IAC-15-D1.6.5

STARSIM: A STAND-ALONE TOOL FOR "IN THE LOOP" VERIFICATION

Fabrizio Stesina
Politecnico di Torino, Italia, fabrizio.stesina@polito.it

Sabrina Corpino
Politecnico di Torino, Italia, sabrina.corpino@polito.it

Lorenzo Feruglio
Politecnico di Torino, Italia, lorenzo.feruglio@polito.it

In the last decades, systems have strongly increased their complexity remarking the importance of defining methods and tools that improve the design, verification and validation of the system process: effectiveness and costs reduction without loss of confidence in the final product are the objectives that have to be pursued.

Within the System Engineering context, the modern Model and Simulation based approach is a promising strategy because it reduces the wasted resources with respect to the traditional methods. Considering a wild range of simulations architectures and methods, crucial stages are defined by algorithm in the loop (AIL), software in the loop (SIL), hardware in the loop (HIL).

This paper presents an in-house tool, developed at Politecnico di Torino, able to perform different simulation sessions in any phase of the space product life-cycle using a unique and self-contained platform, called StarSim: modularity, flexibility, real time operation, fidelity with real world, ease of data management, effectiveness and congruence of the outputs with respect to the inputs are StarSim sought-after features. The main issue is to guarantee the possibility to verify the behavior of the system under test thanks to virtual models, that substitute all those elements not yet available and all the non-reproducible dynamics and environmental conditions. Progressively, pieces of the on board software and hardware can be introduced without stopping the process of design and verification, avoiding delays and loss of resources.

StarSim has been applied for the first time on the e-st@r-II Cubesat, developed the “CubeSat Team Polito” within the ESA Education Office initiative called “Fly Your Satellite”. StarSim has been mainly used for the payload development, an Active Attitude Determination and Control System, but StarSim’s capabilities have also been updated to evaluate functionalities, operations and performances of the entire satellite. AIL, SIL, HIL simulations have been constantly performed, successfully verifying a great number of functional and operational requirements. In particular, attitude determination algorithms, control laws, modes of operation have been selected and verified; software has been developed step by step and the bugs-free executable files have been loaded on the micro-controller. Actuators, logic and electrical circuits have been designed, built and tested and sensors calibrated. Problems such as real time and synchronization have been solved, allowing, at the end of the process, a complete hardware in the loop simulation test for the entire satellite.

The case study has allowed the successfully validation of the first release of StarSim.

I. INTRODUCTION

In the last decade, systems have become more and more complex due to several factors. From the technical point of view, the number of functions implemented in a single system is continuously increasing thanks to the dramatic progress of technology in the last decades. At the same time, the relationships between functions and hardware of the system, as well as the interactions between different elements and disciplines which concur to its definition, are growing up in number and gaining in complexity.

Within the System Engineering (SE) frame, a new approach is taking shape for the design and verification of these complex systems: the Model Based System Engineering (MBSE) [1]. MBSE is increasingly making use of the Model and Simulation (M&S) approach, which is replacing the superseeded “design-produce-test” approach that more expensive in terms of resources (budget and time) needed to get the real hardware as well as a minimum defect in the hardware (due to design or manufacturing) might jeopardize the whole test session thus delaying the project schedule. Moreover, it is sometimes impossible to test some features due to the special environment (Space) in which the system will operate.

Simulation seems to be the perfect means to “test” space systems and it can be used throughout the product life-cycle. In particular, M&S based approach allows verifying solutions before hardware manufacturing.

Actually, on board software and hardware functionalities cannot be verified and validated utilizing only "pure" simulation because it may not be exhaustive. In fact, notwithstanding the modeling effort, pure simulation alone gives only part of the answer to the problem and verifications on actual software and hardware are necessary. The real system may exhibit behaviors which cannot be modelled perfectly and that affects the outputs to a great extent, for example delays and uncertainties are not well predictable by models. For this reason, verification sessions with real on board software, controller/processor and hardware in the loop are necessary. As specified by [2], a few important stages of interaction between the elements of the system along the verification process can be defined:

- Algorithm in the loop (AIL). The algorithms of interest are added to the pure numerical simulation.
- Software in the loop (SIL). Algorithms are translated into the final programming language, reproducing all functionalities, but they run on ground hardware, different with respect to running it on a flight unit architecture (e.g. PC vs embedded-PC)
- Hardware in the loop (HIL). Real hardware is included in the simulation loop, starting from the embedded elements and adding typically of sensors and/or actuators.

One of the main concerns in the verification of space systems using M&S Base Approach (M&SBA) performing "in-the –loop" simulation sessions is to enjoy the use of one or more cooperating simulation platforms.

This paper presents StarSim, a simulation platform developed by the Cubesat Team of Politecnico di Torino (Italy). It is a stand-alone tool that allows building different simulation architectures and verifying algorithms, develop software, evaluate hardware capabilities.

In this paper the general structure and the main features of StarSim are proposed and its use in a real project is presented.

In particular, this section carries on with the analysis of state of art of software, tools and facilities for space product verification. Section II analyses the advantages and drawbacks in the development of an in-house simulator. Section III enters into details of the StarSim description. Section IV shows the use of the first release of StarSim for the verification of an Attitude Determination and Control System (ADCS) of the Cubesat e-st@r-II. Section V remarks the obtained results and the future updates of StarSim.

State of Art

Each phase of the space product life-cycle shows specific needs. For this reason, different features and capabilities are required for the simulation platforms and the tools in charge of leading design, verification and validation.

In the feasibility phase, the design tools shall be able to perform trajectory/orbit simulations, to manage budget analyses through linked spreadsheets, to provide 2D/3D visualization, to manage databases for analysis, and to generate reports. This kind of functions are usually carried out at prime contractor site or in a space agency, such as for instance the *Concurrent Design Facility* at the European Space Agency [3], and the *Project Design Center-TeamX* at NASA/JPL [4].

In the design phase, simulation is used in the SE process at system and subsystem level. Analyses are carried out in many disciplines such as structure mechanics, thermal engineering, and orbit design using dedicated tools. For functional simulations, commercial toolboxes which comprise special libraries for control and system dynamics engineering are used, e.g. Matlab® and Modelica®.

