AI Payload Processing Framework with SpaceROS

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

Duration: 6 months

Expected Start: January 2026

Background

Modern CubeSats increasingly rely on on-board AI/ML processing for autonomous operations and data reduction. This thesis focuses on developing a payload processing framework for the CubeSat FlatSat testbed using Space ROS, with emphasis on deploying and managing multiple AI applications on edge computing hardware (NVIDIA Jetson).

The payload subsystem is part of a hybrid architecture where the satellite bus (C&DH, EPS) uses traditional CubeSat protocols (CSP/CAN), while the AI payload leverages SpaceROS [1] internally for its rich robotics and AI ecosystem. This approach, inspired by JAXA's RACS [2] initiative, enables the payload to benefit from modern ROS2 tools while maintaining compatibility with proven satellite bus architectures through a well-defined CSP communication interface.

Objectives

Develop an AI payload processing system that:

1. Uses Space ROS (ROS2) internally for AI processing pipeline and camera integration
2. Implements CSP interface for command/control communication with C&DH
3. Deploys one ORION-Lab-developed operational AI application (cloud detection) with end-to-end validation
4. Deploys one existing AI application available online to demonstrate framework reusability (TBD)
5. Integrates with C&DH via CSP for commands and telemetry
6. Provides optional containerization framework (Phase 2) for application isolation and comparison
7. Optimizes AI models for edge deployment (latency, power, memory constraints)
8. Demonstrates hybrid communication: CSP/CAN for C&DH interface, ROS2 DDS for internal payload data

Scope of Work

Phase 1: AI Framework Development

Jetson Setup

- Configure NVIDIA Jetson with Ubuntu 22.04
- Install SpaceROS (ROS2 Humble or Iron)
- Install ML frameworks (TensorFlow/PyTorch, TensorRT for optimization)
- Configure development environment and tools

Communication Interface with C&DH

The payload receives commands from C&DH and sends telemetry back.

CSP Interface for C&DH Communication:

- Primary: CSP over CAN for command/control messages
- Implement CSP listener for telecommands from C&DH
- Send telemetry and status via CSP packets
- CSP interface implemented as Python module using libcsp (TBD)
- Converts CSP commands into function calls for Application Manager
- Converts Application Manager status/results into CSP telemetry packets

ROS2 DDS for Internal Payload Communication:

- ROS2 DDS over Gigabit Ethernet for high-bandwidth internal communication
- Camera driver publishes images to ROS2 topics
- AI applications subscribe to camera topics via DDS
- AI applications publish results to ROS2 topics
- Application Manager monitors ROS2 nodes

Native ROS2 Application Architecture (Phase 1)

Camera Driver Node:

- ROS2 node publishing camera data
- RGB camera integration via usb_cam or v4l2_camera ROS2 package
- Publishes to /camera/rgb/image_raw topic (standard sensor_msgs/Image)
- IR camera integration prepared for future hardware

AI Application Nodes:

- Implemented as native ROS2 nodes
- Subscribe to camera topics
- Publish results to /ai/<app_name>/output

- Managed via ROS2 launch system or programmatic node management

Application Manager (ROS2 node on host Jetson):

- Listens for telecommands from C&DH (via CSP interface module)
- Parses commands (e.g., START_APP cloud_detection, STOP_APP, LIST_APPS)
- Manages AI application lifecycle (start/stop ROS2 nodes)
- Subscribes to AI result topics
- Publishes application status
- Coordinates with CSP interface to send telemetry to C&DH
- Maintains application registry (JSON config file listing available apps)

Resource Monitoring

- Track CPU, GPU, memory usage (system-wide and per-process)
- Monitor system resources (Jetson health, temperature, power)
- Publish resource telemetry to ROS2 topic
- Forward critical metrics to C&DH via CSP interface
- Alert on resource threshold violations
- Log resource usage for performance analysis

AI Application Implementation - Cloud Detection

- Phase 1a: Develop and test with static images from memory/files (test dataset)
- Phase 1b: Integrate with RGB camera via ROS2 camera node subscription
- Phase 1c: Prepare for IR camera integration when hardware arrives
- Implemented as ROS2 node
- Model training, optimization (quantization, pruning with TensorRT)
- Publishes cloud detection results to ROS2 topic
- Results forwarded to C&DH as telemetry via CSP interface

