ENABLING AI-IN-THE-LOOP AOCS ALGORITHMS ON IN-FLIGHT HARDWARE: FROM CONCEPTION TO IN-ORBIT DEMONSTRATION IN ESA OPS-SAT

Maria Carrillo Barrenechea [(1)], Ilke Karsli Terjan [(1)], Pierre Lachevre [(2)], Mark Watt [(1)], Carlos Hervas Garcia [(3)]

[(1)] Airbus Defence and Space Ltd., Gunnels Wood Rd, Stevenage SG1 2AS (UK), +44 (0)1438 773000, maria.m.carrillo@airbus.com
[(2)] Airbus Defence and Space SAS, 31 Rue des Cosmonautes, 31400 Toulouse (France), +33 5 62 19 62 19, pierre.lachevre@airbus.com
[(3)] Airbus Defence and Space GmbH, Willy-Messerschmitt-Strasse 1 82024 Taufkirchen (Germany), +491758709277, carlos.hervasgarcia@airbus.com

ABSTRACT

There is a current trend and opportunity in technology to apply Artificial Intelligence (AI) and Machine Learning (ML) to improve systems performance. In the space sector, convincing demonstrators of AI/ML technologies have been developed in simulation environments throughout the industry, showing a great potential for AOCS and GNC applications. However, their adoption on spacecraft flight software has yet to happen in a significant manner.
Among some of the existing opportunities for in-orbit demonstrations, the ESA OPS-SAT CubeSat enables quick prototyping of promising algorithms and in-orbit testing. Past experiments have demonstrated that the platform runs AI code properly, which makes it an ideal candidate for testing the Airbus GNC teams' HOPAS AI algorithm. The Hybrid Online Policy Adaptation Strategy (HOPAS) uses a family of AI techniques called Reinforcement Learning in an online fashion to continuously improve the spacecraft attitude control pointing performance. After showing very convincing performance on a Solar Orbiter spacecraft simulator, the algorithm was proposed for OPS-SAT.
This paper presents the development of a testing infrastructure for AI algorithms in a spacecraft with a particular use case in the work developed over 2022 to adapt and port HOPAS to OPS-SAT, as well as the results of the simulator and experiments.

1 INTRODUCTION

Artificial Intelligence (AI) is a breakthrough technology that has improved the efficiency and capability of many technologies across different sectors: from autonomous vehicles and robots to intelligent chatbots, or even prediction of diseases in medicine. AI on board applications are already present in many sectors, yet it still needs to find its way in the aerospace sector. This is due to a number of factors, including the relatively low computing power available on-board traditional spacecraft and the long development process required in the industry [1] [2].
AI has demonstrated to be an enabling tool when it comes to complex problems, improving performance, new technologies and reducing costs. As such, the field has drawn the attention of many aerospace companies and agencies [3], such as the European Space Agency (ESA) [4]. In fact, within Airbus Defence and Space, AI has been successfully conceptually proven on several use

cases. Particularly in the field of Attitude and Orbit Control Systems (AOCS) and Guidance Navigation and Control (GNC), Airbus has developed the AI algorithm called HOPAS, which uses learning based approaches on board to learn online and continuously improve the AOCS performance. This includes reducing the uncertainties on the dynamics and thus fine tuning the control further, improving the robustness vs performance trade-off.

1.1 Previous work on AI for GNC at Airbus

Airbus Defence and Space has been exploring AI applications for space systems via both ESA projects and internal Research and Development (R&D) for several years. The UK GNC team based in Stevenage has been especially active on this topic with projects. Of particular importance to HOPAS is the GNC-v.AI project, an internal R&D initiative aimed at exploring the effectiveness of artificial intelligence techniques to design controllers for complex spacecraft modes. Reinforcement Learning (RL) [5] was used in order to synthetize controller via a large number of simulations and use cases, gaining hands on experience and know-how.

In 2020, some unexpectedly large disturbance was detected on the Solar Orbiter (SolO) spacecraft. Later attributed to unexpected friction instabilities in the reaction wheels, this caused a loss of AOCS performance. The GNC-v.AI team decided to explore what could be done using AI techniques, which lead to the design, prototyping and refining of the HOPAS algorithm. Very promising results were obtained in a simulation environment of SolO, as the HOPAS algorithm was able to deal with a disturbance representative of what was observed on board. Indeed, pointing performance was better than the one obtained using the handmade retuning almost by a factor of 10. Following this success, the HOPAS team decided to explore new opportunities for this algorithm. This paper describes the process to port the algorithm to OPS-SAT during 2022 [6].

1.2 ESA OPS-SAT

OPS-SAT is a 3U CubeSat launched in 2019 and operated by ESA with the goal of demonstrating future mission capabilities enabled by more powerful flight computers. In fact, it is equipped with an ARM based Linux on board computer, the Satellite Experimental Processing Platform (SEPP). ESA has opened various opportunities for experiments to be run on the spacecraft [7]. It has been recently reported by the OPS-SAT team that 200 experiments are registered on the platform [8].

The platform is also equipped with a comprehensive Attitude Determination and Control System (ADCS), the iADCS-100 [9]. For attitude determination, it is equipped with a sun sensor, a 3 axes gyro, a magnetometer, an accelerometer and a star tracker. The star tracker does not function, but the attitude estimation algorithm is still able to provide an estimated quaternion using the other measurements. For control, it is equipped with 6 reaction wheels and 3 magnetorquers.

HOPAS is not only an ADCS algorithm, but it is also an AI algorithm, which generally requires more computational resources. The SEPP is an ideal candidate for its testing. It not only offers computing resources rarely seen on a spacecraft, but it is also able to run more conventional software (the platform is Linux based and is able to run, among others, python and tensorflow). Past experiments have demonstrated that the platform does run AI code properly [10]. Implementing such a control algorithm on an experimental platform like OPS-SAT is especially challenging. The main challenges to resolve are: time cycle management, as the algorithm needs to be executed at a steady frequency whereas retrieval of attitude telemetry (TM) and sending command executes at varying speed; memory budget, as AI applications generally require a larger computing footprint [1]; implement a closed loop attitude controller, since it was not possible to interface with iADCS from the SEPP; and adapting the software and HOPAS to manage online training [11].

1.3 HOPAS

HOPAS is a satellite ADCS algorithm. Its purpose is the same as a more traditional controller: correct a variable of the dynamics of a system to a desired value. The difference with the standard

control engineering is that HOPAS incorporates AI to the general ADCS, in particular Reinforcement Learning, to try to improve the performance of the control and to adapt to different environments. In RL, an agent is tasked with solving a decision making problem (in this case control) while maximising performance as measured by a reward function. The agent's performance is improved over time and/or successive simulations via interactions and trial and error [4].

