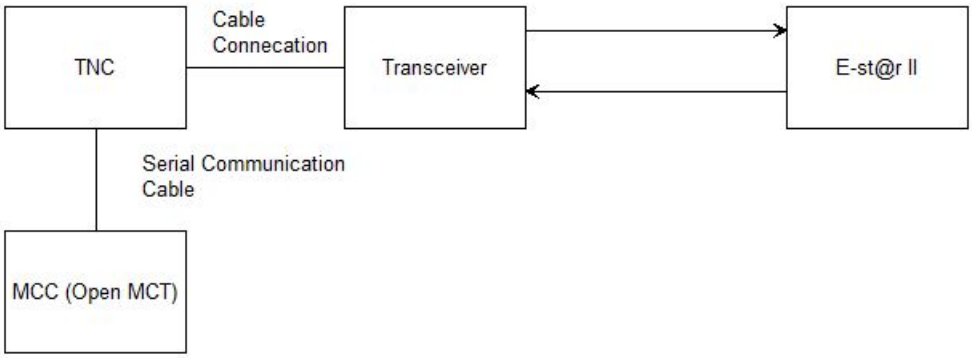


Figure 3.2: Test on E-st@r 2 Block Diagram

E-st@r 2 block represents the Development Board, on which the radio and EPS, consisting of two batteries, are placed.

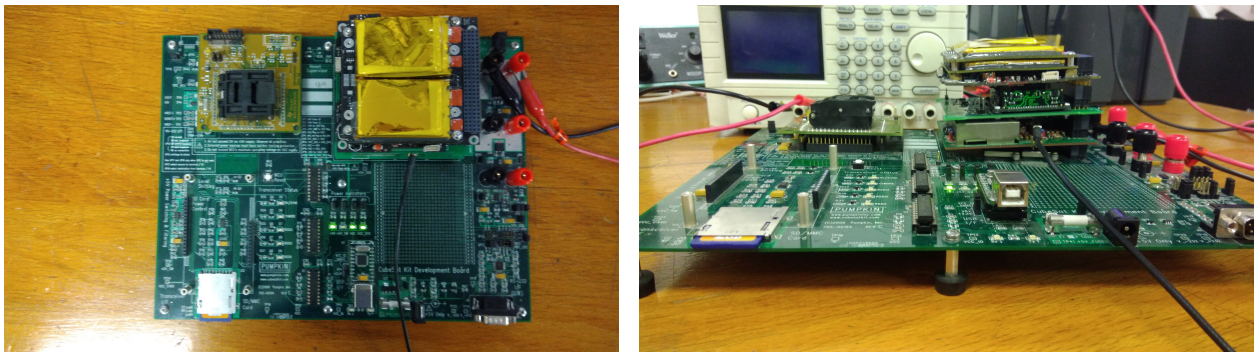


Figure 3.3: Development Board with radio and EPS

To enable the communication with the MCC, we set the Jumper on Development Board to their default values for MSP430 Flight MCU:

Table 3.1: Default Jumper Setting on Development Board for MSP430 Flight MCU

Jumper	**Setting**
JP1	ON
JP2	ON
JP3	ON
JP4	ON
JP5	ON
JP6	OFF
JP7	ON
JP8	ON
JP9	ON
JP10	ON
JP11	OFF
JP12	1-3,2-4
JP13	5-6
JP14	2-3
JP15	ON
JP16	ON

For more details, we suggest consulting User Manual UM-3 [14].
The transceiver component is a KENWOOD TH-F6A/TH-F7E transceiver. It is a small FM portable transceiver features 2 m, 1.25 m , and 70 cm amateur radio band operation plus another all-mode 100 kHz to 1.3 GHz receiver.
The TNC is SCS Tracker/DSP TNC (Terminal Node Controller), a radio network device, which, typically, consists of a dedicated microprocessor, a modem and software that uses AX.25 protocol and provides a command line interface to user. It is designed for custom applications of transferring GPS positions and other small sets of data from remote sensors by HF or VHF radio transmission.

Figure 3.4: KENWOOD FM DUAL BANDER

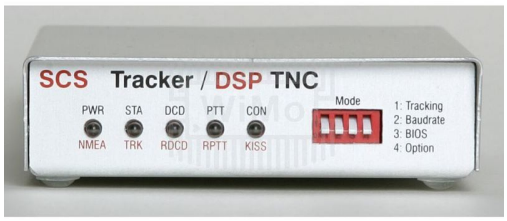


Figure 3.5: SCS Tracker/DSP TNC

The TNC socket is wired as follow:

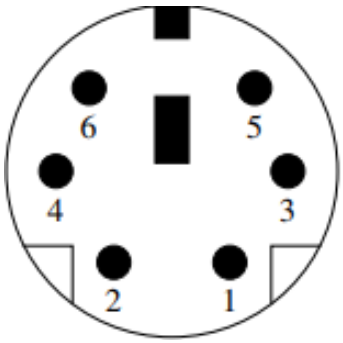
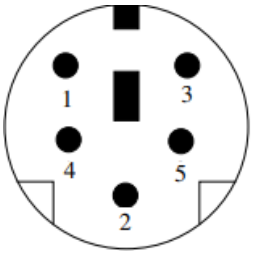


Figure 3.6: TNC PIN Setting

PIN 1: Audio output from the TNC to the transmitter. The TNC supplies a pure audio signal to the microphone (or ACC) input of the transceiver.
PIN 2: Ground (GND). Common ground for all signals.
PIN 3: PTT output. During transmission this output is grounded by the TNC, so that virtually all modern transceivers are usable.
PIN 4: Audio from the receiver to the TNC. The TNC receives signals directly from the speaker output of the receiver. The volume should not be turned up too much. A low volume is quite sufficient.
PIN 5:Optional power supply input. The TNC can be supplied with power via this input.This is especially useful if the transceiver gives a power supply output via the AUX socket.
PIN 6: Not Used

The Kenwood socket is wired:



PIN 1: AF - Output.
PIN 2: AF - IN.
PIN 3: Ground (GND).
PIN 4: Not Used .
PIN 5: PTT output.

Figure 3.7: Kenwood PIN Setting

To connect DSP TNC to Kenwood (HF), we utilize the following pin connection:

Table 3.2: TNC - Kenwood Connection

Signal	TNC	Color	Kenwood	Color
GND	PIN 2	brown	PIN 3	black
PTT	PIN 3	yellow	PIN 5	white
AF-OUT	PIN 1	red	PIN 2	thin cable
AF-IN	PIN 4	orange	PIN 1	thick cable

All this system is connected to the MCC through a Serial Cable Connection with USB adapter as Figure 3.8 shows.

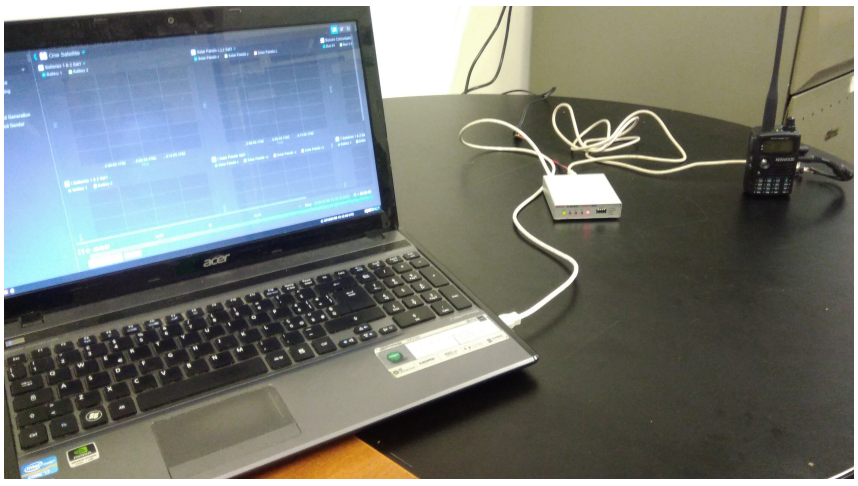


Figure 3.8: Test Setup

3.1 Telemetry Data

To receive telemetry data, we develop ad hoc executable C program "TNC Command User.exe", similar to previous codes. The additional feature is to active KISS protocol on TNC.
This protocol manages the separation of the packets by delimiters, the treatment of the delimiters which may occur inside the data stream and defines a simple set of commands for setting the TNC parameters.
Sending the following strings, with a sleep time of one second, we active KISS Mode:

KISS_MODE={'0x1B','0x40','0x4B','0x0D'},{'0xC0','0x01','0xFF','0xC0'}.

When packet is received, it is delimited by characters in decimal representation '0x2A','0x20' at first, and '0x0A' at the end, that we ignore; in fact, in Figure 3.9 the bytes received are 113, but that visualized are only the 110 characters of the packet.

