AAS 24-084

ELAPSE: A FLATSAT SOFTWARE AND PROCESSING UNIT FOR DEEP-SPACE AUTONOMOUS GNC SYSTEMS TESTING

Davide Perico[*], Gianfranco Di Domenico[†], Gianmario Merisio[‡], and Francesco Topputo[§]

EXTREMA is a European Research Council-funded project challenging the current paradigm in spacecraft guidance, navigation, and control, by enabling CubeSats to autonomously reach their target in interplanetary space. The core of the project is the EXTREMA Simulation Hub, an integrated facility in which to perform closed-loop simulations of the spacecraft-environment interaction, allowing high-fidelity testing of deep-space autonomous guidance, navigation, and control technologies. This work presents EXTREMA's FlatSat Processing System (ELAPSE), developed on the side of the testing hub and aiming to provide a central processing core for hardware-in-the-loop simulations and an execution platform for guidance, navigation, and control algorithms. The system comprises two main interfaced elements, a multi-core central on-board computer and a unit for attitude determination and control. These are populated by highly modular and layered software easing the integration and accelerating the algorithms development. ELAPSE enforces a reactive event-based autonomy paradigm and paths the way for developing deliberate procedures to reach goal-oriented autonomy on system-on-a-chip hardware.

INTRODUCTION

Nowadays, the growth of the space sector sees CubeSats as one of the major actors, thanks to their concept inferencing cost-effective and relatively fast access to space.[1] Moreover, the combination of these features with the recent advancements in onboard technology has pushed their adoption also for deep-space missions.[2] However, the increase in the number of such missions and the high data rates required for operating them from ground will shortly lead to saturation of ground stations.[3] Moreover, the current approach to operations strongly affects the total mission cost, threatening the benefits derived from the adoption of CubeSats. The EXTREMA (Engineering Extremely Rare Events in Astrodynamics for Deep-Space Missions in Autonomy) project, awarded a European Research Council (ERC) Consolidator Grant in 2019, challenges the current paradigm by enabling self-driving interplanetary spacecrafts.[4] Deep-space Guidance, Navigation, and Control (GNC) activities applied in a complex scenario are the core subject of EXTREMA which finds its foundations on three pillars. Pillar I regards autonomous navigation.[5] Pillar II concerns autonomous guidance and control. Pillar III deals with autonomous ballistic capture.[6] The outcome of each Pillar targets the integration in the EXTREMA Simulation Hub (ESH), a Hardware-In-the-Loop (HIL) facility

[*] PhD Student, Department of Aerospace Science and Technology, Politecnico di Milano, Via La Masa 34, 20156, Milan, Italy, davide.perico@polimi.it.

[†] PhD Student, Department of Aerospace Science and Technology, Politecnico di Milano, Via La Masa 34, 20156, Milan, Italy, gianfranco.didomenico@polimi.it.

[‡] Post Doctoral Research Fellow, Department of Aerospace Science and Technology, Politecnico di Milano, Via La Masa 34, 20156, Milan, Italy, gianmario.merisio@polimi.it.

[§] Full Professor, Department of Aerospace Science and Technology, Politecnico di Milano, Via La Masa 34, 20156, Milan, Italy, francesco.topputo@polimi.it.

enabling verification and validation of autonomous GNC technologies,[7] a fundamental step towards the adoption of them on flying spacecraft.
A FlatSat-based architecture enables high flexibility and modularity in the algorithms and systems to be tested and interconnects the other subsystems implementing different purposes. The concept has a long heritage as a tool to assess the functionality of medium-sized and large satellites.[8] Among the spacecraft's different elements, the On-Board Computer (OBC) is a pivotal component, providing the processing capability and hosting the flight software. The latter handles data, executes the onboard routines, and commands and interfaces with the other subsystems. The integration of a FlatSat OBC in the EXTREMA ESH is paramount to frame the Pillars algorithmic output into the GNC flight software, implementing higher-level functionalities and providing the connecting core of the HIL units. Consequently, this leverages the execution of high-fidelity integrated simulations.
In the context of EXTREMA, while the Pillars supply algorithms that autonomously provide the spacecraft with the knowledge of its actual state, compute the path to follow to reach its target, and the thrust commands to follow that trajectory, the GNC software supports and enforces these functionalities at the system level. To this degree, different definitions and levels of autonomy can be recognized. The European Space Agency (ESA), through the European Cooperation for Space Standardization (ECSS) initiative, identifies four levels of autonomy, from E1 to E4, briefly reported in Table 1.[9]

Table 1 System autonomy levels.

Level	Description
E1	Strong ground control, time-tagged commands limited to contingencies.
E2	Pre-planned time-tagged commands managed by a scheduler.
E3	Event-based autonomous operations, onboard adaptation.
E4	Goal-oriented autonomous operations and re-planning.

Considering the highest levels, while E3 is achieved by event-based transitions and state-dependent adaptability, E4 addresses the so-called goal-oriented autonomy, implemented through objective-based re-planning. In recent decades, novel approaches to designing flight software for spacecraft autonomy have been published to provide effective frameworks for robust architectures. Among others, the Multi-mission EXECutive (MEXEC) architecture, developed by the Jet Propulsion Laboratory (JPL), focuses on the planning and execution of flight-related tasks.[10] The idea is to implement an adaptive control to the scheduling of the tasks and modify the plan under the constraints and execution status. Furthermore, as decision-making is a crucial aspect in enabling autonomous selection of actions to take, the Framework for Robust Execution and Scheduling of Commands On-Board (FRESCO) issues the paradigm behind this ability, by proposing high-level fundamentals of software design for on-board autonomy, incorporating also the MEXEC structure.[11] An additional aspect, when the testing capability of the FlatSat is considered, is the simulation environment at the edge. As an example, the Basilisk astrodynamics framework moves in that direction by providing modular integration of models to simulate the space environment and the spacecraft subsystems by exploiting components wrappers in Python.[12] Moreover, the development of autonomous GNC software, facilitated by the FlatSat system, is concomitant with the meticulous selection and prototyping of the underlying hardware. As a matter of fact, the introduction of an autonomous algorithm furnishes the spacecraft with heightened operational autonomy, thereby leading to improved performance. However, the increase in complexity caused by this augmentation necessitates adherence to more stringent reliability requirements. Consequently, the imperative role of testing the software on representative embedded hardware emerges as a pivotal step in certifying the technological maturity of these advancements. In this setting, the literature presents two pertinent testing techniques, namely, Processor-In-the-Loop (PIL) and Hardware-In-the-Loop (HIL). The former entails deploying the software module onto an embedded platform representative of the target processing unit. This deployment is undertaken to verify and validate the software's performance and resource requirements, thereby certifying its maturity for on-board implementation. The latter technique subjects the software to testing within a simulated operational environment and against authentic hardware components or models.[13] Considering the autonomous GNC software development aspect and the HIL testing necessity, this paper aims to present the EXTREMA fLAtSat Processing SystEm (ELAPSE), an apparatus that combines a highly-modular GNC flight software residing on flight-representative embedded hardware. Its objective is to provide the EXTREMA ESH with a core unit that models the processing subsystem of an autonomous interplanetary spacecraft. Its functionalities interconnect the facility units to enable an integrated HIL simulation environment and provide a key platform and software to advance the development of autonomous GNC technologies via validation and verification of the algorithms and the processes to control them at a higher level.

