Received 11 September 2023, accepted 27 October 2023, date of publication 6 November 2023,
date of current version 20 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3330441

RESEARCH ARTICLE

Implementation and Performance Study of the Micro-ROS/ROS2 Framework to Algorithm Design for Attitude Determination and Control System

ALFREDO MAMANI-SAICO[1] AND PABLO RAUL YANYACHI[2], (Senior Member, IEEE)
[1]Electronic Engineering Professional School, Universidad Nacional de San Agustin de Arequipa, Arequipa 04000, Peru
[2]Instituto de Investigación Astronómico y Aeroespacial Pedro Paulet, Universidad Nacional de San Agustin de Arequipa, Arequipa 04000, Peru

Corresponding author: Pablo Raul Yanyachi (raulpab@unsa.edu.pe)

This work was supported by Programa Nacional de Investigación Científica y Estudios Avanzados (PROCIENCIA-CONCYTEC), for the Development of this Research Project under Agreement PE501079885-2022.

ABSTRACT Robot Operating System 2 (ROS2) and Micro-ROS, as advanced robotic platforms, present a potent fusion of versatility, efficiency, and scalability. While ROS 2 continues to evolve with a modular architecture and flexible communication framework, Micro-ROS provides a lightweight and efficient solution tailored for resource-constrained systems. Together, they extend the horizons of robotics across a spectrum of applications, ranging from autonomous robots to highly constrained environments, thus setting a comprehensive standard in the realm of contemporary robotic system development. In this study, we leverage the capabilities of ROS2 and Micro-ROS to realize an experimental platform designed for the development of attitude control algorithms. This platform comprises a dedicated Computer Environment Dedicated to Data Processing (CEDDP) and a module for Attitude Determination & Control System (ADCS). Our investigation entails a measurement of the response time of the CEDDP unit under both exclusive and shared network usage scenarios. Furthermore, we quantify the extent of message loss across three segments during a complete control-task cycle, under various Quality of Service (QoS) configurations. Additionally, we analyze message periodicity at four key points within the ROS2 entities participating in the system. The outcomes of our experimentation reveal a robust experimental platform built upon the ROS2 and Micro-ROS frameworks. Employing a best-effort QoS policy, along with minor adjustments to QoS profiles, emerges as an optimal approach for designing attitude control algorithms.

INDEX TERMS Data distribution service, CubeSat, quality of service, real-time processing, software architecture, ROS2.

I. INTRODUCTION

The Attitude Determination & Control System (ADCS) is a fundamental component of aerospace and robotics engineering, responsible for accurately determining and maintaining the orientation of a vehicle or object in space. It combines sensors, algorithms, and actuators to manage rotational aspects, ensuring precise alignment with desired reference axes for optimal performance and functionality [1]. Several aerospace research topics focus on addressing dynamic models and attitude control problems. For instance, in [2], a model of an artificial satellite is constructed, comprising both a rigid and a flexible component. Subsequently, [3] entails an evaluation of three robust control laws proposed by Boskovic, Dando, and Chen. Furthermore, [4] involves the implementation of algorithms rooted in the theory of the 3D inverted pendulum, employing quaternions as a fundamental framework. However, one of the critical elements in designing attitude control algorithms is the software, proper design and implementation of this software is essential to ensure extensible and reliable operation of the ADCS, since implementing redundant systems is not always the optimal choice [5].

On another note, Robot Operating System 2 (ROS2) emerges as an open-source software platform meticulously engineered with a specific focus on facilitating the

The associate editor coordinating the review of this manuscript and approving it for publication was Alessandro Floris.

development of robotic systems, now extended to encompass industrial environments as well [6]. Since 2018, a progression known as Micro-ROS has been under development, representing a variant of ROS2 tailored for microcontrollers. Its primary objective revolves around bridging the divide between resource-constrained microcontrollers and more robust processors within robotic applications [7]. The transition from ROS to ROS2 has been intricately orchestrated to cater to the demands of contemporary robotic systems. Among the most noteworthy enhancements is the incorporation of the Data Distribution Services (DDS) standard, which significantly enhances real-time data exchange capabilities [8]. DDS functions as middleware, providing the communication layer for both ROS2 and Micro-ROS. Furthermore, a supplementary extension, SROS2, has been introduced to bolster system security through the implementation of authentication, encryption, and access control mechanisms [9]. Additionally, ROS2 introduces heightened control over Quality of Service (QoS), allowing for tailored adaptations to varying performance and reliability requirements [10].

A. RELATED WORKS

The study conducted by Buckner et al. [11] presents one of the initial endeavors that employs the Robot Operating System as a flight software framework for nanosatellite control. This application has been demonstrated in the context of two CubeSat platforms: the Drag De-Orbit Device (D3) and the Passive Thermal Coating Observatory Operating in Low Earth Orbit (PATCOOL) [12]. According to Gonzalez et al. [13], a significant portion of publications detailing CubeSat mission flight software predominantly utilize operating systems such as GNU/Linux or FreeRTOS. Within this context, Micro-ROS has achieved native compatibility with a selection of Real-Time Operating Systems (RTOS), including FreeRTOS, Zephyr, NuttX, as well as Linux and Windows [14]. Consequently, our particular focus resides in the exploration of studies pertaining to the implementation of ROS2 with enhancements in data exchange mechanisms and performance analysis. These themes hold profound importance as we endeavor to establish a reliable attitude control software framework. Subsequently, we present a collection of proposals and outcomes elucidated by various researchers:

- Fernandez et al. [15] have conducted performance studies of ROS2 concerning latency, throughput, and overhead within a two-node network encompassing 50 distinct QoS and cybersecurity configuration scenarios.
- Chen [16] evaluates the impact of combining diverse QoS and security configurations in ROS2, employing a node network analogous to the operation of an unmanned asset group. This study revealed that enabling security led to increased message drop rates across all QoS profiles, and scaling the network to more nodes yielded various consequences contingent upon different QoS configurations.
- The work by Fernandez [17] demonstrated measurable latency and performance variations among different QoS profiles and security settings.
- Park et al. [18] assessed the periodicity of a multi-agent service robot control node in both ROS2 and ROS, indicating that ROS2 holds the potential to meet real-time constraints.
- Jalil et al. [19] analyze the efficiency of local caching, cache management, and QoS balancing optimization to enhance ROS2 node communication when utilizing *RELIABLE* and *KEEP_LAST* options within the Aggregated Robot Processing (ARP) architecture. This study effectively improved latency and reduced packet loss, although susceptibility to low-level hardware impacts was noted.

