76th International Astronautical Congress (IAC 2025), Sydney, Australia, 29 Sep-3 Oct 2025.

IAC-25-A3.3B.4.x102287

Control, Monitoring and Operation Software of a Miniaturized Lidar for Mars Atmospheric Research

J. Samblas Iglesias[a]*, I. Carrasco[a], N. Andrés Santiuste[a], A. Martín-Ortega Rico[a], J. J. Jiménez Martín[a], E. García-Menéndez[a], J. Rivas[a], E. Muñoz García[a], G. Vázquez Yañez[a], S. Montalvo[a], S. Andrés Velasco[a], I. Arruego[a]

[a] *Instituto Nacional de Tecnica Aeroespacial (INTA), Spain. Carretera de Ajalvir, Km 4, 28850, Torrejon de Ardoz, Madrid.*
* Corresponding Author - jsamigl@inta.es

Abstract

The development of a space instrument involves multiple iterations of design, prototyping, and validation campaigns at different levels, supported by Ground Support Equipment (GSE) that includes software applications forming part of the Electrical Ground Support Equipment (EGSE). This work presents the development of the Control and Operation Software (COPS), a modular application created for the Miniaturized LiDAR for Mars Atmospheric Research (MiLi) project, designed to provide real-time control, monitoring, and operation of a [2β + 1δ] LiDAR instrument. The software acts as the interface between the instrument and its operator, enabling telemetry monitoring, scientific data processing, and flexible interaction with subsystems such as laser drivers, detectors, thermal control, and data acquisition through the instrument's processing unit. It incorporates real-time control mechanisms, including a proportional–integral–derivative (PID) controller to maintain the thermal stability of lasers, detectors, and polarizers under diverse operational scenarios. The graphical user interface offers dedicated views for LiDAR signal visualization, system diagnostics, and parameter tracking, while signal processing techniques such as background subtraction, range correction, and signal-to-noise ratio (SNR) optimization enhance data quality for scientific interpretation. Comprehensive telemetry analysis and data logging support detailed post-processing, performance assessment, and long-term validation. The COPS contributes to the operation, validation, and optimization of MiLi, a LiDAR prototype developed to improve the characterization of Martian aerosols and clouds by retrieving key atmospheric parameters such as particle backscatter coefficient, particle linear depolarization ratio, LiDAR ratio, and color ratio. By providing integrated control, adaptive data processing, and robust telemetry management, the software ensures precise system performance under varying environmental conditions, ultimately supporting the reliability and scientific return of the instrument.
Keywords: MiLi; Control and operation software; Mars; Atmospheric LiDAR; Data processing.

Acronyms/Abbreviations

- Technology Readiness Level (TRL).
- Light Detection and Ranging (LiDAR).
- Silicon Photomultipliers (SiPMT).
- Electrical Ground Support Equipment (EGSE).
- Control and Operation Software (COPS).
- Field Programmable Gate Array (FPGA).
- Proportional–Integral–Derivative (PID).
- Universal Asynchronous Receiver-Transmitter (UART).
- JavaScript Object Notation (JSON).
- Power Supply Unit (PSU).
- Data Acquisition (DAQ).
- Visible (VIS).
- Near Infrared (NIR).
- Multi-Pixel Photon Counter (MPPC).
- Central Processing Unit (CPU).
- Global Climate Model (GCM)

1. Introduction

The exploration of Mars requires the development of novel instruments capable of characterizing its atmosphere with high spatial and temporal resolution [1]. Understanding the distribution of aerosols and clouds is essential to improve GCMs, assess environmental variability, and mitigate risks for future robotic and human missions. LiDAR instruments provide comprehensive insights into atmospheric characteristics, including the concentration and spatial distribution of aerosols, dust and clouds. They also enables the retrieval of parameters such as wind speed and direction by analysing different features of the scattered light, such as its intensity, polarization, and wavelength [2-5]. They are particularly suitable for this purpose, as they actively retrieve vertical profiles of dust and cloud layers, complementing passive sensors and enabling continuous operation under variable illumination conditions. Other proposals do exist, but they have a limited range of functionalities [6]. Conventional LiDAR systems are often too heavy, power-demanding, and mechanically complex to be deployed in planetary exploration. To address this limitation, the Miniaturized LiDAR for Mars Atmospheric Research (MiLi) project has been

conceived as an Earth-based prototype to increase the TRL of key technologies such as solid-state lasers, SiPMT, and thermal stabilization mechanisms [7].