The MDVE - Model based Development and Verification Environment, developed in-house by Airbus Defence and Space, represents a good example of simulation suite available in the late stage of the development process [5]. The core elements of the MDVE are the flight computer simulator and a real-time simulator in which the rest of the spacecraft and the environment are modelled.

One drawback of having different platforms for each design phase is the issue of information exchange from one project step to the next. Standardization could be useful to limit this problem. The most common solution is to adopt Model Driven Architecture (MDA) that, among other things, allows managing documents, exchanging/sharing design model Various modelling languages have been defined in order to target different domains such as web development, hardware [6], software [7], and systems [8]. Although SE has been in existence for more than five decades, there has been no dedicated modelling language for this discipline up until recently [9].

II DEVELOPMENT OF IN-HOUSE SIMULATOR

One of the keys for the verification of complex system is the choice among the reuse of an existing simulator, the development of a simulator based on commercial and general purpose software or the development of a new, in-house designed simulator. Probably, the better solution is, in general, the reuse of a simulator previously developed and, possibly, already validated because it confers a higher level of confidence, and it permits to reduce the efforts of development and validation for the engineers and developers as well as the time of learning and training for the operators, analysts and users.

However, if the simulator does not exist two main hypotheses could be investigated: to use and integrate general purpose tools and commercial software or design and build a new simulator.

Commercial software and tools costs are related to the acquisition of licenses: payment of fees could block the use if a low cost solution is pursued. In-house simulator based on free software platforms and programming languages (i.e. C/C++ or Java, and Linux as OS) strongly reduce the costs. This is because they are mainly due to the human resources needed for the development. However the human resources costs are present also considering commercial general purpose tools that require time-consuming training and learning activities.

Moreover, commercial software providing functions and models unused in specific field (i.e. managerial libraries are probably useless for a space vehicles simulator) but within the license the customer pays necessarily also for that. In other case, specific tools or libraries are unavailable or out of the basic licenses and they should be expressly bought with further costs. In-house simulators are developed independently and starting the design from the lower levels so that they are defter and any modification and change results easier to be implemented because a higher control due to the direct development of the code is possible.

In the case of simulators that involves more elements hardware and different software environments as well as simulators that works in different phases of the product life cycle, two other issues should be considered: 1) how to share information and data between tools and development environments deriving by different providers, 2) how to guarantee the compatibility of tools among themselves and with the hardware architecture (e.g. processor), kernel, firmware and operating systems.

Considering the general purpose tools and commercial software, the idea is to put together different tools in the same platform. A recent solution in this sense is provided by the ensemble of UML and SysML. Actually, it seems that another evaluable solution derives from the development of in-house simulators with free or well known software, e.g. TASTE [10].

In-house simulator allows to group all the features required along the entire design cycle within a unique platform, independent from the type of software and hardware that will be designed, developed and verified using the same simulator. Moreover, this kind of simulators remains independent from the general purpose tools, providing a self-contained environment. Pay attention that in-house simulator does not exclude the possibility to interface with commercial software but can properly work without them. On the contrary, their use would help the in-house simulation within an industrial field, where in some case the internal tradition and background tends to be maintained: the in-house simulator could accept inputs generated from other software in previous projects and adapt them to execute simulations.

The bigger drawbacks for a in-house simulator remains the validation and the accreditation. Simulator shall be validated from the hardware, software and models point of view. Validation and accreditation require long and complex technical process and procedures This is a big issue that, in any case, also general purpose tools cannot solve easily, because for example putting together two certified software does not automatically lead to a certified software. On the contrary, validating a unique platform is a big issue but once the capabilities are validated and accredited the simulator has a "secure core" from which updating database and solution and through which verified future products and design with a better confidence level.

III.STARSIM

StarSim is a design & verification platform able to support AIL through HIL simulations. It is suitable for evaluating algorithms and software implementation, and hardware functionality to the desired extent.

StarSim has been designed to adapt to several missions and satellite configurations, and it can be used throughout the development process of the system, from design to qualification, for the verification of functional and operational requirements. The application of the simulator improves the understanding of the behavior of the system early in the life-cycle.

StarSim has been designed taking into account the following key drivers and requirements:

- Flexibility with respect to project phases
- Reusability with respect to projects diversity
- Fidelity with respect to the "real world"
- Real time and off-real time capability
- Multisession operations
- Code auto-generation
- Low cost,

The simulator is based on an intrinsic modular structure to support the project development throughout the life-cycle and to adapt to several missions and system configurations. Fig. 1 illustrates the global architecture of the simulator which is described into the detail in the following paragraphs. Four main blocks can be individuated: the simulation unit (SU) and the test object (TO) are connected by the ground support equipment (GSE) box and the user can operate through a user interface from the control console (CC).

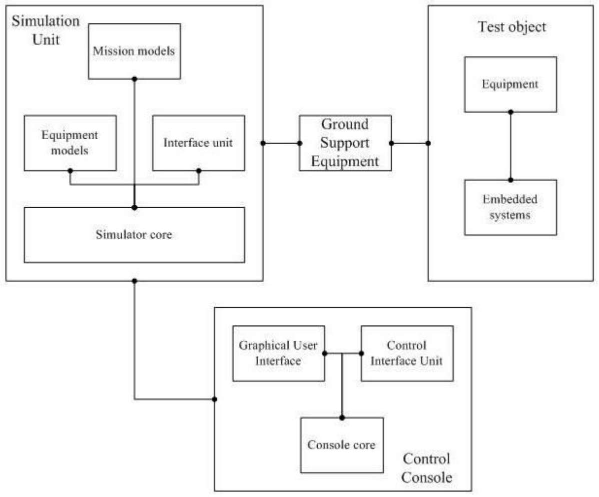

Fig 1: StarSim architecture

III.I. Simulation Unit

The simulation unit is constituted by the simulation core, a model database containing the mission models, and the equipment models, and the interface unit.

III.I.I. Simulation Core

The simulation core is a multi-processor workstation. Linux is the Operating System (OS) and an *ad-hoc* kernel has been built in order to allow different types of simulations (real-time or off real-time) for several arrangements of the *TO* hardware.

StarSim operates through two application software to deal with the simulation setup and with the simulation execution respectively.

The first program, written in Python, allows to setup the simulation in terms of 1) models choice, 2) definition of the logical interfaces with the *TO*, 3) definition of the I/O through the CC, 4) scheduling of tasks according to priorities, time and logical sequence during the execution, and 5) collection of all the information related to the simulation session.

