Journal of Systems Architecture 167 (2025) 103513

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Feasibility of Deep Reinforcement Learning for the real-time attitude control of a satellite system

Ángel-Grover Pérez-Muñoz [a,*], Guillermo López-García [a], Irene García-Villoria [a], Alejandro Alonso [a], Angel Porras-Hermoso [b], María S. Pérez [c]

[a] Information Processing and Telecommunications Center, Universidad Politécnica de Madrid, Avenida Complutense 30, Madrid, 28040, Spain
[b] Instituto Universitario de Microgravedad "Ignacio da Riva", Universidad Politécnica de Madrid, Plaza Cardenal Cisneros 3, Madrid, 28040, Spain
[c] Ontology Engineering Group, ETS de Ingenieros Informáticos, Universidad Politécnica de Madrid, Campus de Montegancedo, Madrid, 28660, Spain

ARTICLE INFO

Keywords:
Artificial intelligence
Neural network controllers
Safety-critical systems
Embedded systems

ABSTRACT

Although Machine Learning (ML) is widely used in a variety of interdisciplinary applications, its implementation in safety-critical systems, such as the Attitude Control System (ACS) of a satellite, poses numerous challenges. While previous studies have shown promising results, there is a lack of information on the design and development process for the application of ML in real-time control systems. This paper presents the implementation of a Deep Reinforcement Learning (DRL) model for a magnetic-based ACS of the UPMSat-2 satellite. The primary objective is not only to design, implement, and validate an RL agent, but also to provide some insights and criteria of the decision-making process to achieve an adequate model. The system was trained and validated on a simulation model with positive results. To further validate non-functional requirements, the resulting trained agent was tested on a real-time embedded system according to safety standards. The obtained quantitative metrics and performance results show the ability of the agent to maintain the satellite's attitude across various operational phases, leveraging its adaptability to dynamic conditions.

1. Introduction

During recent years, there has been a growing interest from companies and organizations in the adoption of Artificial Intelligence (AI) solutions in a wide range of fields. In particular, aviation and space industries are researching towards the use of these techniques in safety-critical systems. AI, especially Machine Learning (ML), can benefit from huge amounts of data to extract patterns and knowledge, learning how to perform tasks without explicit programming. The global research efforts of the authors aim at (i) developing systems based on ML techniques for embedding controllers in safety-critical systems, and (ii) validating that this software can satisfy the stringent requirements of the aforementioned industries.

ML is generally produced by stochastic processes, leading to the ability to generalize unseen cases [1]. RL, a subfield of ML, exploits these capabilities for control systems by automatically learning control laws through the interaction with the environment. This makes RL effective in addressing decision-making and control problems that are complex to solve using hand-coded rules [2]. RL is therefore preferable compared to other methodologies, such as supervised learning. In addition, the control policies generated in RL can use neural networks which can be implemented with bounded time and memory margins, which is mandatory for safety-critical systems.

Despite this, the application of ML/RL techniques in safety-critical systems presents significant challenges, as their failures could result in economic, environmental, or even human losses. Different studies have analyzed the viability of effectively combining these fields, but it is still a challenge [3–5]. An outstanding issue is the identification of ML techniques that can satisfy the strict requirements of the standards of the safety-critical systems. As recognized by Tambon et al. [3], the lack of formalization and limited practical experience make it particularly important to investigate the feasibility of ML and RL in this type of system. As a first step towards standardization, organizations such as European Cooperation for Space Standardization (ECSS) for aerospace [4] and European Union Aviation Safety Agency (EASA) for avionics [5] published general guidelines for AI-based systems with low and mid-criticality levels.

The UPMSat−2 (launched in 2020) and UPMSat−3 (expected to be launched in the last quarter of 2025) projects aim to build experimental microsatellites that will serve as technology demonstrators for

* Corresponding author.
E-mail addresses: angel.perez.munoz@upm.es (Á.-G. Pérez-Muñoz), g.lopezg@upm.es (G. López-García), irene.gvilloria@alumnos.upm.es (I. García-Villoria), alejandro.alonso@upm.es (A. Alonso), angel.porras.hermoso@upm.es (A. Porras-Hermoso), mperez@fi.upm.es (M.S. Pérez).

https://doi.org/10.1016/j.sysarc.2025.103513
Received 7 February 2025; Received in revised form 24 April 2025; Accepted 27 June 2025
Available online 17 July 2025

scientific and educational applications. These projects are led by the Microgravity Research Institute "Ignacio Da Riva" (IDR-UPM), which manages the missions and is responsible for designing and manufacturing most of the satellite components, in collaboration with other research groups and aerospace companies. The authors are members of the Sistemas de Tiempo Real *y* Arquitectura de Servicios Telemáticos (STRAST-IPTC) group, which developed all the software on board the UPMSat−2 satellite and is developing the software on board the UPMSat−3 satellite.

Within this framework, the development of an ACS experiment using RL for the UPMSat−3 satellite was proposed. The ACS is a critical component for the safe operation of satellites, as it ensures proper orientation, directly affecting essential functions such as thermal regulation, power management, and alignment of payload equipment and antennas with ground stations [6]. As the hardware for the UPMSat−3 satellite is not currently available, the experiment was developed within the UPMSat−2 context. The ACS UPMSat−2 software was developed using a Model-Driven Engineering approach [7,8] and Simulink models were built for satellite components that included sensor signal conditioning, control algorithm, and other related operations. Therefore, the development and validation of the ML techniques were based on these models.

Previous research by Elkins et al. [9], Su et al. [10], and Retagne et al. [11] recognized the potential of combining RL techniques in the attitude control domain. Particularly due to their ability to learn control strategies and adapt to the uncertainties of complex and dynamic environments, where classical methods may require precise modeling and manual adjustment. Thus, RL demonstrated superior adaptability in scenarios with high uncertainty and limited observability, making it a suitable choice for real-time control applications on board a satellite.

This paper details the definition of actions and observations, the tuning of the reward function, and the agent training process, offering explanations for each decision made while maintaining a practical approach. Regarding the non-functional aspects, the authors aim to validate that the embedded ML controllers meet the safety-critical standards. This effort relies on aerospace standards from ECSS focusing on the safety and reliability of software systems. Previous research work lacks verification of these non-functional requirements and focuses solely on the functional aspects of the control system [11–13]. Consequently, it is necessary to explore the capabilities of RL in ACS while meeting all requirements of the system in terms of time predictability and resource bound. In summary, this paper (i) describes the design, implementation, and validation of an RL agent for the real-time attitude control of a satellite system; and (ii) deploys the controller in an embedded computer to validate some relevant requirements according to safety standards.

The remainder of this paper is organized as follows. Section 2 gives a background on RL focusing on the formalization of Markov Decision Process (MDP) and the Proximal Policy Optimization (PPO) algorithm used to train the agent. Section 3 reviews recent work on the implementation of ACS by means of RL. Section 4 describes the architecture of UPMSat−2 ACS, its requirements, and the software and hardware materials to train and validate the agent. Section 5 describes the design of the RL controller, discussing iterative decisions on action space, state space, reward function, and training. The results of each iteration are presented and discussed as well. Section 6 discusses the embedded verification after deploying the trained agent. Finally, conclusions are drawn in Section 7 and potential future work is discussed in Section 8.

2. Reinforcement learning background

2.1. Reinforcement learning formalization

RL is a ML method where an agent learns an optimal behavior by interaction with a dynamic environment. The learning aims to improve the agent so that it can choose actions that maximize rewards accumulated over a period of time. As shown in Fig. 1, its main elements are the agent, the action, the state, the environment, and the reward. First, the agent contains the policy that acts as the controller of the system, and the training or learning algorithm to update the policy. The policy is typically implemented using neural networks. Second, the actions are the commands made by the agent given a state, which may be discrete (e.g. turn on/off actuator) or continuous (e.g. applied energy on the actuator). Third, the state represents the status of the environment. In the case of partial observability, the agent is limited to the observable states. Fourth, the environment contains all external elements, including the plant, dynamics, sensors, and actuators. It receives the action from the agent and responds with a new state and reward. Finally, the reward is a numerical value that criticizes the agent's performance based on the state of the environment, with higher values indicating better performance. The reward and state are used by the training algorithm to update the policy so that the actual and future rewards are maximized.

The elements involved in the sequential evolution of the system are formally modeled as a MDP as follows. At each time step t, the agent executes an action $\boldsymbol{u}_t \in \mathcal{U}$ on the environment which returns the new state $\boldsymbol{x}_t \in \mathcal{X}$ and the reward $r_t \in R$. The state $\boldsymbol{x}_t$ is given by the state transition function $F : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$, which depends on the previous state $\boldsymbol{x}_{t-1}$ and the applied action $\boldsymbol{u}_{t-1}$:

$$\boldsymbol{x}_t = F(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}). \tag{1}$$

The received reward r_t is given by the function $R : \mathcal{X} \to \mathbb{R}$, which indicates how well the agent is acting based on the previous state $\boldsymbol{x}_{t-1}$:

$$r_t = R(\boldsymbol{x}_{t-1}). \tag{2}$$

The agent's policy, denoted as π, may be deterministic or stochastic. Deterministic policies directly map states to actions, while stochastic policies define a probability distribution on actions given the states. The ultimate goal of RL is to train a policy π that maximizes the value function, which is given by:

$$V^{\pi}(\boldsymbol{x}) = \mathbb{E}(G_t | \boldsymbol{x}_t = \boldsymbol{x}), \quad \text{with} \quad G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{3}$$

As seen, the value function is the expected sum of future discounted rewards G_t, where $\gamma \in [0, 1]$ represents the discount factor applied to future rewards. Discount factors near 0 cause the agent to optimize the short-term rewards, while factors near 1 cause the agent to have a broader perspective aiming to optimize the current and future rewards.

2.2. Training algorithm: proximal policy optimization (PPO)

The training process consists of a sequence of interactions with the environment, known as episodes. Each episode consists of various steps in which the agent executes an action on the environment, receiving the next state and the reward value in response. The data obtained in each episode are recorded and used to update the agent's policy to maximize the value function (see Eq. (3)). A well-established training algorithm is PPO, a policy gradient method that uses a clipping mechanism to limit the degree to which the policy can change in a single update. This restriction controls the impact of the changes, allowing the training process to be more stable than other alternatives such as Deep Deterministic Policy Gradient (DDPG), Deep Q-Network (DQN), Soft Actor-Critic (SAC) [14]. PPO was shown to outperform traditional feedback controllers in attitude control tasks [15], and to achieve superior learning stability in discrete and constrained action spaces compared to TD3 [9].

The PPO was selected as the training algorithm for ACS due to its ability to support both continuous and discrete action spaces, making it well suited to the PWM-driven magnetorquers used in the UPMSat−2

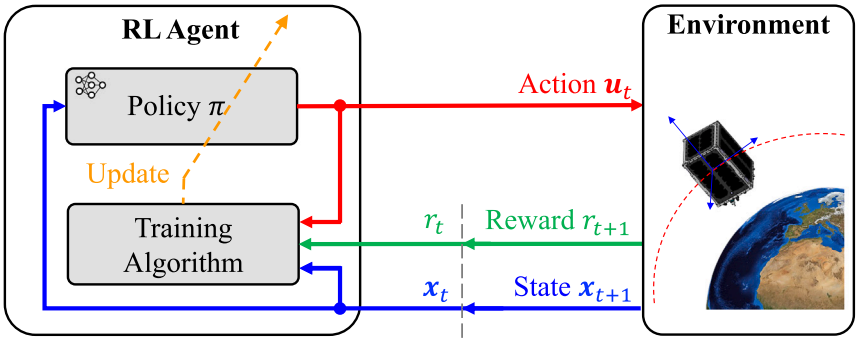


Fig. 1. Reinforcement learning elements.

system. This flexibility allowed us to experiment with both representations and to ultimately achieve stable training and robust performance under the constraints of our system.

To estimate the policy and value functions, PPO uses two neural networks: Actor and Critic. These networks are trained with data collected from interactions with the environment. The Critic $V_{\Phi}(\boldsymbol{x})$ with parameters Φ, takes observation $\boldsymbol{x}$ and predicts the value function V^{π}. The Actor $\pi_{\Theta}(\boldsymbol{u}|\boldsymbol{x})$ with parameters Θ is a stochastic policy that returns the probability of taking action $\boldsymbol{u}$ in state $\boldsymbol{x}$. In a discrete action space, it returns the probability for each action. In a continuous action space, it returns the mean and standard deviation of the learned Gaussian distribution. Note that Actor and Critic are neural networks whose outputs result from matrix multiplications that meet two properties. First, the execution time of each inference is bounded since it can be programmed as a non-recursive function. Second, though the PPO is an stochastic policy, it is deterministic at inference time since their parameters are frozen, producing the same outputs for the same inputs. Therefore, compared to other online optimization-based techniques, they are predictable from the timing and functional perspectives, which is mandatory for safety-critical systems.