Operating and validating such a complex prototype requires dedicated EGSE. The COPS has been developed as the software component of this EGSE, providing a modular and scalable platform for commanding, monitoring, and acquiring scientific and housekeeping data from the MiLi instrument. The COPS interfaces with the instrument's electronics through an FPGA, allowing the configuration of laser triggering, acquisition modes, and detection thresholds, while supporting real-time visualization, signal processing, and telemetry logging. Its architecture incorporates safety mechanisms, thermal control through PID regulation, and visualization tools to assist operators during laboratory and field campaigns. This development ensures robust interaction with the prototype, contributing directly to the validation of its subsystems and to the assessment of overall instrument performance.

The main objective of this manuscript is to present the design, development, and validation of the COPS, emphasizing its architecture and its role in enabling reliable interaction with the MiLi prototype. The document details the overall software architecture and its main modules for control, monitoring, and operation, followed by a more in-depth explanation of the threads and then the experimental setup, test procedures, and results obtained during validation. Finally, it summarizes the main contributions and outlines potential directions for future developments.

2. Software Design and Architecture

The COPS was designed as a modular application that ensures reliable interaction between the operator and the MiLi instrument. Its architecture separates user interaction, hardware control, and data processing into independent components, allowing for a more scalable and robust system. This software coordinates multiple threads dedicated to acquisition sequencing, command and telemetry exchange and data processing, with inter-module communication mechanisms and buffers. This design prevents interface blocking and guarantees deterministic execution of instrument commands.

2.1 Communication Module

The communication module manages all data exchange between the software and the instrument through the FPGA interface. Its main function is to translate user actions into properly formatted telecommands and to decode the received telemetries from the instrument. This ensures that configuration parameters, commands, and data retrieval requests are transmitted and executed reliably.

This module is based on a state-machine architecture that implements the command-response cycle. Each command is associated with explicit preconditions, acknowledgements and completion signals. Safety mechanisms are integrated: parameter validation routines preventing transmission of invalid values, redundancy codes safeguard message integrity and watchdogs and timeouts avoid deadlocks when the instrument does not respond as expected.

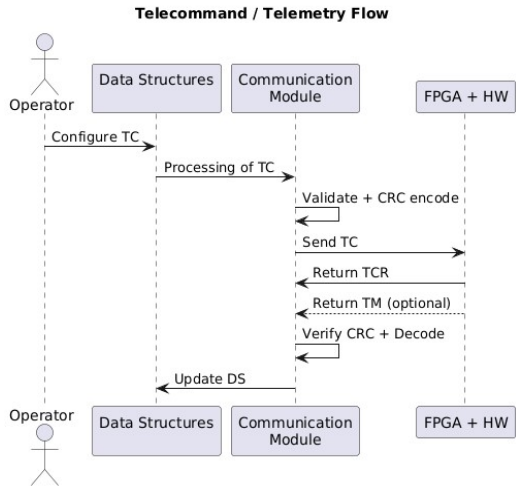

Fig. 1. Sequence diagram of a MiLi communication.

Critical operations are protected and require verification and certain preconditions. This approach minimizes the risk of hardware damage and contributes to operator safety during extended test campaigns where fatigue may occur.

By abstracting the complexity of low-level protocols, the communication module provides a robust and reusable interface between the software and hardware. This design ensures reproducibility of instrument operations, facilitates debugging during integration and establishes a reliable communication for all functionalities of the COPS.

2.2 Processing Module

This module is responsible for handling all telemetry streams produced by the instrument and delivered through the communication module. Its role is to decode, structure and manage both housekeeping and scientific data, ensuring that information is available for monitoring, storage and analysis.