B. SOFTWARE REQUIREMENTS

The functional and non-functional requirements for software design, as previously identified in the study conducted by Gonzales et al. [13], are outlined in Table 1. These requirements encompass five quality attributes pertinent to software design in attitude control and future space missions.

C. PURPOSE OF OUR STUDY

The aim of this study is to implement and evaluate the performance of Micro-ROS and ROS2 as a scalable software solution for prospective aerospace mission projects at the Instituto de Investigación Astronómico y Aeroespacial Pedro Paulet (IAAPP) of the Universidad Nacional de San Agustín de Arequipa (UNSA). The outcomes of this study will offer benefits to researchers and students at UNSA by enabling them to address attitude control challenges in CubeSats or replicate the architecture in emerging research centers as a cost-effective alternative. Furthermore, this study represents an enhancement of a previously developed prototype within the institute, wherein the simulation of a 1U Cubesat Dynamics and a testing system for students were executed, employing cost-effective components [21].

Regarding the technical purpose of our study, our aim is to fulfill real-time communication requirements between the ADCS module and the Computer Environment Dedicated to Data Processing (CEDDP). This objective would be achieved by identifying an appropriate configuration of Micro-ROS QoS policies. Micro-ROS/ROS 2 has been chosen over other frameworks due to its capability to operate efficiently in resource-constrained environments, strong interoperability, and its flexible, modular approach that enables the expansion of possibilities for implementing complex control software. Furthermore, the robust development community and technical support behind Micro-ROS/ROS 2 provide a solid foundation for implementing complex hardware control solutions, as required for our application. Our rationale is based on utilizing a CEDDP, such as a high-

TABLE 1. **Functional and non-functional software requirements.**

Requirements	Description	DDS Approach
Extensibility	It must be able to add new functionalities and commands without affecting the basic structure of the software.	DDS is a data-centric publish/subscribe communication protocol that facilitates open and flexible communication with unregistered or unknown entities [20].
Modularity	It has to be divided into independent modules that can be developed, tested and maintained separately.	By using ROS2 and Micro-ROS, this study illustrates the concept of modularity design in section II. The approach involves constructing systems for ADCS using independent modules.
Reliability	Must perform unit and integrated testing to ensure software reliability and detect potential bugs.	ROS enhances QoS management [15], and in Section III, we conduct a performance evaluation utilizing various QoS policies.
Portability & reusability	It must be able to be adapted to run on different hardware configurations without requiring significant modifications.	This study addresses the general components of a nanosatellite, so the architecture in ROS2 can be adjusted to CubeSats units containing ADCS and OBC (On-Board Computer).
Scalability	It must be scalable for the development and operation of large nanosatellite constellations.	By using ROS2, it can be extended to a network of multi-agent systems, with a central control unit.

performance computer, for intensive algorithm processing, while employing a microcontroller within the ADCS module as a means of data exchange among the components of the experimental platform. This approach facilitates the utilization of cost-effective and accessible hardware.

This article is structured as follows: Section II identifies the general components of the experimental platform and outlines the modular design of both hardware and software. Section III elaborates on our experimental configurations and techniques employed to gather latency, message loss, and data periodicity data. Section IV provides a comprehensive analysis of the measurements and presents the obtained results. Section V offers the conclusions derived from the study, along with insights into the direction of future projects.

II. MODULAR ARCHITECTURE DESIGN AND IMPLEMENTATION

The design of the overarching architecture comprises the following components: Detection, Actuation, Periphery, Control, and Visualization. Each of these components is distributed within the hardware architecture as depicted in Figure 1.

A. HARDWARE ARCHITECTURE

The distribution of the elements constituting the hardware of the system is illustrated in Figure 1. Two main blocks can be identified: the CEDDP block, encompassing control and visualization components, enabling state monitoring and command transmission; and the ADCS block, housing the detection, actuation, and periphery components, physically implemented within the ADCS module. These blocks are interconnected via a wireless link and are discernible by the resources available within their equipment. The ADCS module consists of components with limited resources and low cost, whereas the CEDDP can range from a laptop to a sophisticated server.

In this study, the detection component encompasses an Inertial Measurement Unit (IMU), providing orientation information, along with a voltage measurement sensor facilitating battery state evaluation. On the other hand, the actuation component includes reaction wheels that generate torque within nanosatellite units to achieve the desired orientation. Furthermore, the peripheral component incorporates backup devices and crucial status indicators for calibration purposes. To enable data exchange between the ADCS and CEDDP blocks, the ESP32 microcontroller and a remote computer are employed. Moreover, the microcontroller acts as a central conduit for the data flow among the ADCS components. The details and costs of the primary components employed in constructing the experimental platform are meticulously described in Table 2.

TABLE 2. **List of main components and costs of attitude control experimental platform.**

Component	Model/Detail	Quant.	Cost($)
Remote Computer	Intel Core i7-4790	1	1085
IMU	BNO-055	1	35
Microcontroller	ESP-WROOM-32.	1	7
Driver motor	DEC 24/2	3	45
Motors	Maxon EC45 flat	3	140
Battery	Ion Litio 18650	3	5.5
Display OLED	OLED 0.96"	1	7
SD Module	Micro SD Card	1	2.5
Step down Module	LM2596	1	2.5
Elevator Module	MOSFET BSS138	1	1
		Total	1711.5

B. SOFTWARE ARCHITECTURE

The architecture of the experimental software comprises a robust back-end and an intuitive graphical interface on the front-end. The back-end, associated with the detection, actuation, and control components, is developed within the ROS2 and Micro-ROS environments. It handles data exchange and sends control commands to the ADCS module. On the other hand, the front-end is exclusively linked to the visualization component and is responsible for visual presentation and user interaction, delivering a user-friendly and easily comprehensible experience. This modular architecture allows for a clear separation of responsibilities, streamlining software development, maintenance, and scalability.

