THE EXTREMA SIMULATION HUB: ADVANCEMENTS IN THE DEVELOPMENT OF AN INTEGRATED HARDWARE-IN-THE-LOOP FACILITY FOR AUTONOMOUS CUBESATS GNC TECHNOLOGIES

Gianfranco Di Domenico[1], Davide Perico[2], Fabio Ornati[3], Carmine Giordano[4], Alessandro Morselli[5], Paolo Panicucci[6], Francesco Topputo[7]

Dept. of Aerospace Science and Technology, Politecnico di Milano
Via Giuseppe La Masa 34, 20156, Milano, Italy

The EXTREMA (Engineering Extremely Rare Events in Astrodynamics for Deep- Space Missions in Autonomy) is an ERC-funded project with the goal of enabling CubeSats with autonomous GNC capabilities. The project has received a Consolidator Grant from the European Research Council, a prestigious acknowledgment that funds cutting-edge and disruptive innovation research in Europe. EXTREMA aims to reach its goal by building on three pillars: Autonomous Navigation, Autonomous Guidance, and Autonomous Ballistic capture. Each pillar is associated with a hardware-supported experiment, and the research outcomes are channeled into the EXTREMA Simulation Hub (ESH), a facility in which autonomous algorithms and techniques are to be tested within an integrated hardware-in-the-loop simulation involving both on-board systems and testing support facilities. This work aims to present the advancements in the development of the ESH. In the first part, the progress in the development of the hardware facilities RETINA, an optical facility; ETHILE, a thrust test bench; and STASIS, a spacecraft attitude simulator are illustrated, focusing on the challenges and the adopted solutions. Then, the EXTREMA orbital propagator, SPESI, is presented. SPESI is a high-fidelity, stochastic numerical integrator whose task is to collect all the quantities of interest from the surrounding simulation environment and propagate the state of the simulated spacecraft, along with the associated uncertainties. To do so, SPESI is equipped with a real-time stochastic orbital propagator that guarantees the computation of the spacecraft state in a hard-real-time fashion, keeping the hardware-in-the-loop experiment synchronized. One of the major advancements in realizing the ESH environment is the development of the EXTREMA FlatSat. It consists of a hardware-software stack emulating a typical deep-space probe on-board computer. The software architecture of the EXTREMA FlatSat is described, focusing on the implementation of the flight software, developed with a modular approach in mind that enables quick integration of goal-specific applications and algorithms (such as the guidance optimization and the optical navigation ones) and fast deployment on multiple hardware architectures under a CI/CD framework. Furthermore, the workflow of the EXTREMA Dry Runs is presented. These represent periodic integration tests in which all the advancements carried on in parallel by different teams are integrated and tested on randomly generated mission scenarios. In this context, the EXTREMA Monitoring and Control Application (MoniCA) is also presented, together with the adopted inter-device communication framework. This unified and lightweight solution enables telemetry and status monitoring of all the devices involved in the simulation, enabling at the same time the streamlined development of a user-friendly web-based frontend for the comprehensive monitoring and control of the entire simulation. Eventually, the outcomes of the dry runs are analyzed for a set of significant interplanetary transfers, concluding with a discussion on the potential impacts of the proposed approach and with an outline of the future improvements and roadmap.

[1]PhD Student, gianfranco.didomenico@polimi.it
[2]PhD Student, davide.perico@polimi.it
[3]PhD Student, fabio.ornati@polimi.it
[4]PostDoc Researcher, carmine.giordano@polimi.it
[5]Assistant Professor, alessandro.morselli@polimi.it
[6]Assistant Professor, paolo.panicucci@polimi.it
[7]Full Professor, francesco.topputo@polimi.it

INTRODUCTION

Over the past decade, the space industry has experienced significant growth, largely propelled by CubeSats. Thanks to their reduced manufacturing costs and reduced launch mass, they allowed smaller players to enter the space segment, succesfully contributing to the democratization of space. However, the employment of CubeSats has been mostly focused on the part of space closer to the Earth. Only a few missions employed CubeSats to reach and explore the furthest locations in the Solar System, namely the MarCO (Mars Cube One) mission [1], while others are still in the preliminary design phases (e.g., M-ARGO [2] [3]).
Despite CubeSats offering cost savings in manufacturing and launch, operating deep-space CubeSats incurs expenses comparable to those of larger probes. Deep-space missions often span months or even years, necessitating resource-intensive human-in-the-loop operations that strain mission budgets. Furthermore, despite increased funding, the limited availability of ground communication slots for interplanetary probes is reaching saturation, hindering the proliferation of CubeSats beyond Earth's immediate vicinity.
In thid context, achieving higher levels of autonomy for interplanetary CubeSats represents a promising solution to reduce mission costs and increase the number of missions that can be conducted. This is the goal of the ERC-funded EXTREMA project, led by Prof. Francesco Topputo from Politecnico di Milano. The project aims to catalyze research in this area, with the ultimate objective of revolutionizing how deep-space missions are conducted. This five-year project aims to address a fundamental research question:

To what extent can we navigate the Solar System free of human supervision?

To answer that, EXTREMA project builds on three main Pillars:

- **Pillar I: Autonomous Navigation.** The research activities falling within this pillar focus on the development of navigation algorithm suitable for CubeSats to enable deep space probes to locate themselves in deep space in complete autonomy.
- **Pillar II: Autonomous Guidance and Control.** Current trajectory planning is performed on ground due to limited on-board computational resources. The same applies to correction maneouvers, resulting in huge costs in terms of time and human personnel. EXTREMA aims to develop lightweight guidance algorithms exploiting the knowledge of the spacecraft state obtained through the autonomous navigation to compute a time-definite thrust profile and reach the final target autonomously.
- **Pillar III: Ballistic Capture.** In contrast to the relatively slow-evolving dynamics of deep-space travel, orbital capture around celestial bodies presents significant challenges due to faster orbital periods and associated accelerations. Current electric propulsion technologies often lack the thrust necessary for orbit insertion maneuvers, calling for alternative insertion techniques. EXTREMA aims to exploit the phenomenon of ballistic capture, where specific state-space conditions define a capture set. This allows for the identification of ballistic capture corridors. By strategically maneuvering a spacecraft within these corridors, the multi-body dynamics of the Solar System can be leveraged to achieve a prolonged near-circular orbit around the target body with minimal propulsive expenditure.

The outcome from each Pillar is meant to be integrated in a series of experiments and, ultimately, brought together in the EXTREMA Simulation Hub: a hardware-in-the-loop testing facility that would allow to test integrated guidance, navigation and control (GNC) systems and algorithms.
This paper aims to describe the infrastructure of the ESH, including the simulation facilities that are being developed to support the research activities of the EXTREMA project. Moreover, a detailed description of the on-board computer software architecture is provided, along with the description of MoniCA, the Monitoring and Control Application that gathers data from the involved devices and organizes them in a user-friendly interface. Finally, the results of the dry runs are presented, showing the potential of the developed algorithms and the improvements that can be made in the future.

1 THE EXTREMA SIMULATION HUB

The EXTREMA Simulation Hub is a facility in which the outcomes of the three Pillars at the foundation of EXTREMA will be integrated and tested. It is a hardware-in-the-loop distributed simulation environment in which the autonomous GNC systems interact as they would during an interplanetary transfer. Fig. 1 shows the current status of the ESH within the Politecnico di Milano laboratories. In particular, three facilities have been developed, mimicking the systems of the interplanetary spacecraft: the RETINA (Realistic Experimental faciliTy for vIsion-based NAvigation) optical facility mirrors the onboard optical instrumentation; the ETHILE (EXTREMA Thruster in The Loop Experiment) thrust test bench acts as main propulsion system; the STASIS (SpacecrafT Attitude SImulation System) attitude simulation platform reproduces the attitude evolution of the probe. These devices are commanded by a central computing unit, in the form of a FlatSat, hosted on STASIS in order to be interfaced with inertial sensing units. The FlatSat hosts EXTREMA fLAtsat Processing SystEm (ELAPSE), an on-board computer emulator that runs the navigation, guidance and control algorithms developed in the three Pillars [4]. Additionally, a set of infrastructural systems has been developed in order to carry on the simulation. Among these, the SPESI orbital propagator collects real-time signals from the involved facilities and feeds them to the numerical integrator; this allows to propagate the orbital states of the spacecraft in real-time. All the interactions between the spacecraft and the environment are proxied by SPESI; as an instance, SPESI will be responsible for the generation of the deep-space scene that will be cast on the RETINA screen, or for the variation in maximum thrust magnitude due to varying distance from the Sun characterizing electrical engines.
Fig. 2 shows the simulation loop of the ESH. At each timestep, SPESI interfaces with the sensing infrastructure – recovering the thrust magnitude and direction – and calculates the next spacecraft state. Together with the orientation information, it renders a deep-space scene, that is cast on the microdisplay of the optical facility; this is sensed by the navigation camera. The OBC processes the optical images to run the navigation algorithms, estimating its own 6-DoF state. If necessary, the spacecraft autonomously computes a new trajectory to be followed, relying on the control history of the ETHILE controller, in charge of actuating the latter according to the spacecraft's internal clock. The sensing of the actuated thrust closes the loop, as it is sensed again by SPESI which continues to propagate the translational degrees of freedom.