The Hybrid Online Policy Adaptation Strategy is *Hybrid* in the sense of using both, a Neural Network (NN) architecture and a nominal Control System $K(\theta, \omega)$ (ADCS) to output a torque command T to be applied to the dynamics of the environment, which responds with attitude and rate values (a *state*) that will serve as new input to the hybrid controller NN and $K(\theta, \omega)$ to output a new torque command, or action. This hybrid architecture is shown in the top left block of Figure 1.

The *Online* word refers to the execution of the learning at the same time as the system is in operational phase [11], rather than in a controlled design environment as is often the case with AI applications. The Neural Network is updated and adapted at each time step from observing the output state that its action torque has caused. Here, the learning is performed under the Reinforcement Learning paradigm, and the optimization is performed by an Evolution Strategy based algorithm [12]. This algorithm adapts in an online manner the NN parameters to the model, plant and/or environment dynamics according to the evaluation of the environment answer (state). Hence, the *Adaptation* word in HOPAS refers to the adaptation of the control policy, the NN. A schematic of this general logic is shown in Figure 1.

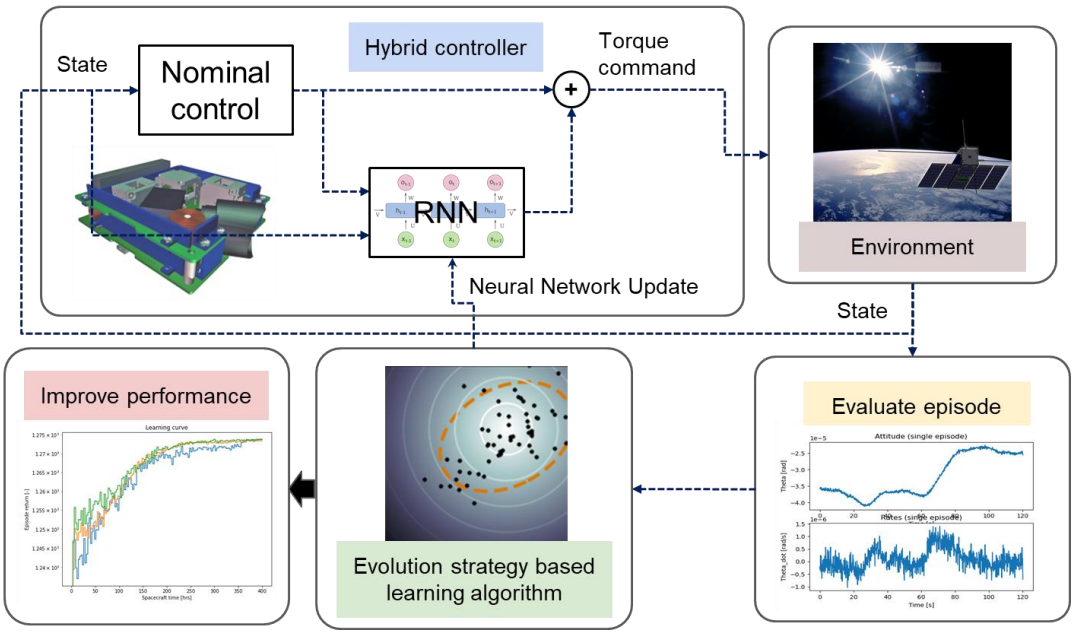

Figure 1. HOPAS algorithm overview

The nominal control is referred to as *baseline controller*. The baseline controller is a core part of the architecture of HOPAS. There are two main reasons for this. First, as the system is learning on board and it needs some time for this at the beginning, at the very start the actions of the system would be completely random, and therefore unsafe. The baseline controller plays that role, and instead of learning a brand new control policy from scratch, HOPAS augments the existing baseline controller to improve performance.

Specific safety and Validation and Verification (V&V) mechanisms were developed for HOPAS, which are out of the scope of this paper. In the OPS-SAT use case, only regular FDIR (Fault Detection Isolation and Recovery) is used. This was deemed sufficient as the experiment can easily be turned off and that the HOPAS specific safety mechanism has not ever been triggered in simulation scenarios (OPS-SAT or SolO use cases).

1.4 Objectives and requirements of the HOPAS on OPS-SAT project

The main objective of this project is the demonstration of the capability to run AI in the loop of an AOCS algorithm in on-board software (OBSW). This paper focuses on the specific use case of the AI algorithm called HOPAS into the ESA in-flight lab OPS-SAT, though general conclusions of this project can be transferred into other potential AI applications for AOCS and GNC.

The HOPAS on OPS-SAT project main objective is the In Orbit Demonstration (IOD) of the AI for AOCS algorithm. In order to do so, several strategic objectives are listed:

- Before anything can be tested in orbit, it shall be encapsulated to be able to be run on board. This in itself is not insignificant, as the step from development ADCS software to flight software is rarely trivial. Doing so with AI in the loop is ambitious and will be an industry first (to the best of our knowledge). The ambition for the project is to use the modern Airbus ADCS code implementation methodology using MATLAB code generation.
- Once the algorithm C code is generated from MATLAB and encapsulated by our software engineering teams, it shall be iteratively validated to make sure all components are working as expected (interfaces, baseline ADCS, AI). Importantly, the team shall make sure the implementation is able to work with the specificities of the OPS-SAT platform.
- Once the two objectives above are completed, an AI performance campaign may be run on board to demonstrate that the algorithm is able to learn during in-flight conditions.

The main objective of the project is to demonstrate that HOPAS is able to learn in-flight, not to evaluate its performance in a realistic scenario. In essence, demonstrate that AI can be run on-board for an AOCS application. Therefore, the experiment is set up in order to put HOPAS in a situation where its effect will be seen clearly and quickly. This fact has effect on several design decisions that will be introduced throughout the paper.

Following a file system corruption issue on the platform in June and the recovery effort that lasted throughout summer 2022, a project extension was agreed on in September 2022. The project was rescoped to focus on objective 2 (code validation), objective 1 having already been achieved at that point in time and objective 3 being unrealistic in the remaining time frame.

Table 1 collects the requirements from an AOCS perspective, software (SW), OPS-SAT and HOPAS team.