The second program is written in C++, and it permits the execution of the simulation. This program 1) contains all the custom developed functions and header files needed to perform the simulation, 2) runs the virtual models, 3) manages the hardware interface with external elements (TO and/or GSE), 4) manages time synchronization and data, and 5) saves the information for post processing activities.

III.I.II. Models database

Using an M&SBA, one of the fundamental part of any simulation is the ability of modelling the elements of the system under study. StarSim has been designed according to the principle that any simulated system should behave as the real one not only in terms of numerical or physical outputs, but also in terms of interface with the other elements involved in the simulation loop. The level of detail for the selected model shall be carefully evaluated led a trade-off between the accuracy and fidelity of the simulation results and the computational cost on the other side. Usually, increasingly more detailed models are used according to the advancement of the design. The models come from literature when available, or they are derived from information given by the manufacturers, or they can be obtained from data acquired during open-loop testing sessions of the specific equipment. The models are organized within a database, divided into main categories:

- *Equipment models* include devices and equipment of the TO, e.g. sensors, actuators, power sources
- *Mission models* include environmental and spacecraft motion models, e.g. the Earth Magnetic Field model, the thermal flows influencing the spacecraft, the orbit (SGP4 propagator) and the satellite rotational dynamics and kinematics
- *GSE models and routines* which include devices/equipment models and their interfaces, e.g power supply unit control and the related software.

Moreover, basic mathematical functions, transformations and conversions, display and visualization elements are part of the models database.

III.I.III. Interface Unit

The interface unit has the main tasks to connect the simulation unit with internal and external modules and part of the entire simulator. Depending on the configuration chosen for the simulator, there will be communication between processes with different features. The interface unit connects the SU with all the other elements involved in the simulation. The interface unit is in charge of managing communications both when two processes are executed by the same processor (*software link*), and in the case the two or more processes are handled by two different controllers (*hardware link*). In both cases, a data link with a precise path for the data exchange between the processes is essential.

StarSim has been designed to support a great number of communication types, in order to provide the user with the widest variety of options, in particular:

- *named pipes* for the software interfaces
- communication standards for the hardware interfaces (serial RS232, USB, LAN, CAN bus and I2C).

III.II Control Unit

The Control Console (CC) gives the operator the authority over the simulation session in any moment, through displays, keyboard and mouse. Before the simulation execution, it favors the definition of the simulation architecture and the parameters setup; during the simulation execution, it permits real time control activities as well as monitoring of values and trends of sensible parameters and variables; at the end of the simulation session, it allows the data handling and post-processing. The CC GUI (Graphical User Interface) is written in Python and it has been designed to simplify the human-machine interface.

III.III. Ground Support Equipment

The GSE are elements that give/receive stimuli to/from the *TO* according to the values computed by the simulator and serve to perform the simulation but they belong neither to the SU nor to the TO. StarSim can manage three main categories of GSE: 1) equipment to stimulate the test object, 2) devices to supply electrical power to the test object and other parts involved in the simulation session, and 3) instruments devoted to the communication/exchange of data between two elements of the simulator.

III.IV. Test object

The test object is the system (or some of its parts) that would be investigated during the verification campaign. It can be constituted by:

- Embedded Systems: electrical board based on a "smart unit" (boards based on micro-processor/controller), circuits and devices needed for the *test object* good working, i.e. timers and synchronization systems, volatile and non volatile memories and input/output logical and physical ports;
- Other equipment including mechanical, electrical, and RF devices. Examples are actuators, sensors, driver logic circuits, signal conditioning boards, motors, transceivers, antennas, solar panels and batteries;
- Algorithms and software.

The test object is peculiar to the specific project and simulation session, and hence a possible TO will be described into the details in the next section, where an example of application of the StarSim is given.

IV. TEST CASE: E-ST@R-II CUBESAT

In this paper, the use of the StarSim is presented as support of the engineering team of the Cubesat Team during E-st@r-II CubeSat mission development. In particular, it aims at supporting the engineering team with the design and AIV activities of small-scale satellites [11].

CubeSats are small satellites which share a common interface and equipment standardization [12]. The basic CubeSat is a 10cm-side cube-shaped platform whose mass is less than 1.33 kilograms [13] By its nature, a CubeSat is a cheap spacecraft which can be produced in only a few months. CubeSats have been mainly developed for educational purposes, but they are increasingly being used for real science and service missions [14]. CubeSats can be useful for the attainment of a broad set of mission goals, including science, technology demonstration, communications, and Earth observation [15], [16]. This will require the development of adequate technology to increase CubeSat performance. Furthermore, it is necessary to improve mission reliability, because educationally-driven missions have often failed. The failure of CubeSats is dominated by infant mortality, which can be traced back to design weakness and/or ineffective Assembly-Integration-Verification (AIV) planning and execution. An effective and exhaustive Verification and Validation (V&V) process may help to increase the reliability of small-scale satellite [17].

IV.I CubeSat Configuration

The payload of e-st@r-II is constituted by an Active Attitude Determination and Control System (A-ADCS) [18]. The A-ADCS is managed by a dedicated micro-controller (RD129-ARM9), on which the algorithms for determination and control run. The ARM9 micro-controller passes data to the OBC through a serial port and it receives sensor measurements on a second serial port. Internal memories (both volatile and non-volatile) are used by the ARM9 to save and load data and models. The attitude is determined using data sensed by an Inertial Measurement Unit (IMU) which includes gyros and one three-axis magnetometer. The actuation is accomplished by three Magnetic Torquers (MTs) which are commanded using a PWM logic circuit driven by three timer ports. The spacecraft bus is constituted by the Electrical Power System (EPS), the Communication System (COMSYS), the On-Board Computer (OBC), and the Structure and Mechanism System (S&M). The EPS collects and transforms solar radiation energy by means of five solar panels made of GaAs cells. Electrical power is stored in two battery packs, then regulated and distributed to other subsystems through two power channels at 3,3 Volt and 5 Volt, and one unregulated battery bus (voltage range is 7 to 8.2 V). The COMSYS is an in-house-developed design and manages downlink and uplink signals. It exchanges information with the OBC by a piece of equipment based on commercial components: a PIC16 modulates/demodulates and checks the signal, a radio transceiver amplifies the signal and a dipole antenna transmits/receives the signal to/from Earth. The OBC performs all on-board activities related to the command and data handling functions, the data storage in the SD memory card, and time synchronization. The CubeSat flight unit and is shown in Fig.2. The main bus represents the common interface among subsystems, for both the data flow and the power line.