1.1 The accelerating framework

To achieve maximum simulation fidelity and ensure synchronization between the hardware components and the virtualized ones, the simulations are run in real-time. In order to guarantee the synchronization between the platform state and the spacecraft state in deep space, the set of operations required in a single step cycle must be performed with a hard limit on the computational time. This means that the filtering of the sensor suite, state propagation, and image rendering and casting procedures must be executed with a hard time constraint.
Interplanetary transfers, especially the ones involving low-thrust and ballistic capture, are usually characterized by prolonged transfer times [5, 6, 7], usually in the order of months or years. EXTREMA aims to validate the developed algorithms and systems with integrated simulations covering all the phases of a typical interplanetary transfer; this is unfeasible with traditional hardware-in-the-loop simulation techniques. In order to attain multiple and repeated testing campaigns, a framework to execute the experiments in reduced time frames is employed [8]. The framework – whose theoretical foundations are currently under development – ensures mathematical equivalence between an original (slow) system and a set of accelerated systems; by tailoring the simulation to reproduce the dynamics of the accelerated system, it is possible to obtain the quantities of interest in a reduced time frame with no approximations.

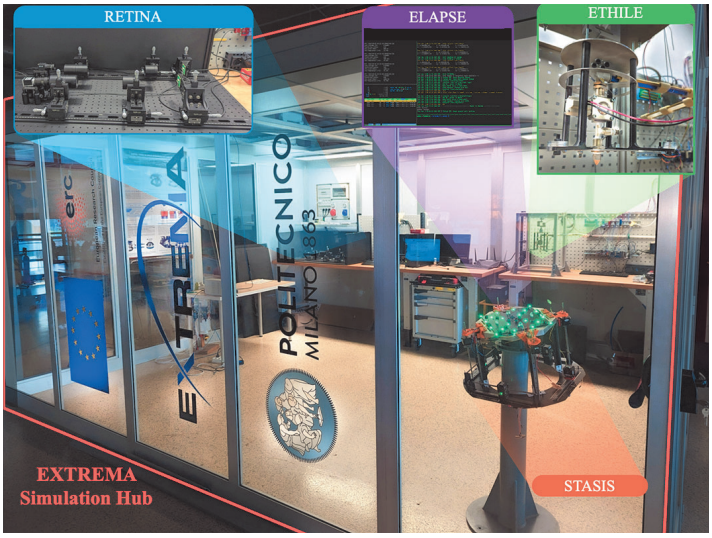


Fig. 1: ESH with highlighted facilities.

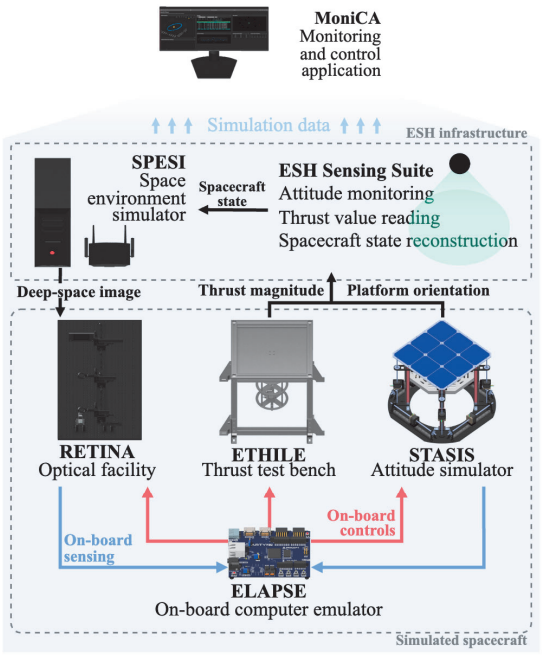


Fig. 2: Full logic scheme of the EXTREMA Simulation Hub.

1.2 RETINA

RETINA is an optical facility designed with the idea of providing a versatile test platform for stimulating CubeSats optical-based navigation systems and cameras [9]. The facility is characterized by a variable-magnification optical design that allows the deployment of cameras with different Field-of-View (FoV) characteristics (between 4 and 22 degrees) without the need to modify any of its hardware components. This range is compatible with the simulation of both the narrow-FoV navigation equipment such as the LUMIO Cam [10] and the lower-end portion of the star-tracker camera FoVs.
RETINA is composed of several opto-mechanical components enclosed in a dark room to prevent disturbances and reflections from external light sources. The architecture has been derived from the one developed by Beierle et al [11] and includes:

1. a WUXGA R5[8] OLED display;
2. a collimator lens system to project the observed scenes at infinity;
3. a relay lens assembly to achieve variable magnification capabilities;
4. a camera and objective to mimic the characteristics of real VBSs.

The two lens groups, shown in Fig. 3, are mounted onto movable optical stages that can be manually adjusted in three axes with micrometric precision. The presence of two different optical groups allows the modification of the magnification and thus the angular size of the image shown to the camera. In this setup, the collimator is placed exactly one focal length away from the location of the virtual image of the screen generated by the relay lens. In this way, the scenes observed by the VBS are collimated at infinity, as in the case of distant objects in space [12]. The different magnification settings are achieved by adjusting the relative position of the optical components. [11, 9]
The screen has been specifically selected for its small size (18.7 x 11.75 mm) and high resolution (1920 x 1200 pixels). Since it relies upon OLED (Organic Light Emitting Diode) technology, it can achieve an extremely high

[8]WUXGA-R5 spec-sheet: `www.emagin.com/products/WUXGA`, last visited April 15, 2024.

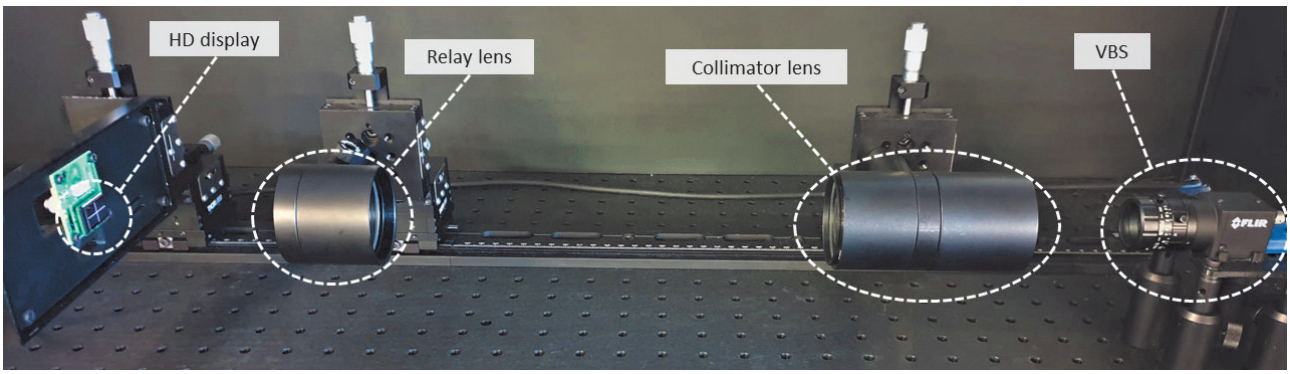


Fig. 3: Main components of RETINA.

dynamic range with virtually inexistent backlight leakage. These features are especially beneficial considering that the purpose of the facility is to simulate optical navigation scenarios that involve celestial objects with large differences in magnitude levels.
The camera currently deployed in the facility is a FLIR BFS-U3-31S4[9] monochrome camera with a 1536 x 2048 pixel 1/1.8" sensor. During the ESH simulations, the camera can be equipped with two objective configurations that can be used to emulate the characteristics of both wide-FoV and narrow-FoV Vision-Based Sensors (VBS). Before simulations, the facility is calibrated both geometrically and radiometrically. The geometric calibration procedure performs the characterization of the optical distortions induced by the lens system and by the misalignment between the components. [13] Thanks to the calibration the distortions can be compensated to achieve images that are geometrically equivalent to the real represented scenarios. The radiometric calibration, instead ensures that the reproduced scenes have realistic levels of luminous intensity. [9, 11]
The facility is coupled with a realistic rendering engine able to generate deep-sky scenes given the state of the spacecraft and the positions of the planets. In the ESH cycle, this information is provided by the SPESI propagator. At each time step a new image is rendered, fed to the screen stimulator, and imaged by the camera. The images are sent to the onboard computer, where they are processed by state-of-the-art far-range optical navigation algorithms to correct the estimate of the current spacecraft state. [14]

1.3 ETHILE

Historically, the modeling of interplanetary transfers has heavily relied on mathematical approximations, system decoupling, and numerical and computational tools to build models for predicting and analyzing the evolution of the quantities of interest. Statistical approaches have been employed to guarantee the robustness of the outcomes against effects not included in the original models. Given the advances in the capabilities of microcomputing and technologies for embedded systems, a different kind of models, mixing numerical tools with physical implementations of the systems of interest, emerged. Depending on which parts of the systems were physically represented, these have been named hardware-in-the-loop (HIL), processor-in-the-loop (PIL), or even human-in-the-loop simulations. The advantages of such approaches can be seen in terms of:

(a) simulation fidelity, since the approximations brought by numerical models of physical elements do not introduce errors in the simulations;

(b) resources required, as it is possible to relieve the numerical integrators from the need to propagate the evolution of physical systems; this benefits, in particular, those simulations involving phenomena happening at very different time scales;

[9]FLIR BFS-U3-31S4 spec-sheet: `http://softwareservices.flir.com/BFS-U3-31S4/latest/Model/spec.html`, last visited April 15, 2024.