Table 1: Requirements

Code	Requirement	Definition
REQ-AOCS-001	Retrieve attitude data	Being HOPAS a closed loop algorithm, the interface between HOPAS and the iADCS shall be able to receive attitude information (quaternion and rates).
REQ-AOCS-002	Send torque commands	Being HOPAS a closed loop algorithm, the interface between HOPAS and the iADCS shall be able to send telecommands (TC) (torque or reaction wheel speed).
REQ-AOCS-003	Constant frequency	The ADCS software shall be run at a constant frequency.
REQ-AOCS-004	Stability	There shall be no loss of pointing stability during execution of the experiment.
REQ-OPSSAT-001	Experiment duration	Due to OPS-SAT operational constraints, the experiment shall not run for more than 4h at a time.
REQ-OPSSAT-002	RAM size limit	RAM memory allocation shall not exceed 256 MB.
REQ-OPSSAT-003	Log file size limit	Log file sizes shall not be more than 10 MB.
REQ-HOPAS-001	Sequenced learning	Derived from REQ-OPSSAT-001. The AI algorithm shall be able to learn in a sequenced manner, pausing the learning and restarting it in the following experiment. This enables effective experiment duration to be greater than 4h.
REQ-HOPAS-002	Total learning time	HOPAS algorithm needs approximately one day to learn, thus, the experiment duration shall be of at least 24 h.
REQ-SW-001	Initialisation	Derived from REQ-HOPAS-001. The harness shall be initialised from configuration file and state file.

REQ-SW-002	Continuity via state file	Derived from REQ-HOPAS-001. The application shall save the necessary data to comply with REQ-HOPAS-001 into state file.
REQ-SW-003	New telemetry flag	Derived from REQ-AOCS-001. Application shall be able to communicate to the harness whether telemetry (TM) information is new.
REQ-SW-004	Constant frequency	Derived from REQ-AOCS-003. The application shall ensure that it runs at a constant frequency.

2 PORTING HOPAS TO ON-BOARD SOFTWARE

OPS-SAT consists of the Satellite Experimental Processing Platform (SEPP), which is a payload on-board computer (OBC) allocated for experiments. The SEPP features a 925 MHz dual-core ARM Cortex A9 Hard Processor System (HPS), 16 GB of flash storage. The SEPP combines a Cyclone V with up to 1 GB of Double Data Rate version 3 (DDR3) Central Processing Unit (CPU) Random-access memory (RAM) with Error Correction Code (ECC) creating a high-bandwidth system for embedded applications [13].

Not only the capability to execute HOPAS AI algorithm on-board is needed, but also the algorithm is required to interact with fine iADCS to retrieve input telemetry and to be able to command torque to the Reaction Wheels (RW) periodically at a fixed frequency. The application that embeds these capabilities is implemented as a standalone process on Linux Operating System (OS), which runs on the SEPP OBC. The SEPP supports several high level programming languages by default (Java, Python, C/C++, etc.). C/C++ language is chosen for performance reasons on-board. To meet the requirements to embed the algorithm on-board, the application consists of two main components: HOPAS Algorithm and Algorithm Driver.

2.1 HOPAS embedded or the "harness"

The inner layer of the application, referred to as the *harness*, contains all the AOCS specific components of the application, including the HOPAS algorithm itself, alongside with some pre- and post-processing for compatibility with the rest of the application. The HOPAS AI algorithm is designed in MATLAB/Simulink, shown in Figure 2. To be able to embed it into a C/C++ application, the auto-code generation option from the Simulink model into C language is used to encapsulate a *harness* containing all of the AOCS related code.

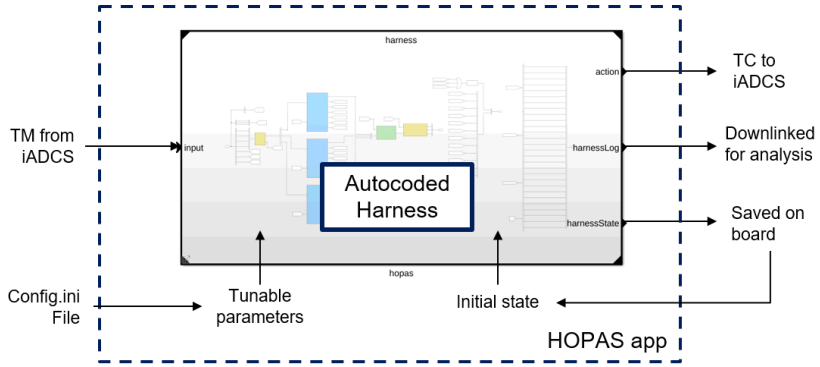

Figure 2. HOPAS harness block in Simulink

The fundamental objective of the harness is to compute a desired *action* (torque command) based on the current *state* of the system (attitude and angular rates). When the full harness is used (v3, see section 3.5), the harness also performs all the functions associated with HOPAS: reward computation, score tracking across episodes, Evolution Strategy algorithm steps and pointing performance monitoring. The harness state (NN parameters) is saved in the state file to comply with REQ-SW-002. All useful intermediate data is also stored in the harness log for analysis.

2.2 Algorithm Driver

The Algorithm Driver is the glue that enables the algorithm on-board. The driver utilises SEPP C++ Application Programming Interface (API) libraries (OPS-SAT development bundle) to interface with the iADCS module. Hence, the driver code is implemented in C++ to interact smoothly with the API library. The driver initialises the SEPP environment modules that the algorithm uses at SW start-up where it sets the epoch time on the iADCS module and enables the required sensors and actuator. Once the Algorithm Driver sets the scene, it starts the cyclic activities to pull input TM using SEPP API library, runs the algorithm and sends the command torque to the reaction wheels.

2.3 Embedding AI into OBSW

The auto-code generation only depends on C Standard Library. The code generated is a pure C SW library agnostic of underlying OS calls (e.g. thread management, semaphore usage, memory allocation, file read/write). The HOPAS Algorithm is quite easy to plug and play.

The challenge has been on the Algorithm Driver in terms of supplying the HOPAS Algorithm inputs from the iADCS module and on providing outputs to the reaction wheels at a steady rate. The implementation of the Algorithm Driver is fortunately not limited by tight CPU load and memory budget on the target platform. Using a standard C++ library sped up the development of the Algorithm Driver instead of dealing with integration into a legacy on-board SW, which provides a limited set of APIs to use with care. The tricky part has been more the integration of target platform's SEPP library to exchange data with the iADCS module. The dynamic behaviour of the SEPP APIs has set certain challenges due to the instability of the I2C bus in the hardware (HW) to communicate with the target module. See section 2.5 for further details.