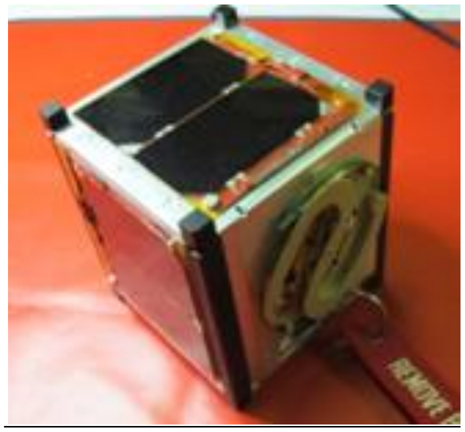
Fig. 2: e-st@r-II cubesat

IV.I.I. A-ADCS

StarSim has been extensively used in the design and verification process of the A-ADCS of e-st@r-II. The simulator helped the engineering team in the definition of the A-ADCS design for choosing the system architecture, the algorithms for the determination and control functions, the operative modes, and the hardware. It has been used for verification purposes by the AIV team to test the execution of each A-ADCS task, such as the transition between operative modes and the ability to determine/control the attitude, while monitoring the power consumption and other parameters of the system, as detailed in the next sections.

IV.II. Verification Campaign

This section presents the main results obtained during the verification campaign. AIL, SIL, HIL simulations are described in terms of configurations and settings of StarSim and the respective results are shown and analysed.

IV.II.I Algorithm In the Loop (AIL) verification

Many AIL simulation sessions have been completed from the feasibility phase to the final qualification of the CubeSat. These sessions supported:

- the evaluation and validation of the virtual models
- the definition, comparison and choice of the attitude determination algorithms for any mission step and operative mode. Fig.3 shows the comparison of the determination algorithms (quest, q-method, and Extended Kalman Filter - EKF). The algorithms have been compared in terms of Mean Knowledge Error (MKE) [19], The EKF algorithm resulted the more accurate, as shown in Table1.
- the definition, comparison and choice of the attitude control laws. For the detumbling phase, a *Y-Thomson* law has been selected. For the the stabilisation phase, the PID and LQR strategies have been assessed and compared.(Table 2and Table 3) highlights the performance of the controller in this phase: the LQR allows to achieve a better and more stable pointing
- the understanding of the A-ADCS behaviour in terms of the time evolution of the state variables such as angular velocity and attitude

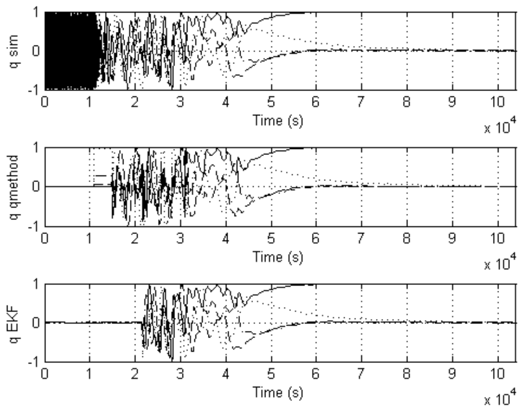

Fig.3: comparison between attitude determination algorithms

	q1	q2	q3	q4
MKE-qmethod	0.0031	0.0019	0.0017	0.0011
MKE – EKF	0.0001	0.0003	0.0007	0.0003

Table 1: MKE evaluation for q-method and EKF algorithms

	PID	**LQR**
x(°)	[5 : 7]	[+0.5 : 1]
y(°)	[-1 : 1]	[-0.5 : 0]
z(°)	[-6 : 0]	[-1 : 0]

Table 2: Pointing accuracy: comparison between PID and LQR

	PID	**LQR**
x(°)	0.5°/10000 s	0.05°/10000 s
y(°)	1°/10000 s	0.1°/ 10000 s
z(°)	1.5°/10000 s	0.5°/ 10000 s

Table 3: Pointing stability: comparison between PID and LQR

The *simulation process* for the AIL simulations is presented Fig.4 as pseudo-code. Each bold function corresponds to a C/C++ routine containing models, management functions and algorithms.

```
SIMULATION PROCESS

Simulation Setup
    Time_set(year); // time config //
    Set_Config(#processes, processes_type, #interfaces, interface_type);
Simulation loop
    while (1){
    Time_mngt (t_start, t_end, // year, time_step);
/Orbit and environment simulation/
    Orbit_propagation (norad, t_step, year, // Lat, Long, Alt);
    EMF (quaternion, year, time_step, Lat, Long, Alt, // B_body, B_orbit);
    Gravity_gradient (quaternion, // gravity_t);
    Aerodynamic_torque (quaternion, // aerodyn_t);
    Magnetic_residual (B_body, // mag_res_t);
    Magnetic_torque (B_body, // mag_moment, magnetic_t);
/Spacecraft dynamics/
    Torques = gravity_t+aerodyn_t+mag_res_t+ magnetic_t;
    DynamicsRK4 (Torques, t_step, quaternion,
                    // w_body@inertial, inertia_matrix);
    Rotation (quaternion, w_body@inertial, // w_body@orbit);
    Kinematics (w_body@orbital, time_step, // quaternion);
/Sensors simulation/
    Gyro_3axis (w_body@inertial, // w_body@inertial_meas);
    Magneotmeter_3axis (B_body, // B_body_meas);
/Attitude determination/
    EKF (quaternion_EKF, w_body@orbit, quaternion_meas,
          w_body@orbit_meas,
         // estimated_quat, estimated_ w_body@orbit);
    Q_method (B_body, B_body_meas, // q_qmet);
/Attitude control/
    Sat_control (estimated_w_body@orbit, estimated_quat, q_qmet,
    desired_quat,B_body_meas, // mag_moment);
    PWM (mag_moment, // duty_cycle);
/Actuators simulation/
    Torquer (duty_cycle, // mag_moment_real);
/Data management/
    Data_mngt (output.txt, simulation_time, Lat, Long, Alt,
         estimated_w_body@orbit, estimated_quat, desired_quat,
         B_body_meas, mag_moment, duty_cycle, Torques);
    }
```

