→DISCOVERY

DEFENCE AND SPACE

HOPAS on OPS-SAT
Executive Summary

1 CONTEXT

While AI technologies have shown promise in various applications in the space industry (see Dario Izzo's review for instance[1]), their adoption on spacecraft flight software has yet to happen in a significant manner. This is due to a number of factors, including the relatively low computing power available on board of traditional spacecrafts, the long development process required in the industry and the lack of opportunities for in-orbit demonstrations. Airbus teams have identified in ESA OPS-SAT a unique opportunity to quickly test promising algorithms in-orbit as it seeks to resolve the issues listed above.

As part of the 2021 idea campaign for OPS-SAT experiments, Airbus teams have submitted the Hybrid Online Policy Adaptation Strategy (HOPAS) concept. HOPAS uses an AI technique called Reinforcement Learning (RL) in an online fashion in order to continuously improve the attitude control performance of a space system and adapt to its environment. It has been prototyped in a representative simulator of the Solar Orbiter spacecraft and it has shown promising performance. This led to an activity that lasted throughout 2022 to adapt the algorithm to the specificities of OPS-SAT, summarised in this document.

[1] Izzo, D., Märtens, M., & Pan, B. (2019). A survey on artificial intelligence trends in spacecraft guidance dynamics and control. *Astrodynamics*, *3*(4), 287-299.

Publishing Date: 7 December 2022
Contract Number: 4000137215/22/NL/GLC/ov
Implemented as ESA OSIP

2 HOPAS

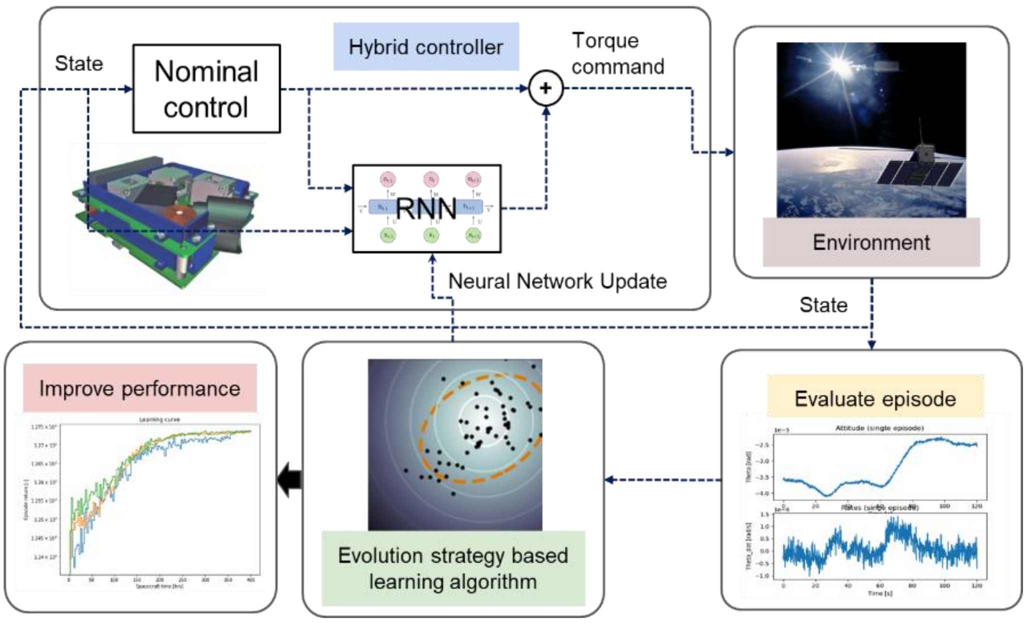

Figure 2-1 HOPAS Algorithm Summary

HOPAS is a closed loop attitude control algorithm, meaning that its role is to compute an appropriate torque command based on measured attitude in order to keep the satellite steadily pointed in the desired direction. Its specificity is that it uses an artificial intelligence technique called reinforcement learning (RL) in order to continuously improve the pointing performance (or in other words, the pointing accuracy) in real time. The guiding principle of RL is the idea of learning by trial and error. The algorithm introduces random variations to the control strategy and learns based on what generates the best result, measured by a *reward* score. Many RL algorithms exist, the one used for HOPAS is inspired by Salimans et al[2], although numerous changes were performed by the Airbus team, in particular to allow it to operate in an online fashion. A key design element in this paradigm is that *evolution strategies* are used as part of the learning algorithm, meaning learning occurs in an episodic fashion. A learning step is taken every *generation* of a given number of *episodes*. The control strategies that yielded the best episodes are used to compute the next generation of episodes.