(c) simulation flexibility, as it is possible to obtain a faithful simulation of harsh environment and phenomena employing tailored sets of sensors and actuators, without leaving the safety of the lab environment.

In particular, ETHILE represents a HIL facility as it integrates hardware in the orbital simulation, namely the thruster. The thrust test bench is composed of three parts:

(a) the thrust balance, where the thruster and the load cell are positioned;

(b) the pneumatics feeding system, which provides compressed air at the required pressure level to the thruster;

(c) the Single-Board Computer, which computes the optimized trajectory, applies the scaling, and actuates the thruster by acting on the solenoid valve and pressure regulators.

The experimental setup is portrayed in Fig. 4 and described more in details in the following paragraphs.

Thrust balance The thrust balance measures the force generated by the thruster. As can be seen in Fig. 5, it is installed on a laboratory workbench and has a vertical layout, with the thrust vector that acts vertically from bottom to top, opposite to the weight of the hosting infrastructure and thruster. The two main drivers of this design choice are:

(a) the magnitude of the scaled thrust forces, which is of the order of a few Newtons instead of the mN or μN of a low-thrust engine;

(b) the simpler calibration process with respect to the horizontal or torsional thrust balances.

More accurate and precise models of thrust balance, like the torsional and horizontal thrust balance, exists and are usually employed for accurate measurement of the performances of low thrust engines. Anyway, their calibration and setup are much more complex and additional care must be taken in the preparation of the experiment. Additionally, they often require amplification mechanisms which might also introduce non-linear effects if the force variation throughout the execution of the experiment varies and gets close to the full scale of the force sensor. The thrust balance hangs from a supporting structure, a $600 \times 400 \times 400$ mm cuboid assembled with aluminum strut profiles which hosts on top a square aluminum. The supporting structure is raised 25 mm from the workbench top to let compressed air flow outside of the supporting structure from below, reducing the air interaction with the profiles at the bottom and thus limiting the onset of vortexes which might disturb the measurements of the thrust balance. The load cell is installed with two knuckle eyes, with the two connection pins inserted on the two adapters aligned with the X and Y directions. This guarantees that momenta acting around these directions, which might result from thruster misalignments errors, are not transferred to the load cell. The thrust balance consists of three aluminum disks: the top one is thicker and is connected to the load cell interface, the other two support instead the pneumatics feeding system, the electrovalve and the thruster. Lightening holes are present on the two lower disks to compensate the non-symmetrical weight distribution of the electrovalve and pressure sensor. The pneumatics feeding line is instead installed at the same level of the elbow connector present on the middle disk. In this way, the load cell is measuring only the weight of electrovalve, thrust balance, fittings, and pressure sensors when the thruster is off. Note that a constraint on the maximum thrust level is imposed by the facility design: the maximum thrust must be smaller than the weight of the thrust balance to keep the load cell working in tensile loading.
Note that the thrust balance can only measure the thrust along one direction. As a consequence, ETHILE could only introduce errors in the magnitude of the thrust. The errors in the directions of thrust are instead introduced by the attitude simulator STASIS.

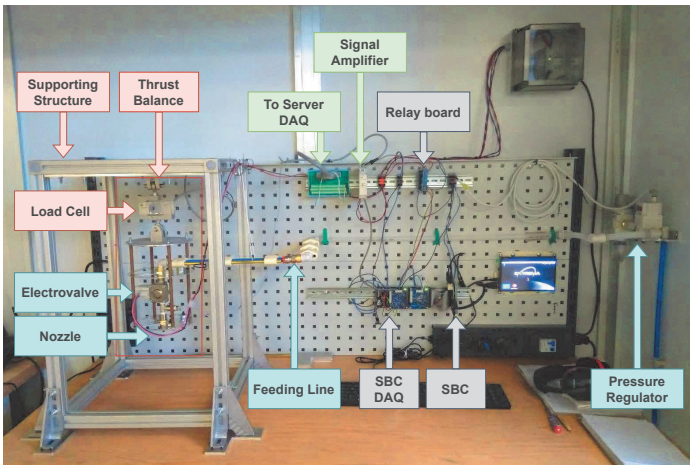


Fig. 4: Layout of the ETHILE facility.

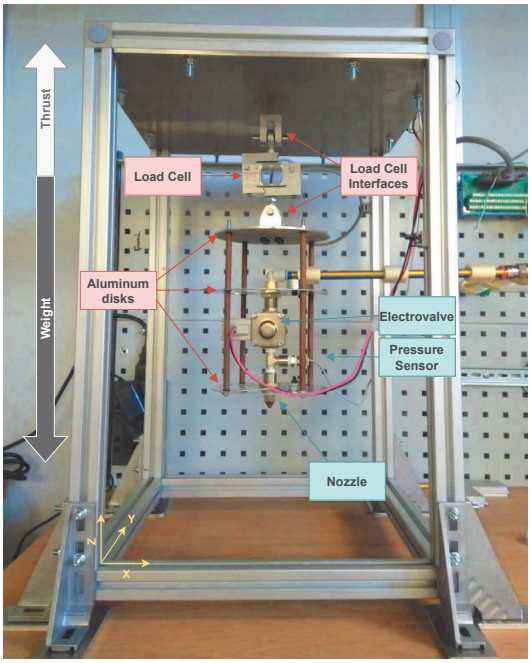


Fig. 5: Thrust balance detail.

Pneumatics feeding system The pneumatics feeding system provides compressed air to the cold gas thruster. The feeding line is connected to the compressed air pipes of the laboratory, which provides a pressure up to 8.0 bars. A manual filter regulator is installed near the pressure regulator on the supply side. The filter was selected in accordance with the requirements on the filtration degree by the pressure regulator. The electro-pneumatic pressure regulator is voltagecontrolled by an analog signal in the range 0-5 VDC, compatible with a SBC voltage output. The feeding line segments are nylon tubes with an internal diameter of 10 mm which are connected together by elbow tube-to-tube adaptors. This solution increases the mechanical degrees of freedom of the feeding line, since the adapters allow each tube to rotate along its axis. In this way, the feeding line can move and accommodate the small vertical displacement generated by the thruster actuation. In addition, this setup eases the levelling and height regulations of the feeding line by simple rotation along the axis of the tubes. The last two segments of the feeding line are kept straight by two brass tubes and are suspended to guarantee the alignment with the elbow connector on the thrust balance. The solenoid valve actuates the thruster and is controlled by the SBC. The selected electrovalve is lightweight and has quick response time (below 20 ms). The analog pressure sensor is placed on a T-joint after the solenoid valve. Its output signal is acquired by the SBC to perform closed-loop control of the pressure, ensuring that the required thrust is achieved. Finally, a convergent nozzle is placed at the end of the circuit. The nozzle diameter shall be such that the flow is chocked for the expected thrust range. Convergent-divergent nozzles were not considered as they would operate in an adapted regime only for one single value of control pressure, which is non-ideal as we will modulate the control pressure. Indeed, possible issues might arise due to the presence of shock waves in the divergent section or expansion waves at the exit of the nozzle. This would likely result in non-symmetrical flow as it is extremely hard to achieve sufficient precision in manufacturing components with inner diameters of a few millimeters.

Single-board computer The Single-Board Computer has a dual function: it runs the guidance algorithm and controls the actuation of ETHILE. Currently, the SBC is a Raspberry Pi 4 Model B with 8 Gb RAM. The SBC functionality could be expanded by connecting physical hardware to the 40-pin GPIO (General Purpose Input Output) connector. Various Hardware Attached on Top (HAT) boards are also available for Raspberry Pi, with dedicated software libraries that ease up software development. The following boards are connected to the Raspberry:

- an environmental sensor to measure pressure and temperature inside the ESH;
- a Data Acquisition (DAQ) HAT to acquire the measurements of the pressure sensor and (optionally) the load cell. By means of these measurement, it is possible to perform a close-loop control to regulate to the desired level of thrust (e.g., counteract pressure deviations, introduce disturbances and simulate misperformance of the thruster).
- a HAT to provide analog voltage output in the 0-5 VDC range, used to control the pressure regulator.
- a 4-channel relay board, to switch on/off the pressure regulator and the electrovalve. These are operated with 24 VDC provided by a power supply and hence cannot be powered directly by the SBC.

This configuration has the advantage of simplifying the interaction between the guidance algorithm and the facility but has the drawback of requiring the execution of the guidance software on the Raspberry Pi. Therefore, it will be the baseline during the testing and validation phase of the ESH but it is foreseen to run the guidance algorithm on a dedicated SBC in the future.
The described setup shall guarantee the repeatability and high fidelity of measurements. This is a key aspect in the EXTREMA simulations: an interplanetary or deep-space transfer has multiple thrust arcs, which are in turn interrupted by multiple navigation sessions. As a consequence, an activation of the cold gas thruster is needed per each thrust arc and it will not be advisable to perform intermediate calibrations during the execution of a simulation. The simpler design shall therefore guarantee an easier setup and smoother execution of the experiment.