This manuscript is organized as follows. The current stage of the HIL facility of EXTREMA is presented in Section II. In Section III, the ELAPSE software architecture is described. Specifically, the two main units constituting the system, the On-Board Computer Emulator, and the ADCU, are delineated, together with the main design principles applied. Section IV presents the embedded processing system hosting the FlatSat software and provides a first representative model of the computing hardware. Finally, the PIL and HIL simulation capabilities of the ELAPSE system are described in Section V. Specifically, three application examples in which the system was successfully employed to perform validation and verification tests are shown.

THE EXTREMA SIMULATION HUB

The EXTREMA HIL facility targets the integrated simulation of a deep-space autonomous spacecraft by following a closed-loop guidance paradigm. Among the others, three main experiment test benches establish the simulation infrastructure. Concretely, the Realistic Experimental faciliTy for vision-based NAvigation (RETINA)[14] renderers deep-sky images with sub-pixel resolution and projects them on a high-resolution screen. A system of lenses and a camera completes the apparatus, acquiring the image of the screen and producing a high-fidelity optical measurement that shall be fed into the optical navigation algorithm. The thrust control is implemented by the EXTREMA THruster In the Loop Experiment (ETHILE)[15] test bench, implementing a compressed-air valve that simulates the pressure exerted by a low-thrust engine. It is composed of a sensing system that senses the force produced by the nozzle firings and transmits the acquired data to the orbit propagator. The latter, called EXTREMA's SPace Environment SImulator (SPESI) oversees the running process by simulating the space dynamics and environment with high fidelity.[16] Finally, the SpacecrafT Attitude SImulation System (STASIS)[17] is an air-bearing platform with 3 degrees-of-freedom that represents the spacecraft rotational dynamics and grants the orientation control via a set of reaction-wheels. The facility's testbeds, modeling the GNC peripherals, are engineered and interfaced following a FLatSat architecture, whose processing core is shaped through ELAPSE. Its sub-units dwell on the STASIS platform and interface with the rest of the test benches via wireless communication. The described framework is schematically reported in Figure 1, where the arrows indicate the functionality implemented by each interface. Furthermore, the entire framework relies on a dynamics accelerating framework to simulate deep-space missions with hardware in a reduced time by exploiting a dynamics similarity approach .[4,18]

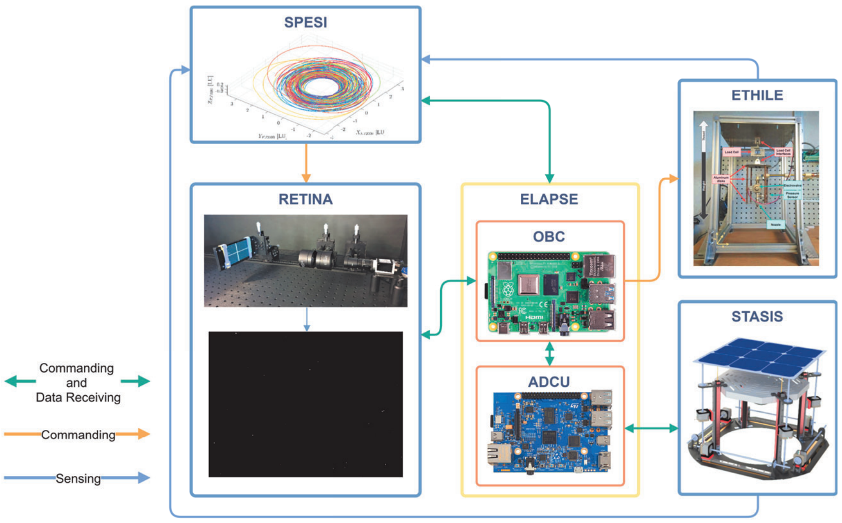


Figure 1 ESH functional breakdown structure including ELAPSE.

ELAPSE SOFTWARE ARCHITECTURE

The FlatSat hardware-software system 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 in terms of orbit and attitude and interfacing with the different ESH hardware units previously described.

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. Moreover, it participates in the effort of archetype shift professed by EXTREMA and enables Hardware–In–the–Loop testing. A highly modular architecture is implemented following the Object-Oriented Programming (OOP) paradigm, orbiting around objects as fundamental elements. Additionally, as the final objective is the on-board implementation, the methodology relies on embedded programming. The main structure, which retraces other flight software framework characteristics, is composed of different levels of abstraction here described. Among the different frameworks, the NASA core Flight System (cFS)[*] has a long space missions flight heritage and pursues reusability through a layered and plug&play architecture. Its main element is the core Flight Executive, contained in a so-called Core Layer. The Abstraction Library Layer provides support and interfaces with different Operative Systems (OS). Additionally, an Application Layer hosts mission-agnostic components and services. Finally, the hardware is integrated through a hardware layer.

[*]https://cfs.gsfc.nasa.gov/ [last visited January 25, 2024]

A similar philosophy, with a different outcome, was followed by ESA with the Space AVionics Open Interface aRchitecture (SAVOIR) initiative that resulted in an On-board Software Reference Architecture (OSRA), based on the principles of reusability and model-based development.[19] The core principle is the presence of components implementing the different functionalities that are executed through an execution platform via mapping through an interaction layer.[19] The same concept of software genericity is retained by the GEneRIC Onboard Software (GERICOS) framework, which focuses on the rapid development of flight software specific to the payload.[20] A three-layer structure includes a core that links a real-time kernel to the active objects. It follows a Blocks layer that provides generic reusable components that can be adapted to the required functionalities for a specific mission. Finally, its Drivers layer is specifically designed for targeting LEON processor's IP cores.[20] Ultimately, other frameworks have been specifically developed for small satellites. One example is the NanoSat MO Framework (NMF), written in Java. It conceives an end-to-end solution to execute custom applications within a modular and flexible approach.[21] A comprehensive comparison of the main characteristics of these frameworks has been performed in the past.[22] Additionally, the Jet Propulsion Laboratory published F Prime, a C++ component-based framework specifically tailored for CubeSats, SmallSats, instruments, and in general entities with sparse processor resources.[23] F Prime is strongly centered on the concept of components and it also includes edge tools for unit and system-level testing. Both NMF and F Prime have been demonstrated in space.[23,24] 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. Figure 2 unveils the OBCE software's three-layer architecture.

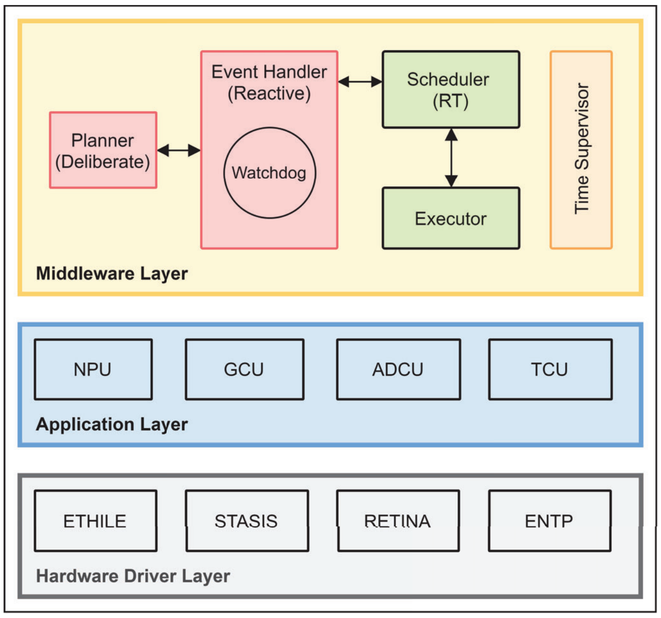


Figure 2 FlatSat OBC software architecture.