Another important specificity of HOPAS is that it is hybrid, meaning that its architecture includes both a classical controller and an AI agent to improve the performance the classical controller alone would achieve (as opposed to using only a neural network to compute the control value). This serves several purposes. Crucially, it makes sure the HOPAS system as a whole is able to control the spacecraft right from the start, which would not be possible if only

[2] Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864.

the neural network was used without pre-training. Additionally, it enables validating the architecture as a whole, as detailed in the technical description of HOPAS.

This algorithm is undergoing a patent process.

3 REQUIREMENTS

Implementing such a control algorithm on an experimental platform like HOPAS is especially challenging. Indeed, the control system is always a critical part for any spacecraft mission, and its design is already challenging. This is true even without AI in the loop, and in fact a majority of the tasks performed during the project were related to setting up a functional AOCS architecture for OPS-SAT that could run on the SEPP (ie the Satellite Experimental Processing Platform, the on board computer on which the experiment will run). Some of the challenges to be resolved include:

- The algorithm needs to be executed continuously over time rather than a single time over the mission. In other words, it needs to be executed at every iteration of a software time cycle. In the specific case of OPS-SAT, HOPAS was run on the SEPP computing hardware, which is separate from the attitude control hardware. This meant time cycle management had to be implemented by the team.
- Furthermore, as a feedback control algorithm, HOPAS requires both receiving attitude telemetry from the iADCS hardware and sending commands to it. Both should happen at a steady frequency on the order of 1Hz.
- AI applications generally require a larger computing footprint. In the case of HOPAS, it is without doubt larger than a classical control algorithm. However, from the very start of the project, specific attention was given to making sure the algorithm was as light as possible in terms of processing and memory requirements. This was achieved by using a small neural network and by simplifying the evolutionary strategy used in the learning algorithm as much as possible without compromising performance.
- Even though HOPAS was optimized to learn relatively quickly, it can still take over a day of in flight time to learn. This can be challenging on a platform like OPS-SAT, and therefore a sequenced training approach, allowing the learning to take place over several phases, was introduced.

Publishing Date: 7 December 2022
Contract Number: 4000137215/22/NL/GLC/ov
Implemented as ESA OSIP

4 KEY IMPLEMENTATION DETAILS

4.1 Architecture

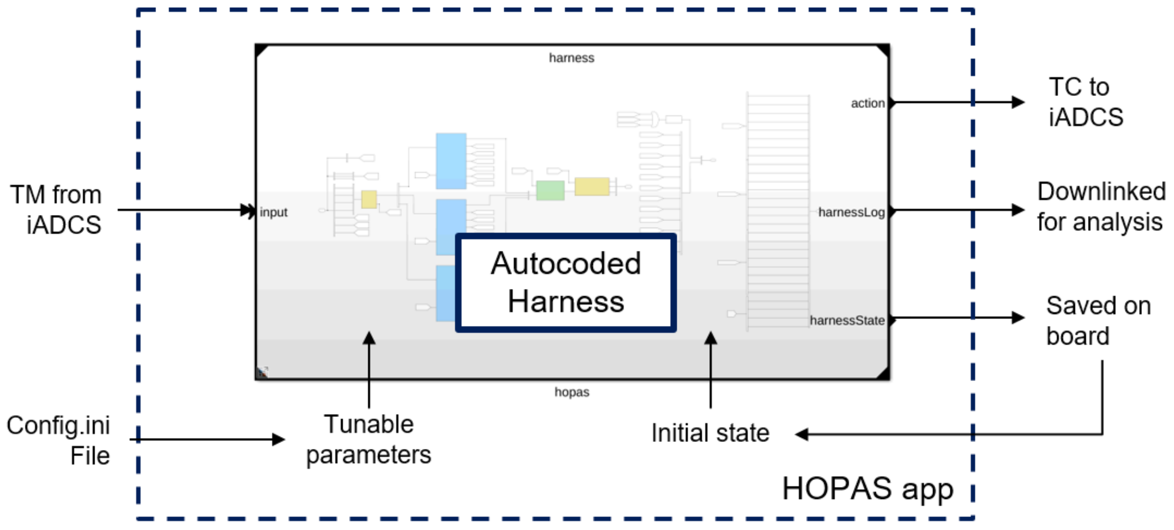

Figure 4-1 HOPAS Software Architecture Summary

The Airbus teams have designed solutions to all of the problems listed above in order to maximise the chances of the algorithm successfully running on OPS-SAT. While the implementation is explained in greater detail in the complete report, some details are provided here.