Telemetry messages received from the communication module are parsed and transformed into structured information such as arrays and dictionaries. Handling of diverse data types like temperatures, voltages, currents or subsystem status flags are measured. Each entry is timestamped to support traceability and correlation across multiple acquisitions or operational scenarios.

To ensure system responsiveness, the processing module operates in a dedicated thread that isolates heavy computations from the graphical interface. This design prevents blocking conditions and allows asynchronous updating of internal buffers while users interact with the system.

2.3 Operation Module

It provides the graphical user interface. It integrates multiple views within a unified graphical interface while remaining decoupled from communication and processing tasks, which run in dedicated threads as has already been stated.

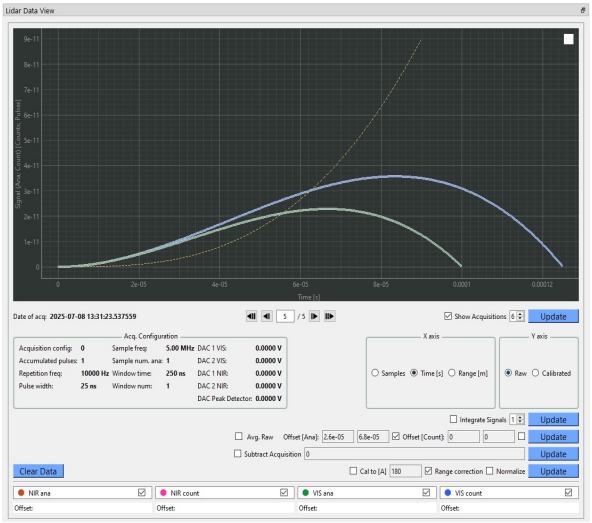

Fig. 2. Lidar Data View of the COPS. Sample data with transformations applied.

The '*Lidar Data View'* offers visualization of scientific signals with a series of transformations (integration, averaging, background subtraction, offset subtraction, division, calibration, range correction and normalization). Users can switch x-axis domains (samples, time and distance) and compare multiples acquisitions at once. Moreover, due to the selected plotting library, more standard operations are available, such as binning and logarithmic transformation in both axes.

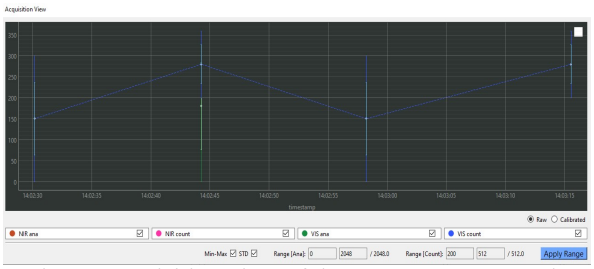

Fig. 3. Acquisition view of the COPS. Some sample data is shown.

The *'Acquisition View'* summarizes each acquisition in a single point, and provides a broad visualization of the signal over time. This view allows changing the x-axis values to those of any of the configuration parameters so we can compare results in contrast of the configuration of the LiDAR at a certain acquisition. Furthermore, this view displays error bars for the maximum and minimum values, as well as the standard deviation of each acquisition.

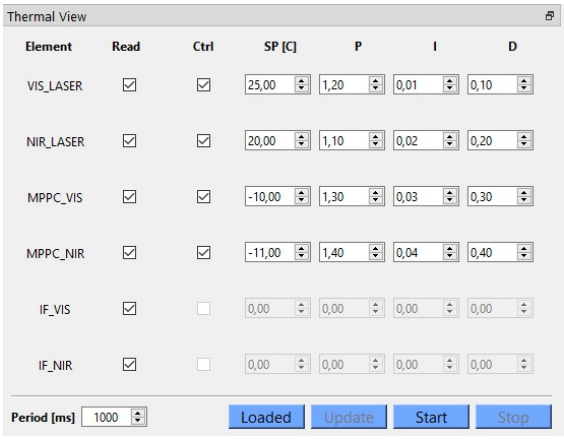

Fig. 4. Thermal View of the COPS with default values.

A *'Thermal Control View'* exposes temperature setpoints and PID parameters for lasers, detectors and interference filters. It permits adjustment of which elements to control and for custom periodic checks.