Each layer serves as a container for Modules, classes that interface with each other by following the OOP four principles of encapsulation, data abstraction, polymorphism, and inheritance. 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 two main modalities, as shown in Figure 3.

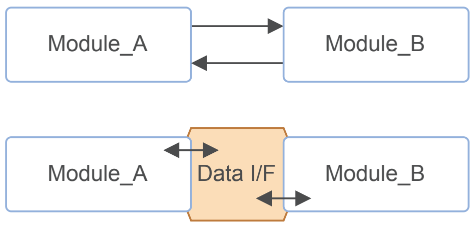


Figure 3 OBCE software modules interfaces.

The first one is end-to-end, with direct passage of data, while the second one relies on additional classes, namely Data Interfaces, that provide more complete 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. This layer is the only one that potentially undergoes localized modifications to adapt the software to different OS.
At the current development stage, an event-based execution policy is targeted. Referencing the aforementioned scheme, in red there are the units proposed to manage it, forming the ELAPSE Policy Manager.
The logic is based on the generation and processing of events. Specifically, a Task Module method returns an execution status merged with a unique module identifier, constituting the event. 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), reported in Listing 1, that provides reactive execution paths without affecting transition latency.

Listing 1 Example of Look-Up-Table Event-Handler.

```
TasksEnum i_LUT_EVENT_DISPATCHER[8][7]
    = {{ABORT, EXADCU, EXNPU, EXNPU, EXDIRGCU, EXINDGCU, EXADCU},
       {ABORT, IDLE, EXNPU, EXNPU, EXNPU, EXADCU, EXDIRGCU},
       {ABORT, EXDIRGCU, EXNPU, EXADCU, IDLE, STOP, NONE},
       {ABORT, EXTCU, EXADCU, EXNPU, IDLE, STOP, NONE},
       {ABORT, EXADCU, EXADCU, EXDIRGCU, IDLE, INIT, NONE},
       {ABORT, EXNPU, IDLE, RECOVER, STOP, NONE, NONE},
```

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.[25] 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 aforementioned MEXEC architecture[10] and implementing the Multi-threading paradigm. The first actor exploits the real-time utilities proper of a real-time OS to schedule the actions provided by the Event-Handler. This is achieved by the use of action priorities and queues built accordingly. Subsequently, the executor is preempted to process the actions on a First-In-First-Out basis related to the queue just updated. The Executor calls the methods mapped from the actions specifying the type of thread to be created. This principle enables high flexibility in multi-threaded and multi-core systems that can potentially run some tasks in parallel or non-synchronized mode.

The last element of the Layer is the Time Supervisor, belonging to the task modules category. Its role is crucial not only 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.[4,18] 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.[5] 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.[26] 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 [*]. This ensures agile development and integration of the EXTREMA Pillars at an acknowledged cost of slightly less efficient implementations. Figure 4 provides an illustrative example of the Continuous Integration (CI) workflow of the algorithms.

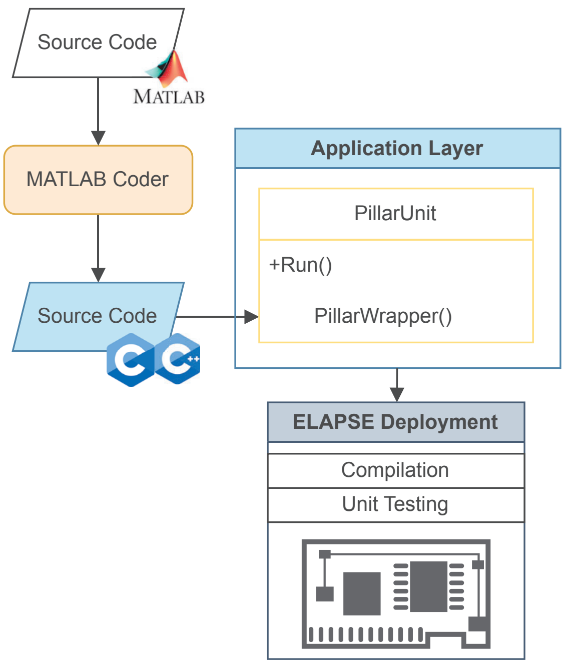


Figure 4 Continuous integration scheme for the automatically coded GNC algorithms.

Finally, the Hardware Driver Layer hosts the rules and interfaces to communicate with the ESH hardware units to receive and send data, commands, and telemetry 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)[†] are employed as proved to be more efficient than other formats.[27] Source ProtoBuf code directly flows from the message inter-

[*]https://mathworks.com/products/embedded-coder.html [last visited January 25, 2024]
[†]https://github.com/protocolbuffers/protobuf [last visited January 25, 2024]

face requirements defined, and the C code to include in the main software is obtained by exploiting the generator from Nanopb [*] 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. Furthermore, the Middleware Layer has been conceived as the connector to the OS which provides functionalities and interfaces with the processor. This approach differs from a so-called bare-metal deployment where the software includes the lowest-level functionalities.

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. One of the most popular open-source RTOS is FreeRTOS, originally design for Internet-of-Things applications and therefore communication-oriented.[28] An alternative is RTEMS[†], that for instance is used on the LEON3 processor, which is the core element of various space on-board computers.[29] It is not rare to find the use of a more general OS, such as Linux, because of the customization and support that they offer. This benefit is normally paid with more resources the system requires to run. IN ELAPSE, the OBCE core software is a C/C++ Linux application. Moreover, the base kernel is aided with a PREEMPT-RT Linux patch to include `pthread` library for 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.[30]

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 Figure 5, is employed. The first loop runs the attitude estimation algorithm, while the second loop executes the attitude control action. Both cycles are designed such that the inclusion of sub-modules designed with the same principles of the OBCE can be included. The unit directly interfaces with the STASIS platform to send attitude control commands and to process the angular rate data sensed by an onboard gyroscope. Moreover, the entire unit connects with the OBCE such that the latter can read the ADCU status and manage the attitude data exchange. Nonetheless, the unit can be used atomically to integrate and compare different control and determination algorithms and hardware drivers.

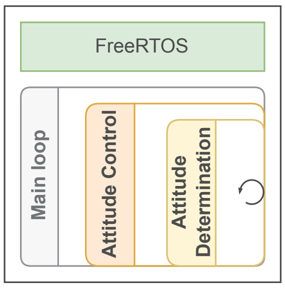


Figure 5 ADCU software architecture.