The authors encourage readers interested in the details of the PPO algorithm to see Schulman et al. [16]. To facilitate readability, the following summarizes the main hyperparameters of PPO that are involved during the training process.

- The "clip factor" (ϵ) controls the range of policy updates. Higher values allow larger changes to the policy, implying faster exploration but increasing the chances of instability. On the other hand, lower values promote slower but more stable and gradual exploration.
- The "learning rate" is the speed at which the network parameters are updated during training. The Actor and the Critic have their own learning rates (denoted as λ_{Ω} and λ_{Φ}, respectively).
- The "experience horizon" is the number of steps taken before training. Each step implies collecting an observation of the state, deciding the next action, and receiving a reward from the environment.
- The number of "epochs" defines how many times the networks process a set of experiences in a single training iteration.
- The "batch size" determines how much data is processed in each training epoch. Each time, the batch is sampled from the experience horizon; therefore, the latter must be equal to or greater than the batch size.
- The advantage function (A_t) represents how much better taking action a_t is compared to the average expected outcome from state s_t.
- The "discount factor" (γ) is used in the advantage function to balance the importance of future rewards. It ranges from 0 to 1 and the higher it is, the more importance is given to future values of rewards.

Table 1
Summary of related and presented work.

Reference	RL agent	Control architecture	Comments
[18]	DQN	Torque on one axis at a time.	Reward function and control method unclear.
[10]	DDPG	Reaction wheels.	Reward function and training process omitted.
[19]	DDPG	Reaction wheels and magnetorquers.	Outputs torque; training process not detailed.
[20]	SAC	Control moment gyroscopes.	RL used only in stabilization phase.
[11]	SAC, PPO	Reaction wheels.	No real-time adaptation to continuous mass variations.
[17]	PPO	Thrusters and reaction wheels.	Reward function and training details not fully explained.
[9]	PPO	Reaction wheels.	Reward function selection unexplained.
Present work	PPO	Three magnetometers, and three magnetorquers.	Functional RL ACS with in-depth discussion of its design and validation.

- The term $r_t(\Theta)$ is the probability ratio between the new and old policies. This ratio measures the relative likelihood of the action $\boldsymbol{u}_t$ under the new policy π_{Θ} compared to the old policy $\pi_{\Theta_{\text{old}}}$.
- The "entropy coefficient" is a parameter that promotes agent exploration. The higher it is, the more uncertain the agent will be to take the next action.

3. Related work

In recent years, research on the use of RL and Deep Reinforcement Learning (DRL) for attitude control has grown due to their adaptability to environmental uncertainties and their excellent performance in non-linear systems [9,11,17]. This section reviews our bibliographic investigation of studies using RL/DRL for satellite attitude control. Our findings cover off-policy methods including DQN and DDPG, as well as on-policy methods such as SAC and PPO. Table 1 summarizes the revised work and highlights the type of RL agent used as the ACS controller, the control architecture including sensors and actuators, and general comments containing the main limitations of these works. For comparison purposes, we include our work in the last row.

Regarding off-policy RL methods, Ma et al. [18] investigated the application of a DQN agent for attitude control, where up to seven discrete actions are available, each corresponding to a specific torque applied along a single axis. However, the control scheme remains ambiguous as the neural network output is directly mapped to the torque without further clarification. Furthermore, the reward function is only briefly described in broad terms, lacking a detailed explanation of its design and impact on the learning process. The use of DQN limited this study to discrete actions. To support continuous action spaces, Su

et al. [10] proposed a DDPG agent for the ACS of simulated satellite using reaction wheels as actuators. Although the study successfully illustrates the potential of using a continuous action space for attitude control, it lacks details on the reward function and does not provide a sufficiently detailed explanation of the training process. Another study from Yadava et al. [19] explored the application of a DDPG agent for attitude control, focusing on a hybrid system that combines reaction wheels with magnetorquers. Their work used the neural network output as the torque to be applied, and not as specific actuator commands. The study lacks of experimental results and does not provide detailed information on the decisions taken during the design of the agent.

Regarding on-policy RL methods, Oghim et al. [20] explored the use of a SAC agent for attitude control using control moment gyroscopes. Their approach integrates RL exclusively during the stabilization phase, after which a conventional control law takes over. The work of Retagne et al. [11] evaluated the SAC and PPO methods for adaptive satellite attitude control under varying spacecraft mass conditions. This method employs a DRL agent trained to stack observations (including the satellite's angular velocity) in such a way that the agent indirectly estimates mass-dependent properties for greater adaptability than conventional control strategies. This study demonstrated the performance of stacked observations in dealing with mass variations, achieving similar results with both algorithms. In general, these studies demonstrate the potential of on-policy RL methods for attitude control, but the specifics of the training mechanism used are not disclosed, leaving a reproducibility gap and a deeper understanding of their approach.

The work carried out by Elkins et al. [9] served as a key reference for this study, as it explores the application of a PPO agent for attitude control. In particular, their work highlights the advantages of PPO in discrete and constrained control problems, showing that with appropriate reward shaping, PPO can outperform other methods under similar conditions. This study implements the simulation environment in Python and provides a functional demonstration of RL for spacecraft control using reaction wheels. Their results indicate that the proposed method performs well, particularly in scenarios where initial disturbances are applied to the satellite. A similar study using PPO was presented by Allison et al. [17] which trains an agent RL for an adaptive attitude controller that varies with the inertias of the spacecraft, which are randomized on purpose throughout the simulation for the enhancement of robustness. The results of these studies demonstrate the adaptability of RL controllers to time-varying changes in the environment, which in some cases is comparable to or better than conventional controllers. However, not much detail is given on the design of its reward function and the actual training procedure itself.

Although these studies explored various RL approaches for attitude control, key distinctions highlight the contributions of our work. There are two differences between this project and those reported in the literature. The first is that this work presents a reinforcement-based approach for a magnetic-based controller for a ACS. Previous studies adapt magnetometer-magnetorquer systems with additional actuators, none of them rely solely on these magnetometers and magnetorquers. The second one is that our work not only focuses on the functional aspects of the controller, it is complemented by a rigorous explanation of reward function design and different training techniques. Unlike most related research, which provides only theoretical descriptions, this study offers deeper insights into the process of developing a RL agent for ACS.

4. Experimental description and setup

4.1. UPMSat–2 attitude control system

This study is based on the original design of UPMSat–2 ACS, whose results were successfully validated in orbit [21] and, therefore, the details of the implementation are applicable to this experiment as well [8,22]. The UPMSat–2 is a 50 kg-class satellite launched on the Vega VV16 flight on 3 September 2020. It was deployed in a Sun-synchronous orbit at 530 km of altitude with a period of 95 min. The ACS of the UPMSat–2 aims to maintain a fixed angular speed and ensure that the rotation axis remains perpendicular to the orbit. Meeting these requirements allows the satellite to maintain a uniform temperature distribution, balance the amount of solar radiation received by the lateral panels, and ensure the visibility of the radio antenna from the ground station. An active magnetic attitude control was selected for the UPMSat–2, which is based on the interaction of the Earth's magnetic field with the satellite. Specifically, a modification of the standard B-dot control law, developed by Cubas et al. [22], was used due to its affordability and reliability for small Low Earth Orbit (LEO) missions with non-demanding pointing and orientation requirements.

Table 2
Discretization of the magnetic moment magnitudes into PWM duty-cycles, where the axis subscript may be X, Y, or Z.

$\boldsymbol{m}_{\text{axis}}$ magnitude [Am^2]	MGT_{axis} duty-cycle [ms]
$[0, 0.05)$	0
$[0.05, 0.15)$	100
$[0.15, 0.25)$	200
$[0.25, 0.35)$	300
$[0.35, 0.45)$	400
$[0.45, 2]$	500

As actuators, the UPMSat–2 was equipped with three single-axis orthogonal magnetorquers producing magnetic moment $\boldsymbol{m}$ [Am^2] in any direction. As sensors, it included three redundant flux-gate magnetometers to measure the magnetic field $\boldsymbol{B}$ [T] on the three axes. The control law was implemented by a computerized system in which magnetometers are read using a 12-bit Analog-to-Digital Converter (ADC). Conversely, due to the lack of Digital-to-Analog Coverters (DACs), magnetorquers are controlled by PWM, a discretization method for controlling analog signals using digital outputs. Specifically, each component of the actuating magnetic moment was discretized for the set of values $\pm\{0, 100, 200, 300, 400, 500\}$, where the sign represents the direction of the torque and the magnitude represents the time in ms the magnetorquer remains active (PWM duty cycle). Although these values may seem excessive, as shorter actuations could make more precise maneuvers, the UPMSat–2 development team decided that they were adequate enough for the requirements of the satellite [8,22]. Table 2 shows the required PWM duty cycles for different magnitudes of the magnetic moment.

During attitude control, the satellite goes through three stages: detumbling, stabilization, and stable phase. The detumbling phase starts after the separation from the launcher. In this stage, the ACS applies magnetic torques to reduce the angular velocity of the X and Y axes, while simultaneously causing rotation around the Z axis to achieve the desired setpoint defined as $\boldsymbol{\omega}^d$, specifically $[\boldsymbol{\omega}_x^d, \boldsymbol{\omega}_y^d, \boldsymbol{\omega}_z^d] = [0, 0, 0.1]\,\text{rad}\,\text{s}^{-1}$. In this phase, the orientation of the Z axis with respect to the orbit normal may vary freely until the next phase. The stabilization phase starts after the angular velocity has been corrected in the detumbling. This phase aligns the spin axis (Z) with the orbit normal (Z_0), as the second Euler angle (θ) tends to the desired value ($\theta^d = 0°$). Lastly, the satellite enters the stable phase maintaining the attitude reached in the previous phases ($\boldsymbol{\omega}$ close to $\boldsymbol{\omega}^d$ and Z parallel to Z_0).

Based on this information, the control architecture of the UPMSat–2 ACS can be depicted in Fig. 2. As seen, the controller acts on the magnetorquers through PWM based on the magnetometer readings and desired angular velocity. The interaction of the magnetorquers magnetic moment and the magnetic field of Earth produces a control torque $\boldsymbol{T}_c = \boldsymbol{m} \times \boldsymbol{B}$. The satellite dynamics receives this control torque, which also interacts with external perturbations. The response of the satellite is feedback to the magnetometers, closing the attitude control loop.

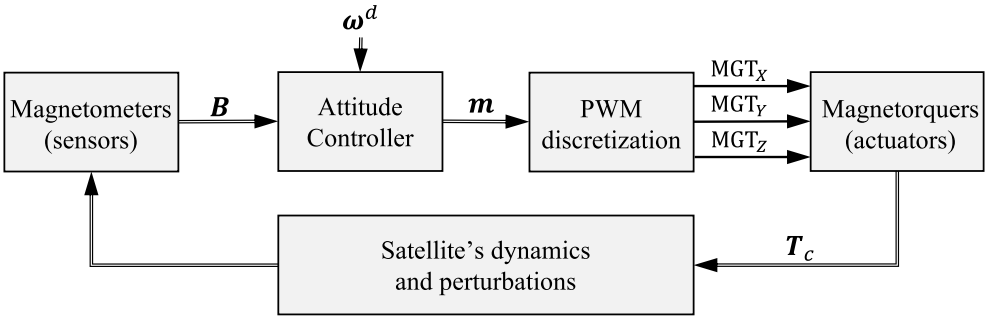


Fig. 2. ACS high-level control architecture.

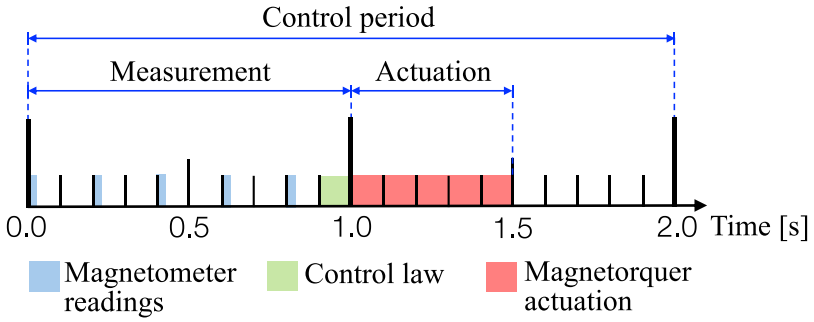


Fig. 3. Control cycle periodically executed by the ACS [8].