State of the art AOCS development process uses code generation (sometimes also referred to as *autocoding*). This allows domain engineers to develop and tune their algorithm in specialised simulation environments (in the case of this project MATLAB/Simulink) and then generate C code that corresponds to the same algorithm. This process was used for HOPAS. The AI algorithm was implemented and autocoded in MATLAB by the AI/AOCS team, and the code generated, referred to as “the harness”, was then passed to the software engineering team.

The software engineering team then encapsulated this *harness* in the overall HOPAS application. One of the key roles of the application was to manage time cycles, as introduced previously. In order to isolate the HOPAS algorithm from the execution of API calls to the ADCS hardware, a multi-threaded design was chosen. This meant long duration calls to the iADCS API would not cause fluctuations on the strict cycle period to run HOPAS algorithm.

4.2 Tuning and Simulator

While HOPAS was designed to be mostly generic, it still requires some tuning to adapt it to the OPS-SAT platform. AI tuning is generally empirical, therefore having a simulator is a requirement to be able to perform the tuning. Since no OPS-SAT simulator was readily available, the team put one together attempting to be representative of the main performance drivers. This includes platform inertia, measurement noise (tuned to telemetry data) and actuator characteristics.

As in-orbit tests were performed, the simulator was further adjusted to better represent what was observed. In particular, software frequency was adjusted, sequenced training was implemented and the I2C performance issues were modelled. This allowed the team to fine tune HOPAS to maximise its ability to learn on board of OPS-SAT, and how fast it could manage to do so.

5 ITERATIVE TESTING APPROACH

As HOPAS required the implementation of numerous critical elements, they were tested on board in an iterative manner. First, "v1" was built to test the API and the time cycle management. Then, v2 adds the harness and tests all basic control functions (telecommands and closed loop functionalities). Finally, v3 adds the AI in the loop.

5.1 Testing 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 09/11/2022
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	
09/11/2022	v2.2.1	sepp-api-b6da662e	X	X	10/11/2022	25/11/2022 26/11/2022 27/11/2022

Table 5-1 Summary of software version tested on EM or FM

The table above summarises the tests performed on the engineering model (EM or FlatSat) and on the flight model (FM, in other words OPS-SAT itself).

It should be noted that following issues on the OPS-SAT platform caused delays in the schedule:

Publishing Date: 7 December 2022
Contract Number: 4000137215/22/NL/GLC/ov
Implemented as ESA OSIP

→ THE EUROPEAN SPACE AGENCY
ESA Discovery & Preparation
From breakthrough ideas to mission feasibility. Discovery & Preparation is laying the groundwork for the future of space in Europe
Learn more on www.esa.int/discovery
Deliverables published on https://nebula.esa.int

5

- A file system corruption issue occurred in late June, causing the v1 FM tests (skeleton app, see 5.2) to be delayed until September. The development was therefore paused and resumed in October.
- Further issues arose in November related to the ground station, delaying the v2 tests (see 5.3) to the end of the month.
- New file system corruption errors occurred at the start of December, preventing further v2 tests from being run.

5.2 Telemetry and time cycle tests

The first series of tests, referred to as v1, were meant to validate the general execution of the code and the telemetry (TM) interfaces. Only a so-called skeleton app was uploaded to the spacecraft, which simply queried for telemetry at a desired rate.

These tests led to the identification of some performance limitations on the interface between the iADCS and the SEPP. On many occasions, queries to the iADCS have failed and/or have taken long durations of several seconds at a time, even though at this stage the target time cycle was 500 ms.