1) FRONT-END SOFTWARE

Figure 2 illustrates the Graphical User Interface (GUI) developed for this study using Tkinter, a standard GUI

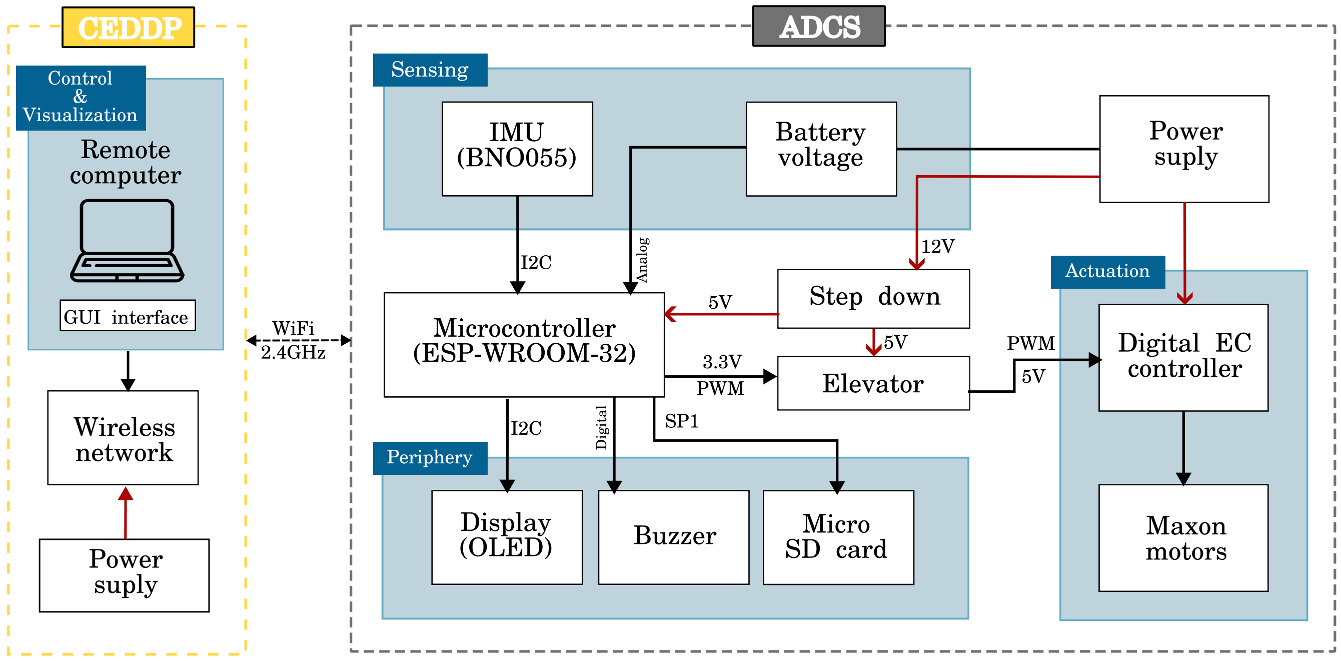


FIGURE 1. Experimental hardware distribution.

programming toolkit for Python. Through the GUI, the operator gains real-time access to data such as angular acceleration, quaternion orientation, Euler angles, ADCS module temperature, and battery status. Additionally, it includes a plotting section to visually assess the behavior of received orientation data and a 3D visual modeling section. The interface further presents two operational modes: "Test Mode", allowing the transmission of velocity commands to the ADCS module to verify proper reaction wheel functionality; and "Active Mode", enabling the reading of Pulse-Width Modulation (PWM) data and direction of rotation of the reaction wheel generated within the control component.

2) BACK-END SOFTWARE

The back-end serves as the internal architecture of the graphical interface, and its design has been realized with the objective of generating reliable velocity commands for three reaction wheels and establishing appropriate routes for the flow of each message type during data exchange. Figure 3 depicts the nodes, topics, and message types constituting the architecture. Two distinct node groups can be discerned:

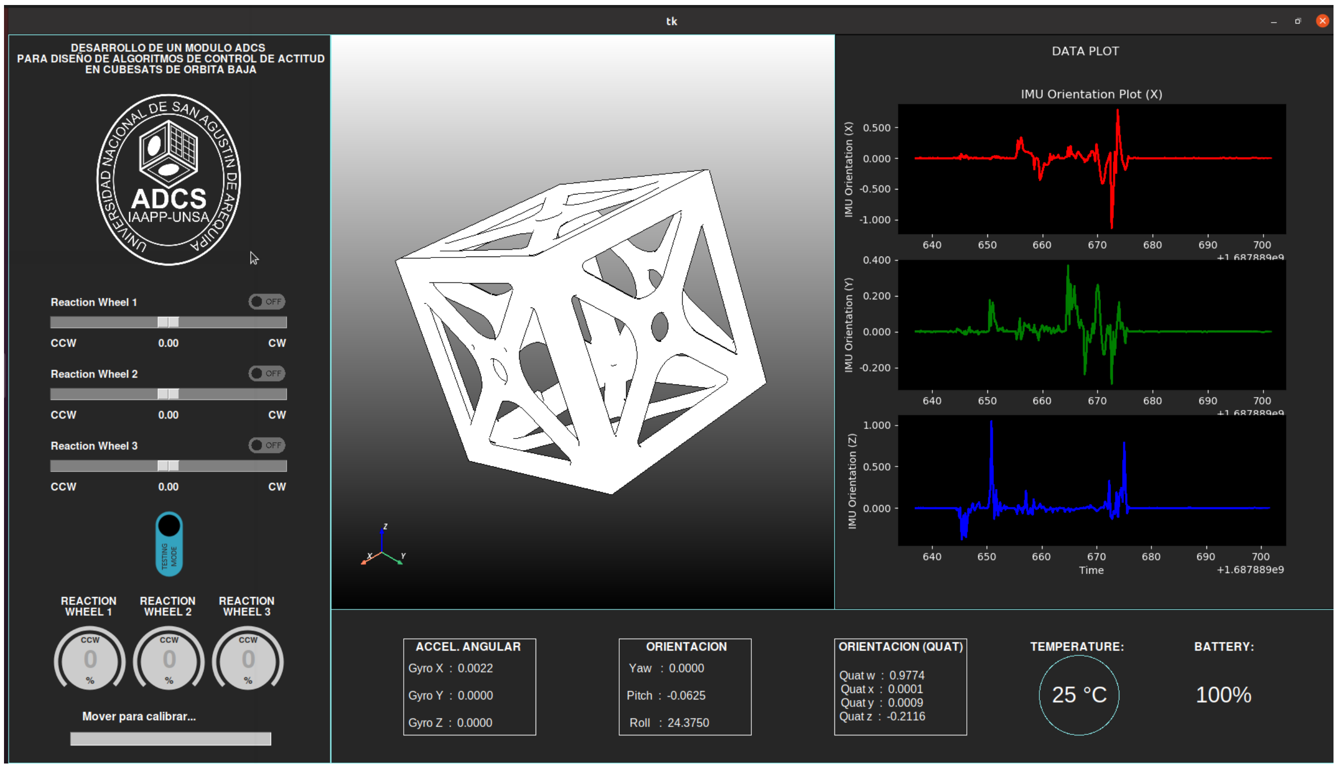
- The MICROROS_GROUP consists solely of the *adcs_node* as its node element, executed on the *ESP-WROOM-32* microcontroller. This node publishes battery state (*/batt_state*), calibration (*/calib_state*), temperature (*/temp*), and IMU values (*/imu*). It subscribes to the */pwm_topic*, which contains PWM values, rotation direction, and enable signal for each reaction wheel.
- The ROS2_GROUP comprises three nodes executed on a remote computer. The *control_node* implements the attitude control algorithm, publishing velocity commands when the interface is in "Active Mode". The *interface_node* publishes velocity commands exclusively when the interface is in "Test Mode". The *pwm_pub_node* conditions the velocity commands into a 12-bit resolution PWM signal, publishing it on the */pwm_topic* at a frequency of 100Hz.