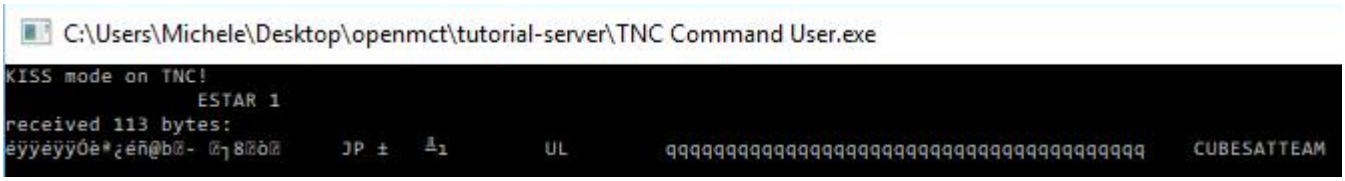


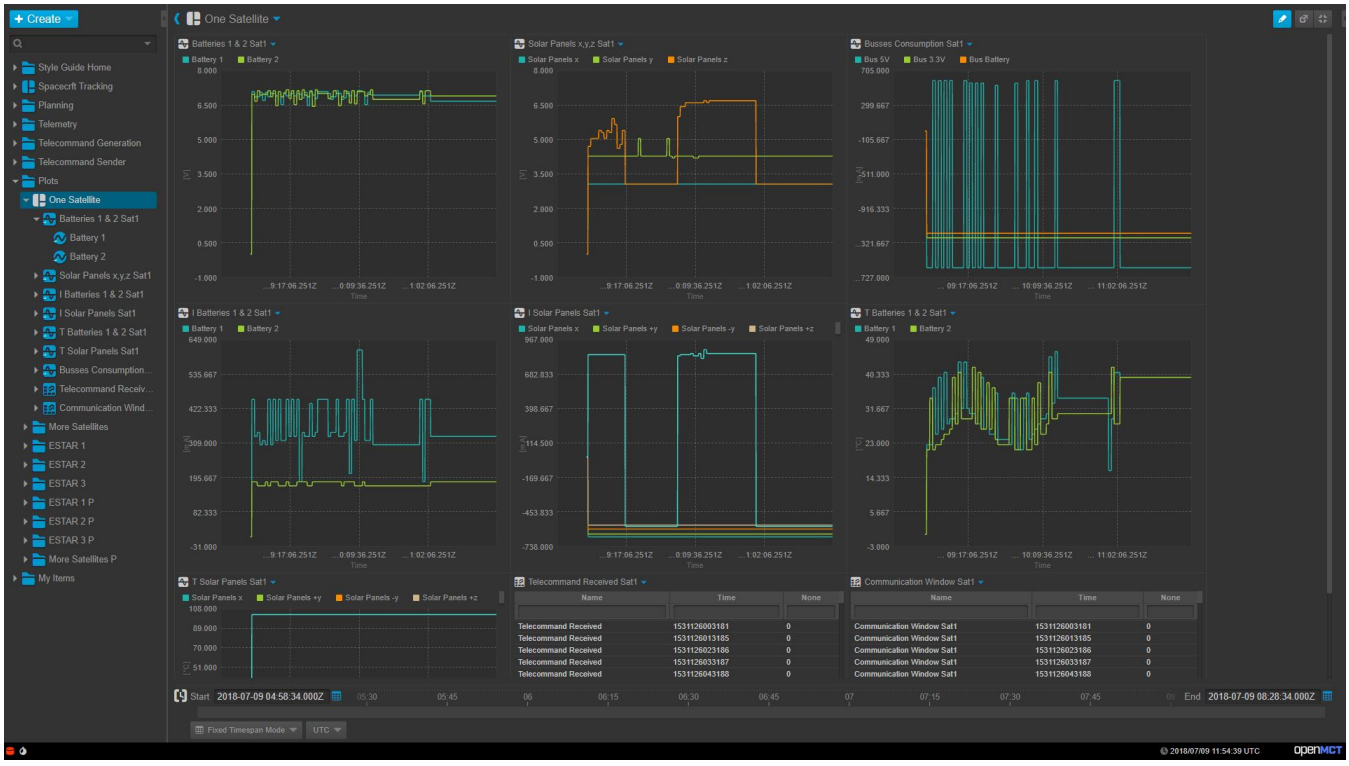
Figure 3.9: KISS MODE Active

During this test, we receive packet each 2 minutes.

3.2 Mission Test

Thanks to thesis work of Roberto Sorba, we can simulate a satellite, through a simulation developed by himself, connected to a power supply that provides current and voltage to EPS. The mission simulated consists in a satellite that performs three orbits a 400 km of Earth, with orbit inclination equal to 96^o and other parameters equal to 0.
All simulation goes on for three hours. In Figures 3.10 (b),(d) and (f), we notice the process of charge and discharge of batteries, highlight by the typical square wave.
Each orbit is characterized by an eclipse period of approximately 30 minutes. This phase is visible in Figures 3.10 (c) and (e), where we can notice that, after a first phase on which Voltage data change between 6V and 4.7, it settles on constant value of 3V. On the other hand, Current value passes from 860.438 mA to a negative number that represents eclipse phase.

Data value of Solar Panels, in this case Solar Panel z for voltage, and Solar Panel -z for current, include data of other panels; this is due to overcome the problem that two of three channels connected to solar panels on Development Boar don't work.
In Figure 3.10 (i), we can see the interval between two packets reception: one at 11:28:14, the next one at 11:30:14.
In all Figure, we notice that the last 45 minutes are characterized by an absence of data reception, this is due to a problem of satellite that doesn't communicated constantly with the MCC.