4.2. Attitude control time requirements

The time requirements in the development of the RL ACS are those defined in the UPMSat−2 and are critical for the correct functioning of the satellite. The timing diagram shown in Fig. 3 illustrates the time requirements of UPMSat−2's ACS, as defined in the original design by Zamorano et al. [8]. The timing behavior of magnetometer readings and magnetorquers actuation is forced by the satellite hardware [8,21]. In the satellite, magnetometers cannot generate readings while the magnetorquers are producing the control torque, as magnetic remanence could cause interference in the sensor readings. Therefore, the original ACS included time requirements to avoid possible errors. If the timing of the control law is not right, the satellite could start to rotate, which could compromise the communication capacity with the ground station or affect the homogeneous charging of the solar panels, leading to a complete loss of the mission.

As seen in Fig. 3, the attitude control cycle has a period of 2000 ms. The cycle starts with magnetometer readings acquired every 200 ms. Subsequently, the control law executes to calculate the actions to be applied on the magnetorquers with a deadline of 100 ms. After that, the actuation is performed with a duration in the range of 0 ms to 500 ms. Finally, it is necessary to leave a minimum of half a second without any action to avoid interference between the magnetometers and the magnetorquers due to residual magnetic forces. The RL controller developed and validated in this paper executes as an alternative control law (green block in Fig. 3). Therefore, these requirements are also applicable, and the RL controller must run between these time limits, with a period of 2000 ms and deadline of 100 ms. For further information on the real-time software architecture and implementation of the UPMSat−2 ACS, the reader is referred to Cubas et al. [22], Zamorano and Garrido [23].

4.3. Training and validation environments

This section describes the materials used to train and test the RL controllers. The environment was simulated using the ACS model designed for UPMSat−2. This model was executed in the Windows-2024b versions of MATLAB/Simulink from MathWorks. Fig. 4 depicts the high-level blocks of the simulation environment. From left to right, it can be explained as follows. The Sun block generates solar radiation and the Earth block generates the Earth's magnetic field. The Satellite block represents the UPMSat−2 satellite simulating the ACS hardware such as magnetometers and magnetorquers. The Perturbations block simulates orbit and attitude perturbations including gravitational torque, magnetic residual torque, and aerodynamic and solar radiation forces and torques. The output signals from the Dynamics block are fed back to the other blocks to complete the simulation loop. Finally, the Outputs block computes additional information including the angular velocity, Z-axis pointing.

The model was configured with a simulation step of 0.1 s to comply with the minimum intervals of the control cycle (see Fig. 3). The simulations were configured with a maximum time of 30 000 s, which consisted of the first 5 orbital periods after the separation from the launcher. This allows us to evaluate the full behavior of the ACS, from the detumbling phase to the stable phase, while also assessing the RL agent ability to adapt to the non-linearities and uncertainties inherent in satellite attitude control. To evaluate the RL controllers, we obtained the following data with a period of 1 s: angular velocity ($\boldsymbol{\omega}$), misalignment on the third Euler angle (θ). The evaluations are considered valid if the satellite maintains attitude objectives with an initial angular velocity of $\boldsymbol{\omega}_0 = [0, 0, 0.1]\,\mathrm{rad\,s^{-1}}$; and the three Euler angles set to $[\phi_0, \theta_0, \psi_0] = [0, 30, 60]\,^\circ$.

The RL controller was implemented using the available Mathworks toolbox for RL [24]. This toolbox was used mainly for two reasons. Firstly, the simulation environment from the UPMSat−2 ACS was already modeled in Simulink. Secondly, Mathworks provides advanced tools for automatic code generation, such as Embedded coder, which enables the transformation of the RL agent into source code to be embedded in the On-Board Computers (OBCs). Therefore, this simplified the integration of the agent into the existing simulation environment.

In terms of hardware, training, and validation of RL agents executed on an "Intel i9-13900" processor featuring 32 GB of RAM and an integrated graphics card "Intel UHD Graphics 770". To further validate the RL agent into a real-time embedded platform, we used the STM32F407 board. This board was chosen for its similarity to the processor used in

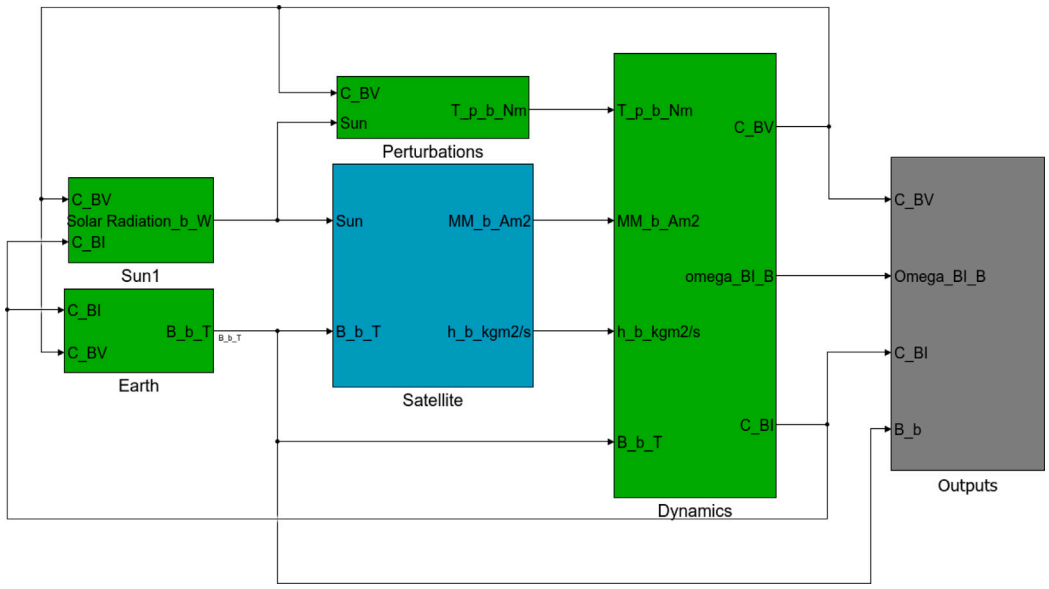


Fig. 4. Simulink model for the satellite's environment.

the UPMSat−3 and its proven reliability in a wide range of applications. In addition, it offers a large number of interfaces, making it easy to connect sensors, actuators, and other equipment. The board features an ARM Cortex-M4 processor running at 168 MHz, 1 MB of flash memory, and 192 KB of SRAM to execute the On-Board Software (OBSW).

4.4. Evaluation metrics

This section discusses the metrics used to quantitatively evaluate the controller results. First, the settling time (t_{set}) was defined as the time it takes for the angular velocity to remain within 5% of the steady-state value according to the requirements of the system. This allows us to determine how fast the attitude has stabilized. Second, the accuracy in each axis was measured using the Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Square Error (RMSE) as shown in Eq. (4)–(6):

$$\mathrm{MAE}_{\mathrm{axis}} = \frac{1}{t_{\mathrm{end}} - t_0} \sum_{t=t_0}^{t_{\mathrm{end}}} |e_{\mathrm{axis}}(t)|, \tag{4}$$

$$\mathrm{MSE}_{\mathrm{axis}} = \frac{1}{t_{\mathrm{end}} - t_0} \sum_{t=t_0}^{t_{\mathrm{end}}} \left(e_{\mathrm{axis}}(t)\right)^2, \tag{5}$$

$$\mathrm{RMSE}_{\mathrm{axis}} = \sqrt{\mathrm{MSE}_{\mathrm{axis}}}. \tag{6}$$

In these equations, the term $e_{\mathrm{axis}}(t)$ denotes the controller error at time t for the X, Y, or Z axis; denoted by the subscript. To compute accuracy, the error quantifies how close the angular velocity is to the setpoint $\boldsymbol{\omega}^d$. In this case, $e_{\mathrm{axis}}(t) = \boldsymbol{\omega}_{\mathrm{axis}}(t) - \boldsymbol{\omega}^d_{\mathrm{axis}}$. Note that these metrics are calculated over the time interval t_0 to t_{end}. Consequently, their values vary depending on the phase in which the controller is evaluated. For the detumbling phase, the evaluation starts when the satellite separates from the launcher and ends when reaching the settling time; therefore $t_0 = 0\,\mathrm{s}$ and $t_{\mathrm{end}} = t_{\mathrm{set}}$. For the stabilization phase, the evaluation starts after detumbling and ends after approximately five orbital periods; therefore, $t_0 = t_{\mathrm{set}}$ and $t_{\mathrm{end}} = 30\,000\,\mathrm{s}$. Finally, if the controller fails to stabilize the satellite ($t_{\mathrm{set}} = \infty$), a complete evaluation is conducted over the entire period; therefore, $t_0 = 0\,\mathrm{s}$ and $t_{\mathrm{end}} = 30\,000\,\mathrm{s}$.

5. Reinforcement learning design: results and discussion

The design of a RL agent is a repetitive process that aims to produce an agent that satisfies the demands of the system (in this case, attitude control). For the sake of clarity, we refer to each repetition in the development process as an iteration, which involves different phases. Fig. 5 provides a high-level scheme of the workflow involved in each iteration. As seen, the state space and data processing are performed first as they remained constant among iterations, allowing us to describe it only once. After that, iterations start in the design phase, where the reward function and action space are defined. The agent is trained and evaluated under the simulation environment, and the results (including quantitative metrics and plots) are analyzed to assess its performance, particularly focusing on the Actor network that implements the control. These results allow us to determine whether new iterations are required, and if so, they will serve as the basis for the next iteration. It is important to note that this evaluation is performed at the model level; therefore, it cannot be considered as a full-validation. The non-functional requirements must also be satisfied to fully validate the controller. The full validation is addressed later in Section 6.

The next subsections describe the most relevant design decisions made among iterations. First, Section 5.1 presents the design of the state space including the processing of the raw data into derived data. The three most important iterations are then presented in Sections 5.2, 5.3, and 5.4. The first iteration discusses the training results of using a continuous action space, that is, real-valued actions. The second iteration significantly improves the previous results by using a discrete action space, that is, a fixed set of actions. Finally, the third iteration ends with the best agent obtained after updating the reward function. In general, the presentation of these iterations and the discussion of their results illustrate design decisions that would be useful for similar systems. Fig. 6 provides a high-level scheme for these three iterations. Note that the state space design is not depicted in the figure as it was kept constant among iterations.

5.1. State space and data processing

The first step before training was to formulate the state space as part of the MDP model. In the case of the UPMSat−2, the magnetic-based ACS included three 3-axis magnetometer that generate readings of the Earth's magnetic field. As explained in the control architecture (see Fig. 3), five magnetometer readings are measured in the first second of the control cycle. Considering the 3 axis, this results in a total of 15 available data. The use of these data obtained through direct observation generally does not provide any insight into the state of

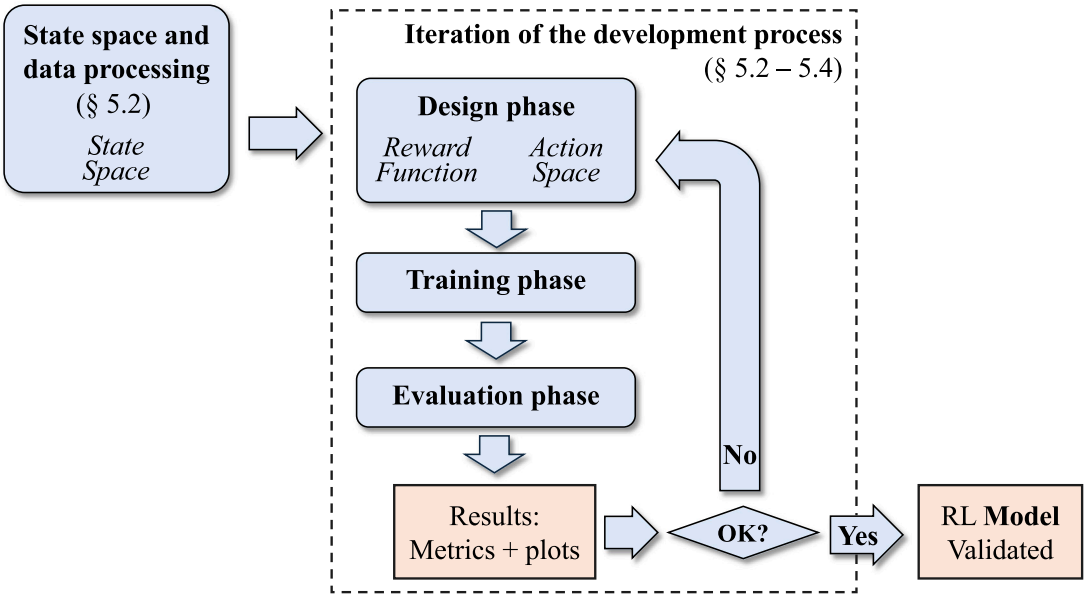


Fig. 5. Flow diagram of the iterations involved in the design process.