III. EXPERIMENTAL SETUP

This section outlines the appropriate conditions under which the requisite data were acquired to analyze the performance of the proposed modular design, as detailed in the preceding section. To achieve this, techniques akin to those expounded upon in the prior study by Delgado et al. [22] were employed. In this context, performance analysis primarily hinges upon three pivotal metrics: periodicity, latency, and data loss.

Figure 4 illustrates the experimental setup stemming from the back-end architecture. For this instance, three nodes have been incorporated to facilitate the execution of a control task. The control task forms a closed-loop cycle encompassing functions denoted as A, B, C, and D. It commences with the transmission of IMU data and concludes with the reception of control signals (PWM and rotation direction). The execution of the attitude control algorithm transpires between points B and C.

Due to the absence of synchronized clocks between the microcontroller and the computer, measuring point-to-point latency between A-B and C-D poses a challenge. A compar-



ACCEL. ANGULAR
Gyro X : 0.0022
Gyro Y : 0.0000
Gyro Z : 0.0000

ORIENTACION
Yaw : 0.0000
Pitch : -0.0625
Roll : 24.3750

ORIENTACION (QUAT)
Quat w : 0.9774
Quat x : 0.0001
Quat y : 0.0009
Quat z : -0.2116

FIGURE 2. Graphical user interface.

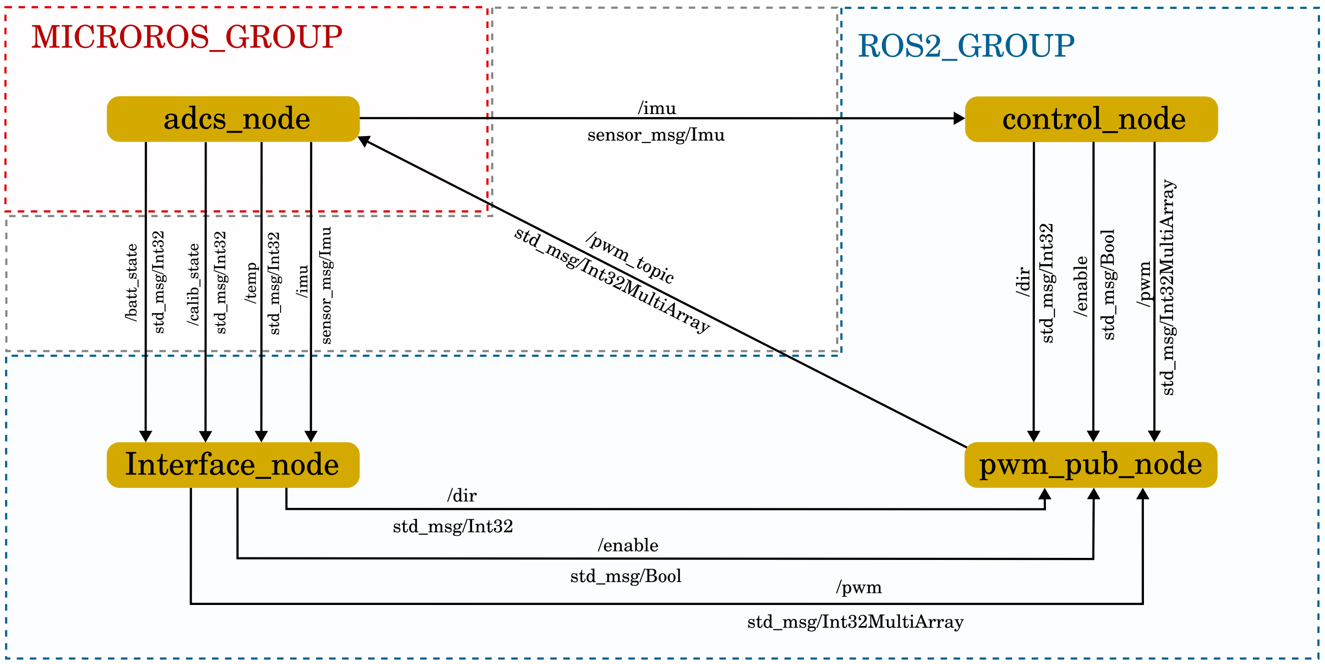


FIGURE 3. Experimental software architecture.

ative analysis of the latency of the ESP32 microcontroller, employed in this study, is elaborated upon in [23].

A. QUALITY OF SERVICE SETTINGS

The implementation of DDS in Micro-ROS, known as Micro XRCE-DDS, allows for configurations related to memory management. Given that these devices possess limited memory resources, it prioritizes the use of static memory over dynamic memory. Consequently, this imposes an upper limit on memory allocations that the user can configure. The Micro XRCE-DDS Client empowers users to create entities for reliable (RELIABLE) or efficient (BEST EFFORT) communications. This is achieved through functions such as `rclc_publisher_init_default`

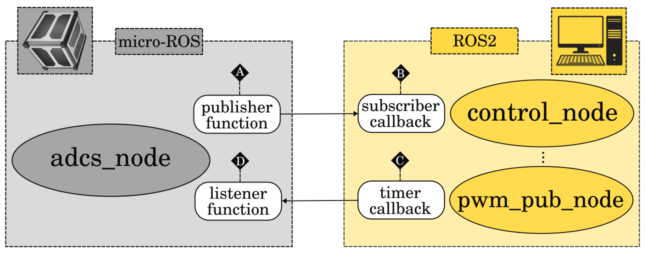


FIGURE 4. Experimental setup.

and `rclc_publisher_init_best_effort`. However, users can customize their own QoS using the `rmw_qos_profile_t` structure. Similarly, within ROS2 entities, QoS profiles can be customized using the `rclpy.qos` Python module.

TABLE 3. Setup QoS in DEFAULT and BEST EFFORT.