2.4 Memory budget

The maximum RAM memory allocated by SEPP OBC for the experiment is 256 MB (REQ-OPSSAT-002). Moreover, the HOPAS AI algorithm memory budget has been strictly monitored during development to minimise memory use and to avoid heap memory as much as possible. Static (Compile-time) memory allocation is strongly preferred over dynamic (run-time) memory allocation. However, for some of the cases where the size of the arrays are determined at run-time, heap memory up to 3.29 MB has been used. The NN size is initialised at the first iteration of the harness, such that the size allocated is constant during the whole execution, managing to fit within the available memory allocation. The static memory allocation is described in the following table:

Table 2: HOPAS Algorithm Memory Footprint

Target Arch	Text (bytes)	Data (bytes)	BSS
64-bit intel	40951	198960	0
64-bit arm	30827	198936	0

2.5 In-Orbit Testing Environment

The Algorithm Driver code (without the HOPAS algorithm) has been tested several times in-orbit to ensure the code skeleton (v1, see section 3.3) executes and performs its tasks on the SEPP (v2, see section 3.4). Testing on-board provided useful feedback that led to refactor design of the driver and AI algorithm to cope mainly with performance limitations of the SEPP and its ripple impact on the algorithm. In-orbit tests have shown that the I2C bus that connects the iADCS module to the SEPP OBC did not have enough bandwidth to cope with the cycle frequency (1 Hz) to poll input TM where it is a strict requirement that the algorithm runs at a fixed frequency. The execution time of retrieval of TM has fluctuated significantly (Figure 3) and in some instances, the retrieval took much longer time than the fixed cycle frequency (especially due to retries). To meet the need to run the algorithm at the steady rate (REQ-AOCS-003), The Algorithm Driver code has been refactored into multithreaded design so that the algorithm can be fed with the last available TM without having

to wait for longer execution of input TM retrieval. Hence, running the algorithm and interfacing with the iADCS module are implemented as parallel activities where the algorithm is still running at constant frequency and the Monitor and Command (M&C) interface is running at diverse rates.

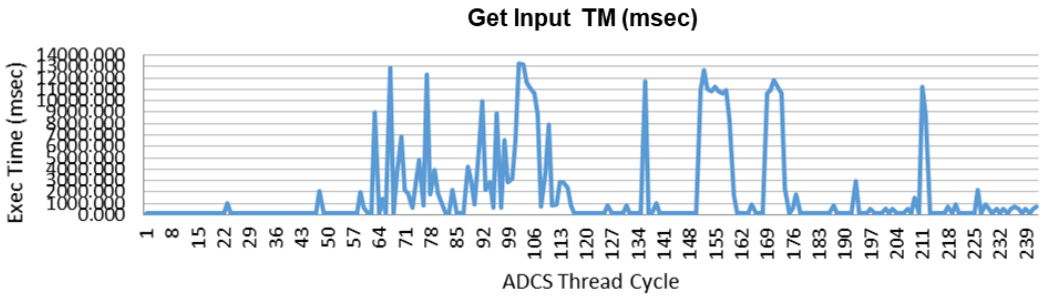

Figure 3. Duration Measurement of Retrieval of Inputs

3 TEST PLAN

3.1 HOPAS learning algorithm evaluation in simulation

In order to evaluate the AI algorithm and the learning performance in the simulator, the tests were divided into 3 different phases, shown in the following Table 3.

Table 3: HOPAS on OPS-SAT algorithm testing phases

Phase	Description
Phase 1: Baseline controller	The OPS-SAT dynamics are controlled by the baseline controller, there is no AI action. This allows the simulation to start and get stabilised dynamics before any NN action is introduced.
Phase 2: Learning phase	The RL algorithm is let to explore and learn.
Phase 3: Frozen network phase	After the smart-mode convergence is achieved, the learning phases ends and the neural network is frozen.

Phase 1 and phase 3 average values are compared to evaluate performance, i.e., comparing the baseline controller to the AI. An interesting point of HOPAS and online learning is that if when flying the AI algorithm a change in the environment is detected, the learning phase can be restarted where the algorithm is able to adapt in a real-time online way to the long-term perturbations or changes in the dynamics. Note that this phase3-phase2 change could also be automated.

3.2 Software test plan

The OPS-SAT platform enables an incremental development approach where successive tests may be performed to test new features. The proposed approach is to test three successive versions, called v1 through v3, each adding more features to the previous version. Before running the tests on OPS-SAT directly, the software should be tested on a representative model on the ground: the Engineering Model (EM). OPS-SAT itself is referred to as the Flight Model (FM).

Table 4: Software test plan summary

Version	Objectives
v1. Interface validation	- Validating that all of the telemetry (TM) interfaces work correctly. - Evaluating the achievable time cycle duration alongside likelihood of telemetries that take longer than usual to come through.

v2. AOCS design validation	- Validating telecommand (TC) interface, in particular to make sure reaction wheels are mapped correctly with the right sign. This will be done by performing an open loop test using a pre-defined torque profile. - Validating the closed loop control tuning and evaluating its performance in order to have a quantified baseline for the AI tests.
v3. AI software validation	- Proper execution of the code over long periods of time (a few hours). - Correct reloading of the internal state of HOPAS in between learning sequences. - No loss of pointing stability during execution of the code.

3.3 Interface Validation (v1)

Before encapsulating the experiment, it is important to validate the code, guaranteeing that it is able to interface properly with OPS-SAT. In the case of HOPAS, the key interface to validate is the interface with iADCS. In the specific case of HOPAS, two strong requirements must be met:

- REQ-AOCS-001 and 002. HOPAS is a closed loop algorithm, meaning it both needs to receive and send commands to the iADCS. This version only tests the reception of the TM data, but their reliability and timeliness are key so that the TC do not get stuck.
- REQ-AOCS-003. The ADCS software needs to be run once every iteration of a software time cycle. Therefore, the v1 code also includes time cycle management utilities.

Note that REQ-AOCS-004 is addressed by the platform internal FDIR system.

3.4 AOCS Design Validation (v2)

As stated in section 1.3, HOPAS requires a baseline controller in the loop. The original idea was to use an existing control mode on iADCS, but it was decided against it for several reasons:

- The exact specifications of the default control mode used by iADCS were not available as it is an off-the-shelf component and not developed by ESA. In fact, the direct use of the gains found in the documentation led to a fairly weak controller, meaning it is likely that even some information is missing.
- There is no existing mechanism to add a corrective torque value to the baseline torque value computed by iADCS. The iADCS modes available are either the control mode which pilots the wheels directly without letting the SEPP interfere, or the measurement mode which provides an estimated attitude that can be read by the SEPP and does not directly send commands to the wheel. This last mode does not compute any control value.
- Lastly, in order for the baseline to be strictly comparable to the hybrid solution proposed by HOPAS, it was deemed preferable to run them both on the same computing hardware (the SEPP), putting both in the same situation with regards to computing resources and latency.