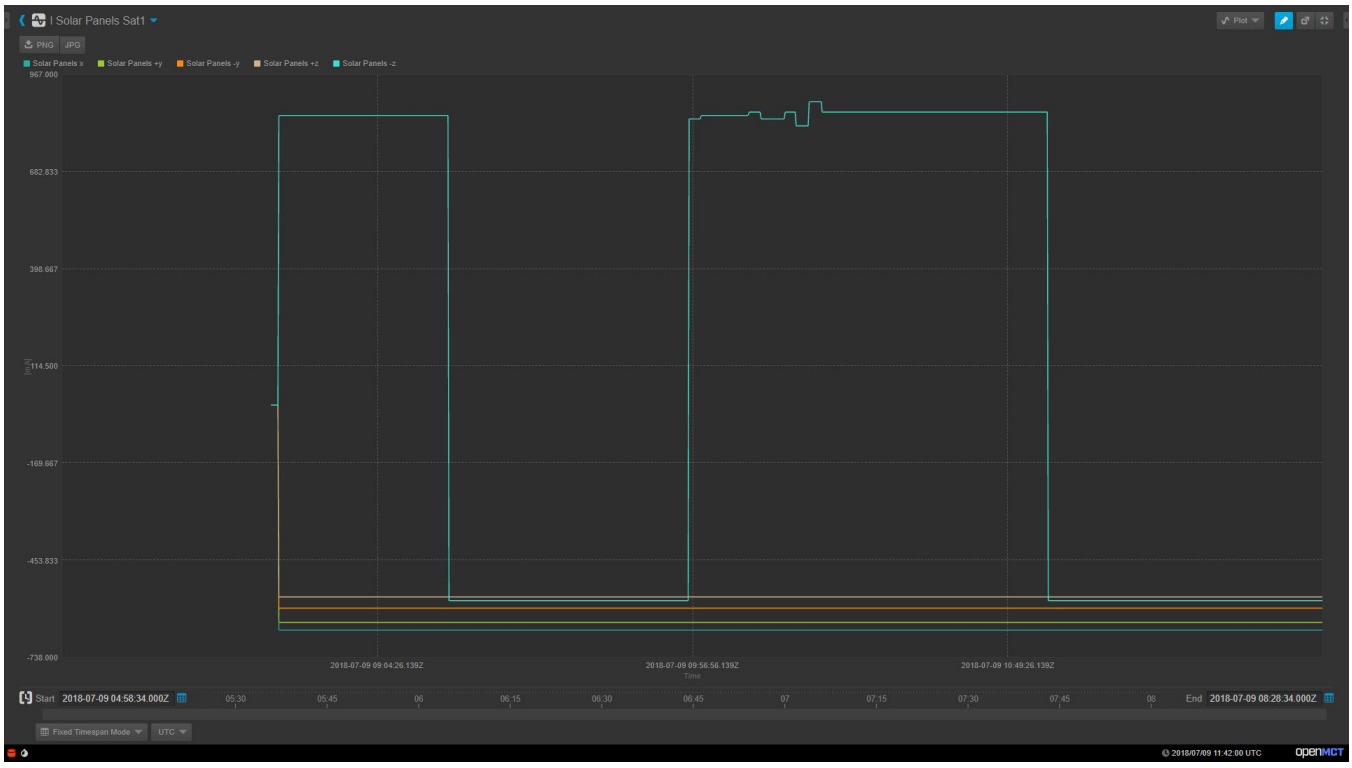


(a) Plot View Window

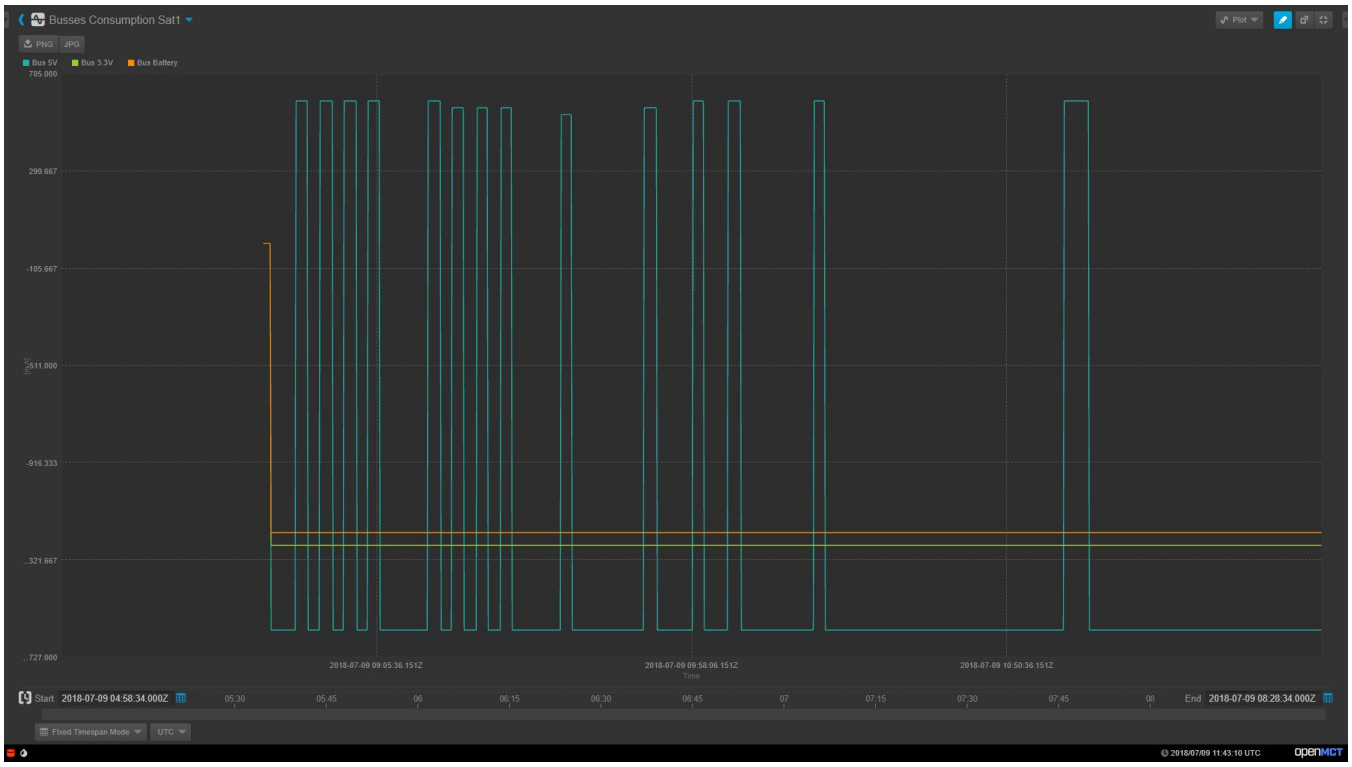
(b) Batteries 1 & 2

(c) Solar Panels

(d) I Batteries 1 & 2

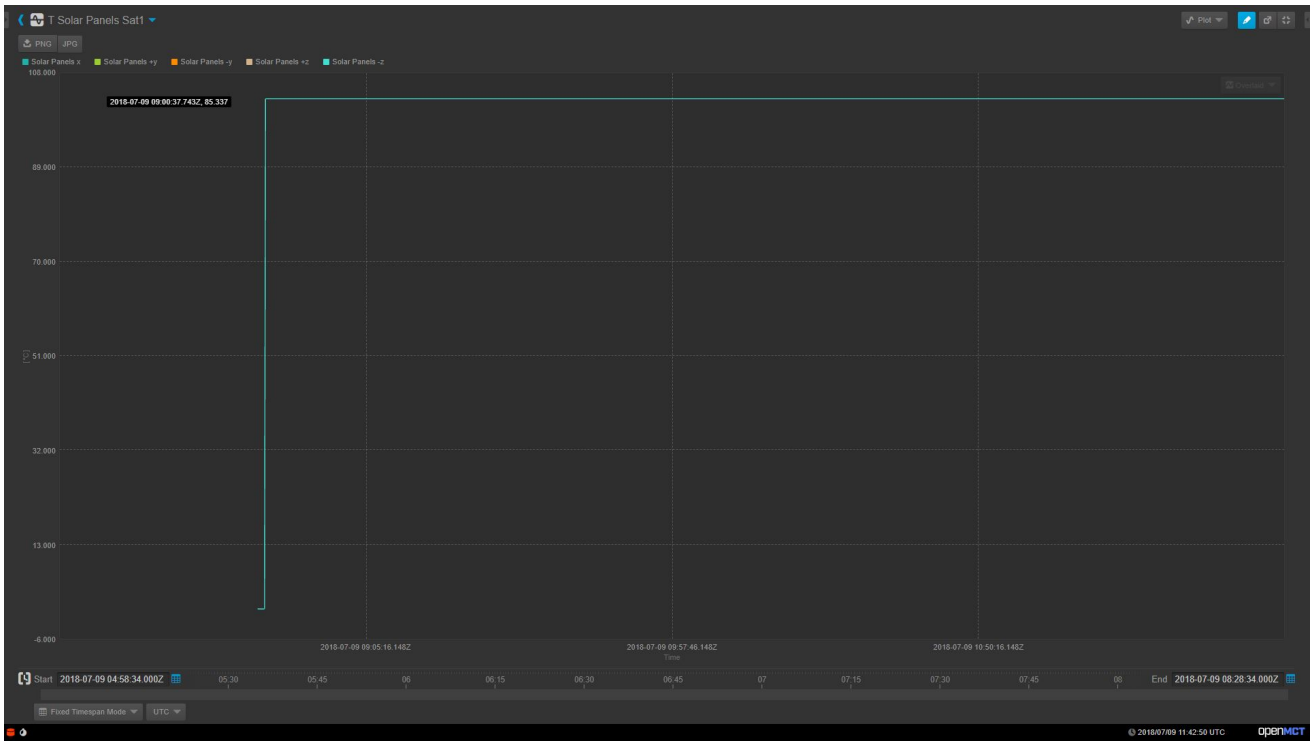


(e) I Solar Panels



(f) Busses Consumption

(g) T Batteries 1 & 2



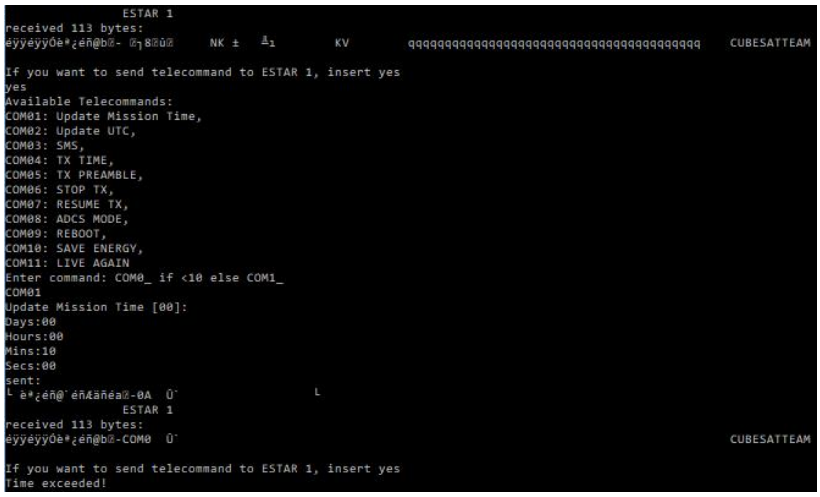
(h) T Solar Panels

(i) Communication Window

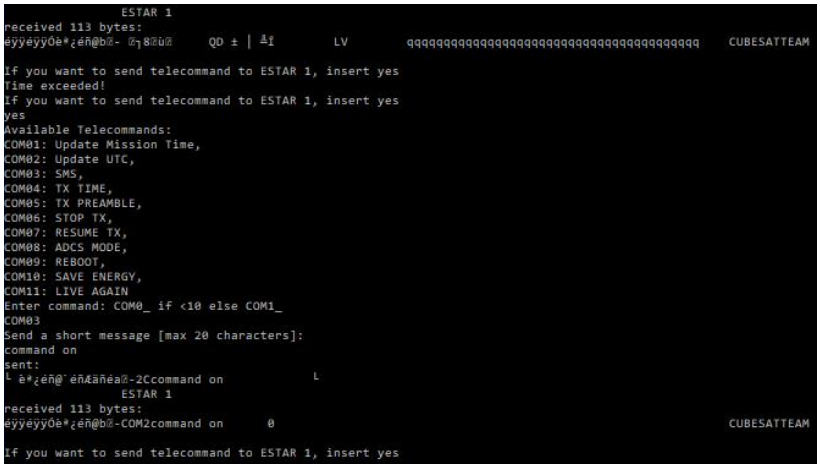
Figure 3.10: Plots of Simulation

3.3 Telecommand

During the simulation, we test also the telecommand sending. We send to satellite COM01 and COM03.



(a) User Interface for COM01



(b) User Interface for COM03

Figure 3.11: User Interface for Commands

In Figures 3.11, we can notice how spacecraft, received command string, sends a confirming packet. The telecommand ID is different, because spacecraft elaborates them starting from ID COM0; so, COM0 corresponds to COM01 and COM2 to COM03 of user interface.
This section has demonstrated that software chosen and additional functions could be utilized in a real mission, because the main requirements imposed, to manage and control satellite in telemetry reception in real time and to be able to send telecommand, have been respected.

Chapter 4

Software Setting for Communication with More Satellites

Usually, in space mission, it is required that satellite continually communicates with a ground station. Depending on how high the satellite orbits, breaks in communication can happen, if satellite is not in line-of-sight. To ensure that satellite for a given purpose is in contact at all times required, it is necessary to deploy more satellites in a constellation.

For this reason, we decide to extend the thesis work to manage more satellites. The constellation that we decide to simulate consists of three satellites, named ESTAR 1, ESTAR 2 and ESTAR 3.

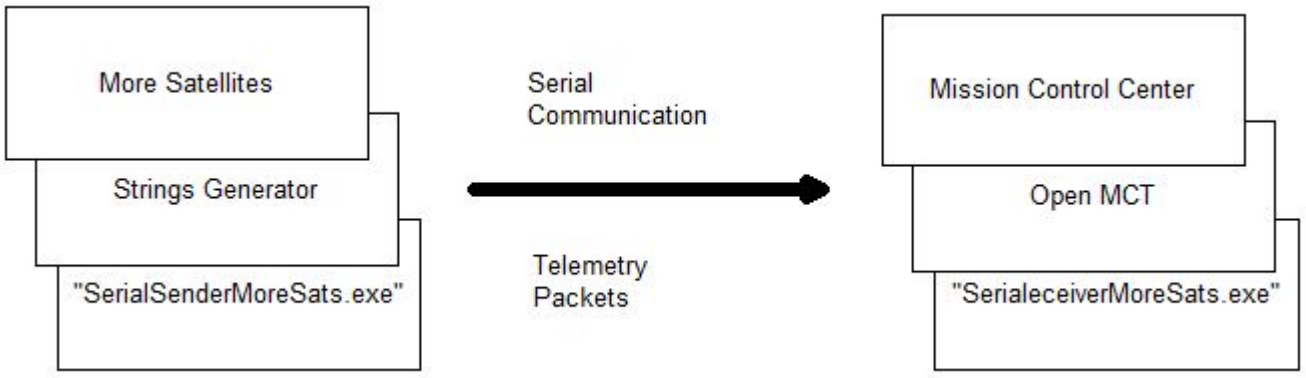


Figure 4.1: More Satellites Block Diagram

As shown in Figure 4.1, we proceed as previously, simulating the three satellites on a second computer, which sends telemetry packets to MCC, through serial communication.
To validate this analysis, the pass/fail criteria are:

- software has to receive consecutive 20 packets;
- it has to recognize different telemetry packets received by each spacecraft;
- it has to visualize data in dedicated satellite sections;
- it has to allow user to send telecommand to each satellite.

The first step of this phase is to test how manage three satellites that send telemetry packets.

We start with 12 packets of data.kss and change only char that identifies the number of satellite:

{ Š¦¨,¤@b } => { ESTAR 1 } , { Š¦¨,¤@d } => { ESTAR 2 }, { Š¦¨,¤@f } => { ESTAR 3 }

In "dictionary.json", we duplicate all sections and modify them for each spacecraft.
User can see on Open MCT the modified window, Figure 4.2, in which appears:

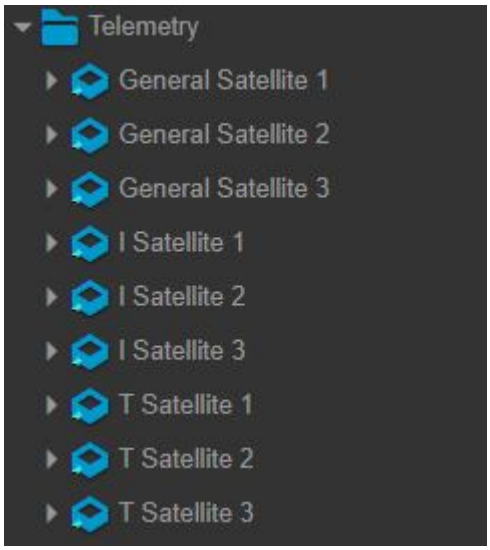


Figure 4.2: Telemetry Folder for More Satellites

- General Satellite 1, General Satellite 2, General Satellite 3: they include general data of, respectively, ESTAR 1, ESTAR 2 and ESTAR 3;
- I Satellite 1, I Satellite 2, I Satellite 3: these sections include information of current subsystem of, respectively, ESTAR 1, ESTAR 2 and ESTAR 3;
- T Satellite 1, T Satellite 2, T Satellite 3: include information of temperature subsystem of, respectively, ESTAR 1, ESTAR 2 and ESTAR 3.

4.1 Serial Communication

4.1.1 First Test

Using the same executable code, "SerialSenderOneSat.exe", that takes string packet by "dati.kss" and sends it to MCC, char by char, we modify only "SerialReceiverOneSat.exe", in "SerialReceiverMoreSats.exe". In fact, we scatter data received in different files, one for each satellite, to allow user to see every single telemetry of each spacecraft.
All data are, also, saved in "data.kss" in decimal representation, used by Open MCT.

In "app.js", we modify JavaScript code to visualize telemetry data of three satellites. To fill each telemetry subsections, software analyzes each packet received and scatters data in specific sections of satellite that sends it.
As previously, user can choose if he wants to analyze telemetry data by length or visualize each data received ("upadatespacecraft3" and "updatespacecraft4" functions).