	Setup 1	
Function QoS profile	Publisher function (A) REALIABLE	Suscriber callback (B) RELIABLE
Function QoS profile	Listener function (D) RELIABLE	Timer callback (C) RELIABLE
	Setup 2	
Function QoS profile	Publisher function (A) BEST EFFORT	Suscriber callback (B) BEST EFFORT
Function QoS profile	Listener function (D) BEST EFFORT	Timer callback (C) RELIABLE
	Setup 3	
Function QoS profile	Publisher function (A) CUSTOM	Suscriber callback (B) CUSTOM
Function QoS profile	Listener function (D) CUSTOM	Timer callback (C) CUSTOM

In this study, we tested three QoS configurations classified in the Micro-ROS tutorials available at https://micro.ros.org/docs/tutorials/programming_rcl_rclc/pub_sub/. As outlined in Table 3, the first configuration utilized the default (RELIABLE) profile for all participating entities. In the second configuration, the BEST EFFORT profile was employed for the Micro XRCE-DDS entities. However, due to reliability compatibility between QoS, the BEST EFFORT profile had to be employed in the ROS2 subscriber callback function. For the third configuration, each entity in both Micro XRCE-DDS and ROS2 was customized in accordance with Table 4. The QoS parameters are defined as follows:

- History: Determines how received samples are stored, and can be set as either *KEEP_LAST* or *KEEP_ALL*.
- Depth: Specifies the size of the queue and is only valid if the "History" policy is configured as *KEEP_LAST*.
- Reliability: Defines how sample delivery is ensured, and can be set as either *RELIABLE* or *BEST EFFORT*.
- Durability: Defines whether samples persist for late subscriptions and can be set as either *TRANSIENT_LOCAL* or *VOLATILE*.

TABLE 4. Setting custom QoS profile.

Function	Parameter	Custom Setting
ALL	History Depth Reliability Durability	KEEP_LAST 2 BEST_EFFORT VOLATILE

IV. ANALYSIS AND RESULTS

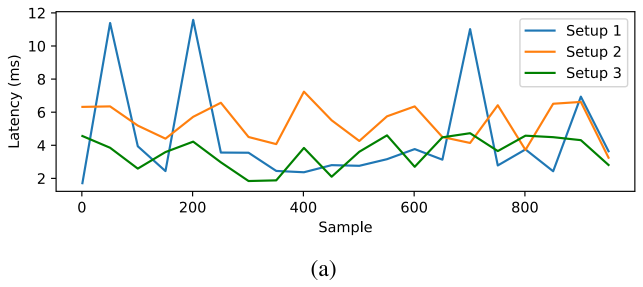
This section analyzes latency, message loss, and periodicity behavior, considering various configurations in which QoS profiles are varied. It is important to note that the XRCE-DDS middleware implementation is recent and still under development. Therefore, this evaluation aims to elucidate the current capabilities of Fast DDS and Micro XRCE-DDS, communication mechanism between ROS2 and Micro-ROS developed by eProsima, within an experimental software framework for control algorithm design. The eProsima Micro XRCE-DDS library, designed for environments with extremely constrained resources, acts as a bridge that enables communication between devices with resource constraints and the DDS network. This client-server protocol facilitates the participation of devices with limited resources in DDS communications, with the eProsima Micro XRCE-DDS Agent acting as an intermediary to enable this communication. Additionally, eProsima Micro XRCE-DDS provides an API layer that allows users to efficiently implement plug-and-play eProsima Micro XRCE-DDS clients. The computational environment resources utilized in the evaluation are specified in Table 5.

TABLE 5. Evaluation environment.

		CEDDP	ADCS
CPU	Model number Frecuency Cores Op-mode(s)	Intel Core i7-4790 3.60GHz 8 32-bit, 64-bit	Xtensa 32-bit LX6 240MHz 2 32-bit
Memory Network ROS2 Distribution OS Distribution DDS Used		7.6GB 100Mbps Foxy Fitzroy Ubuntu 20.04 LTS Fast DDS	4 MB SPI flash 150 Mbps (20 dBm) Micro-ROS - Micro XRCE-DDS

A. LATENCY

We evaluated the latency between points B and C, as depicted in Figure 4. In this evaluation, we measured the response time of the CEDDP unit from the moment it acquires the network topic to the point it sends control commands to a new topic. Measurements were conducted in two scenarios for each configuration type described in Table 3. In the first scenario, exclusive network usage was considered, where only the computer and the ESP-WROOM-32 microcontroller were connected to the network. In the second scenario, network sharing was considered with other common devices like additional computers and mobile phones. Additionally, latency measurements were recorded every 50 samples over a period of 10 seconds after subjecting the system to a stress load for 5 minutes. This approach allowed us to obtain more representative data and analyze latency behavior under both intensive and sustained usage conditions.



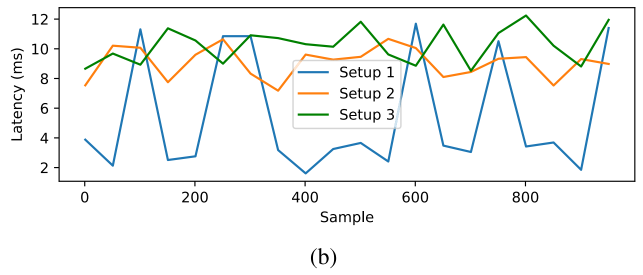


FIGURE 5. CEDDP response time: (a) Latency without network traffic; (b) Latency with network traffic.

Figure 5 presents the results of latency measurements between points B and C, conducted with a consecutive interval of 50 samples. In Figure 5a, the results for an exclusive-use network are shown, where the third configuration exhibits a lower average latency compared to the other two configurations. On the other hand, in a shared-use network, the average latency increased by 70% for the second configuration and 170% for the third configuration, due to network congestion causing delays in the execution of ROS2 nodes. In both scenarios, it can be observed that the first configuration is the least stable in terms of latency.

B. MESSAGE LOSS

The evaluation of message loss has been performed in a closed-loop execution cycle of the control task according to the experimental configuration described in the previous section. The message flow is unidirectional, sent from the microcontroller to the computer, and culminates with the return to the same microcontroller. A total of 1000 messages have been analyzed between point-to-point. Table 6 presents the results of the message loss evaluation for the three configurations described in Table 3.

TABLE 6. Message loss with varying setup of QoS.

Scenario	A->B	B->C	C->D
Setup 1	0/1000	122/1000	9/1000
Setup 2	0/1000	3/1000	0/1000
Setup 3	0/1000	56/1000	0/1000

During the first segment (A->B), no losses were observed in any of the configurations. However, in the second segment (B->C), significant losses were evident in the first configuration, while the second configuration maintained greater stability. During the third segment (C->D), only the first configuration showed slight losses.

These findings suggest that, for data exchange at a frequency of 100 Hz, the implementation of the first configuration is not recommended due to the significant losses observed in the second segment. In contrast, the second configuration proved to be more robust and stable during this time interval. As for the third configuration, it could be considered depending on the requirements for other metrics such as latency and periodicity.