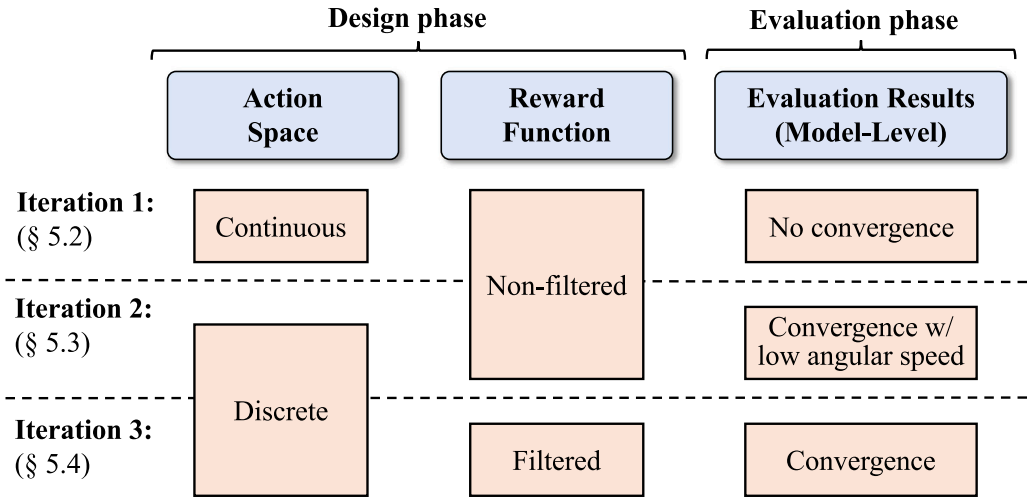


Fig. 6. Overview of the design-decisions and results on the three iterations.

the environment. Alternatively, the processing of data helps to obtain a more compact and organized view of the system. For the RL agent, we combined the acquired measurements to filter out part of the noise by computing their mean value denoted as $\dot{\boldsymbol{B}}$. In addition to that, these measurements were used to compute their derivative $\bar{\boldsymbol{B}}$, which allows the controller to estimate the trend of the Earth's magnetic field. Besides, to allow the agent to understand which states are further from the setpoint, the state includes the error between the angular velocity $\boldsymbol{\omega}$ and its desired value $\boldsymbol{\omega}^d$ for each axis; denoted as $\boldsymbol{\omega}^d - \boldsymbol{\omega}$.

Based on this information, the state variable $\boldsymbol{x}$ consisted of nine components: three for the derivative of the axes X, Y, and Z axes; three for their averaged values; and three for their angular velocity errors. As a summary, Table 3 highlights the main characteristics of the aforementioned elements, including their physical meaning, minimum and maximum values, and units. Before using the state variable as input for the Actor and Critic networks, we needed to ensure that their values were within a predefined range on similar scales. For that purpose, data normalization is applied on the state variable. As seen in Table 3, all numerical values of ACS are within well-defined ranges, therefore

Table 3
List of inputs to the actor containing data derived from the observation. In the case of vectors; minimum, maximum, and units apply at the component level.

Input ($\boldsymbol{x} \in \mathbb{R}^9$)	Physical meaning	Min	Max	Units
$\dot{\boldsymbol{B}} \in \mathbb{R}^3$	Derivative of the magnetometer readings.	-8×10^6	8×10^6	nT s^{-1}
$\bar{\boldsymbol{B}} \in \mathbb{R}^3$	Average of the magnetometer readings.	-5×10^5	5×10^5	nT
$\omega^d - \omega \in \mathbb{R}^3$	Angular velocity error	0	0.25	rad s^{-1}

Min-Max normalization was applied as follows:

$$x_{\text{norm}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}. \tag{7}$$

This operation scales the element x to $x_{\text{norm}} \in [0, 1]$ by using its minimum and maximum values denoted as x_{min} and x_{max}, respectively. In general, using data normalization improves the stability and efficiency of the learning algorithm, allowing better convergence and execution of the process.

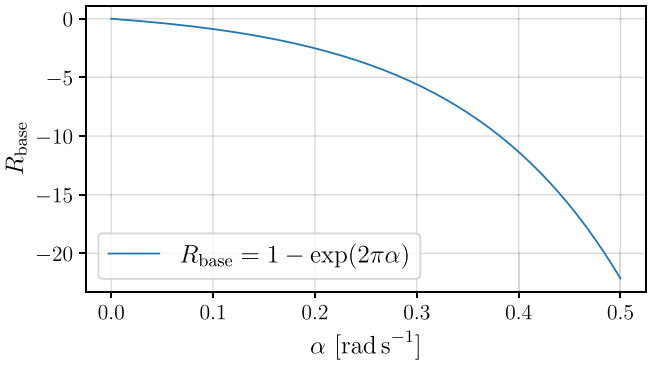


Fig. 7. Base reward function, R_{base}.

Table 4
Hyperparameters used during training.

Parameter	Value
Number of episodes	1000
Number of steps per episode	4000 (8000 s)
Clip factor ϵ	0.2
Entropy loss weight	2e−4
Experience horizon	1024
Minibatch size	512
Discount factor γ	0.99
Advantage estimate method	GAE (factor 0.95)
Actor learning rate λ_{Ω}	10^{-5}
Critic learning rate λ_{Φ}	10^{-4}
Actor hidden Layers	400, 256, 200, 200 (ReLU)
Critic hidden Layers	400, 200, 200 (ReLU)

5.2. Iteration 1: Continuous action space

For the initial iteration, we chose a continuous action space for the output of the neural network. This design is the most intuitive because it aligns with the architecture presented in Fig. 2, where the controller produces a continuous signal $\boldsymbol{m}$ (magnetic moment vector). Based on these considerations, the Actor network contains three output neurons whose value represents the magnetic moment commanded to each magnetorquer. This approach enabled us to map the continuous action outputs to discrete PWM duty cycles using the method described in Table 2.

Regarding the reward function, it was designed to align with the attitude objective, which is to stabilize the satellite angular velocity to the desired value $\boldsymbol{\omega}^d = [0, 0, 0.1]$ rad s^{-1}. To evaluate this objective, the errors between the desired and observed velocity ($\boldsymbol{\omega}$) on the three axes were defined as:

$$[X_{\text{error}}, Y_{\text{error}}, Z_{\text{error}}] = \boldsymbol{\omega}^d - \boldsymbol{\omega}. \tag{8}$$

Since the reward value is one-dimensional, these three error terms were unified by a summation as written in Eq. (9). The absolute value was used to consider the magnitude of the errors without considering the sign, which simplified the reward calculations.

$$\alpha = |X_{\text{error}}| + |Y_{\text{error}}| + |Z_{\text{error}}|. \tag{9}$$

Considering this term, a basic reward function R_{base} was defined as an exponential decay function of the summed angular velocity error α.

$$R_{\text{base}} = 1 - \exp(2\pi\alpha). \tag{10}$$

The choice of this particular expression was given by three factors. Firstly, the range of values of the input parameter (summed absolute error in the angular velocity, α) is relatively small since their initial values are bounded to a minimum of $[-0.1, -0.1, -0.1]$ rad s^{-1} and a maximum of $[0.1, 0.1, 0.1]$ rad s^{-1}. Therefore, an exponential function that changes rapidly with small changes in the abscissa is needed. Secondly, the value of the reward should reflect that larger angular velocity errors imply worse states. This is achieved by an inversely proportional relationship, where large errors lead to lower rewards. Fig. 7 shows that the basic reward function does not vary in the same way when the error is within the range of 0 and 0.2 rad s^{-1} compared to 0.3 and 0.5 rad s^{-1}. Finally, this selection of this reward was supported by the study of Elkins et al. [9] that demonstrated the effectiveness of similar exponential functions for attitude control.

The hyperparameters used for this iteration are summarized in Table 4. The agent was trained during 1000 episodes and 4000 steps per episode. Each step corresponds to a control cycle of 2 s in the ACS, therefore the simulation time is 8000 s per episode. Regarding the Experience Horizon and Minibatch size, their selection was influenced by the hardware characteristics used for training, as larger values could lead to storage issues due to the volume of generated experiences. After different tests, we concluded that the values of 1024 and 512, respectively, provided a balance between better performance and memory usage. The discount factor was set to a relatively high value of 0.99 to ensure that the controller prioritizes both long-term and immediate rewards. The Actor and Critic learning rates were set to 10^{-5} and 10^{-4}, respectively. As suggested in the previous literature, low learning rates are suitable as starting points [1,9]. In addition, the Actor learning rate was set smaller to the Critic to delay the training of the Actor until the Critic is reasonably trained. This relationship has shown good results in previous experiments [25].

Regarding the neural network topologies, the Actor and Critic networks were configured as a Multi-Layer Perceptron (MLP) with four and three hidden layers, respectively. The neurons from the hidden layers used the Rectified Linear Unit (ReLU) activation function: $\text{ReLU}(x) = max(0, x)$. This provided non-linearity relationships between inputs and outputs. The neurons from the output layers used linear activation functions, which are commonly used for regression problems. These design choices were also influenced by the previous studies conducted by one of the authors for the UPMSat−2 ACS [26], in which ReLU layers obtained faster training times and comparable or better results than the Long-Short Term Memory (LSTM) layers.

Fig. 8 presents the evaluation results of the agent trained in this iteration. The figure shows the evolution of the angular velocity for each axis. The setpoint $\boldsymbol{\omega}^d$ is also shown for reference. The agent was evaluated with an initial angular velocity of $[0.1, -0.1, -0.1]$ rad s^{-1} and Euler angles set to $[0, 30, 60]$°. The results show that the actuators controlled by the network have effects in the satellite, but they are insufficient to correctly stabilize the satellite's rotation. Specifically, at time 15 000 s, $\boldsymbol{\omega}_x$ and $\boldsymbol{\omega}_y$ approached values somewhat close to zero but failed to maintain low errors. On the other hand, $\boldsymbol{\omega}_z$ reached the set point near time 4476 s, but it gradually accelerates over time, failing to maintain stability.

For a quantitative evaluation of the agent, Table 5 shows the results of the selected metrics applied to the complete simulation. Note that these metrics represent the average errors on the control, therefore, desired values are close to 0. Like the behavior shown in Fig. 8, the metric values of the X and Y axes are very similar. In both axes, the MAE presents values close to 5.0×10^{-2} rad s^{-1} which are far from the desired setpoint. Concerning the results on the Z axis, it is observed that the error is larger and reaches a value of 3.5×10^{-1} rad s^{-1}, which clearly does not meet the system requirements.

With respect to possible improvements, increasing the number of episodes to 5000 and the steps per episode to 5000 (10 000 s) did not lead to better results in the agent's performance. However, we identified that the continuous action space involved a wide range of possible actions. Specifically, the agent used IEEE-754 double precision floating point format (64 bits), which implies $2^{64} \approx 10^{19}$ representable values for each output neuron. This could limit the agent's ability to converge to an optimal solution during training. Therefore, a new iteration was proposed focusing on discretizing the action space to reduce the possible actions the network can take.