As a consequence, the controller prepared by the HOPAS team needs to be validated on its own before the AI is added in. This is also a good opportunity to quantify baseline performance.

3.5 AI Software Validation (v3)

As stated in section 1.4, AI performance campaigns are out of the scope of the project due to the project re-organisation and extension. The main goal is to validate that learning is occurring, which in practice corresponds to the performance of the AI agent overcoming baseline performance as evaluated during v2 tests.

While HOPAS could theoretically improve performance in a large range of scenarios, it has been demonstrated to work well specifically when a disturbance is acting on the system. It is likely that there will not be any particularly impactful disturbance in the case of OPS-SAT (with the possible exception of hardware issues, such as wheel friction). Therefore, the option to generate an artificial disturbance torque is implemented by sending additional torque signal to the RW, in order to demonstrate that HOPAS is able to compensate it.

4 HOPAS on OPS-SAT: IMPLEMENTATION ADJUSTMENTS

While the HOPAS algorithm is designed to be as generic as possible, some preparation work is still necessary before it can fly. For OPS-SAT in particular, a number of design adjustments were required to ensure compatibility with the platform and iADCS unit. Given the empirical nature of AI algorithms, the tuning efforts need to be performed in a simulation environment. A simulator sufficiently representative of OPS-SAT and its attitude dynamics was therefore developed.

4.1 Reaction wheel interface

The output of the HOPAS algorithm is a command torque value to be passed to the actuators. The SEPP iADCS API did not however provide a function to set a command torque value, only wheel speed commands were available. As such, a functionality was added within the autocoded harness (introduced in section 2.1) to convert torque commands to commanded wheel speed. By the end of 2022 the OPS-SAT team made sending direct torque commands possible. This paper however focuses on the work developed during 2022, which required the RW interface for sending the TC.

4.2 Sequenced learning

Due to OPS-SAT limitation, the HOPAS algorithm can only be run a few hours at a time (3 to 4 hours is expected to be feasible, REQ-OPSSAT-001). As the learning phase 2 normally takes longer, successive segments of learning will be run, turning on and off the learning in a successive manner (REQ-HOPAS-001), which together add up to a single experiment for the application.
The algorithm keeps memory of the last episodes and NN weights values, but while the frozen network is running, the state changes and the learning phase is re-initialised with a different state that it had when the learning was last turned off. This opens the question of how many segments are required for each experiment. In theory, it can be as low as 24 hours of combined training (at least to have observable and undisrupted learning), but depending on time cycle length and reliability of telemetry fetching, it could take longer. In any case, the learning curve will be monitored in between segments to evaluate as early as possible whether learning is occurring as expected.

4.3 Time cycle management and parallelization

Initially it was assumed that the iADCS unit could be polled for Attitude TM @2 Hz where the query would take insignificant time compared to 500 ms. However, EM and FM tests performed using HOPAS V1.x versions (see section 3.3) clearly showed that the retrieving Attitude TM can take up to 13 seconds on its own whereas the harness (the embedded software of HOPAS, see section 2.1) needs to process input data at 2 Hz. As a result, the simple single threaded design had to be refactored into multithreaded design to cope with slower execution time of retrievals from iADCS unit, at the same time to ensure harness runs at 2 Hz even with stale input telemetry.
The first multithreaded design included using one thread to pull input telemetry from iADCS, one thread to run the harness and another thread to command the RW speeds. However, OPS-SAT informed that the SW iADCS class, is not thread-safe. The design was updated to comply with the constraint where the number of worker threads was reduced to two: ADCS Thread and Harness Thread. The ADCS Thread only handles interfacing with the iADCS unit.

4.4 Missed TM events management

Early tests of the application interfaces (see Table 6) showed that it was not possible to get a steady frequency on the TM data. It should be noted that the HOPAS algorithm was designed to operate at a constant frequency, which means it may receive duplicate TM values whenever the call to the API fails or is delayed. In this paper, those events are referred to as “missed telemetry”. In order to better deal with these limitations, some adjustments were made to the harness and the application.

On the application side, due to slower execution time of retrieval of input TM from iADCS unit and faster cycle time of the harness, the application can call the harness-processing function with the same input telemetry several times depending on the length of time to fetch the TM. However, the harness needs to identify whether the given input TM is already processed or not (REQ-SW-003).
On the harness side, the following occurs:

1. On the classical controller side, no specific issues are expected. Since in the default case only proportional and derivative gains are used, the classical controller will simply output the same command value for several steps in a row. Realistically, a missed TM event means that the I2C is stuck and therefore the torque command is not passed directly to iADCS.
2. On the AI side, two elements should be considered: (a) The reward function uses current TM, and therefore episode score may be altered on episodes with many missed events. (b) The neural network itself uses current TM as an input but also has internal memory, therefore the output for the different time steps with the same TM will be different, since the internal state of the neural network will still change over time. It is hard to predict the exact consequences of this it is not possible to anticipate what behaviour the neural network will learn in this scenario. This is definitely something to watch out for however.
3. The last component affected by the missed TM is the conversion from torque command to wheel command. An integrator needs to be used to convert desired torque to desired wheel speed. In the simplest case where only the classical controller is in the loop, a constant value is integrated over the time. Once the situation is resolved, a new commanded wheel value is passed to the iADCS with potentially a large difference from the current value. This will generally lead to a transient in attitude behaviour with a potentially large attitude error.

Because of 2a and especially 3, it was decided to implement a very basic safety mechanism to anticipate this issue. This is achieved by adding a Boolean specifying if the telemetry is new to the input bus and using it in two places:

- In all cases, when the telemetry is not new, the torque passed to the torque to wheel speed conversion blocked is set to 0. This prevents the integrator from building up a large delta on the wheel command. Nothing is lost by doing this since the commands being generated are not relevant for the new state of the system (which is unknown to the harness).
- Even though the output of the NN will not be used, an internal step of HOPAS needs to be taken to keep the clock and NN memory updated. For this, the latest TM is used alongside a reward forced to 0. To avoid a large effect on the learning process, the total return collected each episode is normalized by the number of valid time steps during the episode.