[*]https://github.com/nanopb/nanopb [last visited January 25, 2024]
[†]https://www.rtems.org/ [last visited January 25, 2024]

The cooperation between the two subsystems results in the modes scheme reported in Figure 6. As can be seen, the ADCU modes are hierarchical to the general OBCE GNC modes. Additionally, transitions are also reported, among their types. However, as introduced before, the current implementation inside the OBCE limits all autonomy levels to an event-based one, that is E3.

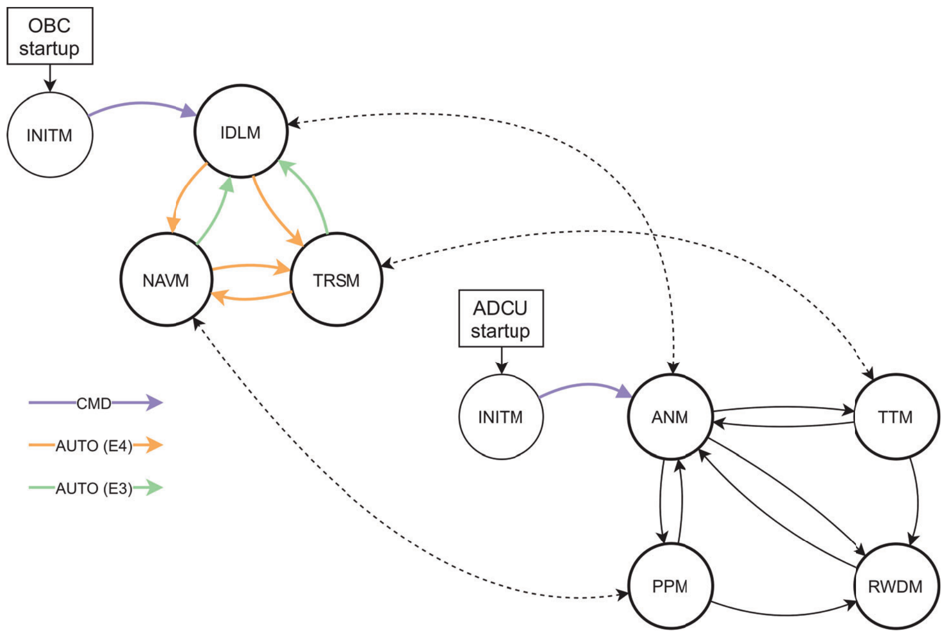


Figure 6 FlatSat OBCE and ADCU modes.

EMBEDDED COMPUTING HARDWARE

The achievement of spacecraft autonomy mandates the development of complex software and consequently requires its deployment on sufficiently high-performance hardware. Nevertheless, deep-space missions, particularly CubeSats, are marked by constrained on-board power, leading to limitations in computing resources. Therefore, a pivotal aspect in the advancement of autonomous GNC algorithms and technology involves ground testing on hardware that accurately emulates on-board characteristics. Hence, the integration of processing systems that faithfully replicate the embedded flight hardware of autonomous systems assumes paramount importance. In this context, the ELAPSE software is augmented by two hosting processing boards designed to fulfill two primary objectives. Firstly, to provide a cost-effective embedded system with restricted computing resources for certifying algorithm maturity. Secondly, to furnish a potent and modular testing platform suitable for diverse validation and verification procedures, capable of executing extensive simulations. Table 2 collects some technical data of processing units currently available for space applications. The products are all based on System-On-Chip (SoC) technology, embodying an integrated electronic device encapsulating diverse components onto a single package. An emerging and well-received configuration entails the integration of a multi-core processing system with Field Programmable Gate Array (FPGA) logic, resulting in a power-efficient and reconfigurable system.[31]

[*]https://kplabs.space/leopard/[last accessed January 25, 2024]
[†]https://xiphos.com/product-details/q7 [last accessed January 25, 2024]
[‡]https://www.satcatalog.com/component/cfc-300/ [last accessed January 25, 2024]

Table 2 Components-Off-the-Shelf spacecraft Data Processing Units.

Board	Vendor	Processor
Leopard DPU[*]	KPLabs	Quad-Core ARM Cortex-A53 1.5GHz
Q7[†]	Xiphos Technologies	Dual-Core ARM Cortex-A9 766MHz
CFC-300[‡]	Innoflight	Dual-Core ARM Cortex-A9 767MHz
Sirius[§]	AAC Clyde Space	Quad-Core LEON4 250MHz
CHP-OBC[¶]	C3S	Quad-Core ARM Cortex-A9 850MHz
NanoMind MK3[‖]	GOMSpace	Dual-Core ARM Cortex-A9 850MHz

However, the utilization of evaluation boards often incurs significant costs, and the associated learning curve is steep. Therefore, as the primary option, the OBCE operates on a Raspberry Pi 4B board[**]. This board, featuring a quad-core ARM processor operating at 1.5GHz and 4GB of Random Access Memory (RAM), is characterized by substantial community support and a straightforward setup. Despite generally exhibiting higher performance than the boards commonly available for deep-space CubeSats and the ones presented, the Raspberry Pi effectively serves as an initial representation of embedded hardware, enhancing the agility of software development and expediting integration within the simulation set-up, as proven by other studies.[32] Additionally, as the baseboard does not include one, it is aided with an external Real-Time Clock (RTC) to precisely keep track of the time.

The ADCU is characterized by a lower computational cost, yet it necessitates meeting stricter real-time deadline constraints to effectively command the attitude hardware. Moreover, the adoption of a second separate board aligns with a modular design philosophy, providing the flexibility to independently update the hardware of the two boards and conduct separate tests. Hence, low-power microcontrollers with modest performance that guarantee adherence to real-time deadlines are preferred. The initial ADCU prototype was implemented on a cost-effective, low-power ESP32 microcontroller[††], interfaced with the OBCE board through the Serial Peripheral Interface (SPI) protocol to minimize data transfer latency. This configuration was employed for testing the interface and subsequently connecting to the gyroscope measuring the angular rate of STASIS platform. A more representative processor was later introduced, specifically a 32-bit Cortex-M4 MPU from the STM32MP157D-DK1[‡‡] board. This 209MHz processor is equipped with a single-precision Floating Point Unit (FPU) and features hard real-time interrupts.

[§]https://www.aac-clyde.space/what-we-do/space-products-components/command-data-handling/sirius-quadcore [last accessed January 25, 2024]

[¶]https://c3s.hu/wp-content/uploads/2022/01/C3S_OBC_datasheet.pdf [last accessed January 25, 2024]

[‖]https://gomspace.com/shop/subsystems/command-and-data-handling/nanomind-hp-mk3-(1).aspx [last accessed January 25, 2024]

[**]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ [last visited January 25, 2024]

[††]https://www.espressif.com/en/products/socs/esp32 [last accessed January 25, 2024]

[‡‡]https://www.st.com/en/evaluation-tools/stm32mp157d-dk1.html [last accessed January 25, 2024]

TESTING CAPABILITIES

As previously introduced, a pivotal functionality of the FlatSat OBC lies in evaluating GNC algorithms to certify their maturity for onboard deployment through a comprehensive verification and validation process. In this trajectory, an incremental approach envisions the utilization of PIL and HIL testing. Typically, this process demands an external set of tools for setup. However, ELAPSE facilitates rapid and continuous testing by incorporating this capability into its implementation. In the following sections, two illustrative examples related to the EXTREMA Pillars I and II testing demonstrate this capability. Figure 7 delineates a comprehensive scheme to set up and perform a PIL simulation employing ELAPSE, emphasizing that, aside from simulation or profiling directives provided as configuration files, the effort is constrained to implementing test runners external to the OBCE and compilation-dependent modules directly within the main software as methods of classes.