Integration of Existing AI Application

- Select suitable open-source AI application (TBD)
- Adapt to ROS2 interface (subscribe to camera topics, publish results)
- Implement as ROS2 node following framework patterns
- Test integration with Application Manager
- Demonstrate framework reusability

Phase 2: Containerization & Trade-off Analysis (<u>Optional</u>)

Containerized Application Architecture

- Dockerize AI applications as alternative deployment method
- Each AI application runs as Docker container with ROS2 inside
- Containers use DDS over host network (--network=host)
- Containers built from ROS2 base images with AI dependencies

- Standard interface: subscribe to camera topics, publish results to /ai/<app_name>/output
- Note: Each container includes its own ROS2 installation - Docker + ROS2 pattern
- Application Manager extended to use Docker SDK (Python) for container management

Trade-off Analysis

- Resource overhead (memory, CPU, startup time)
- Isolation effectiveness
- Deployment ease and flexibility
- Performance (inference latency)
- Documented recommendations for CubeSat use cases

Integration & Testing

C&DH Integration

- Receive telecommands via CSP from C&DH
- Send telemetry and AI results to C&DH via CSP
- End-to-end validation: telecommand → app start → inference → results → telemetry
- Test command acknowledgment and error handling

Camera Integration

- Configure ROS2 camera drivers (RGB: usb_cam or v4l2_camera)
- Image preprocessing pipeline
- IR camera driver setup when hardware arrives
- Test image acquisition and processing pipeline

Performance Benchmarking

- Inference time, accuracy, power consumption
- Multi-application scenarios (running 2+ apps simultaneously)
- Resource usage under various conditions
- Latency analysis for command-to-result pipeline
- (Phase 2) Container overhead measurements if containerization implemented

Technical Requirements

- Operating System: Ubuntu 22.04 on NVIDIA Jetson
- Middleware: SpaceROS (ROS2 Humble or Iron)
- Programming Languages: Python for AI/ML and ROS2 nodes, C/C++ for performance-critical components if needed (Cython)
- ML Frameworks: TensorFlow/PyTorch, TensorRT for optimization
- Communication: CSP over CAN (primary for C&DH interface), ROS2 DDS over Gigabit Ethernet (internal payload)

- Containerization (Phase 2 only): Docker (or alternatives e.g. podman)
- Libraries: libcsp, ROS2 client libraries, Docker SDK (Phase 2), TensorFlow/PyTorch, OpenCV
- Version Control: All code contributed to project GitHub repository
- Testing: Unit tests, integration tests, performance benchmarks

Deliverables

1. AI payload framework running on NVIDIA Jetson with SpaceROS
2. CSP interface module for C&DH communication
3. Application Manager (ROS2 node) with telecommand-based app control
4. Operational cloud detection AI application (ROS2 node) with performance metrics
5. Integration of one existing open-source AI application (ROS2 node)
6. Camera integration (RGB working, IR ready for integration)
7. Integration with C&DH subsystem via CSP
8. Resource monitoring and telemetry system
9. (Optional Phase 2) Containerized deployment framework and trade-off analysis
10. Test results and performance analysis
11. Source code, (dockerfiles if Phase 2) and documentation in GitHub repository

Technical Skills

- Python, C/C++, Cython
- Machine learning frameworks (TensorFlow/PyTorch)
- Computer vision and image processing
- Containerization (Docker)
- ROS2 / SpaceROS
- Embedded systems and resource-constrained computing

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

References

[1] Probe A B, Chambers S W, Oyake A, et al. Space ROS: An Open-Source Framework for Space Robotics and Flight Software. In: Proceedings of the 2023 AIAA Forum; AIAA 2023-2709.

[2] H. Kato, D. Hirano, S. Mitani, T. Saito and S. Kawaguchi, "ROS and cFS System (RACS): Easing Space Robotic Development," 2021 IEEE Aerospace Conference (50100), Big Sky, MT, USA, 2021, pp. 1-8, doi: 10.1109/AERO50100.2021.9438288.