C. PERIODICITY

The periodicity allows for tracking stability in real-time data exchange. This is relevant for the programmability of *control_node* and enables the design of attitude control strategies efficiently and effectively with a high-frequency response. The frequency has been measured for each participating entity, labeled as A, B, C, and D, under different QoS configurations as described in the previous section. Each entity has been programmed to operate at a frequency of 100 Hz. To measure the frequency at points B and C, the commands `ros2 topic hz /imu` and `ros2 topic hz /pwm_pub` were used. To calculate the frequency at points A and D, the following equation was utilized, where $sample_size$, with a value of 50, represents the quantity of measured data points, and $elapsed_time$ is the time taken to record the 50 data points.

$$Frec_{avg}(Hz) = \frac{Sample\_Zise}{Elapsed\_Time(s)} \tag{1}$$

Table 7 displays the detailed experimental data regarding periodicity for the first configuration. The results are presented in terms of average (Avg.), standard deviation (St. d), minimum value (Min.), and maximum value (Max.) of the recorded frequency.

TABLE 7. Periodicity with setup 1 of QoS.

		Parameter			
Entity	Metric	Avg.	St. d	Min.	Max.
A	Frec.(Hz)	97.307	1.928	93.396	100.292
B	Frec.(Hz)	96.993	0.342	96.450	97.756
C	Frec.(Hz)	99.998	0.006	99.969	100.010
D	Frec.(Hz)	94.230	2.125	91.491	98.441

Table 8 presents the empirical data for the second configuration. The results indicate that the frequencies are operating close to the target frequency of 100 Hz. Consequently, the participating entities demonstrate highly precise and stable frequencies, capable of delivering robust performance in terms of quality of service for data exchange at 100 Hz within these entities.

TABLE 8. Periodicity with setup 2 of QoS.

		Parameter			
Entity	Metric	Avg.	St. d	Min.	Max.
A	Frec.(Hz)	100.213	1.024	99.318	102.304
B	Frec.(Hz)	99.982	0.021	99.937	100.011
C	Frec.(Hz)	100.002	0.002	99.999	100.009
D	Frec.(Hz)	100.032	0.725	99.318	101.610

Similar to the second configuration, the third configuration, as shown in Table 4, operates close to the 100 Hz target with a standard deviation lower than that of the previous configurations.

TABLE 9. Periodicity with setup 3 of QoS.

		Parameter			
Entity	Metric	Avg.	St. d	Min.	Max.
A	Frec.(Hz)	100.123	0.910	99.415	101.712
B	Frec.(Hz)	100.007	0.0202	99.935	100.073
C	Frec.(Hz)	99.100	0.001	99.996	100.001
D	Frec.(Hz)	99.887	0.634	99.318	101.365

Periodicity is a key characteristic for the design of event-triggered controllers, as the actual time behavior between events may differ from numerical simulations [24]. For the design of attitude control algorithms on our platform, the controller is triggered by the arrival of IMU data. Hence, in Figure 6a, we depict the periodicity behavior in entity B of the control node. Similarly, Figure 6b illustrates the periodicity behavior for entity D of the *adcs_node* node, which activates an actuator, such as reaction wheels.

The results presented in Figure 6 reaffirm the stability of periodicity in entity B as optimal for designing attitude control algorithms using the architecture outlined in this study. However, it is observed that the default configuration (SETUP 1) of the participating entities exhibits greater instability in terms of periodicity compared to the best-effort (SETUP 2) and custom (SETUP 3) configurations.

D. LAUNCHING INSTRUCTIONS

The operational setup of the experimental platform within the laboratory is depicted in Figure 7 illustrating the ADCS module within a Dyson Sphere. This setup serves as a mechanism for simulating outer space, where a attitude control algorithm implementation is intended. A detailed representation of real-time data transmission with respect to various attitude angles of the ADCS module is illustrated in Figure 8. The incorporation of the Dyson sphere introduces adaptability to accommodate diverse yaw, pitch, and roll orientations.

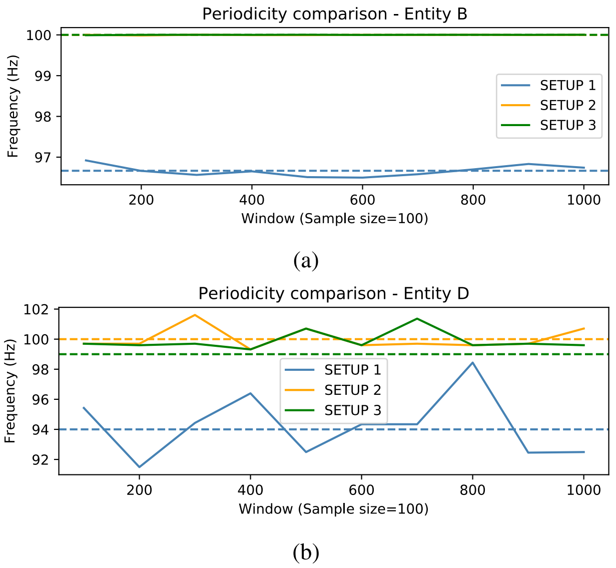


FIGURE 6. QoS Setup Comparison Summary: (a) Entity B; (b) Entity D.