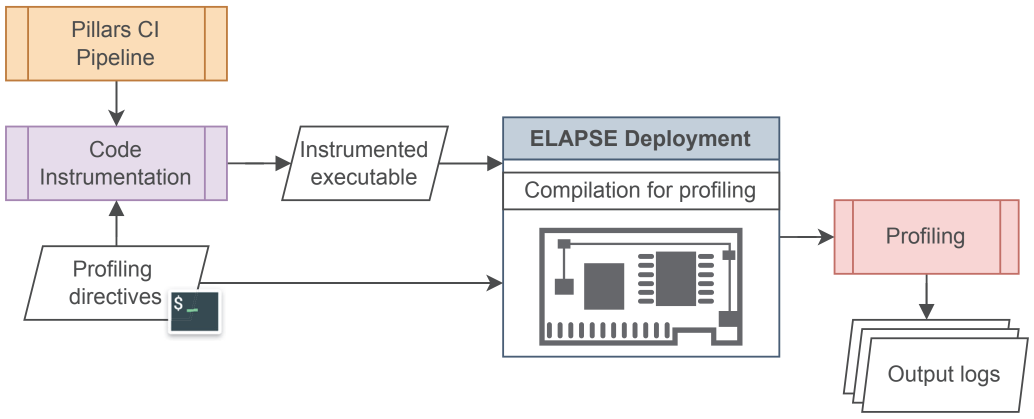


Figure 7 GNC Pillars algorithms profiling general workflow.

Furthermore, the Raspberry Pi 4B multi-core architecture allows for one or multiple cores reserved to run the algorithm under testing and limiting the disturbance of the OS at the minimum. The rationale supporting this approach is rooted in the ability to test the algorithms independently, either in isolation from the OBCE Middleware Layer or by incorporating its functionalities. This is achieved by introducing testing execution modes and compilation preprocessor directives. The latter's role is to provide alternative Run methods to the Task Modules that substitute the base ones to provide optimized and contained testing software.
Furthermore, conducting in-depth profiling necessitates code instrumentation, potentially impacting runtime performance. Consequently, the testing software can be additionally built with different compilation options depending on the type of computational resources measurements required to be tested. All these settings are governed at a high level by a set of scripts, contributing to enabling a systematic and automated approach.

Processor–In–the–Loop simulations

The first pillar of EXTREMA results in a navigation filter that estimates the spacecraft state by processing the deep-space images and detecting the presence of planets, in a way that remembers object triangulation.[5]

Monte Carlo simulations are exploited to validate the optical navigation filter against different images and mission scenarios. Specifically, the trajectory of deep-space scenes is combined with the time series of the NPU operative modes, simulating the optical measurement availability and the duty cycles of an interplanetary cruise.
The PIL workflow adopted is schematically depicted in Figure 8. In detail, two packages are provided to the board. One contains the code generated by the MATLAB Coder, while the other the simulation settings and data to be stored. Mock functions and testing execution paths are created by the developer and the compilation of the software is performed. Once the simulations are started by the aforementioned scripts, every time the camera acquisition is expected, the image is loaded into the Random Access Memory (RAM) to be processed. At the end of the analysis, reports are generated on the board, compressed, and sent back to the host computer to be post-processed.

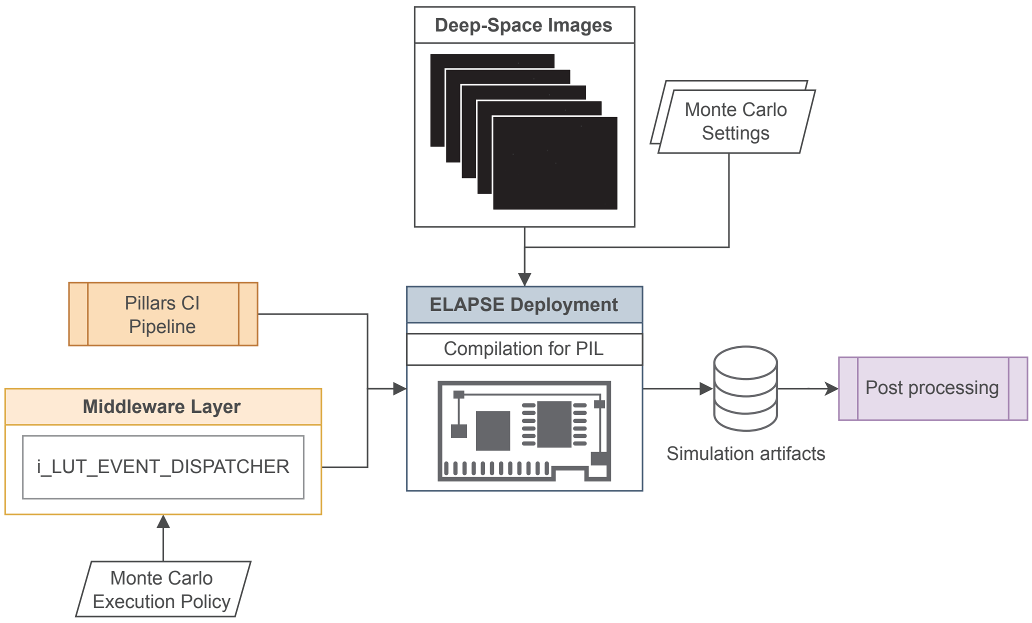


Figure 8 Navigation Processing Unit Monte Carlo simulations workflow.

Processor–In–the–Loop profiling

As previously mentioned, measuring the computational resources of an algorithm in the scenarios it is supposed to be employed is paramount. Thanks to the straightforward exploitation of external operations and memory profiling tools, fully compatible with the OS of the Raspberry Pi 4B board of ELAPSE, such as Valgrind[*] and GNU gprof[†], the aforementioned objective can be achieved. Specifically, Valgrind is used to profile the static, or Stack, and dynamic, or Heap, memory usage, while gprof provides CPU cycles and Instructions estimates.
An in-depth analysis of the time and memory resources consumed by the convex optimization-based guidance algorithm from the EXTREMA Pillar II[33] was performed. Being a direct method, the algorithm resources strongly depend on the number of discretization nodes and the number of

[*]https://valgrind.org/ [last visited January 25, 2024]
[†]https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html [last visited January 25, 2024]

iterations performed. Therefore, characterizing it by profiling it given different scenarios in input is paramount to validate its embedded implementation feasibility and possibly identify areas for improvement.
The workflow previously presented in Figure 7 is followed. A short C++ analysis runner program atomically calls the algorithm such that the overhead of external functions is minimized. Moreover, once the C-sources are generated from the MATLAB code, different compilation options are used to automatically tailor the build to the different profiling analyses. Furthermore, the same policy of processor core reservation used for the first Pillar is employed. Here reported are the results obtained using a batch of 165 scenarios selected from the Zenodo EXTREMA trajectory dataset.[34]
The automatic analysis of reports identifies a predominant use of the Heap over the Stack memory. More in detail, the Heap memory allocation, registered in terms of peaks, is reported in Figure 9 as a function of the number of nodes employed.