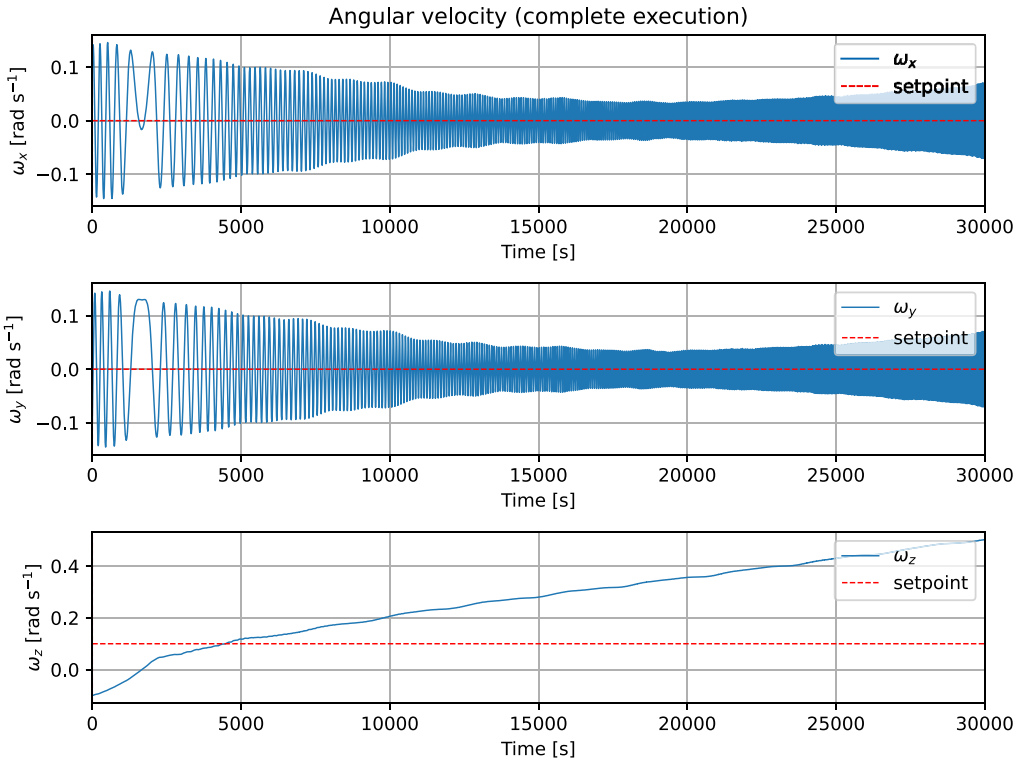


Fig. 8. Evolution of the angular velocity per axis.

Table 5
Evaluation results of the Iteration 1. Metrics calculated from $t_0 = 0\,\text{s}$ up to $t_{\text{end}} = 30\,000\,\text{s}$. Desired values for MAE, MSE, and RMSE are close to 0.

Metric	X axis	Y axis	Z axis
MAE	4.9×10^{-2}	5.8×10^{-2}	3.5×10^{-1}
MSE	4.1×10^{-3}	4.3×10^{-3}	1.4×10^{-1}
RMSE	6.4×10^{-1}	6.6×10^{-2}	3.7×10^{-1}
t_{set} [s]	∞	∞	∞

Table 6
Evaluation results of the Iteration 2 trained with configuration 1 and 2. Metrics calculated from $t_0 = 0\,\text{s}$ up to $t_{\text{end}} = 30\,000\,\text{s}$. Desired values for MAE, MSE, and RMSE are close to 0.

Metric	Configuration 1			Configuration 2		
	X axis	Y axis	Z axis	X axis	Y axis	Z axis
MAE	1.1×10^{-1}	1.2×10^{-1}	1.8×10^{-1}	1.4×10^{-2}	1.3×10^{-2}	3.2×10^{-2}
MSE	1.7×10^{-2}	1.9×10^{-2}	4.0×10^{-2}	1.1×10^{-3}	1.1×10^{-3}	2.2×10^{-3}
RMSE	1.3×10^{-1}	1.4×10^{-1}	2.0×10^{-1}	3.4×10^{-2}	3.3×10^{-2}	4.7×10^{-2}
t_{set} [s]	∞	∞	∞	8000	8000	∞

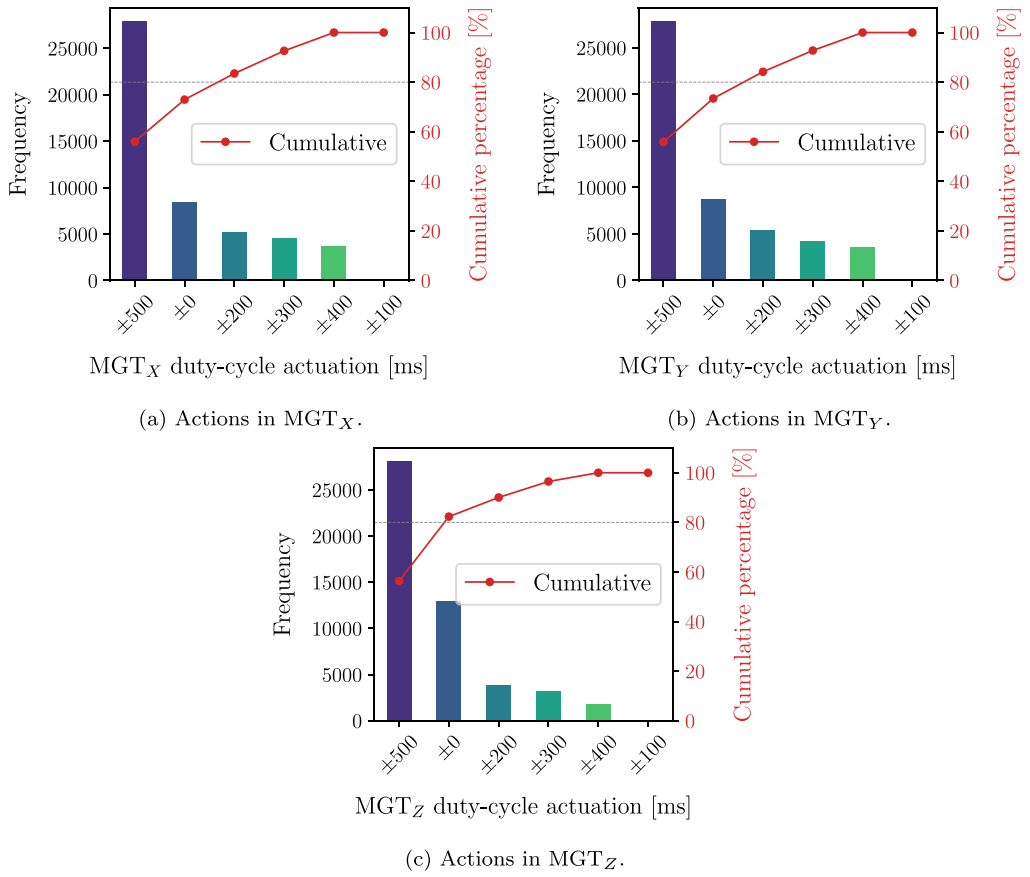
5.3. Iteration 2: Discrete action space

As mentioned in the previous iteration, the continuous action space simplified the Actor network structure but increased the search space, which impacts the convergence of the agent to a solution. For that reason, this iteration presents a new approach by discretizing the output actions into a finite set of values. In this discrete case, the output layer would need to account for all possible combinations of PWM actuations. As previously discussed in Table 2, each of the three magnetorquers can take a value in the set $\pm\{0, 100, 200, 300, 400, 500\}$, resulting in $11^3 = 1331$ possible combinations. With this configuration, the Actor network would have 1331 output neurons, one per combination, which is still a large action space. To reduce it even further, we analyzed the most used actions of the nominal UPMSat−2 ACS. As shown in the Pareto chart in Fig. 9, the three most used PWM duty cycles (above 85% of occurrence) were selected for each magnetorquer. Consequently, each can have five actuations in the subset $\{-500, -200, 0, 200, 500\}$.

Considering the three magnetorquers, this results in $5^3 = 125$ possible outcomes, requiring the Actor network to have 125 neurons in its output layer, where the most probable action is chosen. Therefore, the set of actions can be written as $\mathcal{A} = \{A0, A1, \ldots, A124\}$, where each action corresponds to a unique combination of duty cycles and torque direction for the three magnetorquers. For example, the action $A0$ would be translated into a PWM waveform with a duty cycle of $[500, 500, 500]$ ms in the negative direction, while the action $A124$ would produce the same but in the positive direction.

Apart from the action space, the state space, the reward function, and hyperparameters remained unchanged. In this iteration, the agent was trained twice with different initial conditions, named "Configuration 1" and "Configuration 2". The first training was configured as in the previous iteration with an initial angular velocity of $[0.1, -0.1, -0.1]\,\text{rad s}^{-1}$ and Euler angles of $[0, 30, 60]\,°$. The evaluation results of this first configuration did not lead to significant improvements. As provided in the "Configuration 1" columns from Table 6, the angular velocity did not converge to the desired values in any of its axes ($t_{\text{set}} = \infty$). In addition, the MAE in the three axes reached values of 0.11, 0.12, and $0.18\,\text{rad s}^{-1}$. Considering the initial angular velocities, these values are far from the target error of 0. Similar behaviors are observed in the MSE and RMSE metrics. Although these values are not ideal for a functional ACS, the results on the Z axis are slightly better than the results from the previous iteration.

The second training was configured with a lower initial angular velocity of $[0.05, -0.05, -0.05]\,\text{rad s}^{-1}$. The idea was to facilitate the agent's training by using less strict conditions. The results are shown in the "Configuration 2" columns from Table 6. Comparing the results with "Configuration 1", it is observed that all errors are smaller on all axes. Comparing the results with those of iteration 1, it is observed that the three metrics achieved lower errors. For example, the MAE on the X and Y axes are five times better, and on the Z axis it is about ten times better. Regarding the settling time, though the Z



(a) Actions in MGT$_X$.

(b) Actions in MGT$_Y$.

(c) Actions in MGT$_Z$.

Fig. 9. Pareto chart for the actions of the nominal ACS in the UPMSat−2.

axis did not converge, the X and Y axes reached and maintained the desired value after 8000 s. To complement these results, Fig. 10 presents the evolution of the rotation speed in each axis during the different phases of attitude control; including detumbling and stabilization in the left-hand sided figures. It is observed that the detumbling phase is performed correctly in the three axes, reaching the desired value $\boldsymbol{\omega}^d$. However, the Actor network cannot maintain the speed in the Z axis at values near $0.1\,\mathrm{rad\,s^{-1}}$ after about 10 000 s. Despite these results, it should be noted that the agent was trained in conditions other than evaluation. This shows its ability to generalize knowledge in unfamiliar environments.

In summary, the results for this iteration demonstrate that the discrete agent achieved better results, especially when trained with low angular velocities. In addition, the agent adapted well when tested with higher velocities. It was also observed that when training with a higher angular velocity, the agent did not reach acceptable results. Therefore, the next iteration focused on improving these results by training with higher angular velocities.

5.4. Iteration 3: Reward filtering

In the previous iteration we demonstrated that discrete action spaces reduced the complexity of the network reaching better and faster solutions. However, the improvements were observed only when training and testing with low angular velocities. When the agent was tested under higher speeds (as those found in orbit after separation from the launcher), the agent did not behave as expected. This iteration solves this issue by updating the reward function and maintaining the discrete action space.

To identify which aspects of the reward needed modification, we analyzed its evolution during Iteration 2. It was observed that when the agent was trained under high angular speeds, the evolution of the reward presented significant oscillations, preventing the agent to act effectively. In contrast, when the agent was trained and evaluated under low angular velocities, the reward evolved smoothly, enabling the agent to stabilize the satellite. The Fig. 11 illustrates the evolution of the reward function under these two configurations, namely "R_{base} low speed" and "R_{base} high speed". Therefore, the new reward must be designed to mitigate high-frequency oscillations, even if the satellite starts with high initial angular velocities.

The new reward function was updated with a low-pass filter, specifically, the moving average filter was used. It is commonly used in time series analysis and digital signal processing because it smooths signals by averaging a set of consecutive values, removing part of the high-frequency fluctuations without altering the low-frequency components. The new reward function can be written as follows:

$$R_{\text{filter}}(\boldsymbol{\omega}_t) = 1/n \sum_{k=t-n}^{t} R_{\text{base}}(\boldsymbol{\omega}_t), \tag{11}$$

where the window size n represents the number of past values taken to compute the average. In this case, the window size was kept constant with a 100 s since this value covers one oscillation period in the reward. It can be seen that the average is calculated on the base reward R_{base}, previously defined in Eq. (10). Fig. 11 depicts the application of this filtered reward under high initial angular velocities. It can be observed that the filtered reward has fewer oscillations and shows a smoothness similar to that of the base reward at low angular velocities.