Another view under the name of *'Parameter View'* is created based on a JSON file input and helps track the values of the defined elements. It can, for example, display all the temperatures for the elements of the *Thermal Control View*.

There are many other views dedicated to communication, configuration of the instrument, logging, etc.; but the main scientific views are the ones depicted in this section of the manuscript.

2.4 Implementation Notes

This software was implemented in *Python*, making use of the *PySide6* framework for the graphical interfaces and *NumPy* for numerical calculations and some of the data structures. The COPS has been validated with both a software-based simulator and the MiLi prototype itself, ensuring functionality.

3. Concurrency and Thread Management

The COPS must simultaneously manage communication with the instrument, real-time data processing and user interface responsiveness. A single-threaded approach would be unable to meet these requirements, as blocking operations such as heavy signal processing would directly affect user interaction.

Concurrency is therefore essential to guarantee deterministic command execution, continuous monitoring and real-time visualization of LiDAR signals without compromising user experience.

The multithreading model is based on the *QThread*, a *PySide6* class, which provides native integration between threads and the interface.

Inter-thread communication is managed through *Pyside6's 'signal-slot'* mechanism, which allows asynchronous message passing. This design minimizes the risk of race conditions and deadlocks while ensuring a thread-safe communication across all threads.

3.1 Communication Thread

The Communication thread implements the functionality of the communication module within the COPS. It runs independently of the main application thread and ensures that telecommands are transmitted and telemetry is retrieved without blocking other operations.

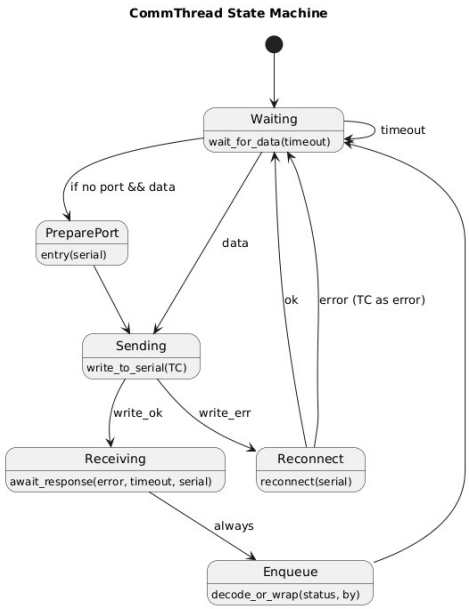

Fig. 5. Communication thread state machine.

Telecommands generated by the application are queued and consumed by the communication thread, which serializes them into byte streams, transmits them through the UART interface of the FPGA, and awaits the corresponding response. On reception, telemetry is read in chunks to avoid buffer overflows and is subject to strict timeouts. Successful receptions are decoded into messages, while error or timeout cases are encapsulated into error reports.

Reconnection routines are used in case of serial errors and integrity checks based on redundancy codes are applied.

The outputs of the communication thread are communication objects containing both the transmitted command and the associated responses, which are then placed into the *'processing queue'*. In this way, the communication thread guarantees a continuous flow of telecommands and telemetry, ensuring the interaction between the COPS and the instrument.

3.2 Processing Thread

The Processing Thread provides the implementation of the processing module. While the module described in section 2.2 defines the functional responsibilities for decoding, structuring and managing telemetry streams, the thread ensures that these operations are executed preserving responsiveness of the application.

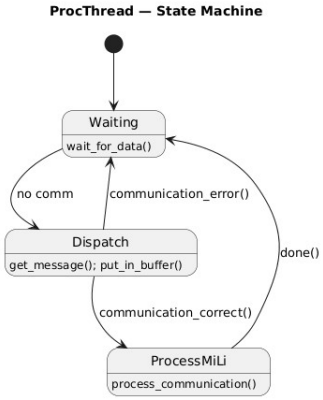

Fig. 6. Processing thread state machine.