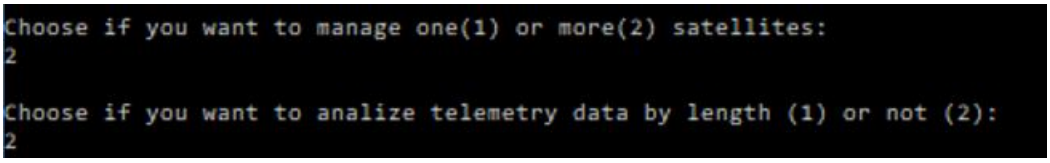


Figure 4.3: Updatespacecraft3 and Upadatespacecraft4 Functions

In this way, user can manage three satellites at the same time.

4.1.2 Strings Generator

To generate the telemetry packets of satellites simulated, we decide to create a new executable code, "SerialSernderMoreSats.exe", that represents a string generator for each satellite.
We consider packets in "dati.kss" as telemetry data of ESTAR 1. To create packets of another spacecraft, we proceed as follow:

- Each packet of ESTAR 1 is copied in a new vector;
- This vector is modified; for example, we decide that ESTAR 2 battery voltage must be 99% of ESTAR 1 battery voltage; than we extract char that identifies this data and put in telemetry packets of ESTAR 2;
- We do similar also for ESTAR 3 telemetry packets.
- Each satellite sends packets approximately each 24 seconds after the previously: ESTAR 1 sends at 0s, ESTAR 2 sends at 24s, ESTAR 3 at 48s, and ESTAR 1 send at 72s, and so on.

A table of parameters modified is in Table 4.1.

Table 4.1: String Generator Parameters

E-ST@R 1(E-1)	**E-ST@R 2(E-2)**	**E-ST@R 3(E-3)**
Battery 1	0.99 Battery 1 (E-1)	0.99 Battery 1 (E-2)
Battery 2	0.97 Battery 2 (E-1)	0.95 Battery 2 (E-2)
Solar Panels x	0.99 Solar Panels x (E-1)	0.95 Solar Panels x (E-2)
Solar Panels y	0.97 Solar Panels y (E-1)	0.99 Solar Panels y (E-2)
Solar Panels z	0.95 Solar Panels z (E-1)	0.95 Solar Panels z (E-2)
I Bus 5V	0.99 I Bus 5V (E-1)	0.99 I Bus 5V (E-2)
I Bus 3.3V	0.98 I Bus 3.3V (E-1)	0.98 I Bus 3.3V (E-2)
I Battery 1	0.9 I Battery 1 (E-1)	0.9 I Battery 1 (E-2)
I Battery 2	0.85 I Battery 2 (E-1)	0.95 I Battery 2 (E-2)
I Bus Battery	0.98 I Bus Battery (E-1)	0.98 I Bus Battery (E-2)
I Solar Panels x	0.98 I Solar Panels x (E-1)	0.92 I Solar Panels x (E-2)
I Solar Panels +y	0.98 I Solar Panels +y (E-1)	0.92 I Solar Panels +y (E-2)
I Solar Panels -y	0.98 I Solar Panels -y (E-1)	0.92 I Solar Panels -y (E-2)
I Solar Panels +z	0.98 I Solar Panels +z (E-1)	0.92 I Solar Panels +z (E-2)
I Solar Panels -z	0.98 I Solar Panels -z (E-1)	0.92 I Solar Panels -z (E-2)
T Battery 1	0.95 T Battery 1 (E-1)	0.85 T Battery 1 (E-2)
T Battery 2	0.95 T Battery 2 (E-1)	0.85 T Battery 2 (E-2)
T Solar Panels x	0.96 T Solar Panels x (E-1)	0.92 T Solar Panels x (E-2)
T Solar Panels +y	0.96 T Solar Panels +y (E-1)	0.92 T Solar Panels +y (E-2)
T Solar Panels -y	0.96 T Solar Panels -y (E-1)	0.92 T Solar Panels -y (E-2)
T Solar Panels +z	0.96 T Solar Panels +z (E-1)	0.92 T Solar Panels +z (E-2)
T Solar Panels -z	0.96 T Solar Panels -z (E-1)	0.92 T Solar Panels -z (E-2)

In this way, we generate the same packets number for each satellite, with different data. These packets are sent to MCC through serial communication which parameters are listed below: Serial communication parameters are listed below:

- Baudrate: 9600;
- Databits:8;
- Parity: N;
- Stopbit: 1;
- ComPort: 2.

Telemetry information of three spacecrafts subsystems aren't very different, because in a satellites constellation, spacecrafts have the same parameters, such as same orbit, altitude and angular velocity.

4.2 Telecommand

The last step of this section is to allow user to send telecommand to more satellites.
To do this, we combine "CommandOneSat.exe" and "SerialSenderMoreSats.exe" to create a new executable program "CommandMoreSats.exe".
This code provides to generate strings for each satellite, as previous section. In addition, user can send telecommand to each spacecraft.
In fact, when telemetry packet is received, code asks user if he wants to send telecommand to ESTAR x.

Figure 4.4: User's Option to Send Telecommand

User has only 3 seconds to insert yes. Then, a list of telecommands appears and he chooses which one send to the respectively spacecraft.

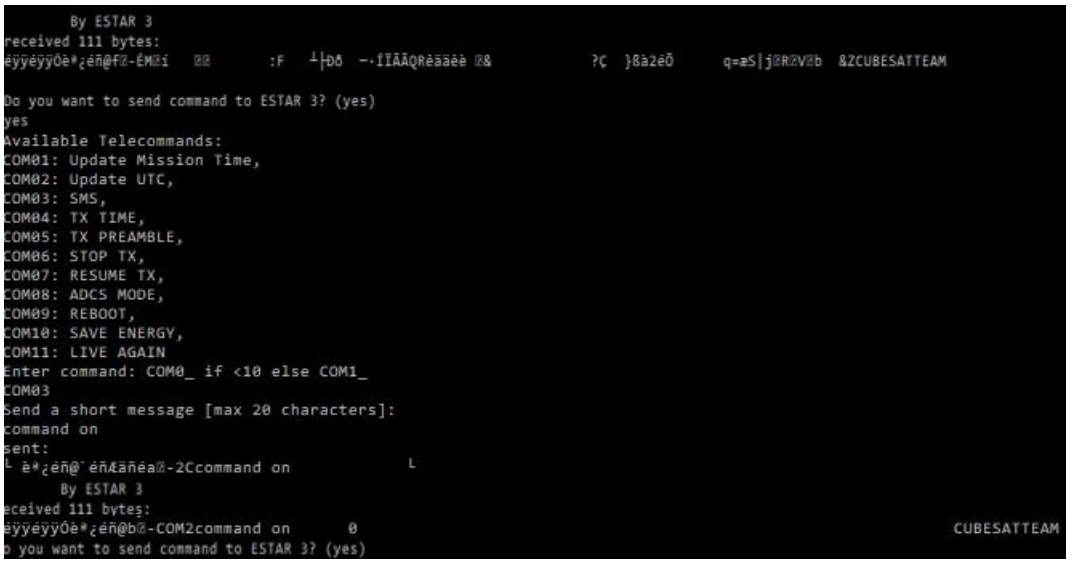


Figure 4.5: Available Telecommands List

To validate the correct telecommand reception by satellite, as previously, user waits the string of confirm reception and checks on Open MCT in “Telecommand Received” window, Figure 4.6.

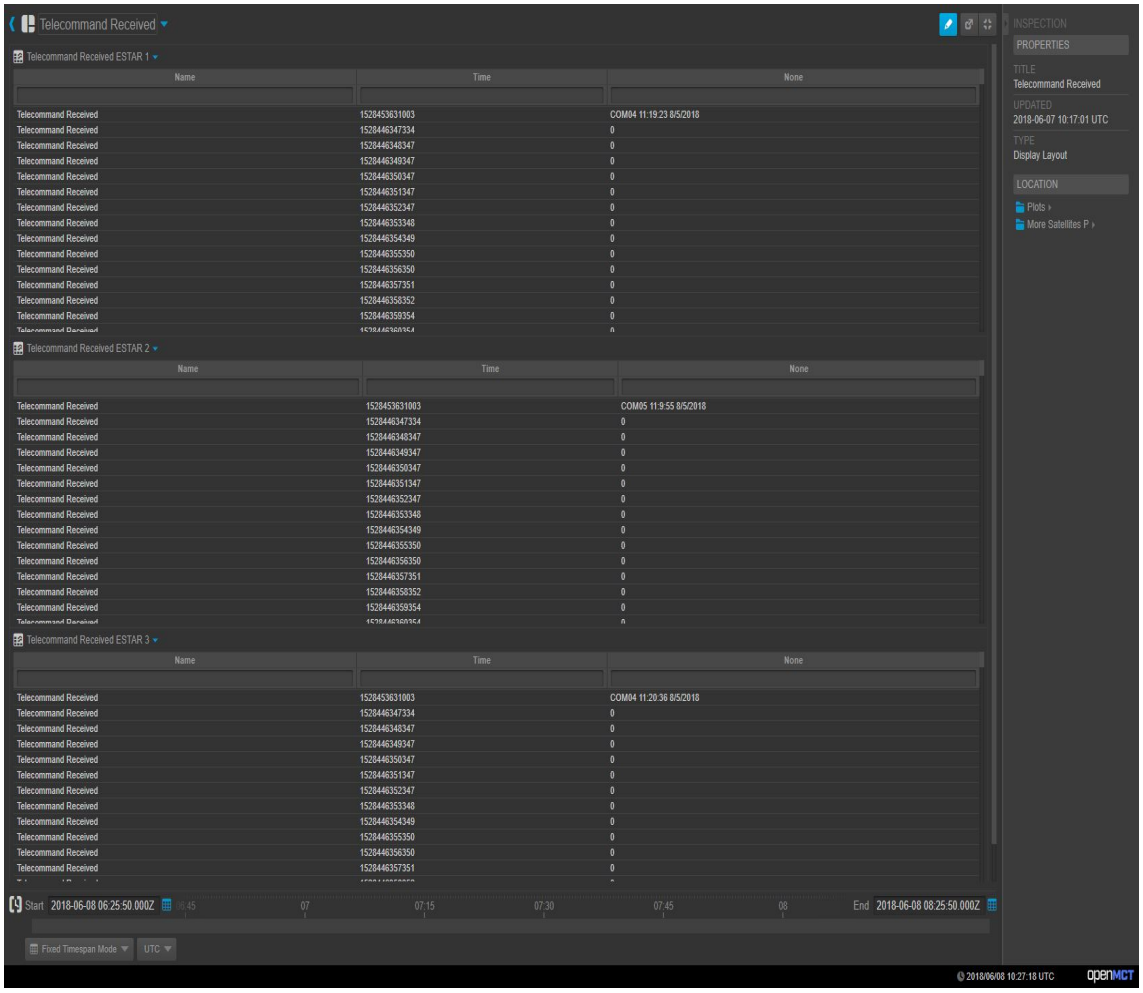


Figure 4.6: Command Visualization on Open MCT

4.3 Plots

To help user to visualize as he wants the data, Open MCT allows to create a fixed display where user can add, move and change the layout of telemetry plugin.
In this thesis, we create a specific folder, “Plot”, in which we dispose and visualize all important data.