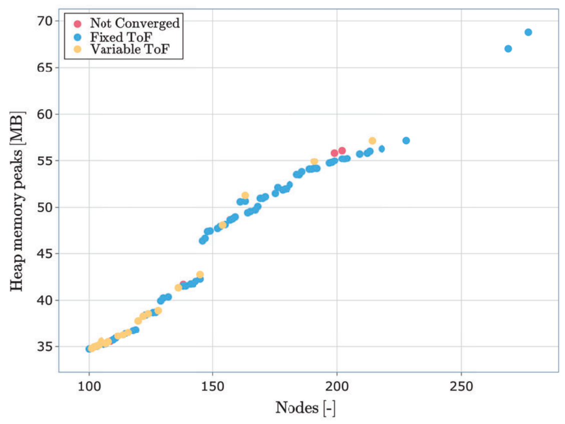


Figure 9 Heap memory peak allocations profiling results.

Furthermore, Figure 10 depicts the CPU time in terms of the number of discretization nodes, showing a quasi-linear dependence. These results are significant as they allow agile and continuous testing of the algorithms that therefore are already deployed on embedded hardware.

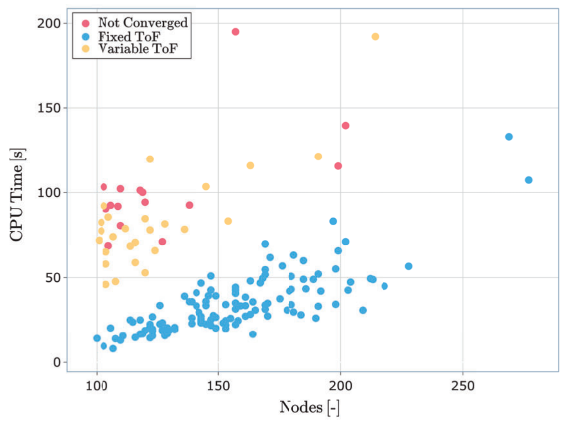


Figure 10 CPU time profiling results.

Hardware–In–the–Loop simulations

The HIL simulation framework represents the last testing stage. The OBCE Hardware Layer contains the Drivers to interface with the different test benches of the ESH. Open-loop simulations are possible by augmenting the baseline ELAPSE system with a monitoring infrastructure and a communication network that synchronizes the events between the different entities involved in the simulation.
Figure 11 schematizes a demonstration of the event-based paradigm of ELAPSE during an open-loop simulation. Specifically, it focuses on the nominal sequence of interactions and actions during a navigation-guidance duty cycle, considering the interplanetary cruise of an autonomous spacecraft.

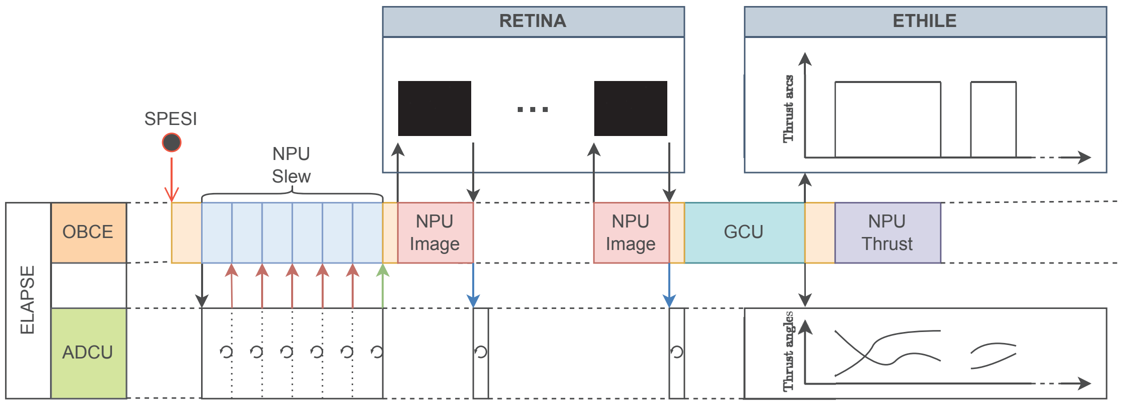
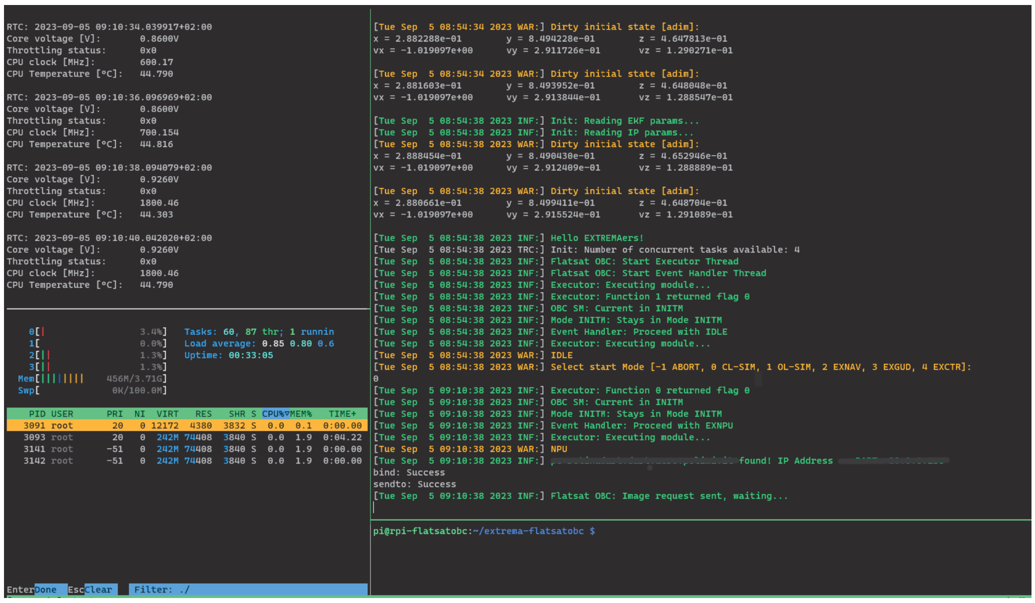


Figure 11 Open-loop HIL events timeline.

The environment numerical propagator SPESI triggers the start point such that the spacecraft processing core and the simulated world are initially synchronized. The transitions between operational phases rely on the reactive policy previously described. In this case, a navigation window with the subsequent observation of two planets is followed by the recomputation of the reference trajectory to reach the final target. The FlatSat OBCE makes a periodic real-time request of the deep-space image to RETINA, the optical facility, and subsequently receives the image taken by the camera. Moreover, upon that request, the ADCU informs the OBCE of the pointing status. Therefore, the latter can pause the optical measurements acquisition and change the NPU operative mode when the attitude falls outside the required pointing requirements. When made available, thrust commands are dispatched to the thruster test bench ETHILE for commissioning, and the pointing trajectory is transferred to ADCU.
Finally, Figure 12 shows a snapshot of the low-level monitoring panel belonging to the control and supervision of ELAPSE. Current software modes and status of operations, and Pillars algorithms results, are printed such that the GNC software procedures can be validated and analyzed in more depth.