This thread continuously consumes communication objects placed in the *'processing queue'* by the communication thread. Each message within the object is broadcasted through signals to other components, such as Parameter Views, Lidar Data View, etc. When the communication exchange is valid (no errors occurred), the processing thread applies domain-specific logic; Mapping of housekeeping values to their configuration channels and detection of acquisition initial conditions. In the case of memory dump telemetries, it extracts analogue and counting modes LiDAR signals, builds acquisition objects, and emits signals with the corresponding scientific data, acquisition configuration and timestamps.

Embedding the processing module in a dedicated thread ensures scientific data is correctly decoded, structured and ready for monitoring, storage and analysis.

3.3 FileWriter Thread

This thread is in charge of the file output functionality of the COPS. It is a low priority thread which stores the sent and retrieved data.

Incoming data are encapsulated in *'FileWriteRequest'* objects. When the thread receives a new request, it determines whether a new file must be created or an existing one should be appended. This decision is based on whether the existing file has reached a size cap of 200 MB. New files are also created whenever the operator issues a command to start a new session, which resets the file path and filename annotations.

To simplify integration, a factory function *'make_slot'* was implemented. It generates Qt *'Slots'* that wrap application signals into write requests automatically.

3.4 FileLoader Thread

The file loader thread complements the writer by providing loading capabilities. When triggered, it iterates over the requested files through a dedicated manager. Each file is opened, parsed, and the acquisitions are reconstructed. For every acquisition found, the loader emits a signal, ensuring all views and buffers in the application react as they would during normal operation.

If a file is of the wrong type or its format is not supported, exceptions are caught and logged without interrupting the remaining processing of files.

This thread supports replay of experimental sessions and offline analysis using the same visualization and processing tools developed for live operation. By doing so, reproducibility and flexibility of the system is reinforced.

3.5 Acquisition Thread

The acquisition thread is responsible for preparing and sending the acquisitions commands.

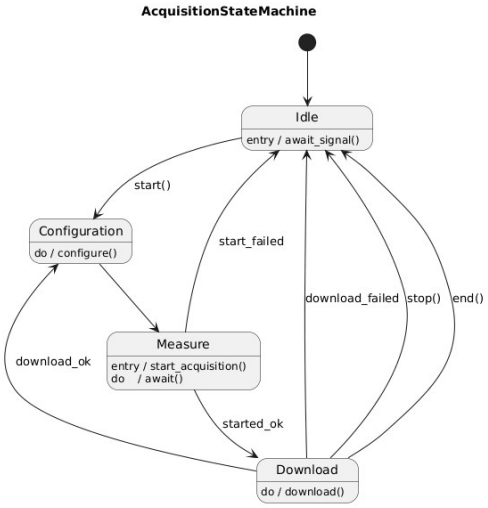

Fig. 7. Acquisition thread state machine.

This thread maintains a state which cycles through four different states: Idle, Configuration, Measure and Download. The Idle state waits for a signal from the interface to begin an observation. Once triggered, the machine transitions to Configuration, where it retrieves the next acquisition setup from the controller. Depending on the observation mode, the configuration may involve sending write-register telecommands to adjust parameters.

In the Measure state, the thread commands the FPGA to start the acquisition by setting the appropriate bit in the configuration register. It then waits for the programmed acquisition duration, checks the completion flags and computes dependent parameters such as the number of bytes retrieved from internal and external memory. If the acquisition has started successfully, the state machine advances to the Download state. Here, the thread issues commands to read both analogue and counting modes memory dumps, verifying that the FPGA acknowledges each request.

After each cycle, the state machine evaluates exit conditions. The observation ends if an error occurs during download, if the user issues a stop request, or if the acquisition controller signals that the predefined number of acquisitions has been completed. Otherwise, the machine transitions back to Configuration to prepare for the next acquisition.

3.6 Thermal Control Thread

The thermal control worker runs in its own high priority thread. When started, it loads and validates a JSON configuration against a schema, checking for duplicate PSU, thermal-devices, etc.

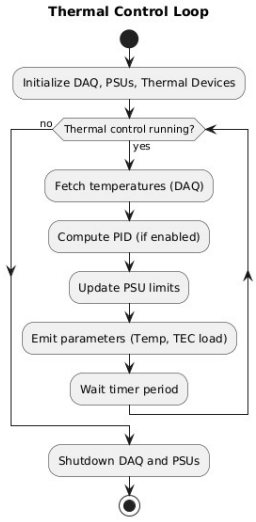

Fig. 8. Thermal control loop activity diagram.