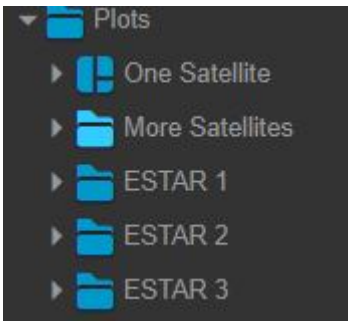


Figure 4.7: Plots Folder on Open MCT

Each section identifies all plot and table we gather. In One Satellite display, we dispose telemetry plot of primarily tests on E-st@r 2. We decide to dispose in same plot similar data, such as Batteries 1 and 2 voltage. This allows user to have specific data of subsystem in the same windows (Figure 4.8).

In Figure 4.9, all subsections are disposed in a fixed display; this allow user to have an overview of all data, monitoring at the same time each subsystem. In particular, if he wants to visualize only one section, he can push on the specific section on the left of the display or on the desired plot. Through this last mode, the selected window moves to the foreground and, closing it, user return immediately on previously fixed display.

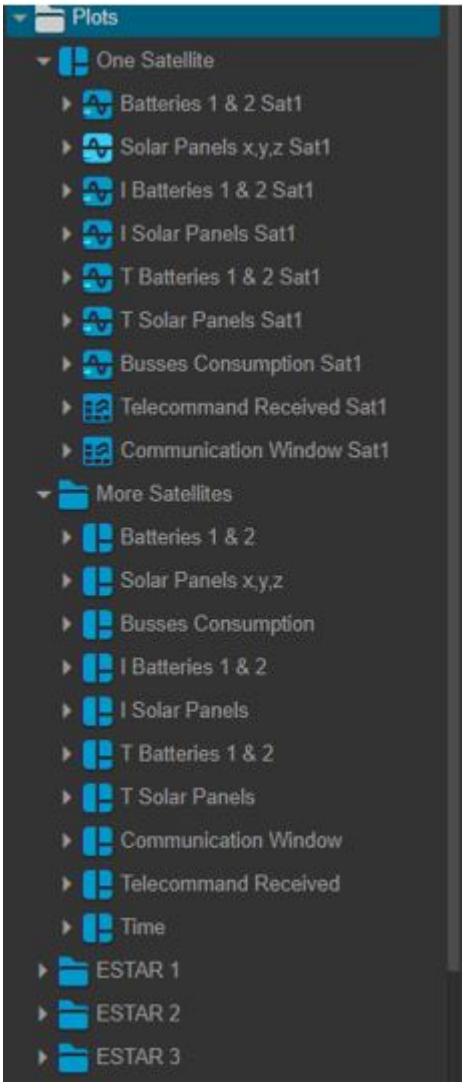


Figure 4.8: Subsections Plots on Open MCT

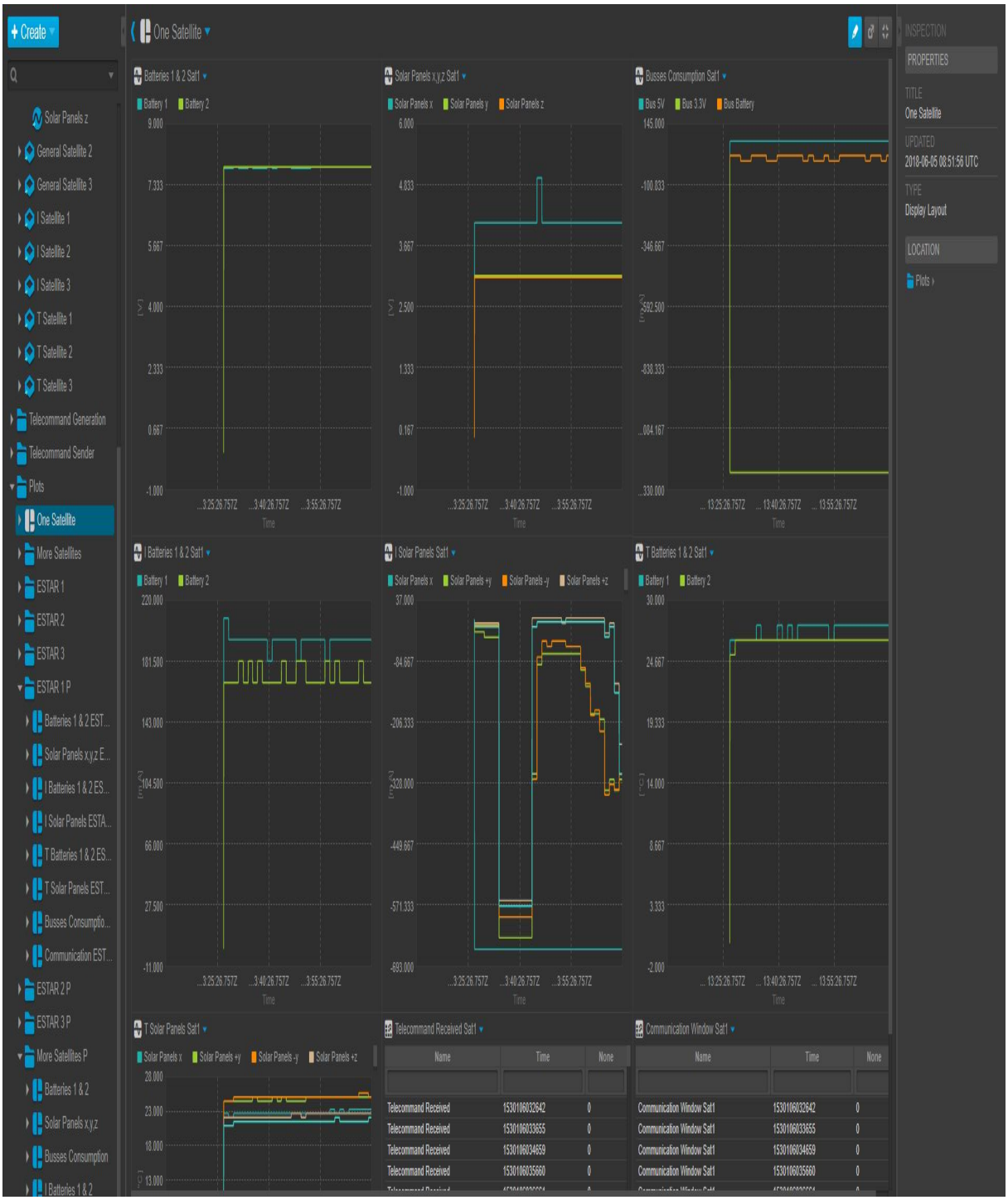


Figure 4.9: One Satellite Plots

In folder More Satellite, we collocate all plots of each satellite of constellation; in such way, user can visualize a specific data, comparing all three satellites.

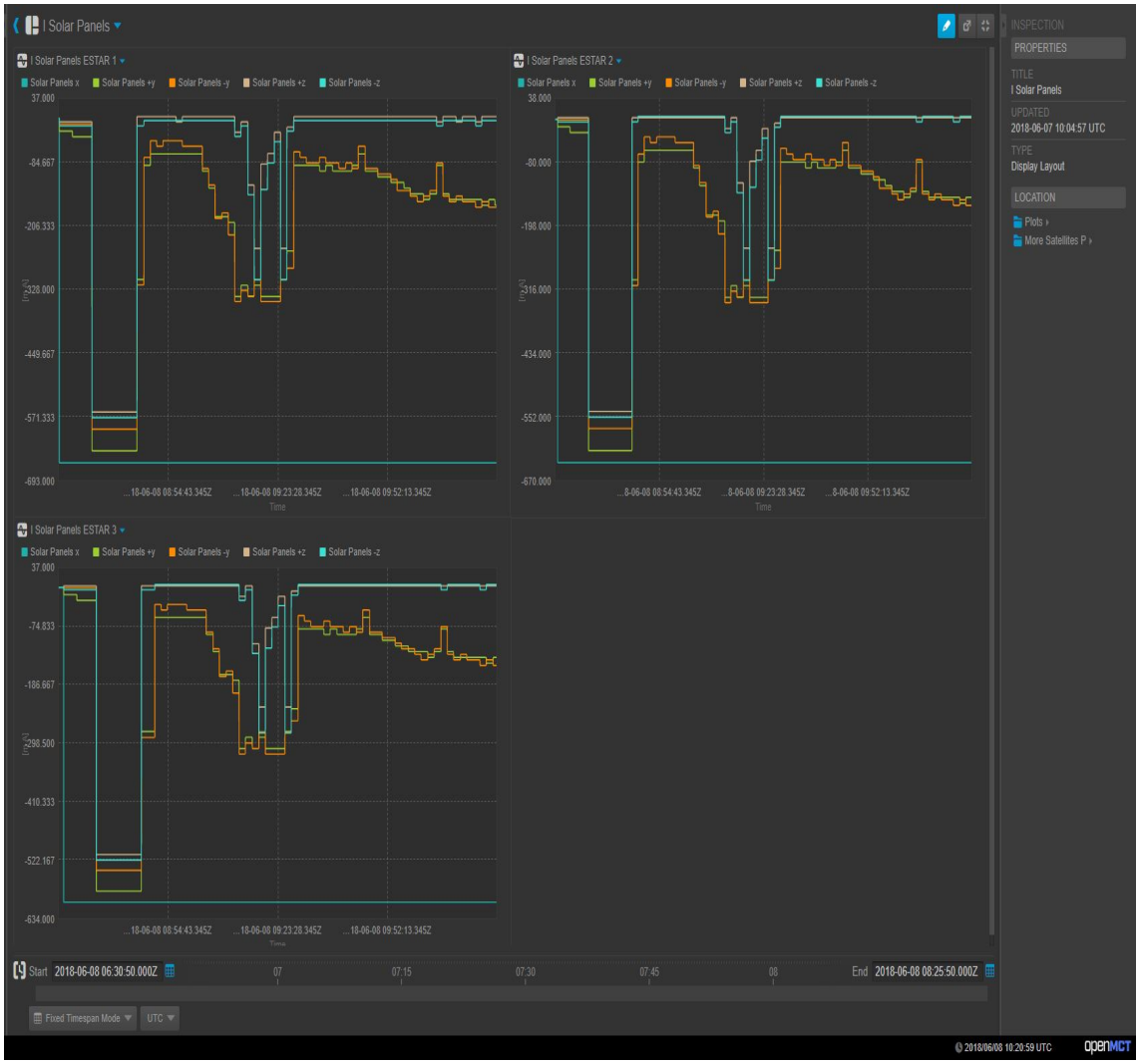


Figure 4.10: I Solar Panels of Three Satellite

The plot folder has been created with the first methodology that Open MCT allows to create new section on software: we utilize the button “Create” and choose display layout; this means that this folder with its subsections, is only visible on this computer. This is a chosen done to allow next user to dispose plots as he wants and take confidence with the software.

To manage more data at the same time, we decide to use a second display, connected with the main computer through a HDMI cable. This is also to simulate a real MCC where on each display, the operator visualizes specific data.