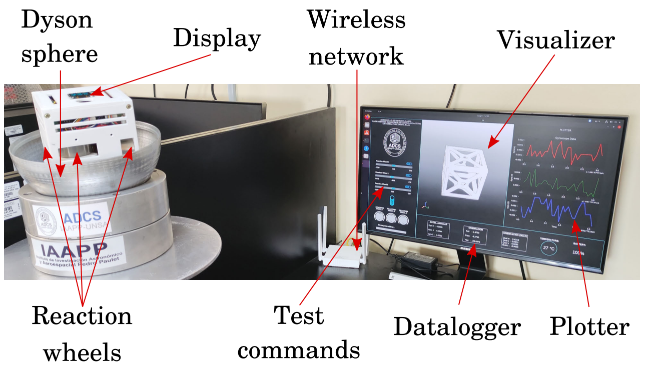
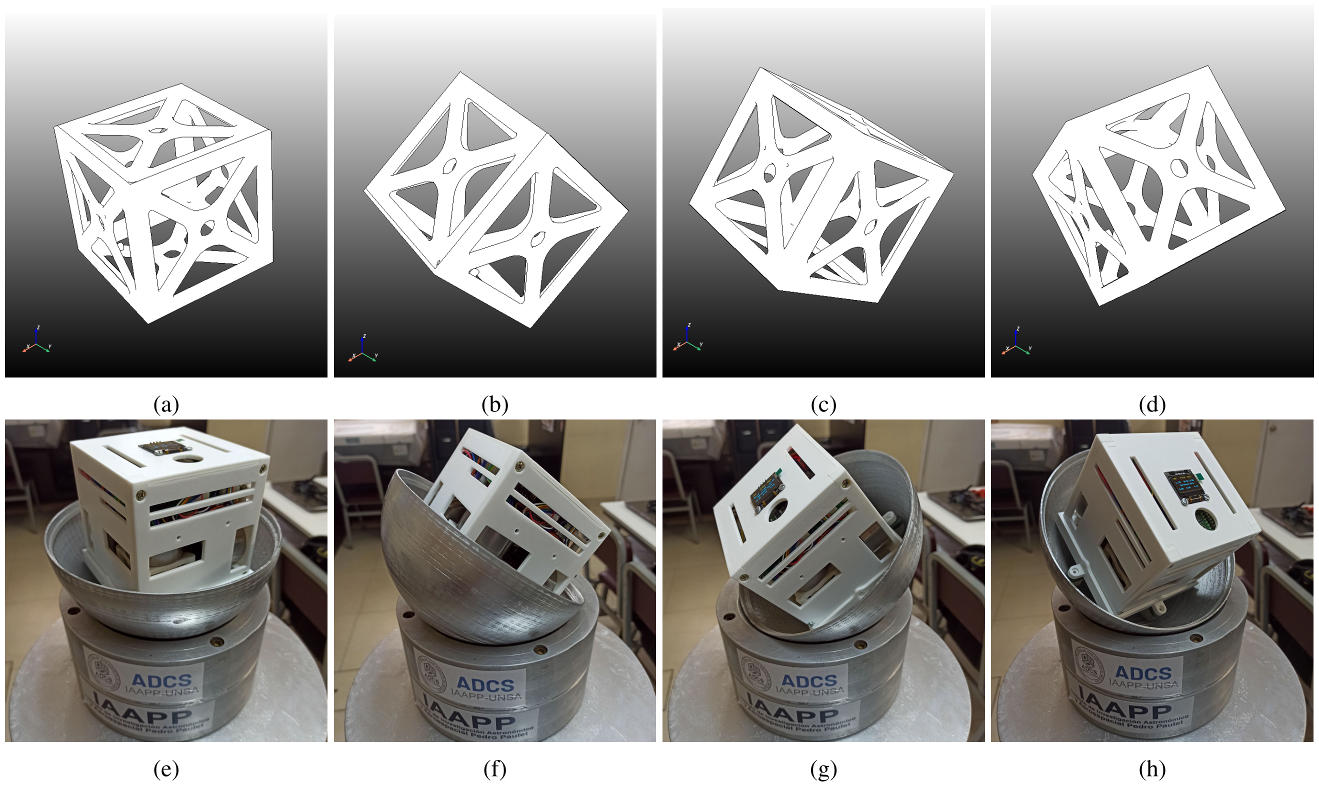


FIGURE 7. Experimental platform ready to use.

It is crucial to note that to establish communication between the ESP32 microcontroller and the computer, an instance of the Micro-ROS agent needs to be executed with a ROS2 Foxy environment using the command `ros2 run micro_ros_agent micro_ros_agent udp4 -port 8888`. Additionally, the Micro-ROS client must be loaded onto the microcontroller utilizing the standard ROS2 client libraries `rclc` and `rcl` [25], enabling the implementation of the *adcs_node* node. The client libraries for the Arduino IDE are available at the following link: https://github.com/micro-ROS/micro_ros_arduino/releases.

V. CONCLUSION AND FUTURE WORK

This research study highlights the successful implementation of an experimental hardware and software architecture platform for attitude control algorithm design. The utilization of low-cost components extends its applicability in academic and research environments. A thorough performance evaluation has been conducted across various configuration scenarios, focusing on three crucial metrics. Latency measurement among the participating ROS2 entities, reflecting computer



(a) (b) (c) (d) (e) (f) (g) (h)

FIGURE 8. Real-time attitude viewer: (a) viewer rotation around the yaw axis; (b) viewer rotation around the negative pitch axis; (c) viewer rotation around the positive pitch axis; (d) viewer rotation around the roll axis; (e) picture rotation around the yaw axis; (f) picture rotation around the negative pitch axis; (g) picture rotation around the positive pitch axis; (h) picture rotation around the roll axis.

response time, reveals that the CEDDP unit exhibits rapid response rates when employed in a dedicated network for the experimental platform. However, within a shared network, a substantial increase in latency is observed. Concerning message loss, the default configuration exhibits a significant loss rate, with the "BEST_EFFORT" QoS policy being the most suitable for mitigating this issue. In terms of periodicity, it has been demonstrated that configurations utilizing "BEST_EFFORT," along with minor adjustments to QoS profiles, prove optimal for algorithm design within the control node. These configurations successfully enhance the standard deviation of periodicity, thereby optimizing system accuracy and reliability.

In future research endeavors, the experimental platform developed in this study can be employed for formulating attitude control algorithms, encompassing both nanosatellites and diverse categories of robots. The integration of ROS2 and Micro-ROS as flight software components could streamline the interconnection among nanosatellite units, enabling the assessment of their effectiveness and feasibility in real-world operations, as well as their capability to manage communication within nanosatellite fleets. Additionally, expanding the study to evaluate performance across varying communication frequencies based on the system's requirements could yield crucial insights. This expansion has the potential to provide essential data for optimizing coordination and communication among units in space missions, thereby contributing to technological advancements in the realm of nanosatellites and their application across various facets of space exploration and scientific research.

ACKNOWLEDGMENT

The authors would like to thank Universidad Nacional de San Agustin de Arequipa (UNSA) and the Instituto de Investigación Astronómico y Aeroespacial Pedro Paulet (IAAPP), for their support in providing the necessary facilities and equipment.

REFERENCES

[1] X. Xia, G. Sun, K. Zhang, S. Wu, T. Wang, L. Xia, and S. Liu, "NanoSats/CubeSats ADCS survey," in *Proc. 29th Chin. Control Decis. Conf. (CCDC)*, May 2017, pp. 5151–5158.

[2] P. R. Yanyachi and P. S. D. Silva, "Modelagem E controle de atitude de satélites artificiais com apendices flexíveis," Ph.D. thesis, Departamento de Engenharia de Telecomunicações e Controle, Univ. de São Paulo, São Paulo, 2005.

[3] B. E. Garcia, A. Martin, and P. Raul, "Non-linear control strategies for attitude maneuvers in a CubeSat with three reaction wheels," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 11, 2020.