On successful start, it initializes the devices. A DAQ is opened using the configured address, and a pool of PSUs is created and preconditioned. Thermal devices are then instantiated from the configuration: VIS/NIR

lasers, MPPCs and interference filters. Each device binds to a DAQ input channel and to a PSU.

The periodic control fetches all DAQ channels, timestamps the read and iterates over the enabled devices. For devices with closed-loop control the load is computed from the latest temperature reading and setpoint, then the corresponding PSU current is set.

Runtime updates adjust the control period and PID/setpoint safely by resetting the configuration with the new values. Stopping the loop turns off the timer, closes the DAQ and returns every PSU to a safe state turning outputs off and closing the device handlers.

4. Results and Validation

The validation process has been conducted through a combination of functional tests, integration tests, unitary tests, static analysis, integration with the prototype hardware, and performance measurements. Each major component of the application has been evaluated individually and within the complete system to ensure reliability, accuracy and usability.

4.1 Test Environment

The validation of the COPS was conducted using a test environment that combined both software tools and real hardware components, allowing for extensive verification at all development stages.

4.1.1 Software Environment

The application was developed and tested primarily on *Windows 10* using *Python 3.12* and the *PySide6* framework. Unit and integration tests were executed using both the built-in library and *pytest*. For each module of the application, multiple *pytest* tests were created to ensure that the application's behavior matched the expected functionality. These tests focused on edge cases and scenarios where the code might fail, allowing the detection and correction of incorrectly implemented logic when necessary.

4.1.2 Simulated Input Data

At all stages of development, simulated acquisitions were used. For this purpose, a software simulator was built that was capable of communicating with the COPS software and sending both telecommands responses and telemetry messages. This allowed the development team to verify the behaviour of the software even when the instrument was not available. Most testing during the early stages of development (and still during later phases) was carried out using this simulator, as it enabled validation of data storage, signal transformation, and data visualization across the various views of the application. All datasets used in simulations were representative and specifically designed to target potential failure points.

4.1.3 Integrated System Testing

Once the final instrument integration was completed, the application was deployed in conjunction with the MiLi electronics. The software controlled the complete acquisition sequence and was used to validate parameters such as timing, gain settings, signal shape, and response to configuration changes. External tools such as oscilloscopes and power supplies were used to control the thermal behaviour of the instrument for already stated causes, instead of using the COPS.

4.2 Validation Methodology

The validation strategy for the COPS software followed a multi-layered methodology, progressively verifying each component and interaction to ensure robustness and reliability throughout the development process.

First, unit tests were implemented for each self-contained software module using the *pytest* framework. These tests targeted the correctness of individual functions and classes and covered a wide range of edge cases. Their purpose was to detect errors in logic and ensure that all internal calculations and conditions behaved correctly.

Next, integration tests were introduced to validate the interaction between software modules. These tests checked whether components such as acquisition control, data transformation, configuration management, and the interface behaved correctly when combined. They also verified that signals and events flowed properly across the application.

Then, functional tests were carried out to simulate real usage scenarios. These included full acquisition sessions, configuration changes, and data visualization workflows. Functional tests were performed manually, ensuring that the software met all functional requirements and behaved consistently in realistic use cases.

In addition to these tests, static code analysis was applied throughout the development. *Pylint* was used. It provided a more comprehensive analysis, checking documentation compliance, identifying unused or misused variables, and reporting potential logical errors. The systematic use of these tools improved maintainability, reduced the probability of hidden defects, and ensured a consistent and coding style across the entire project.

4.3 Performance Analysis

A global performance evaluation was conducted to ensure that the COPS application performs efficiently under real operating conditions. This analysis focused on the responsiveness of the user interface, the computational cost of data transformations, and the scalability of the application when handling multiple acquisitions simultaneously.

The application demonstrated stable behaviour with low latency in common tasks such as changing visualization modes, reading and writing registers, obtaining memory dumps, etc.