Except for the reward function, the agent was trained with the same hyperparameters, state, and action function. The initial conditions for training were established to a high angular speed of $[0.1, -0.1, -0.1]\,\mathrm{rad\,s^{-1}}$ and Euler angles of $[0, 30, 60]\,^\circ$. The agent was then evaluated with the same initial conditions with a duration of 30 000 s. The evaluation results are shown in Fig. 12. In the figures on the left, it can be observed that the Actor network reduces the angular

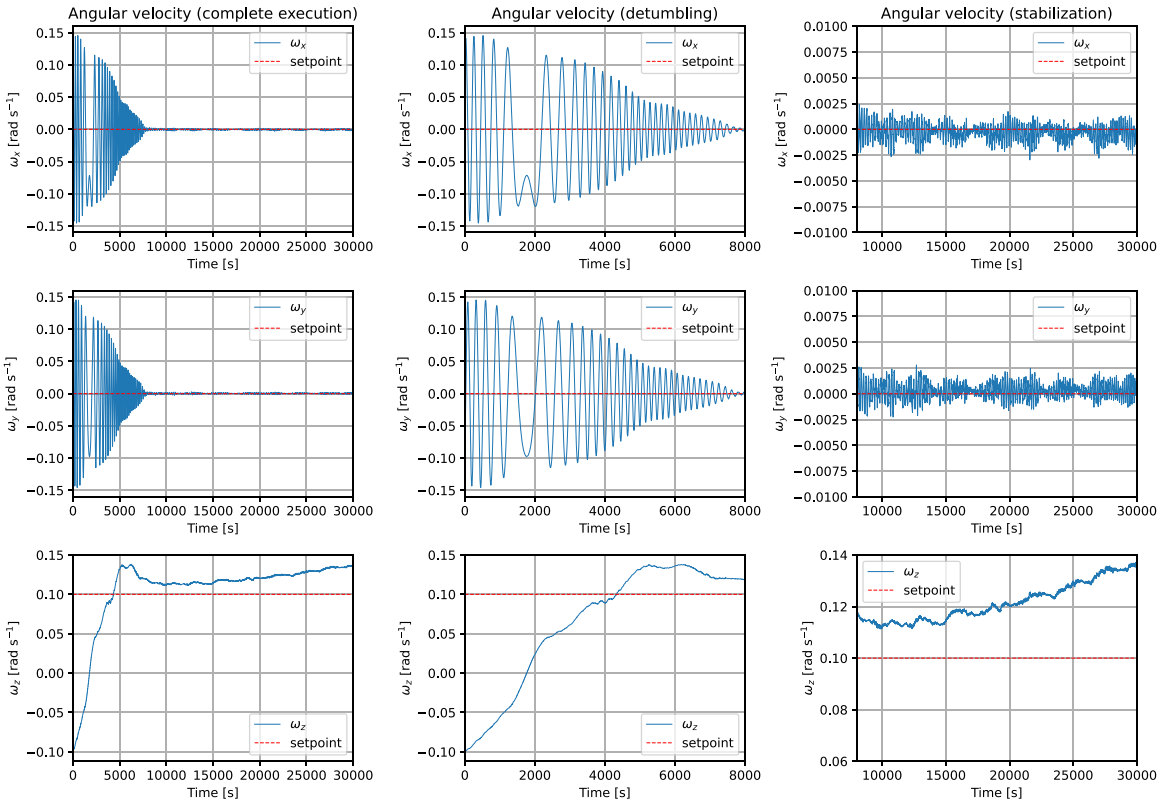


Fig. 10. Angular velocity evolution (per axis) for the RL agent trained with Configuration 2 in Iteration 2. Left: complete evolution over five orbits. Center: zoom in the detumbling phase. Right: zoom in the stabilized phase.

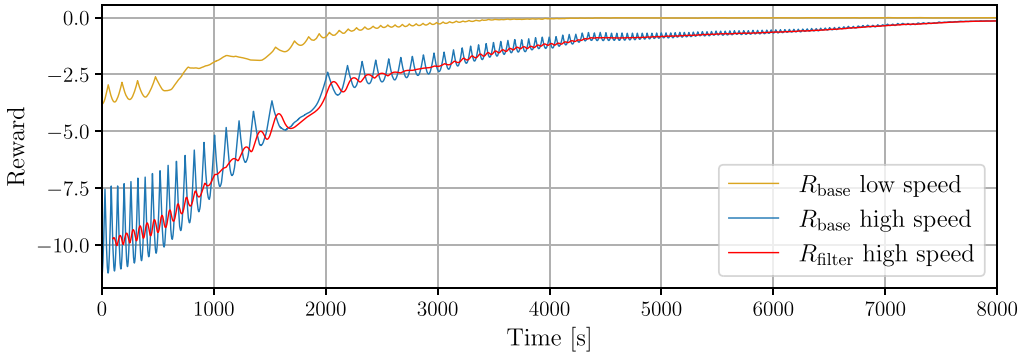


Fig. 11. Comparison of the base reward (R_{base}) under low and high initial angular velocities.

velocity on the three axis by about 8000 s (less than two orbit periods). The figures in the center show that the three axes achieve the desired setpoints; in particular, the rotation on the Z axis stabilizes faster, near time 6500 s. Similarly to the previous iteration, the figures on the right show that during stabilization, the angular velocities on the X and Y axes vary from $-0.0025\,\text{rad}\,\text{s}^{-1}$ to $0.0025\,\text{rad}\,\text{s}^{-1}$. It is also observed that the rotation on the Z axis is slightly more dispersed with values ranging from $0.08\,\text{rad}\,\text{s}^{-1}$ to $0.011\,\text{rad}\,\text{s}^{-1}$. As explained in Cubas et al. [22], the angular velocity around the rotation axis varies naturally with respect to the setpoint. Therefore, these variations cannot be considered a failure or inaccuracy of the controller.

The quantitative metrics of the evaluation results are shown in Table 7. It is observed that the X and Y axes stabilized at the same time as in iteration 2, with slightly more variations in this iteration. However, the results on the Z axis are remarkable, since the stabilization was achieved earlier with a MAE of 2.2×10^{-2}, which is lower than in the previous iteration. The MSE and RMSE exhibit similar behaviors. For a detailed analysis, Table 7 also shows the metrics obtained in the detumbling and stabilization phases. The metrics on the three axes have similar values within the same order of magnitude in the detumbling. During the stabilization phase, the errors on the X and Y axes have similar values, while the errors on the Z axis are ten times bigger compared to X and Y. For reference only, RMSE on Z was of 9.2×10^{-3} and on the X and Y axes it was of 1.1×10^{-3}. Despite this, it is a great improvement, as the Z axis was not stabilized in previous iterations.

In summary, in this iteration, the reward function was updated using a low-pass filter to reduce oscillations. The results demonstrated that training the agent with this filtered reward led to a controller that met the attitude control requirements. It is also remarkable that, although the trained lasted 8000 s, it generalized to unseen cases by maintaining the setpoint throughout 30 000 s (around five orbital periods).

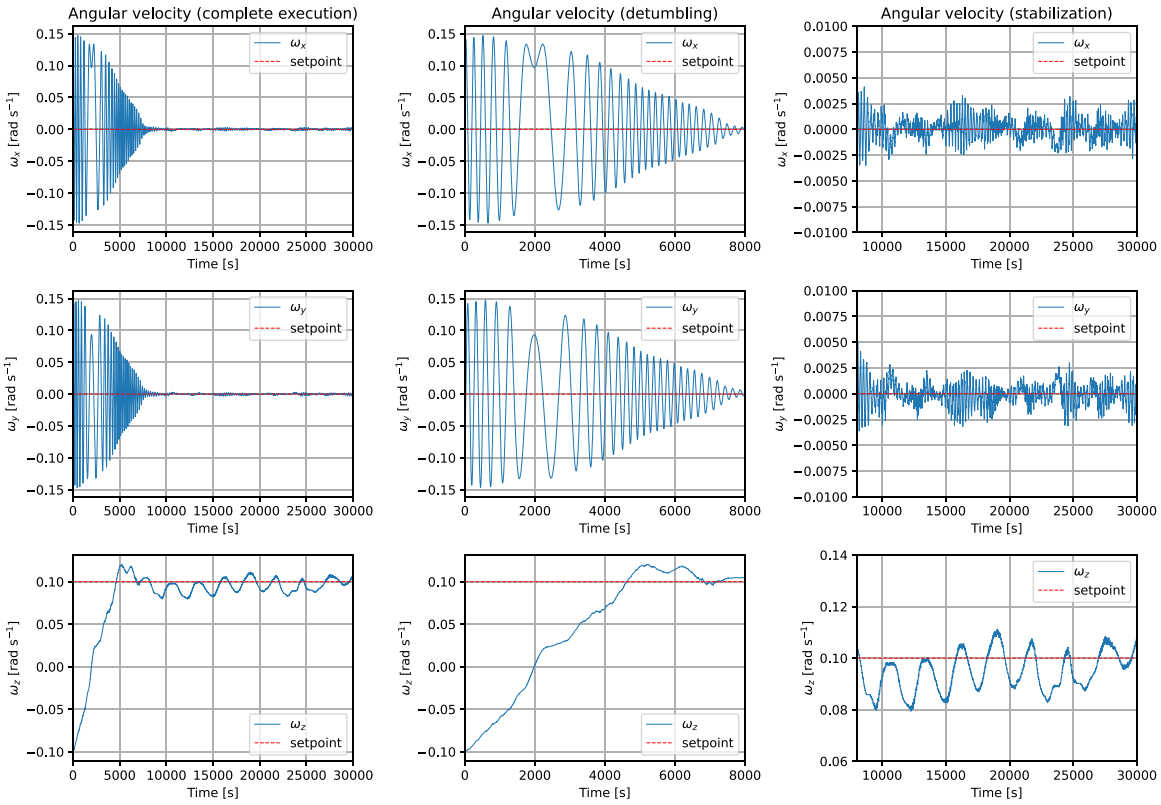


Fig. 12. Angular velocity evolution (per axis) of the RL agent trained in Iteration 3. Left: complete evolution over five orbits. Center: zoom in the detumbling phase. Right: zoom in the stabilized phase.

Table 7
Evaluation results of Iteration 3. Left: complete evolution over five orbits. Center: zoom in the detumbling phase. Right: zoom in the stabilized phase. Desired values for MAE, MSE, and RMSE are close to 0.

Metric	Complete			Detumbling			Stabilization		
	X axis	Y axis	Z axis	X axis	Y axis	Z axis	X axis	Y axis	Z axis
MAE	1.6×10^{-2}	1.5×10^{-2}	2.2×10^{-2}	5.9×10^{-2}	5.6×10^{-2}	5.9×10^{-3}	8.5×10^{-4}	8.6×10^{-4}	7.6×10^{-3}
MSE	1.4×10^{-3}	1.3×10^{-3}	2.1×10^{-3}	5.4×10^{-3}	4.8×10^{-3}	7.5×10^{-3}	1.2×10^{-6}	1.3×10^{-6}	8.6×10^{-5}
RMSE	3.8×10^{-2}	3.6×10^{-2}	4.5×10^{-2}	7.3×10^{-2}	6.9×10^{-2}	8.7×10^{-2}	1.1×10^{-3}	1.1×10^{-3}	9.2×10^{-3}
t_{set} [s]	8000	8000	6500	–	–	–	–	–	–

6. Validation of the embedded controller

This section describes a relevant and initial set of validation activities carried out to ensure the correctness of the RL controller in a more realistic scenario including embedded hardware and software. This work aims to pave the way for future experiments to verify the use of neural networks in safety-critical space systems. The verification and validation setup used in this experiment is representative of a real satellite.

The following sections include the verification of some pertinent requirements that were extracted from standards applicable for space systems (ECSS-Q-ST-80C [27] and ECSS-E-ST-40C [28]). Therefore, the RL agent shall comply with the requirements and guidelines of these standards. Section 6.1 represents the setup of the evaluation platform used for the tests. These tests include (i) the functional validation of the result in a real processor, (ii) analysis of automatically generated code, and (iii) evaluation of the technical budget in terms of processor, memory use, and timing analysis. Sections 6.2, 6.3, and 6.4 describe these three activities in detail, respectively.

6.1. Description of the embedded platform

In order to validate and verify the model, a representative version of the satellite OBSW was developed on top of the embedded system. One of the main purpose was to have a initial setup to explore the integration of neural networks in future space systems. The OBSW of the UPMSat−2 used the Ada programming language and the Open Ravenscar Real-Time Kernel (ORK) execution environment [29] on top of a LEON3 processor. These restrictions of the programming language ensured the temporal schedulability and predictability of the system. For the tests presented in this study, a similar platform was used, with a port of the ORK for the ARM Cortex-M4 processor. This is convenient as the UPMSat−3 will have a similar processor, although with more computational power.