[4] F. Bobrow, B. A. Angelico, F. P. R. Martins, and P. S. P. da Silva, "The Cubli: Modeling and nonlinear attitude control utilizing quaternions," *IEEE Access*, vol. 9, pp. 122425–122442, 2021.

[5] J. Bouwmeester, A. Menicucci, and E. K. A. Gill, "Improving CubeSat reliability: Subsystem redundancy or improved testing?" *Rel. Eng. Syst. Saf.*, vol. 220, Apr. 2022, Art. no. 108288.

[6] J. He, J. Zhang, J. Liu, and X. Fu, "A ROS2-based framework for industrial automation systems," in *Proc. 2nd Int. Conf. Comput., Control Robot. (ICCCR)*, Mar. 2022, pp. 98–102.

[7] Micro-ROS. (2018). *Micro-ROS Puts ROS 2 Onto Microcontrollers*. [Online]. Available: https://micro.ros.org/

[8] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," in *Proc. Int. Conf. Embedded Softw. (EMSOFT)*, Oct. 2016, pp. 1–10.

[9] Y. Liu, Y. Guan, X. Li, R. Wang, and J. Zhang, "Formal analysis and verification of DDS in ROS2," in *Proc. 16th ACM/IEEE Int. Conf. Formal Methods Models Syst. Design (MEMOCODE)*, Oct. 2018, pp. 1–5.

[10] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Sci. Robot.*, vol. 7, no. 66, May 2022, Art. no. eabm6074.

[11] S. Buckner, C. Carrasquillo, M. Elosegui, and R. Bevilacqua, "A novel approach to CubeSat flight software development using robot operating system (ROS)," in *Proc. 34th Annu. AIAA/USU Small Satell. Conf.*, 2020, pp. 1–8.

[12] B. D. Yost, S. Weston, J. Hines, and C. Burkhard, "An overview of the current state of the art on small spacecraft avionics systems," in *Proc. AIAA SCITECH Forum*, Jan. 2022, p. 0521.

[13] C. E. Gonzalez, C. J. Rojas, A. Bergel, and M. A. Diaz, "An architecture-tracking approach to evaluate a modular and extensible flight software for CubeSat nanosatellites," *IEEE Access*, vol. 7, pp. 126409–126429, 2019.

[14] K. Belsare, A. C. Rodriguez, P. G. Sánchez, J. Hierro, T. Kołcon, R. Lange, I. Lütkebohle, A. Malki, J. M. Losa, F. Melendez, M. M. Rodriguez, A. Nordmann, J. Staschulat, and J. von Mendel, "Micro-ROS," in *Robot Operating System (ROS)*. Cham, Switzerland: Springer, 2023, pp. 3–55.

[15] J. Fernandez, B. Allen, P. Thulasiraman, and B. Bingham, "Performance study of the robot operating system 2 with QoS and cyber security settings," in *Proc. IEEE Int. Syst. Conf. (SysCon)*, Aug. 2020, pp. 1–6.

[16] Z. Chen, "Performance analysis of ROS2 networks using variable quality of service and security constraints for autonomous systems," Doctoral dissertation, Dept. Elect. Comput. Eng. (ECE), Naval Postgraduate School, Monterey, CA, USA, Sep. 2019.

[17] J. M. Fernandez, "Quality of service and cybersecurity communication protocols analysis for the robot operating system 2," Doctoral dissertation, Dept. Elect. Comput. Eng. (ECE), Naval Postgraduate School, Monterey, CA, USA, Jun. 2019.

[18] J. Park, R. Delgado, and B. W. Choi, "Real-time characteristics of ROS 2.0 in multiagent robot systems: An empirical study," *IEEE Access*, vol. 8, pp. 154637–154651, 2020.

[19] A. Jalil, J. Kobayashi, and T. Saitoh, "Performance improvement of multi-robot data transmission in aggregated robot processing architecture with caches and QoS balancing optimization," *Robotics*, vol. 12, no. 3, p. 87, Jun. 2023.

[20] M. S. Essers and T. H. J. Vaneker, "Design of a decentralized modular architecture for flexible and extensible production systems," *Mechatronics*, vol. 34, pp. 160–169, Mar. 2015.

[21] P. R. Yanyachi, H. Mamani-Valencia, and B. Espinoza-García, "Low-cost test system for 1U CubeSat attitude control with reaction wheels," in *Proc. IEEE Biennial Congr. Argentina (ARGENCON)*, Sep. 2022, pp. 1–8.

[22] R. Delgado, J. Park, and B. Choi, "Open embedded real-time controllers for industrial distributed control systems," *Electronics*, vol. 8, no. 2, p. 223, Feb. 2019.

[23] D. Eridani, A. F. Rochim, and F. N. Cesara, "Comparative performance study of ESP-NOW, Wi-Fi, Bluetooth protocols based on range, transmission speed, latency, energy usage and barrier resistance," in *Proc. Int. Seminar Appl. Technol. Inf. Commun. (iSemantic)*, Sep. 2021, pp. 322–328.

[24] R. Postoyan, R. G. Sanfelice, and W. P. M. H. Heemels, "Explaining the 'mystery' of periodicity in inter-transmission times in two-dimensional event-triggered controlled systems," *IEEE Trans. Autom. Control*, vol. 68, no. 2, pp. 912–927, Feb. 2023.

[25] J. Staschulat, I. Lütkebohle, and R. Lange, "The rclc executor: Domain-specific deterministic scheduling mechanisms for ROS applications on microcontrollers: Work-in-progress," in *Proc. Int. Conf. Embedded Softw. (EMSOFT)*, Sep. 2020, pp. 18–19.

ALFREDO MAMANI-SAICO was born in Cusco, Peru, in 2000. He is currently pursuing the Bachelor of Electronic Engineering of with Universidad Nacional de San Agustin de Arequipa (UNSA), where he is also a Junior Researcher with Instituto de Investigación Astronómico y Aeroespacial Pedro Paulet (IAAPP). His research interests include robotics, distributed systems, and remote sensing. He has developed the software presented in this paper.

PABLO RAUL YANYACHI (Senior Member, IEEE) received the M.Sc. degree in automatic control from the Polytechnic Institute of Leningrad, and the Ph.D. degree in electrical engineering from the Polytechnic School, University of São Paulo, Brazil. He is currently a main Professor of academic with the Department of Electronic Engineering, Universidad Nacional de San Agustin de Arequipa (UNSA). He is the Station Manager of the Nasa Laser Tracking Station TLRS-3, Arequipa, Peru. He is also the Director of Instituto de Investigación Astronómico y Aeroespacial Pedro Paulet (IAAPP), UNSA.