The efficacy of the first element was verified in simulation, using only the baseline controller in this case, in Figure 4. On the left, no mitigation is used. Transients are observed when the numerous consecutive missed TM events occur (for instance around 1500 s, when a maximum error of over 2 degrees is seen). On the other hand, once the mitigation solution described above is implemented (see right), the transient behaviour is significantly less pronounced.

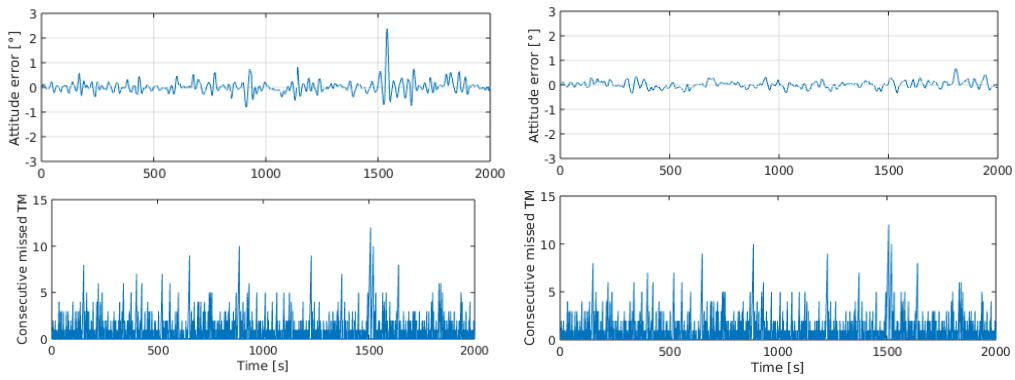

Figure 4. Visualisation of the effect of the missed TM mitigation solution

5 RESULTS

This section presents the HOPAS on OPS-SAT simulator results and HOPAS software validation tests, both on the EM and the FM (summarized in Table 5). It is not possible to test iADCS related functionality of HOPAS on the EM since iADCS unit on the EM is not fully representative of the same unit on FM: (a) Quaternions, which HOPAS uses as input to harness, are received from SEPP iADCS API as 0, which is not physically possible, thus they are not produced on the EM. (b) The iADCS unit on EM does not have the 3 RWs, there is only 1 RW. RW speeds retrieved as input to the harness are either 0 or random values and they do not originate from iADCS unit. The HOPAS application cannot thus command to the missing RWs. (c) The EM is also known to have degraded I2C bus performance compared to the spacecraft.

Table 5: HOPAS software validation tests summary

HOPAS release date	HOPAS Version	Dev. SEPP API Lib Version	EM	FM	EM Test Date	FM Test Date
01/06/2022	v1.0.0	sepp-api-da138de0	X	-	02/06/2022	
07/06/2022	v1.1.0	sepp-api-da138de0	X	-	07/06/2022	
27/06/2022	v1.2.0	sepp-api-b6da662e	X	-	29/06/2022	
01/07/2022	v1.3.0	sepp-api-b6da662e	X	-	05/07/2022	
27/07/2022	v1.4.0	sepp-api-b6da662e	X	X	28/07/2022	11/09/2022
25/10/2022	v1.5.0	sepp-api-b6da662e	X	X	25/10/2022	28/10/2022 11/09/2022
07/10/2022	v2.0.0	sepp-api-b6da662e	X	-	07/11/2022	
08/11/2022	v2.1.0	sepp-api-b6da662e	X	-	09/11/2022	
09/11/2022	v2.2.0	sepp-api-b6da662e	X	-		
09/11/2022	v2.2.1	sepp-api-b6da662e	X	X	10/11/2022	25/11/2022 26/11/2022 27/11/2022
30/11/2022	v2.3.0	sepp-api-b6da662e	X	X	09/12/2022	07/04/2023

5.1 HOPAS on OPS-SAT Simulator results

Figure 5 shows the attitude and rate mean errors during the three different phases of the same test performed with continuous and sequenced learning, the performance metrics oscillate between *in training* performance and *frozen* performance. It can be seen that while the agent is learning, the deviation is high and the values are very variable, as the agent is exploring, till the smart mode converges and the dynamics stabilise. It should be noted that the baseline controller tuning here is quite degraded, and so, the performance of the baseline controller is improved by reducing the attitude error by 0.0347 deg and the rate error by 0.0086 deg/s when comparing to the frozen NN case. The results are similar to the non-sequenced case, even better for the attitude sequenced learning case (0.0647 deg and 0.0070 deg/s), although this training lasts longer to converge (146 h for the continuous and 198 h for the sequenced one without taking into account the 4 h breaks). This test was done to demonstrate the feasibility to pause the training, meeting REQ-HOPAS-001.
Figure 6 shows the results with the final hyperparameters tuning with a standard performance controller tuning for the baseline controller. The average of the values during the 1st phase (baseline controller) and 3rd phase (NN frozen) is also shown in dashed lines. It can be seen that the performance of the baseline controller is improved by 0.0116 deg in attitude and 0.0002 deg/s in rate. Note that the learning takes 150 h because of how the smart mode convergence is tuned, but the learning can be achieved in less than 1 day. As stated in the objectives (section 1.4), the goal for this experiment is to demonstrate that the algorithm learns, rather than improve performance.