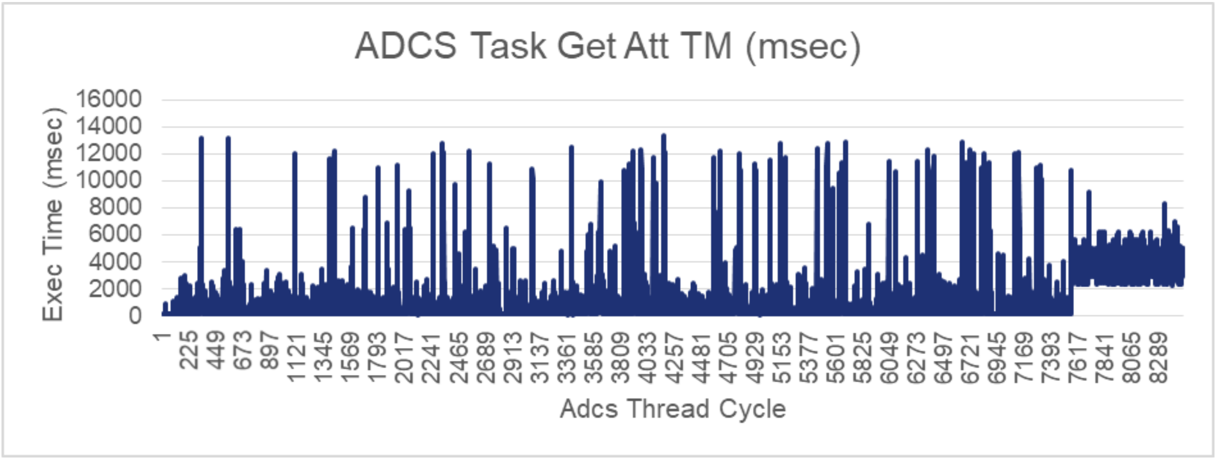

Figure 5-1 Software cycle duration over time

The figure above shows the effective duration of the execution of the API to request TM data from the iADCS (from the v1.5.0 FM tests). The target value is on the order of 500 ms, but the duration of the call may sometimes reach over 10 seconds. Only about 80% of the calls take less than 1000 ms.

A list of mitigation solutions were prepared and are being rolled out at successive tests. This includes moving to a multithreaded design (which is the difference between v1.4 and v1.5), reducing the frequency of the on board software, and reviewing the calls to the iADCS.

Even though the performance of the I2C improves with each test thanks to the mitigation solutions implemented by the team, it is important to remember that this issue could compromise the success of the application. First of all, when the telemetry API takes long execution time, the harness receives the same input several steps in a row, causing the classical controller to continuously output the same value several steps in a row. This could cause issues as the torque to wheel speed conversion algorithm performs an integration, and integrating this constant value could result in large attitude slews being applied to the satellite. Furthermore, this erratic TM behaviour could cause some large bias between episodes of the reinforcement learning algorithm, which could severely compromise the learning. Some mitigation solutions were implemented at harness level to minimise the impact of this behaviour, although an improvement of the performance of the I2C is still required as a 10 seconds gap in TM data could create issues.

5.3 TC and wheel command tests

The objective of the next test was to validate the application's ability to pass Tele Commands (TC) to iADCS, and hence to influence its attitude. It should be noted that this series of test represents the first one with the auto coded harness in the loop, which provided the attitude profile.

It is important to note that it was decided to use reaction wheel speed commands instead of direct torque commands early in the project as the direct torque command was not available then. This required an additional step in the control algorithm and a number of adaptations to the general architecture. A core objective of this test was to validate this design and to make sure the conversion from the torque commands (dictated by a feedforward profile) to wheel speed commands was working properly.

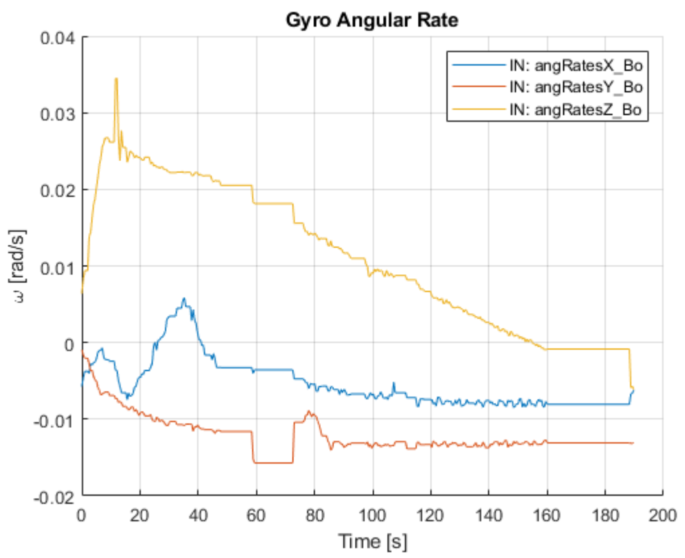

Figure 5-2 OPS-SAT response to open loop reaction wheel speed command profile

The objectives of this test campaign were partially achieved. The harness was run properly and it appears the open loop profile was in part passed correctly to the iADCS unit. However, due to further issues with the interface (flat lines around 60 s on the figure above) and some currently unexplained drift behaviour in the rates, the experiment is currently not considered fully conclusive.