The most time consuming task would be the application of numerous data transformations on the Lidar Data View. Management of 4 different matrices of size 3600x512 and 3600x2048 needed to be transformed. After optimization this task would take a time of 0.3 seconds on average to complete, ensuring adequate response times.

Memory usage remained within acceptable limits during normal operation, even when displaying up to 10 different acquisition in parallel, thanks to the use of efficient data structures. CPU usage temporarily increased during intense operations like signal integration or subtraction, but quickly returned to idle levels after processing.

The graphical interface remained responsive at all times.

In conclusion, the performance analysis confirms that the software meets the minimum requirements for efficient use in the context of a scientific instrument such as the MiLi LiDAR.

5. Discussion

The results demonstrate that the design of the COPS provides a stable foundation for instrument operation, ensuring that communication, data processing, acquisition sequencing and thermal regulation can proceed concurrently without interfering with user interaction. Beyond validating correctness, these outcomes highlight the architecture and its capacity to scale with the instrument's operational demands.

While the current implementation already supports complex operation scenarios, its broader significance lies in showing how a carefully designed EGSE software can contribute directly to instrument reliability and scientific return. Moreover, the software could be adapted for use in other contexts. Its modular design makes it a good candidate to serve as a base for similar instruments or laboratory test benches. By abstracting more of the instrument-specific logic, the COPS architecture could evolve into a general-purpose control and operation framework for all kinds of instrumentation.

6. Conclusions

Thanks to this software, it has been possible to validate the correct functioning of MiLi's subsystems, carry out real time signal analysis, and conduct integration campaigns under controlled laboratory conditions. Its modular design allowed the team to develop the application in parallel with the instrument, supporting changes in firmware, communications and data formats without compromising stability.

In summary, the results obtained demonstrate not only the maturity and utility of the COPS software, but also the contribution of this system to the success of the MiLi project.

Acknowledgements

The authors gratefully acknowledge the financial support provided by the Europe Horizon programme in the framework of the MiLi project, ('Miniaturized LiDAR for MARS Advanced Atmospheric Research') project (ID. 101082451).

References

[1] C. Cordoba, L. Gómez, D. Toledo, Scientific Requirement Document – D1.1, MILI-SP-004-INTA, Issue 2.0, 02 February 2024, Miniaturized Lidar for MARS Advanced Atmospheric Research (MiLi), INTA, Madrid, Spain. url: https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e508789d0c&appId=PPGMS

[2] C. Weitkamp, Ed., LiDAR: range-resolved optical remote sensing of the atmosphere. in Springer series in optical sciences, no. 102. New York, NY: Springer, 2005.

[3] A. Comerón, C. Muñoz-Porcar, F. Rocadenbosch, A. Rodríguez-Gómez, and M. Sicard, "Current Research in LiDAR Technology Used for the Remote Sensing of Atmospheric Aerosols," Sensors, vol. 17, no. 6, Art. no. 6, Jun. 2017, doi: 10.3390/s17061450.

[4] J. Martikainen et al., "Experimental Scattering Matrices of Martian Dust Aerosols with Narrow Particle-size Distributions," Astrophys. J. Suppl. Ser., vol. 273, no. 2, p. 28, Jul. 2024, doi: 10.3847/1538-4365/ad5720.

[5] I. Kuznetsov et al., "Dust Particles in Space: Opportunities for Experimental Research," Астрономический Журнал, vol. 100, pp. 41–69, Jan. 2023, doi: 10.31857/S0004629923010115.

[6] D. R. Cremons, "The future of LiDAR in planetary science," Front. Remote Sens., vol. 3, p. 1042460, Oct. 2022, doi: 10.3389/frsen.2022.1042460.

[7] D. Scaccabarozzi, K. Potemkin, E. Vieira, B. Saggin, et al., Thermo-Mechanical Design of a Miniaturized LiDAR for Mars Advanced Atmospheric Research, IAC-24-E5.3.8, 75th International Astronautical Congress, Milan, Italy, 14-18 October 2024, DOI:10.52202/0783