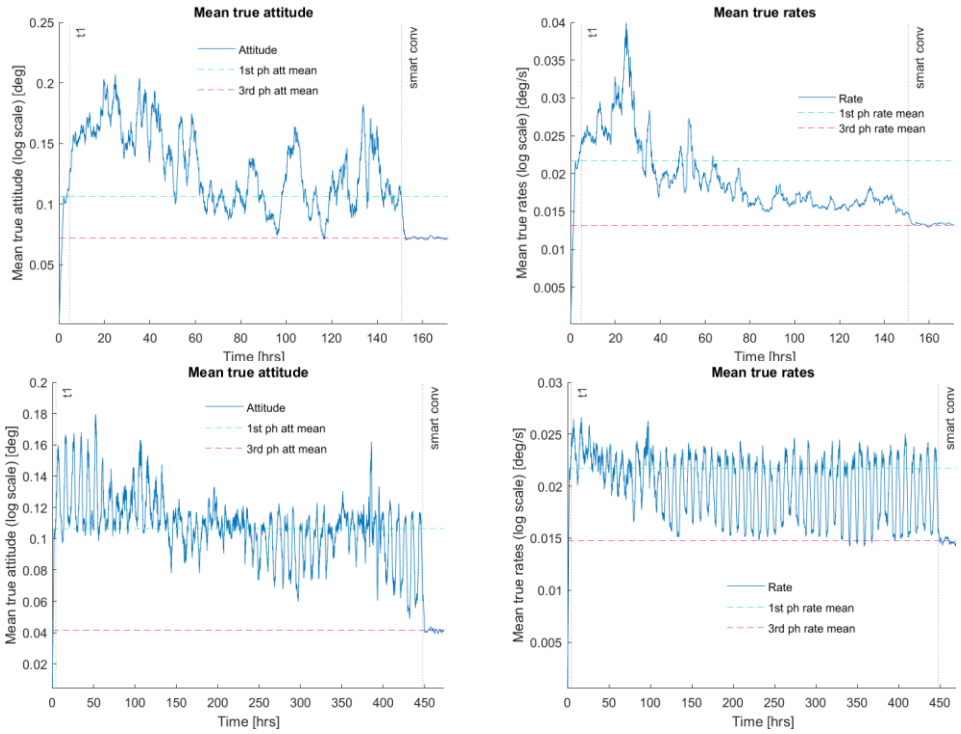

Figure 5. HOPAS on OPS-SAT simulator results: continuous (up) and sequenced (down) training

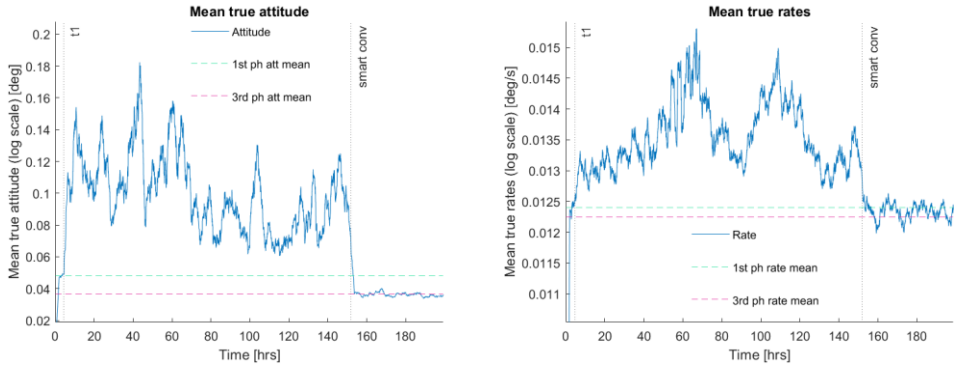

Figure 6. HOPAS on OPS-SAT simulator results with nominal baseline controller

5.2 EM tests

Despite the given limitations on the EM facility, the EM has been used to verify the following:

- HOPAS retrieves TM from iADCS unit over I2C bus periodically (REQ-SW-004).
- HOPAS initialises the SEPP and iADCS environment needs correctly (REQ-SW-001).
- The HOPAS application terminates itself once the experiment duration expires.
- HOPAS does no harm to the system.
- Compressed log file sizes during experiment duration of 3 hours are under 10 MB (REQ-OPSSAT-003).

EM tests have also been useful to get an idea about execution time to retrieve attitude TM and RW speeds. Measuring varying length of durations has led to redesign of application as multi-threaded.

5.3 V1 FM tests

As already introduced in Table 5, only v1.4.0 and v.1.5.0 could be tested on FM thanks to availability of the satellite. The latest FM test campaign performed on v1.5.0 is discussed in this section, as it includes the most significant change: multi-threaded design. In fact, the results received from v1.4.0 FM and EM tests steered the implementation to multithreaded design.

Table 6: v.1.5.0 experiment analysis

Experiment duration	3 h
Average Execution Time (ET)	~ 1 s
Maximum ET	13.35 s
Analytics on ET	72% < 0.5 s, and 17% > 2 s (% of cycles)
Other observations	- 12.64% of the time, SEPP iADCS API failure - I2C Bus in FAILED state permanently for the last hour

5.4 V2 feedforward tests

This open-loop test of version 2 (see section 3.4) involves sending open-loop wheel speed commands to slew the spacecraft about each body axis sequentially. The open-loop wheel speed command is shown in Figure 7 left plot. Note that wheels 0, 1 and 2 do not correspond to the x, y and z body axes due to the orientation of the wheels in the body-frame: wheel1 (red) is aligned with the body x-axis, wheel0 (blue with the y-axis and wheel2 (yellow) with the z-axis. Therefore, the expected behaviour during this open-loop test is to slew about the x, y and z axes consecutively.

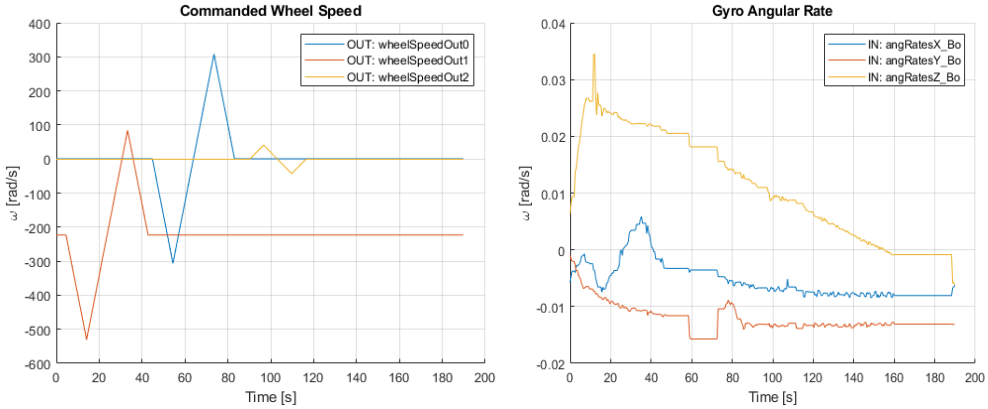

Figure 7. Open-loop test: commanded wheel speed (left) and angular rate (right)

Figure 7 right plot shows the body rates during the open-loop test. In general, it appears that the order of slews (x, y then z) are performed successfully, although the quality of the test results does not allow full confirmation of this. The most prominent evidence of a successful slew is shown on the x-axis which shows clear changes in the rate between 0s and 40s. This test was useful as it identified a polarity error in the harness which was subsequently corrected.

The test starts with a non-zero rate. An unexpected result is that the rates drift significantly when there is no wheel command applied. For example, the change from ~-0.002rad/s to ~-0.012rad/s @ 50seconds on the y-axis rate and the steady decrease in the z-axis rate during the entire test (along with an unexplained transient in z-rate at the start of the test). This drift behaviour will be investigated further prior to performing the closed loop tests.