1.4 STASIS

STASIS (SpacecrafT Attitude SImulation System) is an attitude simulation platform based on a 3-DoF air-bearing spherical joint. The role of STASIS in the EXTREMA simulations is to mimic the attitude evollution of the spacecraft in deep space, while also hosting the OBC [15] (Fig. 6). Air-bearing platforms have an extended heritage in testing, verification, and validation activities for attitude-related spacecraft systems and algorithms. By creating a thin film of air between a socket and a spherical interface, the effect of friction is minimized; if the center of mass (CoM) of the platform is also coincident with the center of rotation (CoR) imposed by the geometrical and aerodynamic configuration of the join, first-order gravitational torques acting on the platform are also nullified, effectively simulating the attitude evolution as it would happen in space. While the concept is promising, such ground-borne platforms are subject to a set of environmental disturbances that threaten their fidelity; among the most prominent ones, residual CoM-to-CoR offsets, aerodynamic perturbances arising from the joint and the surrounding air, elastic deformations, magnetic disturbances, thermal gradients, and effect of battery discharge [16, 17]. Moreover, system identification techniques to retrieve the real platform intertia tensor are subjected to errors – in part due to the effect of the aforementioned disturbances. In order to minimize such effect, STASIS has been developed to be as stiff as possible, employing only rigid connections for the electrical functionality. A set of 8 movable masses – represented by stepper motors – is employed to balance the platform. These are controlled by dedicated controllers, directly interfaced to the stepper motor drivers on a single PCB unit directly attached to the moving mass. The controllers can be wirelessly controlled through simple UDP messages. The entire system is directly powered by the underlying custom-designed board, which acts both as a supportive element for the horizontal moving masses and as power distribution system. The board is interfaced with an overlying solar array, providing maximum power point tracking (MPPT) capabilities; the solar panel itself is fed by a custom-designed power beaming system, represented by a high-power lamp whose spectral and radiative characteristics and pattern have been designed to maximize the collection and conversion efficiency. The efficiency of the system is around 38% , allowing an expected continuous power of about 23 W to power the systems on the board for extended time periods [17].

As the attitude evolution is not integrated, but instead replicated by STASIS, a high-accuracy sensing system has been implemented to track the spacecraft attitude. The EXTREMA Attitude Groun Truth system [18] is made up of a monocular camera placed directly on top of the platform and an array of LED placed directly on the solar array (Fig. 7); the LED tracking system is able to estimate the platform attitude with an accuracy below 50 arcsec at 60Hz[10], following an initial calibration. Such levels of accuracy were attained by using a high-resolution camera and a set of ultra-small LEDs (in a 0201 package) placed directly on an aluminum PCB with automated assembly, in order to ensure the highest possible alignment. The camera is directly interfaced with a computational unit that runs a parameter optimization algorithm to reconstruct the spacecraft orientation; such information is relayed to the orbital propagator in order to align the thrust direction with the one corresponding to the platform attitude and properly propagate the spacecraft translational DoF.

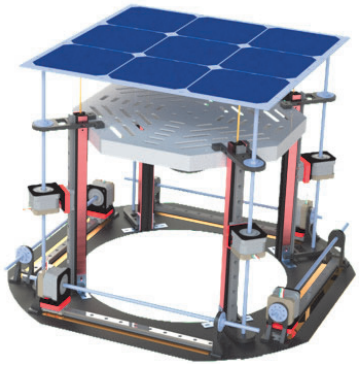


Fig. 6: 3D model of STASIS

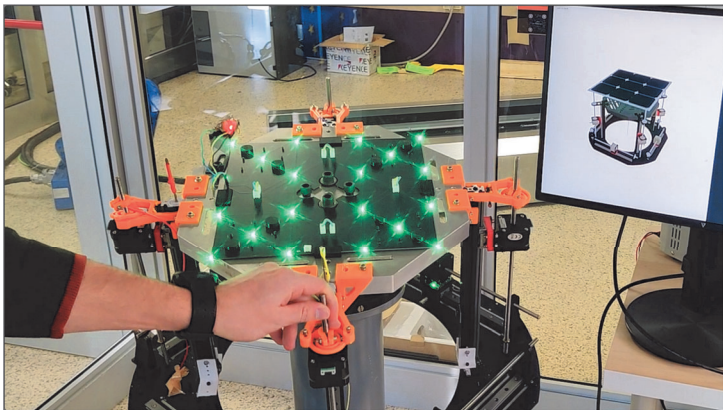


Fig. 7: STASIS Attitude Ground Truth system

1.5 SPESI

SPESI (SPacecraft Environment SImulator) is a hardware-software infrastructure with the task of collecting real-time signals from the involved facilities and integrating them in a numerical integrator to virtually propagate the spacecraft non-reproduced states [19]. In particular, the SPESI software runs the following loop:

1. collect the thrust magnitude and direction from ETHILE;
2. collect the attitude orientation of STASIS;
3. propagate the translational degrees of freedom of the spacecraft;
4. render the updated deep-space scene on the optical facility microdisplay;

As the simulation is performed in real (accelerated) time, all of these must be executed in a hard-real-time fashion (Fig. 8). For this reason, a dedicated SPESI is executed on a dedicated hardware infrastructure featuring an Intel i9 10900X CPU working at 3.7GHz, 64GB DDR4 RAM, and two nVidia Quadro RTX4000, exploited for their fast rendering capabilities, running a Linux kernel version 5.10 customized with the PREEMPT RT patch [20]. Moreover, SPESI employes specific stochastic integration schemes to maximize the numerical stability while also taking into account the constraint that each real-world input must be recovered within the simulation step; this limits the usage of traditional implicit schemes. Among these, a stochastic Runge-Kutta of order (3.0, 1.5) is employed [21].
The implemented dyamical equations use an extended parametrized post-Newtonian formulation (PPN), considering gravitational accelerations of the Solar System bodies, the general relativity correction, Lense-Thirring

[10]Limited by the camera maximum acquisition frequency

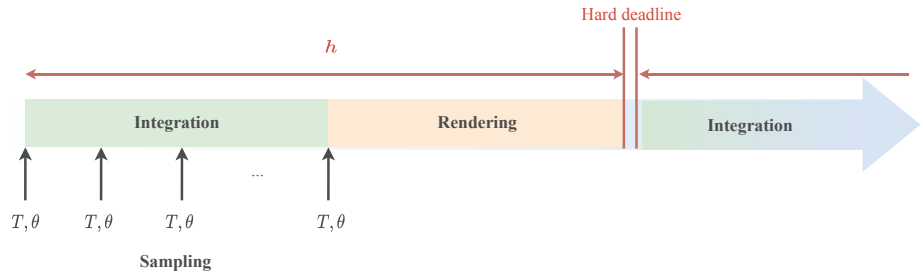


Fig. 8: Visualization of the hard real-time deadlines of SPESI.

precession, solar radiation pressure, and spacecraft thrust [19]. In order to ensure fast ephemeris retrieval, SPESI implements the Fast Ephemerides Retrieval Tool (FERT), also developed internally; it also employes platform-specific single-instruction, multiple-data (SIMD) CPU instruction sets for fast vectorized operations [19].

2 THE FLATSAT SOFTWARE ARCHITECTURE: ELAPSE

The FlatSat hardware-software system ELAPSE, comprises two actors, the On-Board Computer Emulator (OBCE) and the Attitude Determination and Control Unit (ADCU). They contribute to supervising and commanding the autonomous GNC algorithms and interfacing with the different ESH hardware units, enabling Hardware-In-the-Loop testing.

2.1 The On-Board Computer Emulator

The first element of ELAPSE hosts the main software and integrates the algorithmic products of the EXTREMA Pillars. As agile development and testing of autonomous GNC technologies is intrinsic in ESH exploitation, the OBCE software has been designed accordingly. A highly modular architecture is implemented following the Object-Oriented Programming (OOP) paradigm and an embedded programming methodology, towards on-board implementation representativeness. The main structure, which retraces other flight software framework characteristics, is composed of different levels of abstraction described in the following paragraphs. This structure, together with a modular-based philosophy, can be found in different state-of-the-art flight-proven frameworks such as the NASA core Flight System (cFS)[11], the On-board Software Reference Architecture (OSRA) by ESA [22], the GEneRIC Onboard Software (GERICOS) framework [23], the small satellites-optimized NanoSat MO Framework (NMF) [24], and Jet Propulsion Laboratory-factored F Prime [25]. Being flexible, generic, and enforcing the reusability principle, those examples paved the foundations for the design of the OBCE software, sharing their philosophies but specifically tailored for the EXTREMA ESH environment and agile GNC algorithms integration. Fig. 9 unveils the OBCE softwares three-layer architecture.
Each layer serves as a container for Modules, classes that interface with each other by following the OOP encapsulation, data abstraction, polymorphism, and inheritance principles. Two types of Modules can be identified, Task Modules, which contain a running task, and Core Modules, which instead implement atomic functionalities. The input/output interfaces between elements are implemented via end-to-end data transmission or through Data Interface classes, that provide more complex structures specific to GNC states. The top layer, called Middleware, governs the execution of the Task Modules and provides high-level operational functionalities and interfaces with the application-specific modules. Its architecture eases the adaptation of the core software to different purposes and Operative Systems (OS), with only localized modifications to be implemented. At the current development stage, an event-based execution policy governs the tasks transitions through the red units in the aforementioned scheme. The logic relies on the generation and processing of events. Specifically, the

[11] https://cfs.gsfc.nasa.gov/ [last visited April 15, 2024]

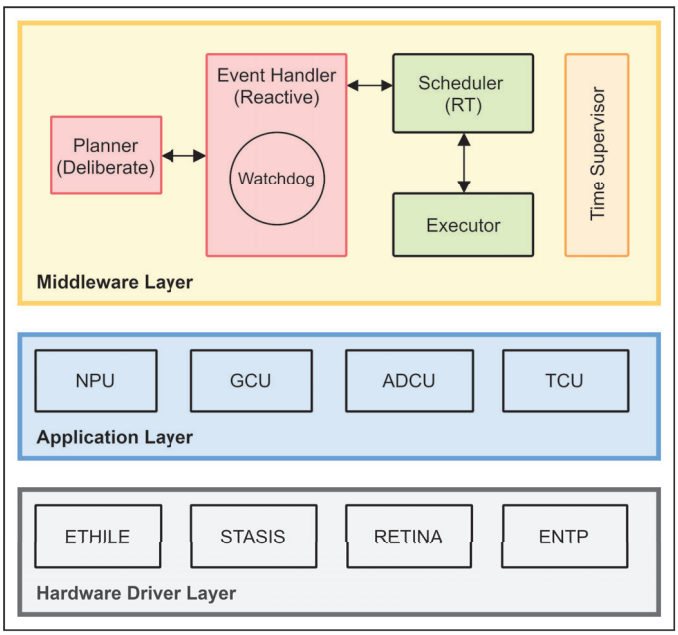


Fig. 9: FlatSat OBC software architecture.

latters are built on a couple of an execution status and an identifier of the Task Module returning the status. Subsequently, this is watched by the Event Handler via a watchdog mechanism and dispatched into one or a sequence of actions to take. The link between events and actions finds its realization in a completely customizable and easily expandable Look-Up Table (LUT) that provides reactive execution paths without affecting transition latency.