Fig. 4: pseudo code of simulation process for AIL

The process is constituted by the following elements:

- Simulation setup functions
- Time management function
- Orbit propagator (SGP4)
- Earth Magnetic Field - EMF (IGRF2011-2015)
- Attitude disturbance torques (aerodynamic torques, gravity gradient, magnetic residual due to electronics)
- Rotational dynamics and kinematics of the satellite
- Sensors. IMU model (constituted by the magnetometers and gyroscopes models) conveniently setup, calibrated and including noise and other non-ideal features
- Attitude determination algorithms. Deterministic algorithms (*quest*, *q-method*) using the simulated magnetometer measurements and the EMF model, and *Kalman Filters* on the state variables of the satellite have been implemented and used alternatively in different simulation sessions
- Control laws. A *Y-Thomson* law is used for the detumbling phase. *PID* (Proportional Integrative and Derivative) and *LQR* (Linear Quadratic Regulator) controllers are considered for the stabilization phase. Only *PD* (Proportional Derivative) law has been implemented in order to control slew maneuvers
- Control logic for the actuators is made by the PWM logic algorithms
- Actuators. MTs model is defined and calibrated taking into account physical and electrical parameters
- Data management functions to update variables and save data into file for post-processing activity.

This code represents the backbone of all simulations, and it can be modified according to the project stage and/or spacecraft configuration by substituting the relevant models.

IV.II.II Software In the Loop (SIL) verification

In the SIL configuration, the verification sessions have the main objectives to verify:

- the protocol and data extraction for IMU/ARM9 communication
- the protocols and data extraction for A-ADCS/OBC communication
- the A-ADCS operative modes and their transitions.

The SIL configuration (Fig 5) foresees the presence of more processes but they all run on the SU.

There is a process for each simulated processing unit able to execute a series of functions and operations: the *ADCS process* and *OBC process* emulate the A-ADCS and the OBC processor respectively, and the *simulation process* executes the supporting functions. The exchange of data/information among the processes occurs through named pipes. The *simulation process* is similar to the AIL simulation process, apart from the determination and control functions that are carried out by the *ADCS process*. Moreover, a few functions to support the exchange of information between the two processes have been included in the *simulation process*. IMU's data are passed to the *ADCS process* through a pipe (*pipe_imu*)

The *ADCS process* contains the functions that runs in the ADCS computer. It manages the operative modes, acquires the IMU's data, and computes the algorithms for attitude determination and control until the definition of the duty cycle command for the MTs is done. These values are passed to the *simulation process* through the *pipe_debug* channel.

The *OBC process* refers to the functions related to the A-ADCS/OBC communications. Only telemetries and commands exchanged between the two subsystems are taken into consideration. Commands can be sent from the CC to the *OBC process* in order to simulate the capability of changing the A-ADCS operative mode or updating A-ADCS parameters from the GCS.

```
SIMULATION PROCESS

Simulation Setup
Simulation loop
    while (1){
/Orbit and environment simulation/
/Spacecraft dynamics/
/Sensors simulation/
/Data exchange with ADCS process/
    ADC (w_body@inertial_meas, B_body_meas, //
            w_body@inertial_format, B_body_format);
    IMU_packet (w_body@inertial_format, B_body_format,
                    // IMU_string);
    write (pipes_imu, IMU_string, 40);
    read (pipes_debug, command_string, 20);
    extract_data (command_string, // duty_cycle)
/Actuators simulation/
/Data management/
    }

ADCS PROCESS

Process Setup
    Time_set (year);
    Set_interface (#interfaces, interface_type);
Simulation loop
    while (1) {
    Time_mngt (t_start, t_end, // year, time_step);
/Data exchange with OBC process/
    read (pipe_obc, O2A_string, 20);
    extract_data (O2A_string, op_mode);
/Data exchange with  simulation process/
    read (pipe_IMU, IMU_string, 40);
    extract_data (IMU_string, w_body@inertial_meas, B_body_meas);
/Attitude determination/
    EKF (quaternion_EKF, w_body@orbit, quaternion_meas,
            w_body@orbit_meas,
          // estimated_quat, estimated_ w_body@orbit);
    Q_method (B_body, B_body_meas, // q_qmet);
/Attitude control/
    Sat_control (estimated_w_body@orbit, estimated_quat, q_qmet,
    desired_quat,B_body_meas, // mag_moment);
    PWM  (mag_moment, // duty_cycle);
/Data exchange with  simulation process/
    string_generation (duty_cycle, command_string);
    write (pipe_debug, command_string, 20);
/Data exchange with OBC process/
    string_generation (simulation_time, estimated_quat,
          w_body@inertial_meas, B_body_meas duty_cycle, // A20_string);
    write (pipe_obc, A2O_string, 50);
/Data management/
    Data_mngt (output_ADCS.txt, simulation_time, estimated_
          w_body@orbital, estimated_quat, desired_quat, B_body_meas,
          mag_moment, duty_cycle, O2A_string, A2O_string);
}

OBC PROCESS

Process Setup
    Time_set (year);
    Set_interface (#interfaces, interface_type);
Simulation loop
    while (1) {
    Time_mngt (t_start, t_end, // year, time_step);
/Data exchange with  Control Console/
    read ("command_console", console_command_string, 10);
    string_generation (console_command_string, O2A_string, 20);
/Data exchange with  OBC/
    write (pipe_obc, O2A_string, 20);
    read (pipe_obc,A2O_string 50);
    extract_data (A2O_string, // estimated_quat, w_body@inertial_meas,
     B_body_meas duty_cycle);
/Data management/
    Data_mngt (output_OBC.txt, simulation_time, O2A_string, A2O_string);
}
```

Fig.5: SIL pseudo-code configuration

For the verification of the IMU protocol and data extraction, only two processes and two pipes are necessary. The *simulation process* generates the formatted IMU's outputs computing the angular velocity values, the EMF, the linear accelerations, and formatting these data according to the sensor protocol. The formatted string is passed for elaboration to the *ADCS process* through the *pipe_imu* pipe. In this process, the packets are validated and data are extracted (angular velocities, accelerations and EMF components). The resulting string is passed back to the *simulation process* through the *pipe_debug*.