CONCLUSION

This work presented the software-hardware architecture of the EXTREMA FlatSat processing system ELAPSE. The computing unit, providing highly modular embedded software, is paramount in integrating, validating, and verifying autonomous GNC algorithms and leveraging closed-loop guidance HIL simulations. Specifically, the system combines flight software that enables the agile



```
RTC: 2023-09-05 09:10:34.039917+02:00
Core voltage [V]:        0.8600V
Throttling status:       0x0
CPU clock [MHz]:         600.17
CPU Temperature [°C]:    44.790

RTC: 2023-09-05 09:10:36.096969+02:00
Core voltage [V]:        0.8600V
Throttling status:       0x0
CPU clock [MHz]:         700.154
CPU Temperature [°C]:    44.816

RTC: 2023-09-05 09:10:38.094079+02:00
Core voltage [V]:        0.9260V
Throttling status:       0x0
CPU clock [MHz]:         1800.46
CPU Temperature [°C]:    44.303

RTC: 2023-09-05 09:10:40.042020+02:00
Core voltage [V]:        0.9260V
Throttling status:       0x0
CPU clock [MHz]:         1800.46
CPU Temperature [°C]:    44.790

    0[|                  3.4%]   Tasks: 60, 87 thr; 1 runnin
    1[                   0.0%]   Load average: 0.85 0.80 0.6
    2[||                 1.3%]   Uptime: 00:33:05
    3[||                 1.3%]
  Mem[|||||||||      456M/3.71G]
  Swp[                0K/100.0M]

    PID USER       PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+
   3091 root        20   0 12172  4380  3832 S  0.0  0.1  0:00.00
   3093 root        20   0  242M 74408  3840 S  0.0  1.9  0:04.22
   3141 root       -51   0  242M 74408  3840 S  0.0  1.9  0:00.00
   3142 root       -51   0  242M 74408  3840 S  0.0  1.9  0:00.00

EnterDone  EscClear   Filter: ./
```

```
[Tue Sep  5 08:54:34 2023 WAR:] Dirty initial state [adim]:
x = 2.882288e-01        y = 8.494228e-01        z = 4.647813e-01
vx = -1.019097e+00      vy = 2.911726e-01       vz = 1.290271e-01

[Tue Sep  5 08:54:34 2023 WAR:] Dirty initial state [adim]:
x = 2.881603e-01        y = 8.493952e-01        z = 4.648048e-01
vx = -1.019097e+00      vy = 2.913844e-01       vz = 1.288547e-01

[Tue Sep  5 08:54:38 2023 INF:] Init: Reading EKF params...
[Tue Sep  5 08:54:38 2023 INF:] Init: Reading IP params...
[Tue Sep  5 08:54:38 2023 WAR:] Dirty initial state [adim]:
x = 2.888454e-01        y = 8.490430e-01        z = 4.652946e-01
vx = -1.019097e+00      vy = 2.912409e-01       vz = 1.288889e-01

[Tue Sep  5 08:54:38 2023 WAR:] Dirty initial state [adim]:
x = 2.880661e-01        y = 8.499411e-01        z = 4.648704e-01
vx = -1.019097e+00      vy = 2.915524e-01       vz = 1.291089e-01

[Tue Sep  5 08:54:38 2023 INF:] Hello EXTREMAers!
[Tue Sep  5 08:54:38 2023 TRC:] Init: Number of concurrent tasks available: 4
[Tue Sep  5 08:54:38 2023 INF:] Flatsat OBC: Start Executor Thread
[Tue Sep  5 08:54:38 2023 INF:] Flatsat OBC: Start Event Handler Thread
[Tue Sep  5 08:54:38 2023 INF:] Executor: Executing module...
[Tue Sep  5 08:54:38 2023 INF:] Executor: Function 1 returned flag 0
[Tue Sep  5 08:54:38 2023 INF:] OBC SM: Current in INITM
[Tue Sep  5 08:54:38 2023 INF:] Mode INITM: Stays in Mode INITM
[Tue Sep  5 08:54:38 2023 INF:] Event Handler: Proceed with IDLE
[Tue Sep  5 08:54:38 2023 INF:] Executor: Executing module...
[Tue Sep  5 08:54:38 2023 WAR:] IDLE
[Tue Sep  5 08:54:38 2023 WAR:] Select start Mode [-1 ABORT, 0 CL-SIM, 1 OL-SIM, 2 EXNAV, 3 EXGUD, 4 EXCTR]:
0
[Tue Sep  5 09:10:38 2023 INF:] Executor: Function 0 returned flag 0
[Tue Sep  5 09:10:38 2023 INF:] OBC SM: Current in INITM
[Tue Sep  5 09:10:38 2023 INF:] Mode INITM: Stays in Mode INITM
[Tue Sep  5 09:10:38 2023 INF:] Event Handler: Proceed with EXNPU
[Tue Sep  5 09:10:38 2023 INF:] Executor: Executing module...
[Tue Sep  5 09:10:38 2023 WAR:] NPU
[Tue Sep  5 09:10:38 2023 INF:] [redacted] found! IP Address [redacted]
bind: Success
sendto: Success
[Tue Sep  5 09:10:38 2023 INF:] Flatsat OBC: Image request sent, waiting...

pi@rpi-flatsatobc:~/extrema-flatsatobc $
```

Figure 12   ELAPSE monitoring panel during an open-loop HIL simulation.

integration of autonomous algorithms with a framework fostering their testing on embedded processors. The peculiarity of the system elements is highlighted as playing a major role in improving the development and testing of the algorithms. To best represent the typical limited CubeSats onboard computational conditions, the software is deployed on embedded processors serving as prototypes for the future adoption of SoC reconfigurable computing technology. Finally, the work demonstrates, via two examples regarding optical navigation and onboard guidance, that the system can methodically perform both PIL and HIL simulations. Furthermore, reactive sequencing of operations achieved by ELAPSE during open-loop simulation is reported as illustrative of its testing capabilities and event-based autonomy, the first step to target goal-oriented autonomy.

ACKNOWLEDGMENT

This research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant Agreement No. 864697).

REFERENCES

[1] E. Kulu, "Nanosatellite Launch Forecasts - Track Record and Latest Prediction," *Small Satellite Conference 2022*, Logan, Utah, USA, 2022.

[2] A. Freeman and C. Norton, "Exploring our Solar System with Cubesats and Nanosats," *Proceedings of the 13th Reinventing Space Conference*, Oxford, UK, 2018. DOI: 10.1007/978-3-319-32817-11.

[3] L. J. Deutsch, P. E. Townes, P. E. Lieberecht, P. A. Vrotsos, and D. M. Cornwell, "Deep Space network: The Next 50 Years.," *In: 14th International Conference on Space Operations*, Daejeon, Korea, 2016. DOI: 10.2514/6.2016-2373.

[4] G. Di Domenico, E. Andreis, A. Carlo Morelli, G. Merisio, V. Franzese, C. Giordano, A. Morselli, P. Panicucci, F. Ferrari, and F. Topputo, "The ERC-Funded EXTREMA Project: Achieving Self-Driving Interplanetary CubeSats," *Modeling and Optimization in Space Engineering: New Concepts and Approaches* (G. Fasano and J. D. Pintér, eds.), pp. 167–199, Springer International Publishing, 2023. DOI: 10.1007/978-3-031-24812-2_6.