With a view to the implementation of a goal-oriented policy, the Planner is implemented as a class that deliberately modifies the policy, that is adapting the execution of tasks by reasoning on a goal [26]. At the current development stage, only the baseline structure of this element is included in the architecture. The second group of units dwelling the Middleware is the Scheduler-Executor tandem, inspired by the Multi-mission EXECutive (MEXEC) architecture [27] and implementing a Multi-threading paradigm. The first actor exploits the real-time utilities to schedule the actions provided by the Event-Handler, using action priorities and queues built accordingly. Subsequently, the Executor is preempted to process the actions on a First-In-First-Out basis, by setting and calling specific thread types. This principle enables high flexibility in multi-threaded and multi-core systems that can potentially run some tasks in parallel or in an asynchronous mode.

The last element of the Layer is the Time Supervisor. Its role is crucial not only in maintaining the synchronization between the ELAPSE and the rest of the hardware and the SPESI propagator but also in obeying the simulation accelerating framework proper of EXTREMA [28, 29]. The GNC algorithms and all the other FlatSat algorithmic functionalities are hosted inside the Application Layer as Task Modules. Specifically, the Navigation Processing Unit (NPU) wraps the optical navigation algorithm that takes a deep-space image as input and estimates the spacecraft state thanks to the identification of stars and planets [14]. The Guidance and Control Unit (GCU) follows, which computes the trajectory for the spacecraft to follow to reach the final target and the corresponding thrust commands [30]. The Thurst Control Unit (TCU), post-processes the thrust commands and organizes the queue to be sent to the thruster test bench. Finally, the Attitude Determination and Control Unit manages the attitude guidance and supervises the state of the ADCU hardware. The NPU and GCU are designed by following the automatic code generation framework, consisting of the creation of C/C++ source code from the original code developed in MATLAB, via the MATLAB Embedded Coder [12]. This ensures agile development and integration of the EXTREMA Pillars at an acknowledged cost of slightly less efficient implementations. Fig. 10 provides an illustrative example of the Continuous Integration (CI) workflow

[12]https://mathworks.com/products/embedded-coder.html [last visited April 15, 2024]

of the algorithms.

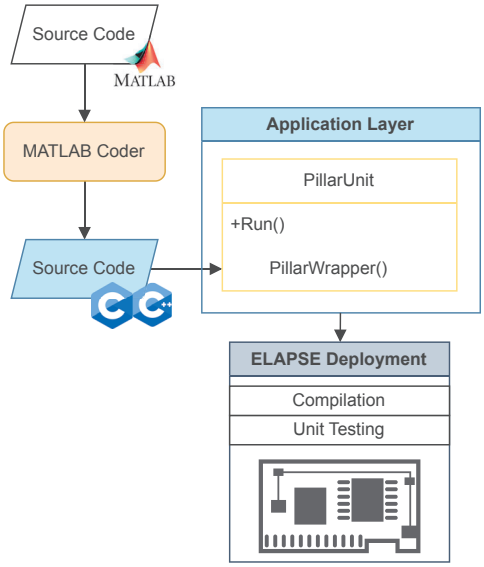


Fig. 10: Continuous integration scheme for the automatically coded GNC algorithms.

Finally, the Hardware Driver Layer hosts the rules and interfaces to exchange data, send commands and telemetry with the ESH hardware units, via wireless or cable connections. The requirement of a data exchange format suitable for resource-constrained embedded systems and with the least latency is paramount to minimize communication delays and to target a uniform real-time infrastructure. To this purpose, Protocol Buffers (ProtoBuf)[13] are employed as proven to be more efficient than other formats [31]. Source ProtoBuf code directly flows from the message interface requirements defined, and the C code to include in the main software is obtained by exploiting the generator from Nanopb [14] library, an embedded systems-oriented C implementation of the protocol. During the simulations, the exchange of messages between devices is performed by exploiting the low-latency User Datagram Protocol (UDP) and message compression.
Normally, embedded systems like the FlatSat OBC, are required to run applications with deadlines to respect, therefore, Real-Time Operative Systems (RTOS) are mainly considered. Some of the most popular open-source RTOS are FreeRTOS [32] and RTEMS[15], which for instance is used on the LEON3 processor, which is the core element of various space on-board computers [33]. However, the current base OS supporting the OBCE C/C++ software is Linux which provides wide customization at a cost of higher computing resources it requires. Moreover, the base kernel is aided with a PREEMPT-RT Linux patch minimizing the kernel latency to implement real-time routines.
It is finally important to mention that the whole architecture targets a static memory allocation philosophy. Where this is not purely achievable, the elements are anyway stored in a bounded scheme. Consequently, the upper bound of memory required by the system is known at compilation time, a fundamental prerequisite for embedded software [34].

2.2 The Attitude Determination and Control Unit

The ADCU software is a C++ FreeRTOS application that runs the attitude determination and control tasks. A dual loop scheme with frequencies in fixed ratios, schematized in Fig. 11, is employed. The attitude determination algorithm is run by the first loop, while the second loop executes the attitude controller. A third loop

[13]https://github.com/protocolbuffers/protobuf [last visited April 15, 2024]
[14]https://github.com/nanopb/nanopb [last visited April 15, 2024]
[15]https://www.rtems.org/ [last visited April 15, 2024]

is taking care of the communication between the ADCU and the OBCE. All the cycles are designed such that the inclusion of modules is agile. Moreover, the unit directly interfaces with the STASIS platform and will command the reaction wheels via the attitude control feedback laws implemented. The platform also hosts a gyroscope that senses its angular rate and is sampled by the software. Moreover, the connection with the OBCE enables the exchange of the ADCU status and attitude telemetry data. Nonetheless, the unit can be used atomically to integrate and test different control and determination algorithms and hardware drivers.

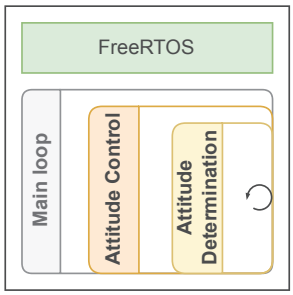


Fig. 11: ADCU software architecture.

The cooperation between the two subsystems results in the modes scheme reported in Fig. 12. As shown, the ADCU modes are hierarchical to the general OBCE GNC modes and during the integrated simulations interact as the dashed-lines highlight. Additionally, transitions of different autonomy levels are also reported. However, the software present design stage bounds all autonomy levels to the event-based ones, that is E3.

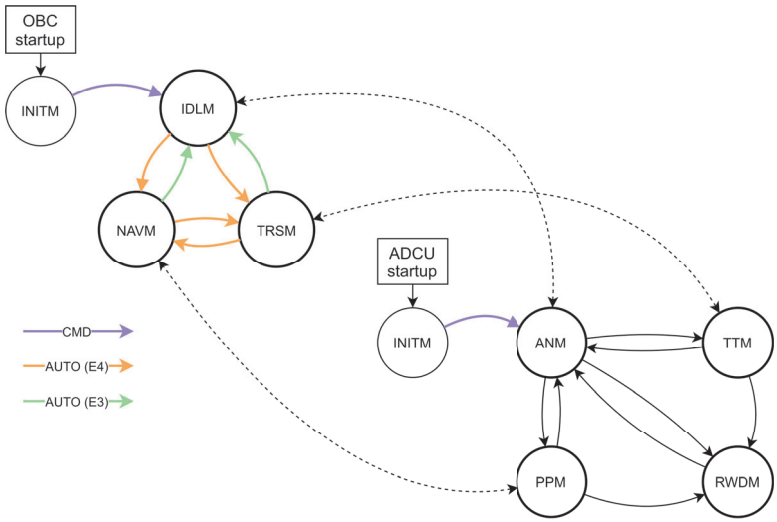


Fig. 12: FlatSat OBCE and ADCU modes.