Three processes are involved for the verification of the communication protocol between the A-ADCS and the OBC systems: the *simulation process*, the *OBC process* and the *ADCS process*. The communication between the processes occurs through pipes, specifically generated in order to share the related information between the processes. Information is shared in the form of formatted strings, with a custom header and closer.

For the verification of the execution of the operative modes and the transition between operative modes all processes and interfaces are necessary. The *simulation process* passes the simulated IMU strings through the *pipe_imu* to the *ADCS process*, which communicates the *A2O_string* to the *OBC process* thanks to *pipe_obc*. On the same software interface, the *OBC process* passes the *O2A_string* to the *ADCS process*. Finally, the *ADCS process* sends the values of the actuators commands to the *simulation process* that will use them to calculate the control torque to be applied.
Main results of the SIL simulations are:

- No errors have been detected in the communication between processes
- The A-ADCS works properly, switching from operative modes as per design and performing the satellite control
- Software bugs have been fixed

IV.II.III. Hardware In the Loop (HIL) verification

The HIL verification was developed on a step-by-step basis, i.e. including gradually several pieces of the flight equipment in the simulation loop.

First, a CIL session has been set to test:

- the set up of the ARM9 timer outputs
- the serial communication between the IMU and the A-ADCS computer
- the serial communication between the *OBC process* and the A-ADCS computer
- the synchronization among all the elements and processes and the real time capability

In this configuration, the TO is the ARM9 micro-controller of the A-ADCS board. All the other elements of the A-ADCS board are not under test, and are still simulated. With respect to the AIL configuration, main differences are that there is an *ADCS process* constituted by the flight software running on the ARM9, the interface between processes occurs through serial ports and some pieces of GSE are needed, such as power supply units, instruments and cables.

Main results of the CIL test campaign are:

- PWM timer outputs are properly setup
- No packet is lost in the IMU/A-ADCS communication but about 2% of packets are corrupted
- No packet is lost in A-ADCS/OBC communication but about 1.2% of packets are corrupted
- Real time is achieved

In the second step of the HIL testing campaign, the test object is the whole A-ADCS, with the objectives to:

- Test and calibrate the PWM circuits
- Test and calibrate the circuits for current sensing
- Test and calibrate the IMU
- Test the A-ADCS/OBC communication
- Verify the full integrated A-ADCS system

With respect to the CIL configuration, main difference is that all A-ADCS elements are included in the loop, i.e. sensors and actuators. Moreover, the *OBC process* is replaced by the flight *OBC board* to test the A-ADCS/OBC communication. The pseudo-codes of the three processes (simulation process, ADCS board and OBC board) are summarised in Fig 6.

For the PWM and sensing circuits calibration and functional test, the configuration is based on two processes. The *simulation process* generates the commands in order to emulate the attitude control values as computed by the microcontroller. The *ADCS board* receives the data through the *serial_debug* port and the duty cycle is computed from the values. The *ADCS board* sets these values on *timer_outputs* connected to the PWM logic circuits in charge of setting the corresponding currents flowing into the MTs

Another circuit senses these currents and generates output signals in the range [0-3.3] V according to the sensed current. They are acquired by the microcontroller on GPIO pins (*gpio-adc)* interfaces through an internal 10-bits Analog to Digital Converter (ADC). The result of this operation are values that are transmitted to the simulation process together with the set PWM's values and then saved for post-processing use. In the same way, the *simulation process* acquires the MT telemetry strings, displays, and saves the related values.

For testing data acquisition from the IMU, the *simulation process* receives information from the *A-ADCS board* through the *serial_debug* port. The *A-ADCS board* acquires information from the sensor mounted on the board through the *serial_imu* port. The output (expressed in terms of angular velocities, EMF and accelerometers components) passes to the *simulation process*. In this configuration, the *ADCS board* must be supplied with 3.3V, 5 V and a voltage in the range [7-8.2] V regulated, and other GSE as RS232 9pins cables and a MAX232 adapter circuit are required to complete the setup.

To test all the functions of the A-ADCS, the simulation architecture is depicted in Fig.6.

The *simulation process* passes the IMU strings through the *serial_imu* port and receives the PWM commands on the *serial_debug* port. A front-end GSE is required to adapt the signals level on both links. The data exchange between the *ADCS board* and the *OBC board* is guaranteed by a serial connection (*serial_obc*).

. The *A-ADCS* and *OBC boards* are powered with three different voltages (3.3V, 5V and [7-8.2]V) by a dedicated GSE (a three channels power supply unit).

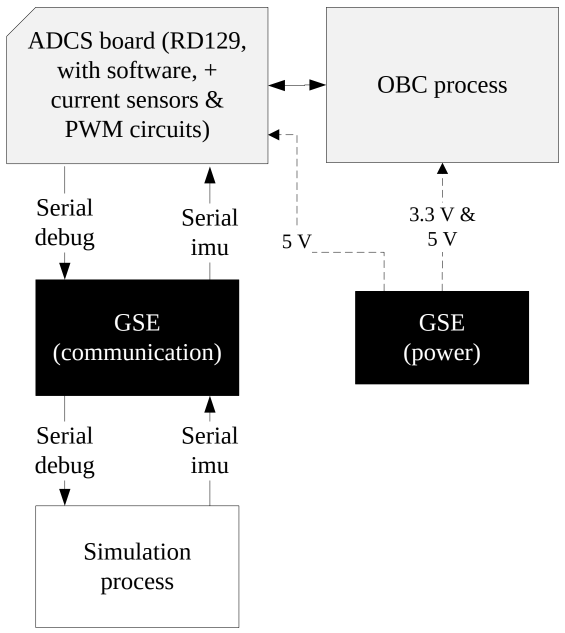

Fig.6: HIL simulation architecture

Fig. 7 highlights the model flows of the three processes.In this figure, only the OBC functions necessary to interact with the A-ADCS are reported: they refer to the communication of *O2A*- and *A2O-strings* on the *serial_obc* port.