In particular, the representative version of the satellite included the most relevant tasks whose periods and deadlines were those of the UPMSat−2 [8], as follows:

- Housekeeping: This task reads a temperature sensor (embedded in the board) and stores its value in the Storage protected object. It runs with a period of 1000 ms.
- Telemetry, Tracking, and Command: This task is responsible for the communication with the ground. It reads the temperatures from the Storage and sends it to the ground station by a Universal Asynchronous Receiver Transmitter (UART) interface. It runs with a period of 10 000 ms.
- ACS: This task executes the attitude control. Specifically, it imports the neural network code (autogenerated in C) and performs

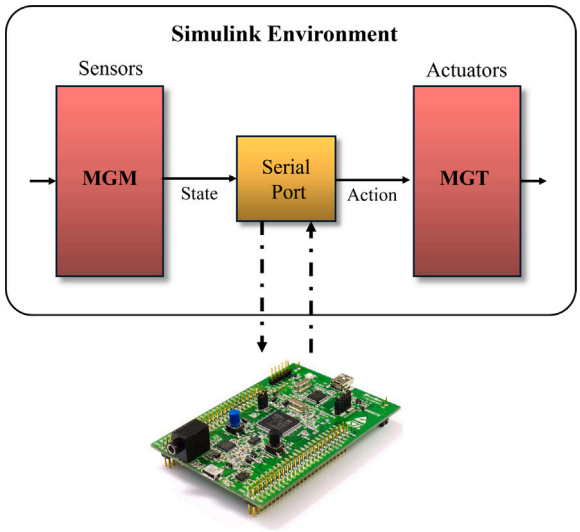


Fig. 13. Configuration of the validation on an embedded system. In the figure, MGM refers to magnetometers and MGT to magnetorquers.

an inference to the Actor. The inputs and outputs are transmitted via UART interface to the simulation environment. It runs with a period of 2000 ms with a deadline of 100 ms, as described in the control loop (Fig. 3).

Fig. 13 illustrates the setup of the test. The board runs the OBSW and is connected via serial port to a host computer, which runs the Simulink simulation model of the UPMSat−2. This configuration allows us to simulate the physical environment found in orbit and exchange the state and actions with the neural network executing on the board.

6.2. Functional validation

The ECSS-Q-ST-80 standard [27] (Section 6.3.5) includes requirements for testing and validation. In this subsection, we describe the functional tests, in which the behavior of the system is compared to the requirements baselines. As a remainder, the main requirement of the UPMSat−2 ACS is to reach an angular velocity of $[0, 0, 0.1]\,\mathrm{rad\,s^{-1}}$. This functional requirement was previously verified at the model level in MATLAB as part of the evaluation in each iteration. The next step is to verify that the RL agent maintains the functional validity executing in a real-time embedded system. In the literature, this step is commonly known as processor-in-the-loop validation.

The validation results in Fig. 14 show that the performance of the agent in the embedded system is similar to the results obtained in the simulation environment. In both cases, the angular velocity is stabilized in the three axis to the desired values. In the case of the Z axis, the same oscillations are seen around the setpoint value, but again, they are negligible as they are slow and confined between $\pm 0.05\ \mathrm{rad\,s^{-1}}$. This test allowed us to verify that the functional behavior of the RL model is still valid after embedding it on a real processor.

6.3. Evaluation of the automatic generated code

The MATLAB automatic code generation tool was used to transform the RL agent into C code to facilitate its deployment in a embedded system. In this context, the ECSS-Q-ST-80 standard [27] (Section 6.2.8) requires evaluating code metrics and tool configuration, including parameters for customization and its compliance with standards. This subsection includes the explanation of the tools and characteristics used for those purposes.

Table 8
Static analysis results of the autogenerated code from the neural network.

File name	LOC	Stmts.	Max Comp.	Max Depth
callPredict.c	43.303	42	7	2
FullyConnectedLayer.c	70	18	2	2
make_prediction.c	68	23	3	2
make_prediction_data.c	19	3	0	0
make_prediction_initialize.c	30	8	1	1
make_prediction_rtwutil.c	277	176	15	5
make_prediction_terminate.c	29	7	1	1
predict.c	32	6	1	1
rt_nonfinite.c	70	18	1	1
rtGetInf.c	58	10	1	1
rtGetNaN.c	43	6	1	1
SoftmaxLayer.c	70	38	14	4

The controller obtained in Section 5 is a RL agent that contains a neural network (the Actor) capable of effectively controlling the satellite's rotation. After having validated its behavior in a simulated environment, the next step would involve transforming this model into source code that can be deployed on the embedded computer. As previously stated, MATLAB provides the functionality to automatically transform the neural network into C99 source code. The code generator was configured to comply with the MISRA C:2012 coding standard from the Motor Industry Software Reliability Association (MISRA) to restrict the usage of unsafe coding practices like dynamic memory and implicit type casting. The tool was also configured to protect against arithmetic exceptions like division by zero.

The aforementioned standard and ECSS-E-ST-40C [28] (section 5.8) propose verification of code activities that should be performed to obtain a evaluable collection of metrics. The static analysis tool SourceMonitor was used to obtain the values of those metrics for all the 12 files generated by MATLAB. Results can be observed in Table 8. The analyzed metrics are lines of code (LOC), statements (Stmts., lines that end with a semicolon), maximum complexity (Max Comp., maximum number of control statements in a function like *if/else*) and maximum depth (Max Depth, maximum number of nested control statements in a function). For more information on these metrics, the reader is recommended to read the software metrication guidelines from the ECSS-Q-HB-80-04A handbook [30].

The ECSS-Q-HB-80-04A handbook also includes a set of reference values for those metrics based on experiences from real projects and depending on the criticality category of the analyzed code, from the least (level D, minor or negligible) to the most critical (level A, catastrophic). The proposed values are included in Table 9 and should be considered as indicators as they may not be appropriate for every project. For instance, the LOC metric in the neural network generated code does not match the estimated values. In particular, the file "callPredict.c" has 43.303 lines, which is larger than the range suggested in the handbool (between 50 and 75). In any case, the code generator toolset should be qualified for use in a safety system.

Regarding the maximum complexity, "make_prediction_rtwutil.c" is the most complex file with a value of 15, which means that there is a function that contains 15 control statements in its code. This kind of statements increases the level of complexity of the code as more possible paths are introduced in the execution of the program. Although the value obtained is not very high, it would only satisfy the criticality category C, referring to critical systems that could cause a degradation in mission quality. In the case of the maximum depth or nesting level, the code in the "make_prediction_rtwutil.c" file has a peak value of 5. This satisfies all levels of criticality. As stated before, compliance with these values should be taken as an indicator, not a mandatory requirement.

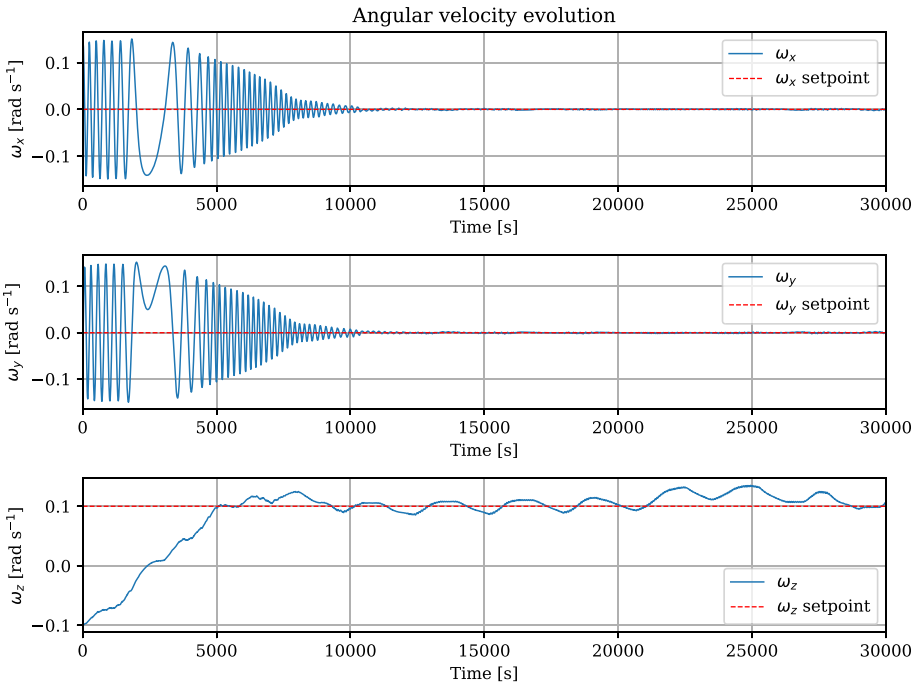


Fig. 14. Results of the processor-in-the-loop validation performed with the STM32F407.

Table 9
Reference values for the software metrics proposed in ECSS-Q-HB-80-04A [4].

Metric	Criticality category			
	A	B	C	D
LOC	50	50	50	75
Max. Comp.	10	10	15	20
Max. Depth	5	5	5	7

6.4. Technical budget and timing analysis

The ECSS-E-ST-40C standard [28] (section 5.8.3) indicates that the software product shall be analyzed in terms of memory size and processor utilization to ensure a predictable behavior. For this reason, the code footprint and timing characteristics must be checked and compared with the margins of the system. Firstly, regarding memory bounds, the executable occupied approximately 370 kB of memory. With the UPMSat−2 OBC having 2 MB [8] of memory, the code is well suited in terms of size. In addition, UPMSat−3 has more powerful features with up to 32 MB of memory, so the size of the experiment should not be a problem in both cases.

Secondly, with regards to the timing analysis, the Open Toolbox for Adaptive WCET Analysis (OTAWA) [31] tool was used to calculate the Worst-Case Execution Time (WCET) for the execution of the neural network. This tool analyzes the executable statically and returns the number of processor cycles that it takes to follow the longest execution path. OTAWA is open source and straightforward tool, but it tends to be pessimistic as it considers the theoretical longest combination of paths, even if in practice they are never going to be executed. However, they are useful to ensure a top limit execution time that will not be surpassed. In the case of the ACS, the number of cycles obtained for the longest theoretical path was 12.421.975. Knowing that the processor has a frequency of 168 MHz (cycles per second), the resulting WCET with OTAWA was 73.9 ms. To obtain a more realistic approximation, a dynamic test was also performed by measuring the execution time of thousands of inferences to the neural network in the real processor. The resulting average maximum execution time was 9.72 ms.

Compared with the control cycle shown previously in Fig. 3, the controller had a period of 2000 ms and a deadline $D_{\text{ACS}} = 100$ ms to decide the actions applied to the magnetorquers. In the representative version of UPMSat−2 that was made for these validation and verification activities, there are no interferences between the ACS task and the other two, so the response time of the ACS task (R_{ACS}) is considered to be the same as its WCET ($C_{\text{ACS}} = 73.9$ ms), therefore $R_{\text{ACS}} = 73.9$ ms. Since the response time of the task is less than the deadline $R_{\text{ACS}} < D_{\text{ACS}}$, the system would execute in time and can be considered as schedulable. In future experiments, when the UPMSat−3 system is available, a detailed response analysis should be performed with all tasks, similar to the study by Zamorano and Garrido [23]. In addition, the WCET is expected to decrease significantly due to the better performance of the processor in the UPMSat−3 and the use of more qualified toolsets such as RapiTime. These toolsets provide a safe upper bound on the WCET while minimizing pessimism.

The results obtained are very promising, providing initial insights into the feasibility of using neural networks in safety-critical systems. The most important result of this analysis is that the obtained RL agent was proven to be both timely and spatially bounded and deterministic, while keeping the functional behavior executing in an embedded system. This time and memory determinism is a necessary condition for the schedulability of the system.

7. Conclusions

This paper demonstrates the successful application of a RL agent in the ACS of an experimental satellite, achieving the desired control behavior. Beyond the results obtained, the decision-making process and design challenges faced during development are also documented. First, the reasons for the choice of inputs for the neural network and data pre-processing were introduced. Second, the selection of an appropriate reward function with a detailed explanation of its mathematical representation and its impact on agent performance. Third, a comparison between continuous and discrete action spaces was made, concluding that, for this case study, discretizing the agent's actions obtained better results. Fourth, a slight modification of the reward function using a moving average filter led to the final model with promising results. In addition, each step is supported by both graphical and quantitative metrics to validate the agent's performance.

Lastly, the paper includes a preliminary validation test in which the neural network was executed on an embedded system. The trained agent was transformed into a lightweight, standalone C program using MATLAB's automatic code generation tool. Not only was the generated code proved to perform as well as the simulation, but it was capable of running within the limited resources of the embedded board by means of time analysis, memory and processor utilization. The authors acknowledge that these positive results would help in the development of future systems, although more research is needed to study the integration of ML techniques into safety-critical systems.