3 EXTREMA DRY RUNS

The workforce currently involved in the EXTREMA project counts about 10 people. In order to streamline research activities within the single pillars, the team is divided into multiple subgroups, each focusing on a specific aspect of the project – either developing algorithms or setting up the ESH infrastructure. The development of the ESH is carried out in parallel, with the goal of integrating the advancements on a specified schedule, following a Continuous Integration (CI) approach. To ensure the correct integration of the different components, the EXTREMA Dry Runs are periodically performed. These are integration tests and collective debugging sessions aimed at verifying the correct interaction between the different components and assessing

the overall performance statistics and simulation readiness of the ESH. The Dry Runs are carried out on randomly generated mission scenarios, which are designed to test the robustness of the systems. The outcomes of the Dry Runs are downstream analyzed to identify issues, improve the capabilities of the ESH, and identify subsequent development directions to follow.

The typical schedule of the EXTREMA development cycle is depicted in Fig. 13. The schedule has a period of four weeks. Three weeks before a Dry Run, the new features and improvements to develop and implement are declared. Typically, these are divided in high-priority and nice-to-have features. The feature declaration is collectively analyzed in order to spot and solve cross-dependency between development branches. Once agreed, the implementation phase – lasting about 14 days – starts. During the latter, effort is put in developing the declared features without focusing on integrating the modifications with other parties; this is because, given the complexity of the distributed hardware-in-the-loop simulation, unit and regression tests are preferred over integration ones. Seven days before the Dry Run, an assessment of the development results is made. A final call is made on the feature readiness level: the functionalities whose improvement has not been completed are rollbacked to the previous version, in order to guarantee the execution of the Dry Run without simulation-stopping errors. Features that are considered ready are subsequently deployed – where applicable – on the target hardware, and the last round of tests is performed; these can include small integration tests involving only a reduced number of devices. The Dry Run is then performed. A single Dry Run can last from about 3 hours to 6 hours, and foresees the entire EXTREMA team reunited to monitor and evaluate the prosecution of the simulation. The activities are followed according to one or multiple pre-determined schedules; among these:

- random generation and execution of the mission scenario;
- live tracking and debugging of errors and issues arising during the simulation;
- live issue tracking using the available development platform (i.e., GitLab);
- collection and filling of activity reports;
- generation of simulation logs.

While validating the effectiveness of the algorithm and facilities is of interest during the sessions, the main focus is to verify the correct interaction between the different components of the ESH and assess the integrated performance of the systems. While the inter-device loop is the one described in Section 1, the schedule of a single Dry Run is pre-determined, and can foresee the spacecraft in different phases of the transfer. In order to ensure test coverage, each Dry Run is performed on a randomly-generated scenario. The scenario could be randomly picked from the publicly-available EXTREMA scenario database [35] or computed on-the-fly; in this case, an indirect optimization algorithm is run over a random initial condition in order to generate a feasible initial guess for the trajectory optimization algorithm. The spacecraft orientation is carefully picked in order to guarantee a minimum number of observable planets. This information is then relayed to a) the SPESI orbital propagator, and b) the FlatSat on-board computer. This is the only piece of information the FlatSat receives from the external world. An initial clock calibration procedure is performed in order to minimize any pre-existing time drift among the device clocks. Once the initial conditions and scenario information are passed to the devices, the simulation starts and the devices are followed while they interact in complete autonomy.

After the Dry Run, a debriefing session is held, where the results are analyzed and the root causes of the issues encountered are identified. Immediate actions – such as version rollback or consolidation – are taken. The next development cycle is then planned, and the schedule is repeated.

The idea behind the EXTREMA Dry Run is to simulate a single leg of an interplanetary transfer simulation under the supervision of the people involved.

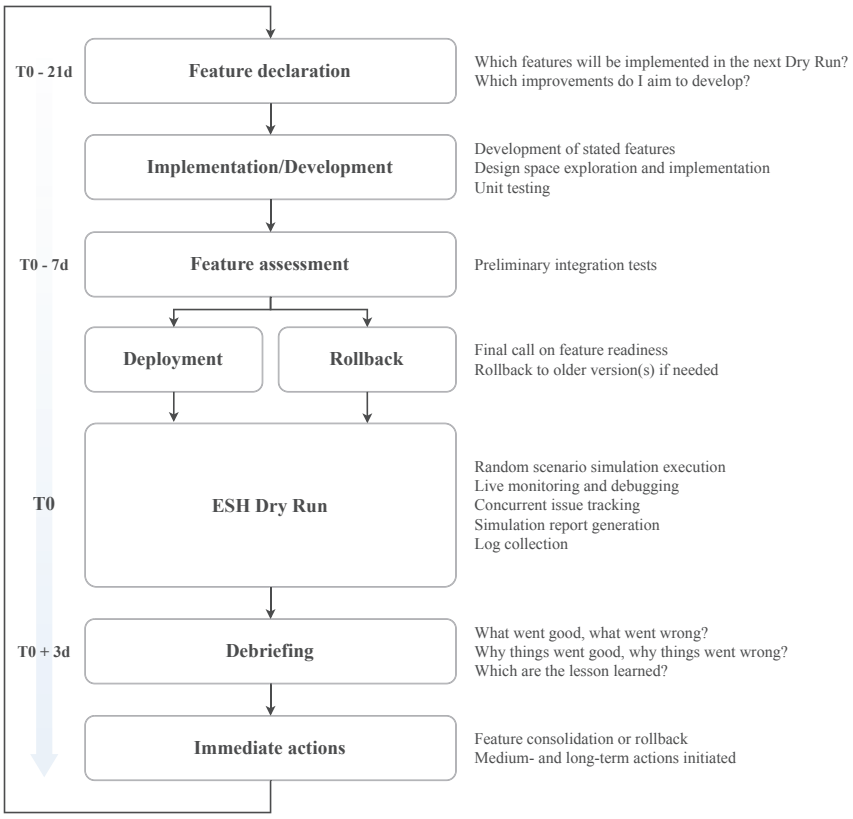


Fig. 13: Typical schedule of a Dry Run development cycle

3.1 MoniCA: the Monitoring and Control Application

In order to guarantee the Dry Run operators access to the information being exchanged and computed during a single dry run, a tailored monitoring framework has been set up. The setup was developed with the following concepts in mind: a) the monitoring stack on each device should not interfere with other operations; b) it should be compatible with a wide gamma of devices and programming languages; c) the architecture should be easily extendable with the addition of new devices within the simulation. Referring to the OSI layers for communication protocols, the existance of Wi-Fi interface on every device involved allowed us to select the UDP datagram protocol to cover the network stack up to the Transport Layer [36]. When available, a cabled PHY interface toward the central switch was preferred in order to optimize latency and minimize packet loss. The UDP solution was preferred over the TCP/IP because – while not guaranteeing hard real-timeness – it does not rely on the receiving device being up and responsive. For what concerns the upper layers, different encapsulations for the message were tested. Among these, the JSON format was initially employed, due to its user-friendliness and human readability; similar considerations were performed due to the YAML protocol. However, the need of external plug-ins in order to guarantee the consistency of a schema moved the authors to select Protocol Buffers – or protobufs – as encapsulation method. Among the advantages of protocol buffers:

- they are language-agnostic, meaning that the same message can be sent and received by devices programmed in different languages;
- they are schema-based, meaning that the message structure is defined in a separate file, and the compiler generates the necessary code to serialize and deserialize the message;
- they are binary, meaning that the message is sent in a compact form, reducing the network load and the time needed to send and receive the message.

- they are extensible, meaning that the message structure can be easily modified by adding or removing fields, without breaking the compatibility with the previous versions of the message.
- they are fast, meaning that the serialization and deserialization process is optimized for speed, reducing the time needed to send and receive the message.

Regarding last points, [31] shows how protocol buffers are at least one order of magnitude faster than other approaches. Moreover, existing compiler for protobuf are easily implemented in CI/CD pipelines and guarantee a predictable signature of the generated files.

3.1.1 MoniCA Software Architecture

MoniCA is thought as a central node to monitor and control the simulation without interfering with inter-device exchanges. Monica consists of two main components: a web server to serve the clients connected through the web interface, and a device server to monitor the telemetry and communication between the devices. Both have been developed in C++, employing the Crow.cpp web framework [16]. This also allows an in-memory data path faster than traditional file- or database-based approaches. MoniCA leverages heavily on the advantages of protocol buffers and CI/CD practices. Upon building, a pre-build script scans the entire EXTREMA GitLab repository for .proto files; these files are developed with the following template:

Listing 1: Example of a .proto file for the deviceMessage structure.

```
syntax = "proto2";
import "nanopb.proto";

message messageTypeOne {
    required int32 timestamp = 1;
    required int32 fieldOne = 2;
    required int32 fieldTwo = 3;
    required int32 fieldThree = 4;
}

    message messageTypeTwo {
    required int32 timestamp = 1;
    required int32 fieldOne = 2;
    required int32 fieldTwo = 3;
    required int32 fieldThree = 4;
}

message deviceMessage {
    required uint32 uid = 1 [default = 1234];
    oneof msg {
        messageTypeOne msgOne = 2;
        messageTypeTwo msgTwo = 3;
    };
}
```

Each device has a default unique ID (UID); each UID is mapped to a single protobuf specification file. MoniCA ensures no repeated UIDs are found. When a `deviceMessage` type is individuated - MoniCA compiles the proto with the `nanopb` utility and generates `.pb.h` headers and `.pb.c` source files for the respective structure. The generated structure is consistent with the protobuf structure, and features static memory allocation; this means that the same implementation can be employed on any device - from embedded to higher level ones. A set of macros generate modern C++ classes encapsulating the generated C structure and providing higher-level functionality.