Two main conclusions were drawn at the end of this experiment: the calls to the API need to be further optimised in order to reduce the occurrence of the TM hanging events (this may be achieved by a combination of reviewing the calls to the API and reducing their frequency), and it was preferable to go back to direct torque control, made available again by the OPS-SAT team. This last point also partially resolves the concerns raised at the end of the previous section (regarding integration of a constant torque value). The test to evaluate the efficiency of these improvements is ready and should be flown at the next available opportunity.

6 CONCLUSIONS AND NEXT STEPS

Although some implementation and schedule challenges prevented the team from testing the AI algorithm itself, some significant progress was made towards the implementation of a close loop AOCS algorithm running on the SEPP. This was a significant challenge in itself, but thanks to the iterative approach the limitations of the interface between the SEPP and the iADCS were characterised and solutions designed. The team is confident reasonable performance can be achieved, although it will require reducing the time cycle frequency to some extent.

The team is ready and hoping to continue testing the application on the platform, to iteratively add the final improvements to the time cycle management discussed above, the closed loop controller, and finally the AI algorithm itself. The figure below summarises the current achievements and what remains to be done.

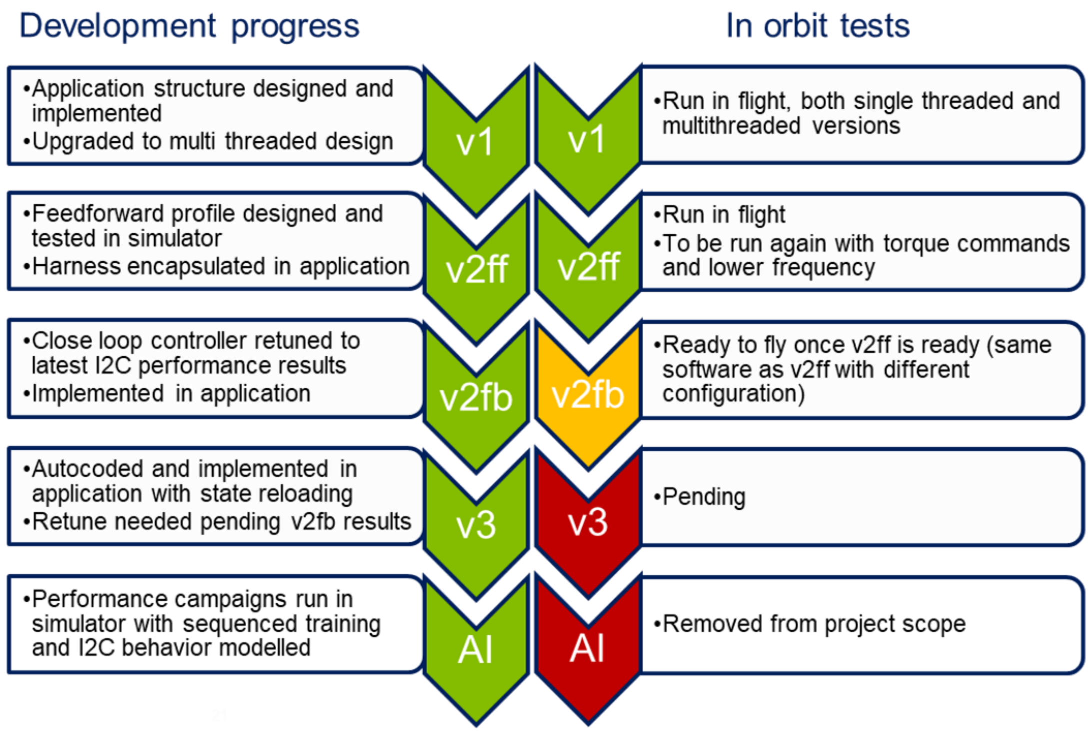
Development progress
In orbit tests
•Application structure designed and implemented
•Upgraded to multi threaded design
v1
v1
•Run in flight, both single threaded and multithreaded versions
•Feedforward profile designed and tested in simulator
•Harness encapsulated in application
v2ff
v2ff
•Run in flight
•To be run again with torque commands and lower frequency
•Close loop controller retuned to latest I2C performance results
•Implemented in application
v2fb
v2fb
•Ready to fly once v2ff is ready (same software as v2ff with different configuration)
•Autocoded and implemented in application with state reloading
•Retune needed pending v2fb results
v3
v3
•Pending
•Performance campaigns run in simulator with sequenced training and I2C behavior modelled
AI
AI
•Removed from project scope

Figure 6-1 Progress summary

Publishing Date: 7 December 2022
Contract Number: 4000137215/22/NL/GLC/ov
Implemented as ESA OSIP