The *A-ADCS board* sends the PWM commands to the MTs, by setting the values on the proper ARM9 configuration files that are amplified in specific electronics circuits. campaign The *ADCS board* reads the *O2A-string*s and verifies what operative mode shall be activated and extracts the telemetry of interest. Accordingly to the active operative mode, the *ADCS board* executes the specific algorithms for attitude determination and/or control.

```
SIMULATION PROCESS

Simulation Setup
    Time_set(year); // time config //
    Set_Config(#processes, processes_type, #interfaces, interface_type);
Simulation loop
    while (1){
/Orbit and environment simulation/
/Spacecraft dynamics/
/Sensors simulation/
/Data exchange with ADCS board/
    ADC (w_body@inertial_meas, B_body_meas, //
          w_body@inertial_format, B_body_format);
    IMU_packet (w_body@inertial_format, B_body_format,
                   // IMU_string);
    write (serial_imu, IMU_string, 40);
    read (serial_debug, command_string, 20);
    extract_data (command_string, // duty_cycle)
/Actuators simulation/
/Data management/
    }

ADCS BOARD

Controller Setup
    Time_set (year);
    Set_interface (#interfaces, interface_type);
Main loop
    while (1) {
    Time_mngt (t_start, t_end, // year, time_step);
/Data exchange with OBC process/
    read (serial_obc, O2A_string, 20);
    extract_data (O2A_string, op_mode);
/Data exchange with  simulation process/
    read (serial_IMU, IMU_string, 40);
    extract_data (IMU_string, w_body@inertial_meas, B_body_meas);
/Attitude determination/
    EKF (quaternion_EKF, w_body@orbit, quaternion_meas,
          w_body@orbit_meas,
         // estimated_quat, estimated_ w_body@orbit);
    Q_method (B_body, B_body_meas, // q_qmet);
/Attitude control/
    Sat_control (estimated_w_body@orbit, estimated_quat, q_qmet,
    desired_quat,B_body_meas, // mag_moment);
    PWM (mag_moment, // duty_cycle);
/Data exchange with  simulation process/
    string_generation (duty_cycle, command_string);
    write (serial_debug, command_string, 20);
/PWM setting/
    write (GPIO_port, duty_cycle, 20)
/current telemetry acqusition/
    read (GPIO_port, MT_currents, 3)
/Data exchange with OBC process/
    string_generation (simulation_time, estimated_quat,
         w_body@inertial_meas, B_body_meas duty_cycle, // A20_string);
    write (serial_obc, A2O_string, 50);
/Data management/
    Data_mngt (output_ADCS.txt, simulation_time, estimated_
         w_body@orbital, estimated_quat, desired_quat, B_body_meas,
         mag_moment, duty_cycle, O2A_string, A2O_string);
    }

OBC BOARD

Processor Setup
    Time_set (year);
    Set_interface (#interfaces, interface_type);
Main loop
    while (1) {
    Time_mngt (t_start, t_end, // year, time_step);
/Data exchange with  Control Console/
    read ("command_console", console_command_string, 10);
    string_generation (console_command_string, O2A_string, 20);
/Data exchange with  OBC/
    write (serial_obc, O2A_string, 20);
    read (serial_obc, A2O_string 50);
    extract_data (A2O_string, // estimated_quat, w_body@inertial_meas,
```

Fig. 7: HIL configuration and model flow

For the final simulation, the most accurate models have been selected for the equipment, the environment, and the spacecraft motion. For example, the IMU model is characterized with bias, misalignment, noise, and scale factor stability of the real sensor and taking into account the temperature changes. The *IMU_string* is transmitted (through the *serial_imu*) from the simulation process to the IMU-connector on the *ADCS board*. It receives also the PWM duty cycle commands calculated by the *ADCS board* and simulates the MT behavior. Finally, all the variables values and trends are saved for post processing.

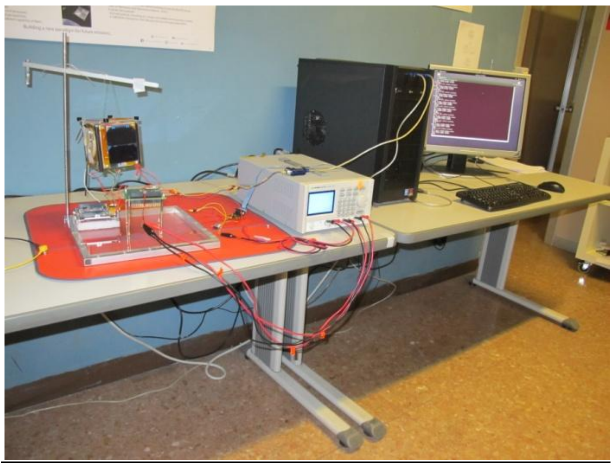
Fig. 8: test bench in HIL configuration

Fig.8 shows the HIL verification test setup arrangement: it represents an overview of the test bench with SU and CC, the Power GSE (power supplier) and the TO. In particular on the left of picture, the details of the test object are highlighted: the A-ADCS board is connected to the power supplier output to receive the regulated power on the main bus pins through the 104-pin connector. On the right of the picture, it is visible the SU and in the middle there is the GSEs (Power unit and MAX232 adapter).

Considering two days of simulation, the main results are summed up hereafter:

- About 4% of IMU packets have been lost but no error arose after the data extraction
- IMU measurements are characterized and an analog to digital linear conversion law is confirmed
- Telemetry confirms that the PWM driver circuit is able to set the desired current flowing in each torquer according to the duty cycle commands set by the controller
- Telemetry confirms that he sensing circuit the measures the right currents set according to the duty cycle commands
- Performance requirements are verified. Fig.10 shows that the angular velocities are damped and then maintained cloase to 0 rad/s. Fig. 11 highlights the

trend of the attitude (expressed in quaternion): at the end of the stabilization phase the desired pointing ($q_{desired}$ = [0 0 0 1]) is reached. The visible discontinuities on the graphs are due to wrong acquisitions of the values that, however, do not compromise reaching the final goal with the required accuracy (less than 5 degrees).