[16] https://github.com/CrowCpp/Crow

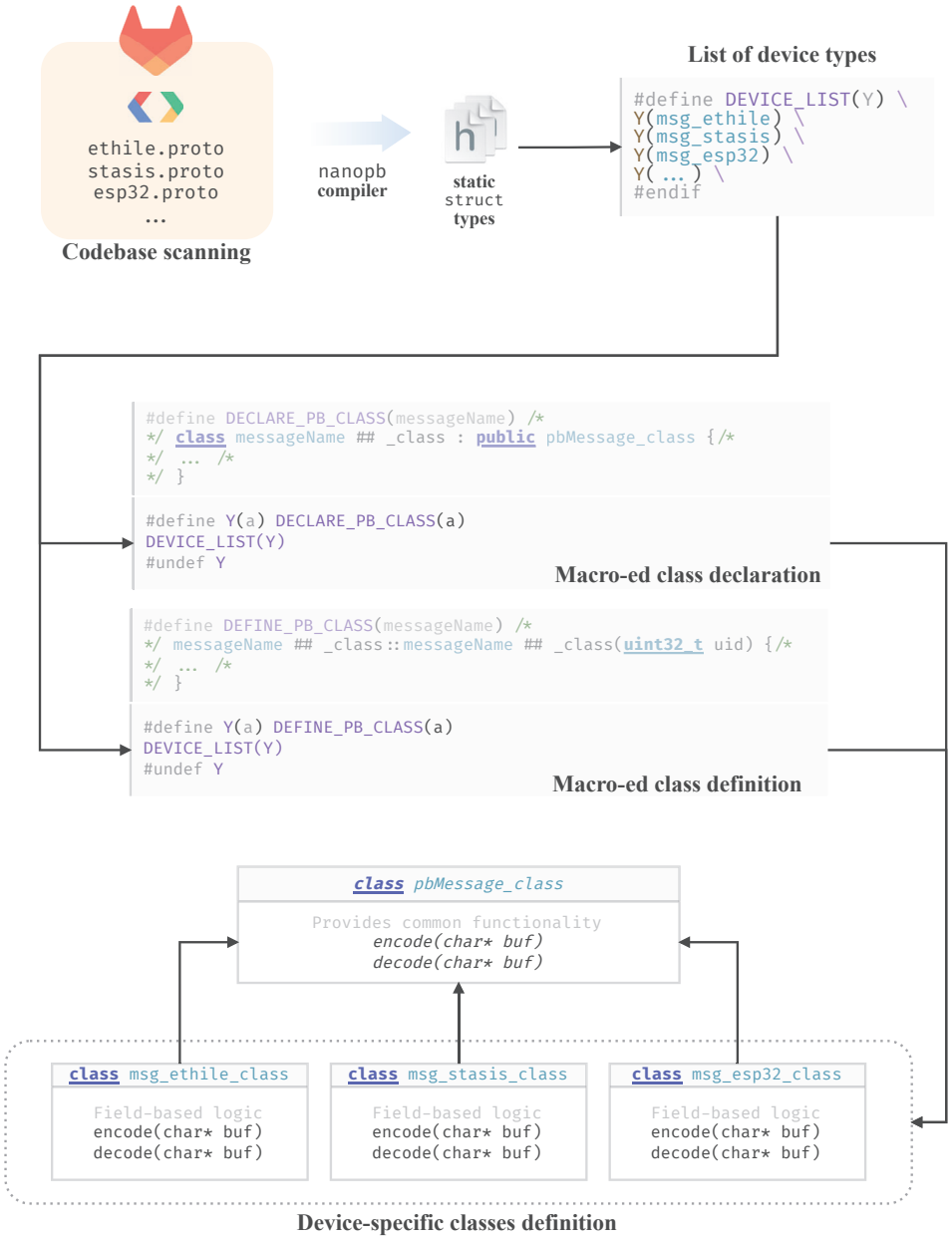


Fig. 14: MoniCA macro-ed class structure

When the device turns on, it must send a subscription message to MoniCA. The subscription message should contain the UID of the subscribing device and wether the communication channel is multicasted over the network; this allows MoniCA to bind the IP address of the device to the UID and subscribe to the multicast group if necessary. Whenever an UDP datagram is received, the IP address is matched against the UID, allowing proper de-serialization of the message into one of its sub-messages. The deserialization is trivially executed thanks to the polymorphism class structure of the message specification, allowable thanks to `nanopb` predicatble signature.

On the client side, the frontend is being developed with a widget-like structure (Fig. 16). JavaScript bindings to protobuf are employed to ensure the deserialization of protobuf to the proper `Object` prototype. Upon loading, the interface sends a subscription message to MoniCA, indicating which devices and which sub-messages it wants to monitor. This allows to define a set of callbacks.

When MoniCA receives a protobuf via UDP, it checks the frontend subscription list for clients interested in that update; the encoded sub-message is relayed to all the frontends via WebSocket connection. The described

approach allows to extend the monitoring capabilities without adding non-specific code functionality; new widgets can be developed focusing entirely on the visualization side, exploiting the automation guaranteed by protobuf compilers and tailor-made C++ macros. The choice of a JavaScript frontend allows all data visualization tools to run on the client device, reducing the load on the server side and allowing the MoniCA backend to process data with minimal latency.

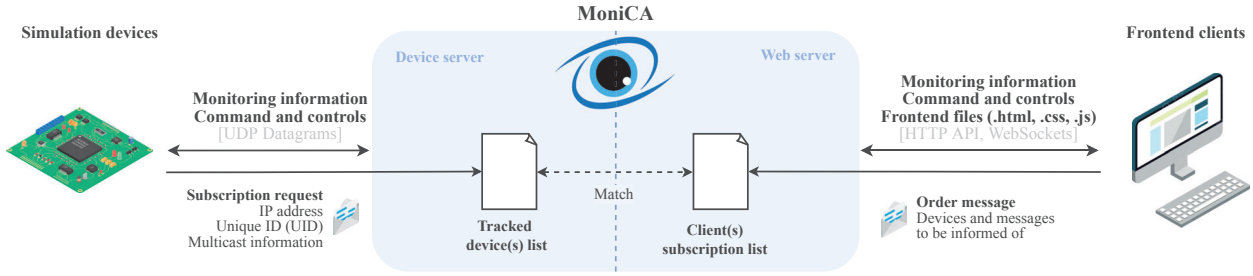
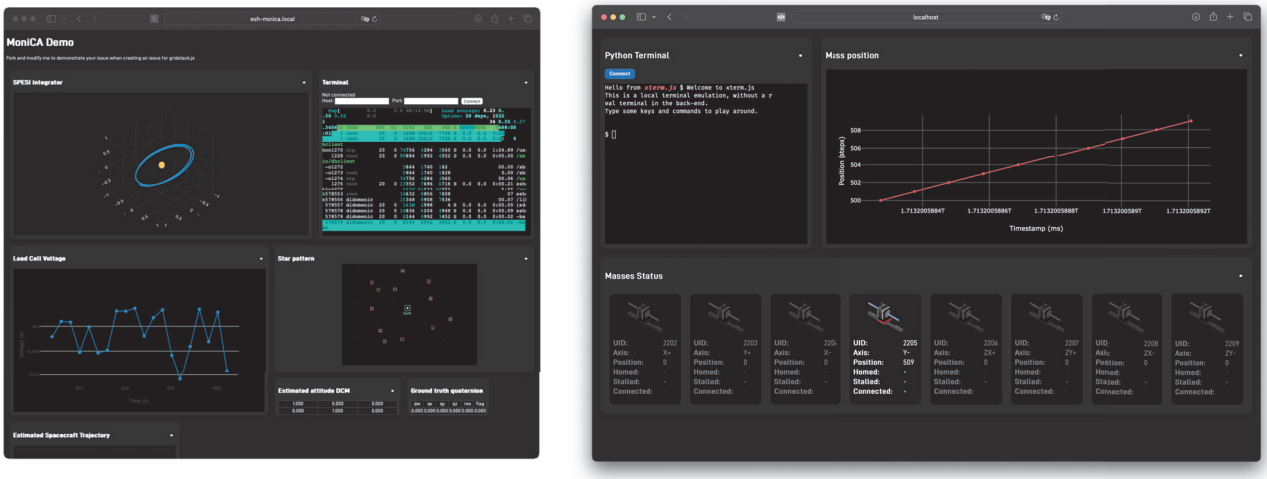


Fig. 15: MoniCA interaction with devices and clients



(a) Logging and control interface

(b) Balancing system interface

Fig. 16: Visualization of MoniCA web interface

CONCLUSIONS

This paper presented the advancements made in the development of an integrated hardware-in-the-loop facility for autonomous CubeSat GNC technologies. The EXTREMA Simulation Hub has already proven to be realistic and efficient testing environment for CubeSat missions. Through the integration of hardware and software components, the HIL facility enables the evaluation and validation of CubeSat GNC systems in a controlled and repeatable manner.

In particular, the advancements made in the development of the EXTREMA Simulation Hub were discussed. The paper highlighted the progress made in the development of the hardware facilities, including the RETINA optical facility, the ETHILE thrust test bench, the STASIS spacecraft attitude simulator and the SPESI stochastic orbital propagator. These facilities provide the necessary hardware support for the testing of CubeSat GNC algorithms and the validation of their performance in realistic space environments. The ELAPSE software architecture was also presented, focusing on the implementation of the flight software. The software was designed with a modular approach in mind, enabling the quick integration of goal-specific applications and algorithms. This approach allows for the rapid deployment of the software on multiple hardware architectures under a Continuous Integration/Continuous Deployment (CI/CD) framework. The software architecture was designed to be lightweight and efficient, ensuring the real-time execution of the CubeSat GNC algorithms.