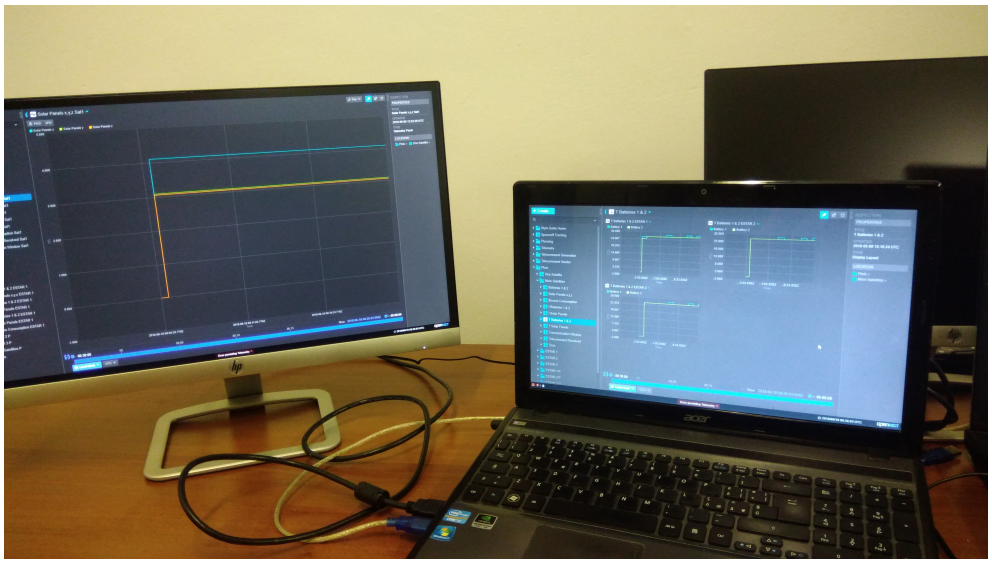


Figure 4.11: Second Display

All available plots of the management of three satellites are placed in Appendix B.

Chapter 5

Verification of the Software: Warning Identification

To complete the thesis, we decide to insert warning, luminous objects that advice user to the state of health of each subsystem.

5.1 Warning Definition and Rules

To create these elements, we use a plugin included in Open MCT: "Summary Widget". This plugin allows to create rectangles and insert a text that alerts user, if a data is out of valid range. It allows to insert rules that when respected, change warning color and text.
To use this plugin, user has to select a telemetry data to associate with warning, drag it in specific window, as shown in Figure 5.1.

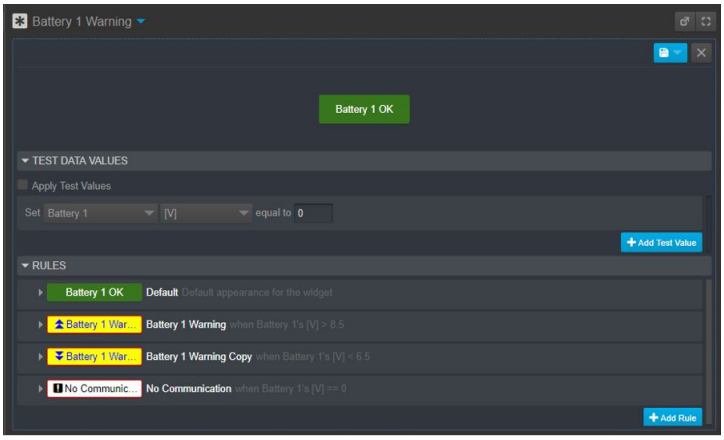


Figure 5.1: Warning Rules

A summary of rules for each subsystem used is listed below:

Telemetry Data	Lower Limit	Upper Limit
Battery 1 & 2 [V]	6.5	8.5
Solar Panels x,y,z [V]	3.3	5
I Battery 1 & 2 [mA]	0	300
I Solar Panels [mA]	0	500
I Busses 5V,3.3V, Battery [mA]	40	90
T Battery 1 & 2 [°C]	20	45
T Solar Panels [°C]	-120	120

Table 5.1: Rules for Telemetry Data Warning

When MCC has not received any telemetry data of one spacecraft, its respective warnings are set through a white rectangle with “No Communication” text.

Figure 5.2: No Communication Warning

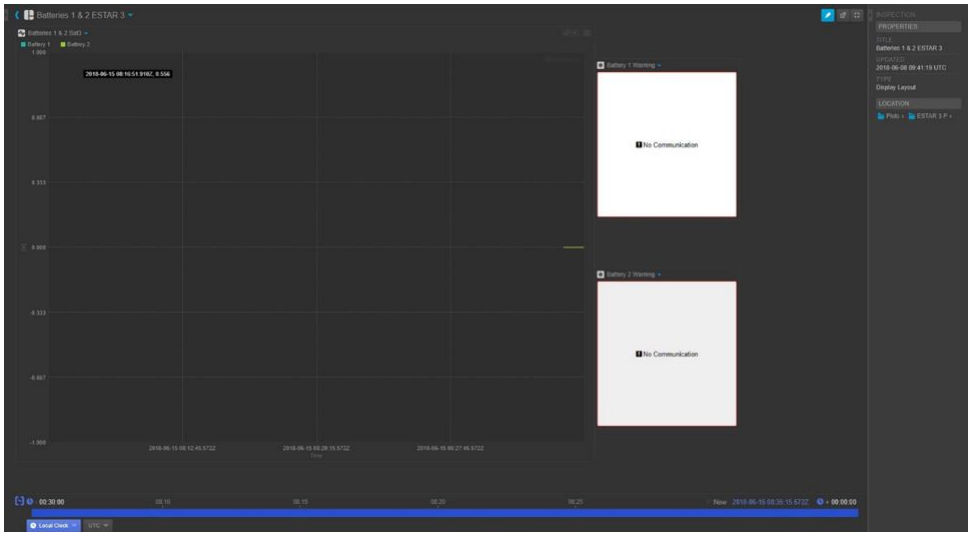


Figure 5.3: No Data Received

When telemetry data is between lower and upper limits, the warning is green and shows "xxx OK", where xxx indicates telemetry data.
When data exceeds limits, the rectangle becomes yellow with text "xxx Waring" and an icon that indicates if data has exceeded upper or lower limits.

Figure 5.4: Plot with Warning

For I Solar Panels, we insert a further rule: if telemetry data is lower than 0, appears a warning that indicates Eclipse; this means that satellite not receive light by Sun or albedo.

Figure 5.5: Eclipse Warning

All available plots are placed in Appendix B.

Conclusion

This thesis aims to develop a Mission Control Software that manages and validates telemetry data sent by small spacecraft or a satellites constellation.
This theme is object of interest for space industries and research groups of universities because, nowadays, their attention is switch to involve Nano Satellites in space environment.
The software has been developed using an existing open source software, Open MCT, that allows to integrate additional functions developed and tested, in order to reach the final goal.
JavaScript is the code language utilized to develop them, whereas the interfaces between Open MCT and the satellite, and simulated satellite itself have been created in C language.
Software has been tested with telemetry packets of E-st@r 2, received during the phase test of development of the satellite. The first test respects the requirements imposed; in fact, we compared the result with those of CubeSat Team and data correspond.
Through serial communication, we simulate satellite that sends packets from a second computer. This test has been a preliminary simulation of a mission.
In the next step, in fact, we test if software is able to manage and control data in real time; to do this, we set up the Development Board of E-st@r 2 with radio and EPS of satellite prototype.
Thanks to a mission simulator developed by another student, we organize a mission test, simulating a spacecraft that, during three orbits, sends telemetry packets. During this test, software proves his ability to manage, control, validate and visualize telemetry data in real time, demonstrating that it can be use in real mission.
In the last step, we decide to manage and validate telemetry data of a satellites constellation consisting of three Nano Satellites, which telemetry is created virtually through a string generator. Software passes also this test, showing its flexibility and adaptability to the needs and requirements of current missions.

The first future improvement of this work should be to develop the interface between Open MCT and satellite in the same language of Open MCT; this will allow future user to open just one software. Other suggestions could be the following:

- Develop a plugin inside of Open MCT that performs the tracking of the spacecraft, in order that user, managing telemetry data, can follow and have information about the satellite position and orbit parameters;

- Test the software with a real satellite constellation;
- Simulate a whole space mission, establishing phase mission and duration, through the Timeline plugin available on Open MCT.

In conclusion, it has been demonstrated that, starting from an existing open source software, it is possible to develop a "new software" that performs and ensures the needs and requirements of the user, without starting from zero in the development of ad hoc software, or, employing a high cost software, that if mission requirements change, has additional cost to be modified.

Appendix A: “Dictionary.json”

The “Dictionary.json” is the code written to enable Open MCT to show telemetry data subsections.
JSON (JavaScript Object Notation) is a text syntax for storing and exchanging data which can easily be sent to a server.
We create a folder “Telemetry”, which contains the following sections:

- **General Satellite**: it shows all general information received by satellite, such as “Destination Address”, “Source Address”, “Time”;
- **I Satellite**: it shows information about current data of subsystems;
- **T Satellite**: it shows temperature data of subsystems.

Each section has “measurements” subsections,which require the following informations:

- **name**: it identifies the title of data;
- **identifier**: it identifies the code location;
- **units**: it identifies measure’s unit of telemetry data;
- **type**:it identifies what kind of data, such as string, float number.

```
{
    ``name``: ``Telemetry``,
   ``identifier``: ``sc``,
    ``subsystems``: [
        {
            ``name``: ``General Satellite 1``,
            ``identifier``: ``gen``,
            ``measurements``: [
                {
                    ``name``: ``Destination Address``,
                    ``identifier``: ``gen.da``,
                    ``units``: ``None``,
                    ``type``: ``None``
                },
                {
                    ``name``: ``Source Address``,
                    ``identifier``: ``gen.sa``,
```

```
            ``units``: ``None``,
            ``type``: ``None``
        },
        {
            ``name``: ``Time Sat1``,
            ``identifier``: ``gen.tem``,
            ``units``: ``[s]``,
            ``type``: ``float``
        },
        {
            ``name``: ``Communication Window Sat1``,
            ``identifier``: ``gen.cw``,
            ``units``: ``None``,
            ``type``: ``None``
        },
        {
            ``name``: ``TLM Packets``,
            ``identifier``: ``gen.tpac``,
            ``units``: ``[Number packets]``,
            ``type``: ``float``
        },
        {
            ``name``: ``Last command``,
            ``identifier``: ``gen.lc``,
            ``units``: ``None``,
            ``type``: ``None``
        },
        {
           ``name``: ``Battery 1``,
            ``identifier``: ``gen.bat1``,
            ``units``: ``[V]``,
            ``type``: ``float``
                    },
        {
            ``name``: ``Battery 2``,
            ``identifier``: ``gen.bat2``,
            ``units``: `` [V]``,
            ``type``: ``float``

        },
        {
            ``name``: ``Solar Pannel x``,
            ``identifier``: ``gen.spx``,
            ``units``: ``[V]``,
            ``type``: ``float``
        },
        {
            ``name``: ``Solar Pannel y``,
            ``identifier``: ``gen.spy``,
            ``units``: ``[V]``,
            ``type``: ``float``
        },
        {
            ``name``: ``Solar Pannel z``,
            ``identifier``: ``gen.spz``,
            ``units``: ``[V]``,
            ``type``: ``float``
        }
                ]
},
{
    ``name``: ``I Satellite 1``,
    ``identifier``: ``in``,
```

```
``measurements``: [
    {
        ``name``: ``Bus 5V``,
        ``identifier``: ``in.bus5``,
        ``units``: ``[mA]``,
        ``type``: ``float``
    },
    {
        ``name``: ``Bus 3.3V``,
        ``identifier``: ``in.bus3``,
        ``units``: ``[mA]``,
        ``type``: ``float``
    },
    {
        ``name``: ``Battery 1``,
        ``identifier``: ``in.b1``,
        ``units``: ``[mA]``,
        ``type``: ``float``
    },
    {
        ``name``: ``Battery 2``,
        ``identifier``: ``in.b2``,
        ``units``: ``[mA]``,
        ``type``: ``float``
    },
    {
        ``name``: ``Battery 1 Direction``,
        ``identifier``: ``in.b1d``,
        ``units``: ``None``,
        ``type``: ``None``
    },
    {
        ``name``: ``Battery 2 Direction``,
        ``identifier``: ``in.b2d``,
        ``units``: ``None``,
        ``type``: ``None``
    },
    {
        ``name``: ``Bus Battery``,
        ``identifier``: ``in.bs``,
        ``units``: ``[mA]``,
        ``type``: ``float``
    },
    {
        ``name``: ``Solar Pannel x``,
        ``identifier``: ``in.spx``,
        ``units``: ``[mA]``,
        ``type``: ``float``
    },
    {
        ``name``: ``Solar Pannel +y``,
        ``identifier``: ``in.spy``,
        ``units``: ``[mA]``,
        ``type``: ``float``
    },
    {
        ``name``: ``Solar Pannel -y``,
        ``identifier``: ``in.spmy``,
        ``units``: ``[mA]``,
        ``type``: ``float``
    },
    {
        ``name``: ``Solar Pannel +z``,
```

```
                ``identifier``: ``in.spz``,
                ``units``: ``[mA]``,
                ``type``: ``float``
            },
            {
                ``name``: ``Solar Pannel -z``,
                ``identifier``: ``in.spmz``,
                ``units``: ``[mA]``,
                ``type``: ``float``
            }

        ]
    },
    {
        ``name``: ``T Satellite 1``,
        ``identifier``: ``t``,
        ``measurements``: [
            {
                ``name``: ``Battery 1 ``,
                ``identifier``: ``t.bat1t``,
                ``units``: ``[ C]``,
                ``type``: ``float``
            },
            {
                ``name``: ``Battery 2 ``,
                ``identifier``: ``t.bat2t``,
                ``units``: ``[ C]``,
                ``type``: ``float``
            },
            {
                ``name``: ``Solar Pannel x``,
                ``identifier``: ``t.spx``,
                ``units``: ``[ C]``,
                ``type``: ``float``
            },
            {
                ``name``: ``Solar Pannel +y``,
                ``identifier``: ``t.spy``,
                ``units``: ``[ C]``,
                ``type``: ``float``
            },
            {
                ``name``: ``Solar Pannel -y``,
                ``identifier``: ``t.spmy``,
                ``units``: ``[ C]``,
                ``type``: ``float``
            },
            {
                ``name``: ``Solar Pannel +z``,
                ``identifier``: ``t.spz``,
                ``units``: ``[ C]``,
                ``type``: ``float``
            },
            {
                ``name``: ``Solar Pannel -z``,
                ``identifier``: ``t.spmz``,
                ``units``: ``[ C]``,
                ``type``: ``float``
            }
        ]
    }
]
}
```

Appendix B: Available Plots

In this appendix are shown definitive plots of all subsystems data, including warnings for the satellite constellation.

ESTAR 1
This section shows plots of one satellite. We can see process of charge and discharge of Batteries 1 and 2 in Figures 6.6 (a) and (c).
Figure 6.6 (b) shows Voltage of Solar Panels and Figure 6.6 (d) shows warning for satellite in eclipse.

(a) Batteries 1 & 2

(b) Solar Panels

(c) I Batteries 1 & 2

(d) I Solar Panels

(e) Busses Consumption

(f) T Batteries 1 & 2

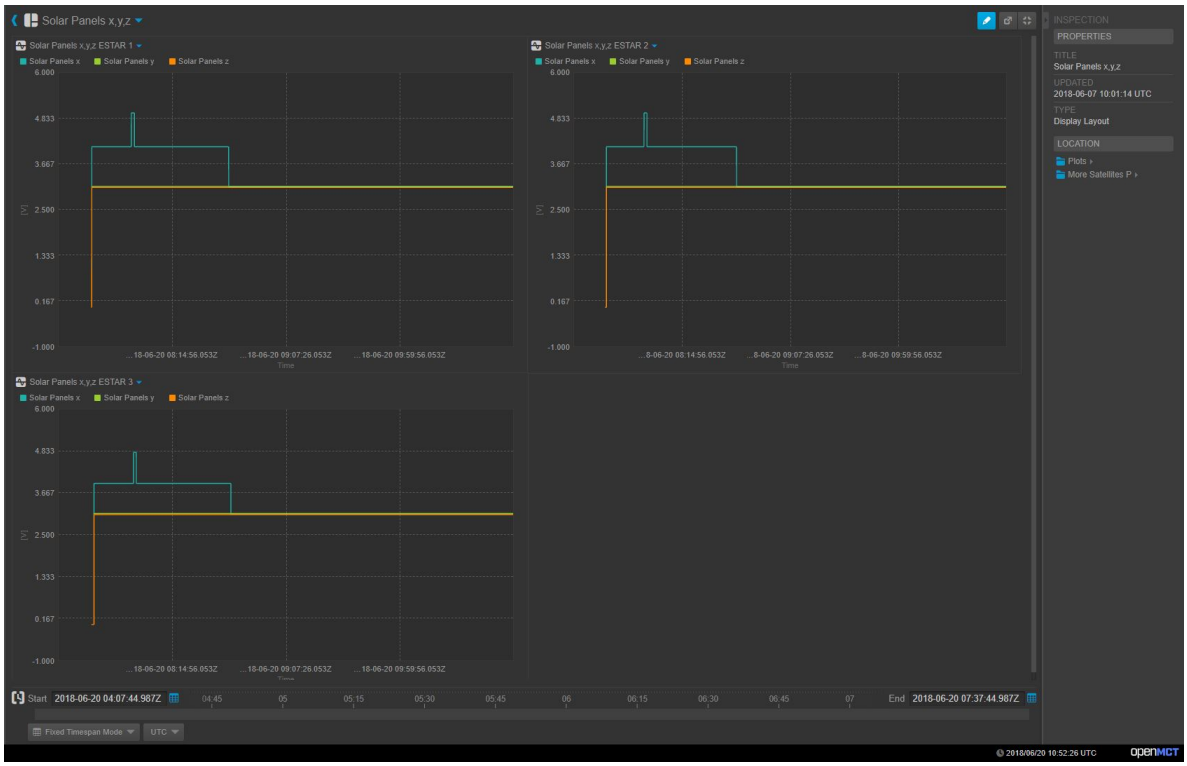
(g) T Solar Panels

Figure 5.6: ESTAR 1 Plots

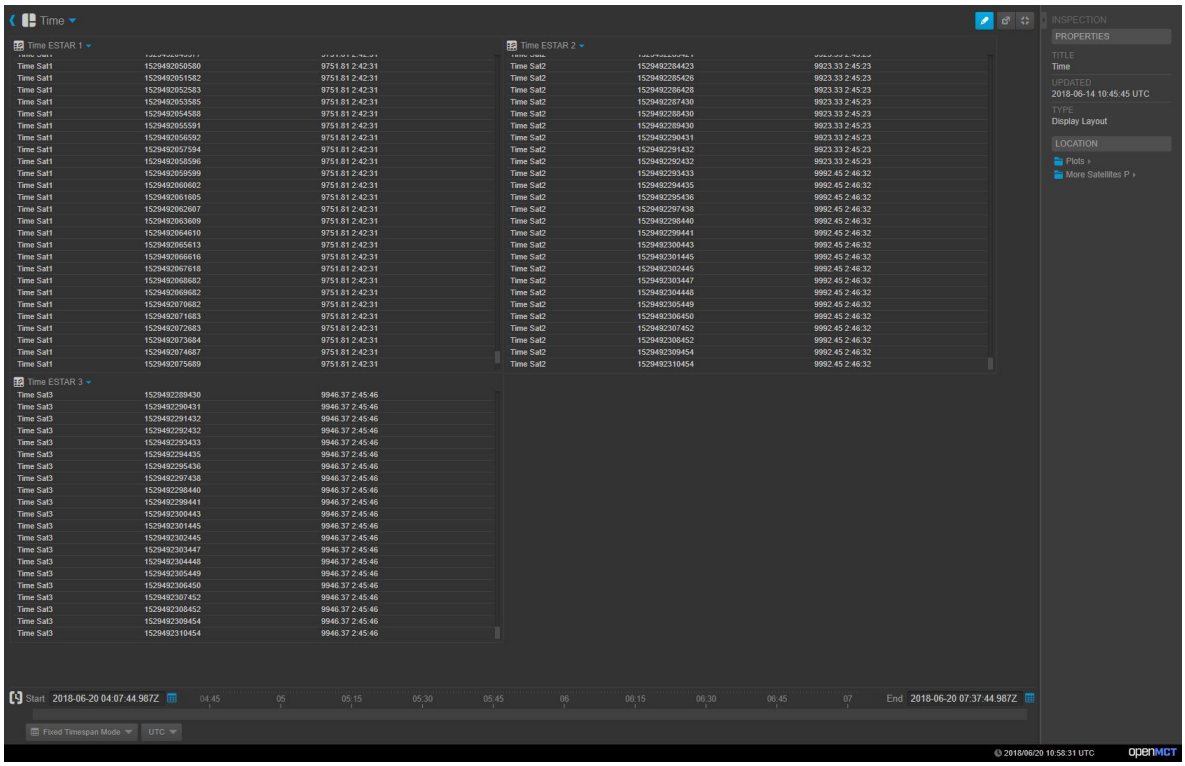
More Satellites

This section, on the other hand, shows plots of three satellite of constellation analyzed. Plots are set up to allow user to see a specific telemetry data, comparing with that of other satellites.