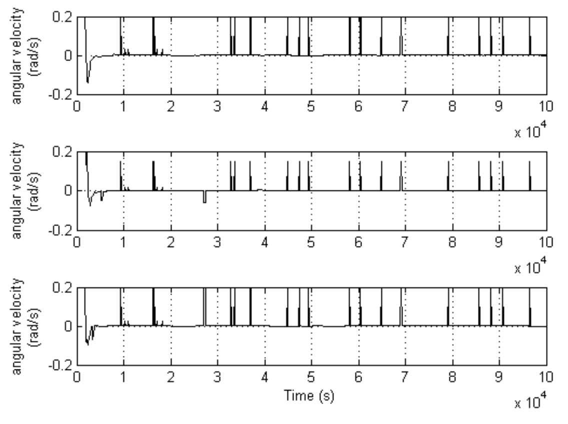

Fig 9: trend of the body angular velocity wrt the inertial frame [rad/s]

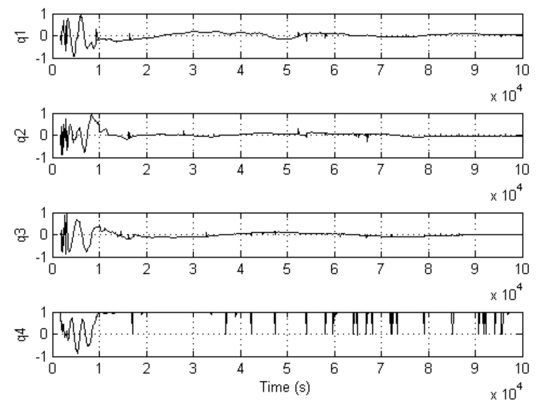

Fig. 10: trend of attitude (quaternion)

V- CONCLUSION

The paper presents a in-house simulator developed by the Cubesat Team of Politecnico di Torino. The StarSim simulator is a means to support the engineering team with a versatile tool devoted to the design and development process of a small space system. Thanks to its modular architecture, StarSim provides the desired flexibility with respect to different project stages and types of missions/systems, that can be chosen by setting the simulator in terms of specific mission parameters and systems involved in the simulation session. In particular, StarSim is able to perform “in the loop” simulation. The capability of the simulator to be used in different phases of the life-cycle and for several kinds of missions and systems is one of the most important characteristics of StarSim and its main strength. Small projects might benefit from this feature to a great extent, while for bigger programs this might not be fully applicable. It is instead true that it could be used for AIV of other-than-CubeSat small systems, such as micro-rovers and/or –landers, by adding the adequate mission and equipment models to the database.

StarSim supported the verification of the A-ADCS functional and operational requirements of the e-st@r-II CubeSat. Several simulation sessions have been performed in different “in-the-loop” configurations and the CubeSat’s A-ADCS has shown to be compliant with the required functions and to operate as expected. It should be noticed that a large number of requirements have been verified within few test sessions, thus saving resources in terms of time for test set-up and running, man-hours and costs, if compared to other testing techniques. Moreover, the tests allowed following out the behavior of the equipment included in the simulation loop. AIL, SIL, and HIL simulation have concurred to design and verify the determination and control algorithms, the software and, the hardware (microcontroller, sensors, actuators and associate items).

In the near future, the simulator will be upgraded in the areas of: 1) autonomy, in order to facilitate the user operation during the setup and the post-processing activities; 2) models database, to include other equipment e mission models, and 3) addition of the software support for other GSE.

VI REFERENCE

[1] ECSS-E-TM-10-21A – Space Engineering – Space modelling and simulation – (16/04/2010)

[2] Eickhoff, J. (2009). Simulating Spacecraft Systems. Springer Aerospace Technology, Springer Berlin Heidelberg, Berlin, Heidelberg.

[3] Bandecchi, M. (ESA), Gardini, B. (ESA), Melton, B. (ESA), and Ongaro, F. (ESA). (2000). “The ESA/ESTEC Concurrent Design Facility.” 2nd European Systems Engineering Conference (EuSEC2000), 329–336.

[4] Judnick, D. C. (2010). “Concept Design Center Pre-work Overview.” AIAA Space Conference & Exposition, The American Institute of Aeronautics and Astronautics, ed., Reston, VA.

[5] Eickhoff, J., Falke, A., and Röser, H.-P. (2007). “Model-based design and verification—State of the art from Galileo constellation down to small university satellites.” Acta Astronautica, 61(1-6), 383–390.

[6] Agnew, D., Claesen, L., and Camposano, R. (Eds.). (1993). Computer Hardware Description Languages and their Applications. Computer Hardware Description Languages and their Applications, Elsevier.

[7] Object Management Group, O. (2014). "Introduction To OMG's Unified Modeling Language^{TM} (UML®)." (Jan. 1, 2015).

[8] Weilkiens, T. (2007). Systems Engineering with SysML/UML. Systems Engineering with SysML/UML, Morgan Kaufmann Publishers Inc., 1–22.

[9] Hoffmann, H.-P. (2005). "UML 2.0-Based Systems Engineering Using a Model-Driven Development Approach." CrossTalk The Journal of Defense Software Engineering, 1–18.

[10] M. Pollina, Y. Leclerc , E. Conquet, M. Perrotin, G. Bois, L. Moss, The ASSERT Set of Tools for Engineering (TASTE): Demonstrator, HW/SW codesign, and future.

[11] Corpino, S., and Stesina, F. (2014). "Verification of a CubeSat via hardware-in-the-loop simulation." IEEE Transactions on Aerospace and Electronic Systems, 50(4), 2807–2818.

[12] Toorian, A., Diaz, K., and Lee, S. (2008). "The CubeSat approach to space access." IEEE Aerospace Conference Proceedings, 1(1), 1–14.

[13] "CubeSat Design Specification R13." (2014). .

[14] Bouwmeester, J., and Guo, J. (2010). "Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology." Acta Astronautica, Elsevier, 67(7-8), 854–862.

[15] Viscio, M. A., Viola, N., Corpino, S., Stesina, F., Fineschi, S., Fumenti, F., and Circi, C. (2014). "Interplanetary CubeSats system for space weather evaluations and technology demonstration." Acta Astronautica, Elsevier Ltd, 104(2), 516–525.

[16] Dubos, G. F., Castet, J. F., and Saleh, J. H. (2010). "Statistical reliability analysis of satellites by mass category: Does spacecraft size matter?" Acta Astronautica, 67, 584–595.

[17] Stesina, F., Corpino, S., Mozzillo, R., and Rabasa, G. O. (2012). "Design of the active attitude determination and control system for the E-ST@R CUBESAT." Proceedings of the International Astronautical Congress, IAC, International Astronautical Federation, Naples, 4585–4594.

[18] ECSS-E-HB-60A - Space Engineering Control engineering handbook – (14/12/2010)