Command & Data Handling Software for CubeSat FlatSat Testbed

Lab: Ω-Space Group, ORION Lab, National Technical University of Athens

Duration: 3-6 months

Level: Master Thesis

Background

The Ω-Space Group of the ORION lab is developing a CubeSat FlatSat testbed for end-to-end software testing and AI algorithm validation. This thesis focuses on implementing the core Command & Data Handling (C&DH) subsystem using Space ROS (ROS2 for space applications). The C&DH serves as the central flight computer managing satellite operations, subsystem coordination, and telemetry/telecommand handling. This work explores the applicability of Space ROS to CubeSat missions, aligning with ESA's research interests in modern space software architectures.

Objectives

Develop a functional Space ROS-based C&DH software system that:

1. Implements core satellite control functionality as ROS2 nodes (mode management, telecommand handling, telemetry generation, housekeeping)
2. Communicates with payload subsystem via Space ROS DDS over Gigabit Ethernet
3. Provides time synchronization across subsystems
4. Demonstrates end-to-end command execution and telemetry collection with the AI payload
5. Serves as foundation for future fault detection, isolation, and recovery mechanisms

Scope of Work

Phase 1: Core C&DH Implementation

Space ROS Setup

- Configure Space ROS environment on Raspberry Pi 4 (Ubuntu 22.04 + ROS2 Humble/Iron)

C&DH Architecture as ROS2 Nodes

Mode Manager Node:

- Implements satellite operational state machine
- Safe Mode (minimal operations, fault recovery)
- Nominal Mode (standard housekeeping, waiting for commands)
- Payload Mode (AI processing active)
- Mode transitions based on commands or system conditions

Telecommand Handler Node:

- Subscribes to /telecommand topic
- Command validation and parsing
- Command execution (e.g., START_PAYLOAD, STOP_PAYLOAD, REQUEST_TELEMETRY, CHANGE_MODE)
- Command acknowledgment and status reporting

Telemetry Manager Node:

- Collects telemetry from all C&DH nodes and payload
- Packages telemetry in JSON format with timestamps
- Publishes to /telemetry/cdh/* topics
- Stores telemetry locally (rosbag or database)

Housekeeping Node:

- Periodic collection (every 5-10 seconds) of system health data
- CPU temperature, load, memory usage (via psutil)
- Disk space, uptime, network status
- Subsystem status (payload alive/responding)
- Publishes to /housekeeping topic

Time Synchronization Node:

- C&DH as time master (system assumes offline operation)
- Periodic broadcast of current time to other subsystems
- Synchronization message format and handling

Integration & Testing

- **Command Injection Interface:** ROS2 topic/service for testing (publish commands to /telecommand for validation)
- **Integration with Payload:** Subscribe to payload AI results (e.g., /ai/cloud_detection/output), package as telemetry
- **Testing & Validation:** Unit tests for each ROS2 node, integration tests with payload subsystem, end-to-end scenarios (command → execution → telemetry), performance analysis (command latency, telemetry rate)

Phase 2: STM32 Port (Optional)

- Port core C&DH functionality to STM32 Nucleo with micro-ROS + FreeRTOS
- Maintain compatibility with Space ROS architecture
- Document porting process and challenges
- Compare performance (RPi4 vs STM32)

Technical Requirements

- **Operating System:** Ubuntu 22.04 on Raspberry Pi 4
- **Middleware:** Space ROS (ROS2 Humble or Iron)
- **Programming Languages:** Python for ROS2 nodes (rapid development), C/C++ for performance-critical components if needed
- **Communication:** ROS2 DDS over Gigabit Ethernet
- **Version Control:** All code contributed to project GitHub repository
- **Testing:** ROS2 testing framework, integration tests, rosbag for data recording

Deliverables

1. Working C&DH software as Space ROS nodes on Raspberry Pi 4
2. Complete ROS2 node implementations (Mode Manager, Telecommand Handler, Telemetry Manager, Housekeeping, Time Sync)
3. Integration with payload subsystem (end-to-end command/telemetry flow)
4. Test results and performance analysis (latency, throughput, resource usage)
5. Source code with documentation in GitHub repository
6. ROS2 package documentation (launch files, node descriptions, topic/service interfaces)
7. Master's thesis report documenting design, implementation, and validation
8. (Optional) Partial STM32/micro-ROS port with migration guide

Prerequisites

- Programming skills in Python, familiarity with C/C++
- Understanding of embedded systems and real-time operating systems
- Familiarity with Linux development environment
- ROS2 knowledge beneficial but not required
- Basic understanding of satellite operations helpful but not required

Why This Thesis?

- **Cutting-edge Technology:** Work with Space ROS, a modern framework for space applications
- **ESA Alignment:** Research area of interest to European Space Agency
- **Hands-on Experience:** Real hardware testbed with multiple subsystems
- **Open Source:** All work contributes to open-source CubeSat community (GPL-3.0)
- **Team Building:** Foundation for growing student team and future collaborations
- **Industry Relevance:** Skills directly applicable to space industry and robotics

Management

- Weekly coordination meetings ensure project tracking and knowledge sharing.
- All work is documented and shared via GitHub, following open-source guidelines and contributing to the open-source community.
- Master thesis can overlap with other theses and/or ongoing developments in the working group. Collaboration is encouraged but all topics have been designed to not introduce blocking points.

Contact

Simon Vellas: svellas@mail.ntua.gr

Alexis Apostolakis: alexis.apostolakis@gmail.com

Giorgos Athanasiou: georgios.athanasiou.ntua@gmail.com

Expected Start: January 2026