(a) Batteries 1 & 2



(b) Solar Panels



(c) Simulation Time

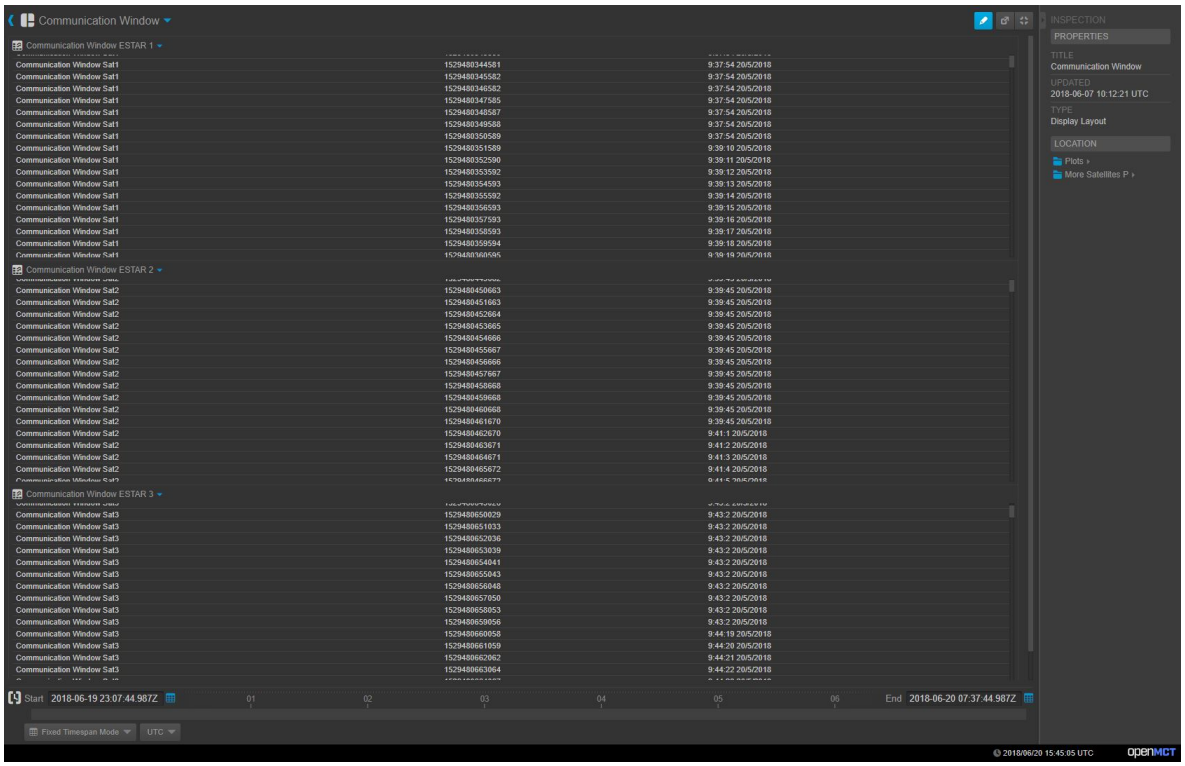
(d) I Batteries 1 & 2

(e) I Solar Panels

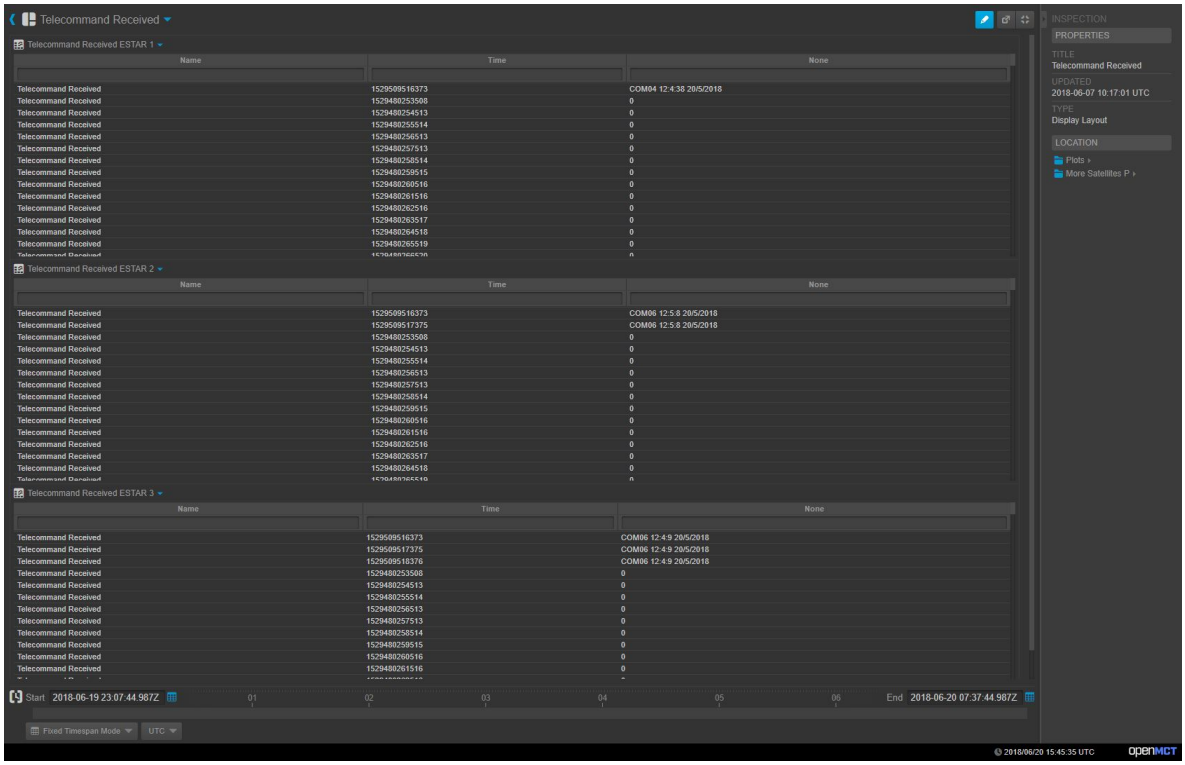
(f) Busses Consumption

(g) T Batteries 1 & 2

(h) T Solar Panels



(i) Communication Window



(j) Telecommand Received

Figure 5.7: More Satellites Plots

Bibliography

[1] Ham Radio Club, AIR, section BRA
`http://www.aribra.it/hf/info_e.php`

[2] Justin Morris, Scott Zemerick, Matt Grubb,John Lucas and Robert Bishop, *" Simulation-To-Flight (STF-1): A Mission to Enable CubeSat Software-based Validation and Verification"*, American Institute of Aeronautics and Astronautics,2005

[3] Justin R Morris,Matthew Grubb, *"Simulation-to-Flight (STF-1) Proposal"*, West Virginia University,2014

[4] *"Simulation-to-Flight(STF-1)"*,West Virginia University,2014,
`http://www.stf1.com/simulation.php`

[5] Hawaii Space Flight Laboratory (HSFL), *"Comprehensive Open-architecture Solution for Mission Operations Systems (COSMOS)"*,University of Hawaii, Manoa,2015,
`http://cosmos-project.org/`

[6] Hawaii Space Flight Laboratory (HSFL), *"COSMOS Manual"*,University of Hawaii, Manoa,2015

[7] Trevor C. Sorensen, Eric J. Pilger, Mark S. Wood, Elizabeth D. Gregory, Miguel A. Nunes, *"Development of the Mission Operations Support Tool (MOST)"*, Hawaii Space Flight Laboratory, University of Hawaii, Honolulu, HI, 96822 (2010)

[8] *"Open Source Mission Control Technologies (Open MCT)"* ,Ames Research Center, Jet Propulsion Laboratory,2017
`https://github.com/nasa/openmct`

[9] Ames Research Center, *"Open Source Mission Control Technologies (Open MCT)"*, Jet Propulsion Laboratory,2017
`https://nasa.github.io/openmct/`

[10] *"Open MCT Tutorials"* ,Ames Research Center, Jet Propulsion Laboratory,2017
`https://github.com/nasa/openmct-tutorial`

[11] *"Open MCT To-do-List"* ,Ames Research Center, Jet Propulsion Laboratory,2017
`https://nasa.github.io/openmct/docs/tutorials/\#to-do-list`

[12] *"Open MCT Mars Rover"* ,Ames Research Center, Jet Propulsion Laboratory,2017
`https://openmct-demo.herokuapp.com/`

[13] Teunis van Beelen, *"RS-232 Serial Communication Libriary"*
`https://www.teuniz.net/RS-232/`

[14] PUMPKIN, Rreal Time Software, *"Pumpkin CubeSat Kit User Manual"*, 750 Naples Street, San Francisco,CA 94112

[15] SCS, *"Tracker/DSP TNC Installation Guide"*,Roentgenstr. 36, D-63454 Hanau, GERMANY

[16] Trevor Sorensen, Eric Pilger, Mark S. Wood, Miguel A Nunes, *"Development of a Comprehensive Mission Operations System Designed to Operate Multiple Small Satellites"*, University of Hawaii, 1680 East-West Rd. POST 501, Honolulu,August 2011

[17] S. Corpino, S. Chiesa, F. Stesina, N. Viola, *"CUBESATS DEVELOPMENT AT POLITECNICO DI TORINO: THE E-ST@R PROGRAM"*, 61st International Astronautical Congress, Prague, CZ

[18] Fabrizio Stesina, *Space Systems course notes Lesson 19*, Politecnico di Torino

[19] PolySat team, *"The Cal Poly Earth Stations"*
`http://www.polysat.org/earth-station/`

[20] N. Peccia, *"EGOS - The European Space Agency (ESA) Ground Operations Software System"*, ESA/ESOC, Robert Bosch Strae 5, 64293 Darmstadt, Germany, 2008

[21] Paul Wooster, David Boswell, Patrick Stakem, Jessy Cowan-Sharp, *"Open Source Software for Small Satellites"*

Riassunto

Questa tesi si basa sullo sviluppo, l'implementazione e la validazione di un software utilizzato nelle Ground Control Stations (GCS) per missioni di Nano Satelliti, il quale si occupa della gestione e del controllo della telemetria e dei telecomandi dei sottosistemi di veicoli spaziali. In genere, le operazioni a terra durante la missione sono sempre complesse e richiedono molti sforzi in termini di tempo, costi e risorse necessarie.
Un modo per ridurre tempi e costi consiste nell'utilizzare un software specifico che consente all'utente di gestire contemporaneamente telemetria e comandi di tutti i sottosistemi in modo autonomo.
Per tale motivo, è stata effettuata una ricerca bibliografica di software esistenti che svolgessero le funzioni richieste, valutando se, per il nostro obiettivo, fosse necessario impiegarne uno già sviluppato e testato, scegliendo tra "open" e "close" source, o se fosse necessario progettarne uno "in-house developed".

Il software finale sviluppato e utilizzato è una combinazione di un software open source esistente con un layout grafico che ne facilita l'utilizzo da parte dell'utente e un'implementazione di funzioni per il controllo e la gestione dei dati di telemetria e comandi.

Per validare tale software, abbiamo utilizzato i dati di telemetria ricevuti da E-st@r 2, satellite sviluppato dal CubeSat Team del Politecnico di Torino, durante un test preliminare, simulando il satellite che invia pacchetti di telemetria attraverso un programma esterno. Successivamente, si è proceduto alla gestione, controllo e analisi dei pacchetti di telemetria ricevuti, in tempo reale, dalla Development Board, su cui sono stati posizionati la radio e l'EPS del prototipo di E-st@r 2.

Infine, il software sviluppato è stato ulteriormente modificato e migliorato in modo tale da renderlo in grado di gestire una costellazione costituita da tre Nano Satelliti, uno dei quali, E-st@r 2 e, tramite un generatore di stringhe, è stata elaborata la telemetria degli altri due satelliti.