To conclude, this test does not represent a particularly 'clean' result, partly due to missing data during the y-axis slew; therefore, an additional FM open-loop test has been planned. The open-loop tests done during 2023 have an additional idle phase before and after the slews in order to allow the spacecraft to settle and avoid polluting the results with any initial transients.

6 CONCLUSIONS AND FUTURE WORK

The HOPAS team was the first to attempt implementing an alternative closed loop controller on OPS-SAT (and hence one of the firsts to interface continuously with iADCS, since iADCS itself is not reprogrammable). This was a challenge from the start, and several unforeseen hurdles had to be

dealt with throughout the project.
The core achievements of the team include implementing and autocoding the HOPAS algorithm, creating a parallelized application able to interface with the iADCS while running the harness at a constant frequency and validating the ability to both receive telemetry and pass commands computed in the autocoded harness. The team has been able to adapt to the specifics and constraints of the OPS-SAT platform to reach these goals.
One key takeaway from the project is that having access to such an experimental board was immensely beneficial. A reprogrammable board with such an architecture offers an amount of flexibility almost never seen in the space industry and enables rapid iteration and testing. The fact that such experimental boards are being considered for more missions is extremely promising and should accelerate software development tremendously.
However, the challenges related to developing an AOCS algorithm on a computing device separated from the specialized ADCS hardware are not negligible and were somewhat underestimated by the team. In particular, managing a steady time cycle and building a reliable interface were challenging, and some OPS-SAT related issues further complicated the topic. This alongside other OPS-SAT issues (such as a segmentation fault) delayed the schedule, meaning the experimental campaign is not complete yet. In particular, closed loop control of the platform and on orbit execution of the AI algorithm have yet to be achieved.
Regardless, many lessons learned were drawn, and can be extrapolated for other AI application in the AOCS of a flying software:

- The team maintained a philosophy of keeping resource footprint of the application low throughout the project. Indeed, while state of the art methods often advertise large networks, some use cases can be solved with small networks and streamlined learning algorithms. This simplified on board executions, and proved especially useful when available RAM and storage were reduced following the memory issues over the summer. No adjustments were required at that time.
- This project has shown that validating the interfaces and application requirements should be done as early as possible in the project. The HOPAS software was de-risked in the simulator before the interfaces and I2C could be (v1 tests). It is recommended that de-risking of the interface should be an absolute priority for future similar IOD experiments as this is where most unexpected issues arose in the case of this project. It should be noted that this was the OPS-SAT team's recommendation, which could not be followed due to schedule conflict.
- IOD opportunities might differ greatly from what the algorithms to be tested were designed to operate on. For instance, OPS-SAT is very different from SolO, which was the reference use case during HOPAS prototyping. However, this does not prevent validating the algorithm principle, and since software is being considered, some elements can be simulated on-board to some extent (fake failures for FDIR, software driven disturbances for control, etc.). Overall, IOD design decisions should be made to maximise the chances of the SW working on the platform and for the application to demonstrate appropriate behaviour.
- Adjusting to the specifics of the IOD platform should be expected to generate additional complexity regardless of the platform, and adequate time should be allocated for it.
- Engineering model tests can save time even if not fully representative (the OPS-SAT EM did not have representative iADCS hardware).
- Trial and error approaches are possible, in particular in the environment OPS-SAT enables. It is however very challenging and requires quick adaptation and the ability to precisely diagnose the issues that arise.

For the future work, the HOPAS team is looking forward to running AI on-board. The code is ready, pending any possible small adaptations depending on the results of the feedforward and feedback v2 tests. Performance campaigns for HOPAS may be run after proving the flying software

to complete all project objectives and demonstrate the ability to train a neural network in real time on a platform such as OPS-SAT or any other opportunities the HOPAS team may have for IOD.
In any case, the HOPAS on OPS-SAT campaign was a constructive example of the potential of experimental computing payloads on flight models, and could pave the way for on board AI applications.

7 REFERENCES

[1] G. Furano et al., *Towards the Use of Artificial Intelligence on the Edge in Space Systems: Challenges and Opportunities*, IEEE Aerospace and Electronic Systems Magazine, Vol. 35, No. 12, 44-56, 2020.

[2] V. Leon, G. Lentaris, D. Soudris, S. Vellas and M. Bernou, *Towards Employing FPGA and ASIP Acceleration to Enable Onboard AI/ML in Space Applications*, 2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC), 1-4, Patras, Greece, 2022.

[3] D. Izzo, M. Märtens and B. Pan, *A survey on artificial intelligence trends in spacecraft guidance dynamics and control*, Astrodynamics, Vol. 3, 287-299, 2019.

[4] ESA, *Artificial intelligence in space*, 31 March 2022, https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/Artificial_intelligence_in_space.

[5] Sutton, Richard S., and Andrew G. Barto. *Introduction to reinforcement learning*, Cambridge: MIT press, Vol. 2. No. 4, 1998.

[6] P. Lachevre, I.K. Terjan, M. Carrillo Barrenechea, M. Watt and C. Hervas Garcia, *HOPAS (Hybrid online policy adaptations strategy) on OPS-SAT*, ESA Nebula Public Library, 30 January 2023, https://nebula.esa.int/content/hopas-hybrid-online-policy-adaptations-strategy-ops-sat.

[7] M. Henkel, P. Romano and R. Zeif, *OPS-SAT Phase B2/C/D/E1 Experimenter ICD*, Issue 0.5, 2016.

[8] O. P. Vasilakis, *ESA OPS-SAT News*, 3 November 2022, https://opssat1.esoc.esa.int/news/52.

[9] Berlin Space Technologies GmbH, *OPS-SAT iADCS-100 ICD*, Issue 3.4.0, 2019.

[10] ESA, *Interplanetary internet & cameras in space: ESA's OPS-SAT first results*, 18 December 2020, https://www.esa.int/Enabling_Support/Operations/Interplanetary_internet_cameras_in_space_ESA_s_OPS-SAT_first_results.

[11] S. C.H. Hoi, D. Sahoo, J. Lu and P. Zhao, *Online learning: A comprehensive survey*, Neurocomputing, Vol. 459, 249-289, 2021.

[12] H.G. Beyer and H.P. Schwefel, *Evolution strategies – A comprehensive introduction*, Natural Computing, Vol. 1, 3-52, 2002.

[13] ESA OPS-SAT, *SEPP C++ API Library Documentation*, Issue 1.0, 2022.