[5] E. Andreis, V. Franzese, and F. Topputo, "Flight Software Development, Migration, and Testing in Desktop and Embedded Environments," *Journal of Guidance, Control, and Dynamics*, Vol. 45, No. 8, 2022, pp. 1466–1480. DOI: 10.2514/1.G006294.

[6] M. Gianmario, *Engineering ballistic capture for autonomous interplanetary spacecraft with limited onboard resources*. PhD thesis, Politecnico di Milano, 2023.

[7] A. Morselli, G. Di Domenico, E. Andreis, A. C. Morelli, G. Merisio, V. Franzese, C. Giordano, F. Ferrari, and F. Topputo, "The EXTREMA Orbital Simulation Hub: a Facility for GNC Testing of Autonomous Interplanetary CubeSat," *4S Symposium Proceedings*, Vilamoura, Portugal, 2022.

[8] J. C. E. Barcellos, A. W. Spengler, L. O. Seman, R. D. C. e. Silva, H. P. Roldán, and E. A. Bezerra, "FlatSat Platforms for Small Satellites: A Systematic Mapping and Classification," *IEEE Journal on Miniaturization for Air and Space Systems*, Vol. 4, No. 2, 2023, pp. 186–198. DOI: 10.1109/JMASS.2023.3249044.

[9] ESA Requirements and Standards Division, "Space Segment Operability, ECSS-E-ST-70-11C," stardard, Noordwijk, The Netherlands, July 2008.

[10] M. Troesch, F. Mirza, K. Hughes, A. Rothstein-Dowden, A. Donner, R. Bocchino, M. Feather, B. Smith, L. Fesq, B. Barker, and C. B., "MEXEC: An Onboard Integrated Planning and Execution Approach for Spacecraft Commanding," *30th International Conference on Automated Planning and Scheduling*, Nancy, France, 2020.

[11] R. Amini, L. Fesq, R. Mackley, F. Mirza, R. Rasmussen, M. Troesch, and K. Kolcio, "FRESCO: A Framework for Spacecraft Systems Autonomy," *2021 IEEE Aerospace Conference*, Big Sky, Montana, USA, 2021. DOI: 10.1109/AERO50100.2021.9438470.

[12] P. W. Kenneally, S. Piggott, and H. Schaub, "Basilisk: A Flexible, Scalable and Modular Astrodynamics Simulation Framework," *Journal of Aerospace Information Systems*, Vol. 17, No. 9, 2020, pp. 496–507. DOI: 10.2514/1.I010762.

[13] B. Broekman and N. Edwin, *Testing Embedded Software*. Addison-Wesley, 2003.

[14] P. Panicucci, E. Andreis, V. Franzese, and F. Topputo, "An overview of the EXTREMA deep-space optical navigation experiment," *3rd Space Imaging Workshop*, Atlanta, Georgia, USA, 2022.

[15] A. Morselli, A. C. Morelli, and F. Topputo, "ETHILE: A Thruster-In-the-Loop Facility to Enable Autonomous Guidance and Control for Autonomous Interplanetary Cubesats," *73rd International Astronautical Congress (IAC 2022)*, Paris, France, 2022.

[16] C. Giordano and T. F, "SPESI: A Real-Time Space Environment Simulator for the EXTREMA Project," *33rd AAS/AIAA Space Flight Mechanics Meeting*, Austin, Texas, USA, 2023.

[17] G. Di Domenico and T. F, "STASIS: An Attitude Testbed for Hardware-in-the-Loop Simulations of Autonomous Guidance, Navigation, and Control Systems," *73rd International Astronautical Congress (IAC 2022)*, Paris, France, 2022.

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

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

[20] P. Plasson, C. Cuomo, G. Gabriel, N. Gauthier, L. Gueguen, and L. Malac-Allain, "GERICOS: A Generic Framework for the development of on-board software," *DASIA 2016-Data Systems In Aerospace*, Vol. 736, 2016, p. 39.

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

[22] D. J. F. Miranda, M. Ferreira, F. Kucinskis, and D. McComas, "A comparative survey on flight software frameworks for 'new space'nanosatellite missions," *Journal of Aerospace Technology and Management*, Vol. 11, 2019.

[23] R. Bocchino, T. Canham, G. Watney, L. Reder, and J. Levison, "F Prime: an open-source framework for small-scale flight software systems," *32nd Annual AIAA/USU Conference Small Satellites*, 2018.

[24] C. Coelho, M. Merri, O. Koudelka, and M. Sarkarati, "OPS-SAT Experiments' Software Management with the NanoSat MO Framework," *AIAA SPACE 2016*, 2016. DOI: 10.2514/6.2016-5301.

[25] F. Ingrand and M. Ghallab, "Robotics and artificial intelligence: A perspective on deliberation functions," *AI communications*, Vol. 27, No. 1, 2014, pp. 63–80.

[26] A. C. Morelli, A. Morselli, C. Giordano, and F. Topputo, "Convex Trajectory Optimization Using Thrust Regularization," *Journal of Guidance, Control, and Dynamics*, 2023, pp. 1–8. DOI: 10.2514/1.G007646.

[27] D. Friesel and O. Spinczyk, "Data serialization formats for the internet of things," *Electronic Communications of the EASST*, Vol. 80, 2021. DOI: 10.14279/tuj.eceasst.80.1134.

[28] D. Déharbe, S. Galvão, and A. M. Moreira, "Formalizing FreeRTOS: First Steps," *Formal Methods: Foundations and Applications*, 2009, pp. 101–117. DOI: 10.1007/978-3-642-10452-7_8.

[29] D. Guzman, M. Prieto, S. Sanchez, J. Almena, O. Rodriguez, and D. Meziat, "Improving the LEON spacecraft computer processor for real-time performance analysis," *Journal of Spacecraft and Rockets*, Vol. 48, No. 4, 2011, pp. 671–678.

[30] G. J. Holzmann, “The power of ten–rules for developing safety critical code1,” *Software Technology*, Vol. 10, 2018.
[31] A. D. George and C. M. Wilson, “Onboard Processing With Hybrid and Reconfigurable Computing on Small Satellites,” *Proceedings of the IEEE*, 2018. DOI: 10.1109/JPROC.2018.2802438.
[32] M. Cols Margenet, H. Schaub, and S. Piggott, “Flight Software Development, Migration, and Testing in Desktop and Embedded Environments,” *Journal of Aerospace Information Systems*, Vol. 18, No. 4, 2021, pp. 157–174. DOI: 10.2514/1.I010820.
[33] C. Hofmann, A. C. Morelli, and F. Topputo, “Performance Assessment of Convex Low-Thrust Trajectory Optimization Methods,” *Journal of Spacecraft and Rockets*, Vol. 60, No. 1, 2023, pp. 299–314. DOI: 10.2514/1.A35461.
[34] G. Merisio, “EXTREMA: Random autonomous interplanetary mission scenarios for EXTREMA Simulation Hub (ESH) hardware-in-the-loop simulations,” Aug. 2023. DOI: 10.5281/zenodo.8300632.