In conclusion, this paper is intended to be a beneficial resource for similar works combining AI/ML with embedded control systems, as it provides insights and lessons learned throughout the process of developing an effective model. While the proposed solutions may not be universally applicable to all RL problems, the authors believe that many of the ideas presented can be adapted and reused in related contexts.

8. Future work

The presented work focused on the development and validation of an RL attitude controller in a simulation environment and an embedded computer. A further step towards full verification and validation would be to perform the hardware-in-the-loop validation. This would allow us to verify the behavior of the neural networks in a physical platform; including sensors and actuators, electronic boards, and test beds simulating conditions found in orbit. Work is underway to prepare the dynamic test bed available at IDR-UPM facilities for the attitude control of the future UPMSat−3. In addition, it would be interesting to incorporate multiple objectives as part of the reward function, including energy consumption and generation. In this way, the agent would learn to control the attitude while optimizing performance. Another interesting research line is to generalize the lessons learned in this paper so that they can be applied as a systematic methodology for other safety-critical and real-time control systems. Although the work presented already serves as a basis for the future development of the ACS in UPMSat−3, obtaining a generic methodology will be valuable for studies other than attitude control.

CRediT authorship contribution statement

Ángel-Grover Pérez-Muñoz: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Guillermo López-García:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Investigation, Formal analysis, Data curation. **Irene García-Villoria:** Writing – review & editing, Writing – original draft, Visualization, Software, Investigation, Data curation. **Alejandro Alonso:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Investigation, Funding acquisition, Conceptualization. **Angel Porras-Hermoso:** Visualization, Validation, Resources, Conceptualization. **María S. Pérez:** Visualization, Validation, Supervision, Resources, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was mainly developed under the project OAPES-CM "Operación Avanzada de Pequeños Satélites" (Ref.: Y2020/NMT-6427) and PRESECREL (PID2021-124502OB-C43). The authors also acknowledge the financial support of the Ministerio de Ciencia e Innovación, Spain, and the Comunidad de Madrid Proyectos Sinérgicos from the I+D plan (Spain), as well as the collaboration with the partners in these projects.

References

[1] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, first ed., The MIT Press, 2016.

[2] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, second ed., A Bradford Book, Cambridge, MA, USA, 2018.

[3] F. Tambon, G. Laberge, L. An, A. Nikanjam, P.S.N. Mindom, Y. Pequignot, F. Khomh, G. Antoniol, E. Merlo, F. Laviolette, How to certify machine learning based safety-critical systems? A systematic literature review, Autom. Softw. Eng. 29 (2) (2022) 38, http://dx.doi.org/10.1007/s10515-022-00337-x.

[4] ECSS, ECSS-E-HB-40-02A. Space Engineering: Machine Learning Qualification Handbook, ESA Requirements and Standards Section, Noordwijk, 2024, URL: https://ecss.nl/home/ecss-e-hb-40-02a-15-november-2024/, (Accessed 14 November 2024).

[5] EASA, Daedalean, Concepts of design assurance for neural networks (CoDANN) II with appendix b, 2024, URL: https://www.easa.europa.eu/en/downloads/128161/en, (Accessed 5 August 2024).

[6] P.W. Fortescue, G.G. Swinerd, Attitude control, in: Spacecraft Systems Engineering, John Wiley & Sons, Ltd, 2011, pp. 289–326, http://dx.doi.org/10.1002/9781119971009.ch9.

[7] J.A. de la Puente, J. Garrido, J. Zamorano, A. Alonso, Model-driven design of real-time software for an experimental satellite, IFAC Proc. Vol. 47 (3) (2014) 1592–1598, http://dx.doi.org/10.3182/20140824-6-ZA-1003.01967, 19th IFAC World Congress.

[8] J. Zamorano, J. Garrido, J. Cubas, A. Alonso, J.A. de la Puente, The design and implementation of the UPMSAT-2 attitude control system, IFAC Pap. 50 (1) (2017) 11245–11250, http://dx.doi.org/10.1016/j.ifacol.2017.08.1607, 20th IFAC World Congress.

[9] J. Elkins, R. Sood, C. Rumpf, Autonomous spacecraft attitude control using deep reinforcement learning, 2020, URL: https://ntrs.nasa.gov/api/citations/20205008891/downloads/elkins_iac_RLADCS_v2_2_reformat.pdf.

[10] R. Su, F. Wu, J. Zhao, Deep reinforcement learning method based on DDPG with simulated annealing for satellite attitude control system, in: 2019 Chinese Automation Congress, CAC, 2019, pp. 390–395, http://dx.doi.org/10.1109/CAC48633.2019.8996860.

[11] W. Retagne, J. Dauer, G. Waxenegger-Wilfing, Adaptive satellite attitude control for varying masses using deep reinforcement learning, Front. Robot. AI 11 (2024) http://dx.doi.org/10.3389/frobt.2024.1402846.

[12] A. Barzegar, D.-J. Lee, Deep reinforcement learning-based adaptive controller for trajectory tracking and altitude control of an aerial robot, Appl. Sci. 12 (9) (2022) http://dx.doi.org/10.3390/app12094764, URL: https://www.mdpi.com/2076-3417/12/9/4764.

[13] J.-A.R. Sarmiento, V.H. Tan, M.C.R. Talampas, P.C. Naval, Sample efficient deep reinforcement learning for diwata microsatellite reaction wheel attitude control, Aerosp. Syst. 6 (1) (2023) 61–69, http://dx.doi.org/10.1007/s42401-022-00169-3.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, arXiv:1707.06347, https://arxiv.org/abs/1707.06347.

[15] J.T. Allison, M. West, A. Ghosh, F. Vedant, Reinforcement learning for spacecraft attitude control, in: Proceedings of the International Astronautical Congress, IAC, International Astronautical Federation, 2019, pp. IAC–19–C1.5.2.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, http://dx.doi.org/10.48550/arXiv.1707.06347.

[17] J. Allison, M. West, A. Ghosh, F. Vedant, Reinforcement learning for spacecraft attitude control, 2019, URL: https://par.nsf.gov/biblio/10156483.

[18] Z. Ma, Y. Wang, Y. Yang, Z. Wang, L. Tang, S. Ackland, Reinforcement learning-based satellite attitude stabilization method for non-cooperative target capturing, Sensors 18 (12) (2018) http://dx.doi.org/10.3390/s18124331.

[19] D. Yadava, R. Hosangadi, S. Krishna, P. Paliwal, A. Jain, Attitude control of a nanosatellite system using reinforcement learning and neural networks, in: 2018 IEEE Aerospace Conference, 2018, pp. 1–8, http://dx.doi.org/10.1109/AERO.2018.8396409.

[20] S. Oghim, J. Park, H. Bang, H. Leeghim, Deep reinforcement learning-based attitude control for spacecraft using control moment gyros, Adv. Space Res. (2024) http://dx.doi.org/10.1016/j.asr.2024.07.078.

[21] A. Porras-Hermoso, J. Piqueras, J. Cubas, E. Roibás-Millán, On-orbit performance analysis of spinning spacecraft magnetic control laws. Application to the upmsat-2 mission, Measurement 225 (2024) 113962, http://dx.doi.org/10.1016/j.measurement.2023.113962, URL: https://www.sciencedirect.com/science/article/pii/S0263224123015269.

[22] J. Cubas, A. Farrahi, S. Pindado, Magnetic attitude control for satellites in polar or sun-synchronous orbits, J. Guid. Control. Dyn. 38 (2015) 1947–1958, http://dx.doi.org/10.2514/1.G000751.

[23] J. Zamorano, J. Garrido, Schedulability analysis of PWM tasks for the upmsat-2 ADCS, in: J.A. de la Puente, T. Vardanega (Eds.), Reliable Software Technologies – Ada-Europe 2015, Springer International Publishing, Cham, 2015, pp. 85–99, http://dx.doi.org/10.1007/978-3-319-19584-1_6.

[24] Mathworks, Reinforcement learning toolbox, 2024, URL: https://www.mathworks.com/products/reinforcement-learning.html, (Accessed 29 November 2024).
[25] K.M. Patel, A practical reinforcement learning implementation approach for continuous process control, Comput. Chem. Eng. 174 (2023) 108232, http://dx.doi.org/10.1016/j.compchemeng.2023.108232.
[26] G. López García, Diseño, desarrollo y validación de un modelo de aprendizaje automático para el control del satélite UPMSat-3 (Master's thesis), Universidad Politécnica de Madrid – Escuela Técnica Superior de Ingenieros Informaticos, 2024, URL: https://oa.upm.es/82862/, (Accessed 3 February 2025).
[27] ECSS, ECSS-q-ST-80c rev.1 – software product assurance, 2017, URL: https://ecss.nl/standard/ecss-q-st-80c-rev-1-software-product-assurance-15-february-2017/, (Accessed 11 September 2024).
[28] ECSS, ECSS-e-ST-40c – software (6 march 2009), 2017, URL: https://ecss.nl/standard/ecss-e-st-40c-software-general-requirements/, (Accessed 14 April 2025).
[29] J.A. de la Puente, J.F. Ruiz, J. Zamorano, An open ravenscar real-time kernel for GNAT, in: H.B. Keller, E. Plödereder (Eds.), Reliable Software Technologies Ada-Europe 2000, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 5–15.
[30] ECSS, ECSS-q-HB-80-04a – software metrication handbook (30 march 2011), 2011, URL: https://ecss.nl/hbstms/ecss-q-hb-80-04a-software-metrication-handbook/, (Accessed 22 April 2025).
[31] C. Ballabriga, H. Cassé, C. Rochange, P. Sainrat, OTAWA: An open toolbox for adaptive WCET analysis, in: S.L. Min, R. Pettit, P. Puschner, T. Ungerer (Eds.), Software Technologies for Embedded and Ubiquitous Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 35–46, http://dx.doi.org/10.1007/978-3-642-16256-5_6.

Ángel-Grover Pérez-Muñoz is an Assistant Professor at the Escuela Técnica Superior de Ingenieros Informáticos at the Universidad politécnica de Madrid. He is also a researcher at the Information Processing and Telecommunications Center (IPTC-UPM). He is currently pursuing a Ph.D. in Computer Science from the UPM. His main research fields include real-time embedded systems, software for aerospace systems, artificial intelligence, and safety-critical systems.

Guillermo López-García is a researcher at the Universidad Politécnica de Madrid (UPM) for the STRAST-IPTC group. He received his B.Sc. and M.Sc. degrees in Computer Science from UPM in 2022 and 2024, respectively. Currently, his main research interests include the interdisciplinary use of Artificial Intelligence, especially Reinforcement Learning, in real-time embedded systems.

Irene García-Villoria was born in Madrid, Spain, where she completed her education. She holds both a bachelor's and a master's degree in Electrical and Computer Engineering, focused on machine learning and telecommunications, from the Universidad Politécnica de Madrid. She has participated in a research project on artificial intelligence for space systems and coauthored both a scientific paper on reinforcement learning for satellite attitude control and a book chapter on machine learning in safety-critical systems. She completed an academic exchange at the University of the Technischen Universität Wien and has professional experience as a data analyst.

Alejandro Alonso is a Full Professor at the Information Processing and Telecommunications Center of the Technical University of Madrid (IPTC-UPM). His current research interests are in real-time and embedded systems, including design methodologies, software architectures, resource management, and operating systems. He has participated in several EU-funded projects, as well as national government and industry-funded research projects, such as EICACS, MultiPARTES, OAPES, HIJA, PRESECREL or RoboCop. He is co-author of more than 100 technical papers and book chapters.

Angel Porras-Hermoso received his Ph.D. title in 2024 and started a career in the academia. He currently holds the position of assistant professor at the Technical School of Aerospace Engineering at the Universidad Politécnica de Madrid. His main fields of study encompass the study of attitude determination and control algorithms for small spacecrafts, the study and design of the electrical subsystem in satellites and determination of orbit and parameters of space debris. He has participated in different small satellite missions as an ADCS engineer including the participation in a stratospheric balloon mission.

María S. Pérez is a Full Professor with the Universidad Politécnica de Madrid. She is also a member of the Big Data Value Association Board of Directors. She has coauthored four books and seven book chapters. She has published more than 100 articles in international journals and conferences. She has been involved in the organization of several workshops and conferences. She has participated in a large number of EU and Spanish R&D projects. Her research interests include artificial intelligence, big data, HPC, and large-scale computing. She is serving as a program committee member for many relevant conferences.