Finally, the workflow of the EXTREMA Dry Runs was presented. These periodic integration tests are essential for verifying the correct interaction between the different components of the EXTREMA Simulation Hub and assessing its overall performance. The Dry Runs are carried out on randomly generated mission scenarios, which are designed to test the robustness of the systems. The Dry Runs leverage MoniCA, the Monitoring and Control Application, to enable telemetry and status monitoring of all the devices involved in the simulation. This unified and lightweight solution provides a user-friendly web-based frontend for the comprehensive monitoring and control of the entire simulation.

Overall, we consider the realization of the ESH a significant contribution in the field of CubeSat GNC technologies. It provides researchers and companies with a powerful tool for testing and validating CubeSat GNC systems, ultimately contributing to the advancement of autonomous small spacecraft missions. By targeting cost-effective probes, EXTREMA has the potential of impacting also larger ones, usually characterized by less operational constraints. Future work will focus on further enhancing the capabilities of the ESH, formally assessing its validation through integration of third-party technologies and systems.

ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 864697).

REFERENCES

[1] J. Schoolcraft, A. Klesh, and T. Werne. "MarCO: interplanetary mission development on a CubeSat scale". In: *Space Operations: Contributions from the Global Community*. Springer, 2017, pp. 221–231. DOI: 10.2514/6.2016-2491.

[2] R. Walker et al. "Miniaturised Asteroid Remote Geophysical Observer (M-ARGO): a stand-alone deep space CubeSat system for low-cost science and exploration missions". In: *6th Interplanetary CubeSat Workshop, Cambridge, UK*. Vol. 30. 05. 2017.

[3] F. Topputo et al. "Envelop of reachable asteroids by M-ARGO CubeSat". In: *Advances in Space Research* 67.12 (2021), pp. 4193–4221. DOI: 10.1016/j.asr.2021.02.031.

[4] D. Perico et al. "ELAPSE: A FlatSat Software and Processing Unit for Deep-Space Autonomous GNC Systems Testing". In: *46th AAS Guidance, Navigation and Control Conference*. 2024.

[5] C. A. Kluever. "Heliospheric boundary exploration using ion propulsion spacecraft". In: *Journal of Spacecraft and Rockets* 34.3 (1997), pp. 365–371. DOI: 10.2514/2.3218.

[6] R. Nah, S. Vadali, and E. Braden. "Fuel-optimal, low-thrust, three-dimensional Earth-Mars trajectories". In: *Journal of Guidance, Control, and Dynamics* 24.6 (2001), pp. 1100–1107. DOI: 10.2514/2.4844.

[7] V. Franzese et al. "Target Selection for M-ARGO Interplanetary CubeSat". In: *71st International Astronautical Congress (IAC 2020)*. 2020, pp. 1–15.

[8] G. Di Domenico. "Development of a hardware-in-the-loop simulation framework for interplanetary transfers on smaller timescales". MA thesis. Politecnico di Milano, 2020.

[9] F. Ornati et al. "RETINA: a Highly-Versatile Optical Facility for Camera-In-the-loop Testing of Spaceborne Vision-Based Sensors". In: *46th AAS Guidance, Navigation and Control Conference*. 2024, pp. 1–19.

[10] P. Panicucci et al. "Vision-Based Navigation for the LUMIO CubeSat Mission". In: *46th AAS Guidance, Navigation and Control Conference*. 2024, pp. 1–20.

[11] C. Beierle and S. DAmico. "Variable-Magnification Optical Stimulator for Training and Validation of Spaceborne Vision-Based Navigation". In: *Journal of Spacecraft and Rockets* 56.4 (2019), pp. 1060–1072. DOI: 10.2514/1.A34337.

[12] G. Rufino and A. Moccia. "Laboratory test system for performance evaluation of advanced star sensors". In: *Journal of Guidance, Control, and Dynamics* 25.2 (2002), pp. 200–208. DOI: 10.2514/2.4888.

[13] P. Panicucci and F. Topputo. "The TinyV3RSE Hardware-in-the-Loop Vision-Based Navigation Facility". In: *Sensors* 22.23 (2022), p. 9333.

[14] E. Andreis, V. Franzese, and F. Topputo. "Onboard Orbit Determination for Deep-Space CubeSats". In: *Journal of Guidance, Control, and Dynamics* 45.8 (2022), pp. 1466–1480. DOI: 10.2514/1.G006294.

[15] G. Di Domenico, F. Topputo, et al. "STASIS: an Attitude Testbed for Hardware-in-the-Loop Simulations of Autonomous Guidance, Navigation, and Control Systems". In: *73rd International Astronautical Congress (IAC 2022)*. 2022, pp. 1–20.

[16] G. A. Smith. "Dynamic simulators for test of space vehicle attitude control systems". In: *VA. POLYTECH. INST. PROC. OF THE CONF. ON THE ROLE OF SIMULATION IN SPACE TECHNOL., PT. C¡ 1965¡(SEE N69-18979 08-11)*. 1965.

[17] G. Di Domenico and F. Topputo. “Disturbances Quantification for Air-Bearing Spacecraft Attitude Simulation Platforms”. In: *AIAA Scitech 2024 Forum*. 2024, p. 0379.

[18] F. Ornati et al. “High-accuracy vision-based attitude estimation system for air-bearing spacecraft simulators”. In: *arXiv preprint arXiv:2312.08146* (2023).

[19] C. Giordano, F. Topputo, et al. “SPESI: A Real-Time Space Environment Simulator for the EXTREMA Project”. In: *33rd AAS/AIAA Space Flight Mechanics Meeting*. 2023, pp. 1–13.

[20] F. Reghenzani, G. Massari, and W. Fornaciari. “The real-time linux kernel: A survey on preempt_rt”. In: *ACM Computing Surveys (CSUR)* 52.1 (2019), pp. 1–36.

[21] A. Rler. “Runge–Kutta methods for the strong approximation of solutions of stochastic differential equations”. In: *SIAM Journal on Numerical Analysis* 48.3 (2010), pp. 922–952.

[22] J.-L. Terraillon. “SAVOIR: Reusing specifications to improve the way we deliver avionics”. In: *Embedded Real Time Software and Systems (ERTS2012)*. Toulouse, France, 2012.

[23] P. Plasson et al. “GERICOS: A Generic Framework for the development of on-board software”. In: *DASIA 2016-Data Systems In Aerospace* 736 (2016), p. 39.

[24] C. Coelho, O. Koudelka, and M. Merri. “NanoSat MO framework: When OBSW turns into apps”. In: *2017 IEEE Aerospace Conference*. 2017, pp. 1–8. DOI: 10.1109/AERO.2017.7943951.

[25] R. Bocchino et al. “F Prime: an open-source framework for small-scale flight software systems”. In: *32nd Annual AIAA/USU Conference Small Satellites*. 2018.

[26] F. Ingrand and M. Ghallab. “Robotics and artificial intelligence: A perspective on deliberation functions”. In: *AI communications* 27.1 (2014), pp. 63–80.

[27] M. Troesch et al. “MEXEC: An Onboard Integrated Planning and Execution Approach for Spacecraft Commanding”. In: *30th International Conference on Automated Planning and Scheduling*. Nancy, France, 2020.

[28] G. Di Domenico. *Development of a hardware-in-the-loop simulation framework for interplanetary transfers on smaller timescales*. 2019.

[29] G. Di Domenico et al. “The ERC-Funded EXTREMA Project: Achieving Self-Driving Interplanetary CubeSats”. In: *Modeling and Optimization in Space Engineering: New Concepts and Approaches*. Ed. by G. Fasano and J. D. Pintr. Springer International Publishing, 2023, pp. 167–199. DOI: 10.1007/978-3-031-24812-2_6.

[30] A. C. Morelli et al. “Convex Trajectory Optimization Using Thrust Regularization”. In: *Journal of Guidance, Control, and Dynamics* (2023), pp. 1–8. DOI: 10.2514/1.G007646.

[31] D. Friesel and O. Spinczyk. “Data serialization formats for the internet of things”. In: *Electronic Communications of the EASST* 80 (2021). DOI: 10.14279/tuj.eceasst.80.1134.

[32] D. Dharbe, S. Galvo, and A. M. Moreira. “Formalizing FreeRTOS: First Steps”. In: *Formal Methods: Foundations and Applications*. 2009, pp. 101–117. DOI: 10.1007/978-3-642-10452-7_8.

[33] D. Guzman et al. “Improving the LEON spacecraft computer processor for real-time performance analysis”. In: *Journal of Spacecraft and Rockets* 48.4 (2011), pp. 671–678.

[34] G. J. Holzmann. “The power of ten–rules for developing safety critical code”. In: *Software Technology* 10 (2018).

[35] G. Merisio. *EXTREMA: Random autonomous interplanetary mission scenarios for EXTREMA Simulation Hub (ESH) hardware-in-the-loop simulations*. Version 1.3.0. Zenodo, Aug. 2023. DOI: 10.5281/zenodo.8300807. URL: https://doi.org/10.5281/zenodo.8300807.

[36] H. Zimmermann. “OSI reference model-the ISO model of architecture for open systems interconnection”. In: *IEEE Transactions on communications* 28.4 (1980), pp. 425–432.