Module Guide for RoCam

Team #3, SpaceY
Zifan Si
Jianqing Liu
Mike Chen
Xiaotian Lou

January 12, 2026

1 Revision History

Date	Version	Notes
Nov. 10, 2025	Rev -1	Initial Draft
Nov. 16, 2025	-	Fix Peer Review Comments
Jan. 21, 2026	Rev 0	Complete module guide first version

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
RoCam	Explanation of program name
UC	Unlikely Change

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules as is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules laid out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The computer vision model. Frequent changes to the vision model might be required to adapt the system to new deployment environments.

AC2: The user interface. The user interface may need to be updated based on the feedback of our customers during the field test.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: The hardware platform. The system relies hardware acceleration for running the computer vision model, encoding, and video output. Changing the hardware platform may require refactoring all the related algorithms.

UC2: The tracking algorithm. The system is designed to track only rockets. The tracking algorithms are designed to only handle the mostly deterministic trajectory of a rocket. Changing the tracked object to something other than a rocket will make the tracking algorithm less accurate.

UC3: Fundamental changes to the use case. The system is designed to be operated remotely by a human operator.

5 Module Hierarchy

This section provides an overview of the module design. Modules are organized in a hierarchy decomposed by secrets. The modules listed below represent the complete module hierarchy. Container modules (M1, M6, and M17) will not actually be implemented since they are not leaf modules.

M1: Jetson Module

M2: CV Process Module

M3: Live Video Process Module

M4: Transcode Process Module

M5: Video Stream Abstraction Module

M6: Control Process Module

M7: CV Process Management Module

M8: Live Video Process Management Module

M9: Transcode Process Management Module

M10: Tracking Module

M11: Gimbal Abstraction Module

M12: Serial Abstraction Module

M13: System Status Module

M14: Recording Database Module

M15: State Management Module

M16: API Gateway Module

M17: UI Module

M18: Preview Module

M19: Manual Control Module

M20: Recording Management Module

M21: Configuration Module

6 Connection Between Requirements and Design

This design decomposes RoCam into modules that directly realize the SRS requirements. Tables in Section 8 provide requirement-to-module traceability (e.g., Table 1). This section complements the tables by recording the main architectural decisions that map the SRS requirements to responsible modules and interfaces.

6.1 Design Decisions Traced to SRS Requirements

This section records key architectural decisions made to satisfy the SRS. For each decision, we summarize the intent, the modules responsible for the decision, and the traced SRS requirement identifiers.

DD-1: Single control surface via REST API

- **Decision:** All operator commands and status queries are mediated by a single API Gateway. The UI does not directly access Jetson processes or hardware.

- **Rationale:** A single, stable interface supports remote operation over a local network and isolates UI changes from backend changes. It also provides one place to enforce validation, authorization, and consistent error handling.
- **Modules:** M16 (primary), M17 (client), M15 (state/query source)
- **SRS Trace:** SCR-3; FR-6, FR-10, FR-11, FR-12

DD-2: Explicit state machine with safe transitions

- **Decision:** System operation is modeled as explicit modes (e.g., Disarmed/Armed/Tracking/Rec with all transitions owned and validated by the State Management module.
- **Rationale:** A centralized state machine prevents invalid mode combinations, makes safety constraints enforceable, and provides predictable behavior for core workflows. Manual override is treated as a controlled transition rather than ad-hoc interruption.
- **Modules:** M15 (primary), M6 (enforces/propagates), M16 (transition entry point)
- **SRS Trace:** SCR-1, SCR-4; FR-4, FR-5, FR-6

DD-3: Validated command path for hardware safety

- **Decision:** All gimbal motion commands (manual or automatic) pass through a validated command path before actuation (range checks, rate limiting, and state checks).
- **Rationale:** Hardware actuation is safety-critical. Validation prevents invalid or unsafe commands (e.g., out-of-range angles) and ensures manual control is only available in permitted modes while tracking commands remain autonomous during tracking.
- **Modules:** M16 (validation entry), M15 (mode gating), M11 (actuation), M19 (manual input), M10 (automatic commands)
- **SRS Trace:** IR-1; FR-2, FR-6, FR-7

DD-4: Process separation for real-time performance

- **Decision:** Vision/tracking, live output, and offline transcoding are separated into dedicated worker processes.
- **Rationale:** Separating CPU/GPU-heavy workloads prevents offline work (e.g., transcode) from degrading tracking responsiveness. It also allows independent scheduling and failure isolation to maintain sustained throughput for 1080p 60 fps operation and continuous output.
- **Modules:** M2, M3, M4 (worker processes), M5 (stream abstraction)
- **SRS Trace:** SLR-2, SLR-3

DD-5: Dedicated worker supervision and recovery

- **Decision:** Worker lifecycles (start/stop/health monitoring/restart) are managed by explicit management modules under the Control Process.
- **Rationale:** Supervising workers improves robustness and observability: failures can be detected, surfaced to the operator, and recovered when possible; unrecoverable faults trigger safe behavior.
- **Modules:** M6 (orchestration), M7, M8, M9
- **SRS Trace:** RFR-1, RFR-2

DD-6: Offline-first operation on local network

- **Decision:** RoCam operates without public internet access; all control and status exchange occurs over the local network through the API.
- **Rationale:** Field deployments may lack reliable internet. Offline-first design improves reliability, reduces external dependencies, and aligns with the remote-operation constraint over LAN.
- **Modules:** M16, M17 (LAN client), M15 (local status), M14 (local storage)
- **SRS Trace:** RFR-3, AS-1; SCR-3

DD-7: Recording and recording management as a first-class workflow

- **Decision:** Recording is initiated from the UI and stored with metadata to support listing, downloading, and deletion; transcoding is performed asynchronously from tracking.
- **Rationale:** Recording is a primary operational requirement and must not degrade real-time tracking. Separating recording storage/metadata from transcoding supports post-flight review while preserving real-time performance.
- **Modules:** M20 (UI workflow), M14 (metadata/files), M2 (capture), M4 (offline conversion), M16 (endpoints)
- **SRS Trace:** FR-11, FR-12; SLR-2, SLR-3

DD-8: HDMI live output via stream abstraction

- **Decision:** The live output pipeline is treated as a required external interface and is implemented through a stream abstraction module to isolate encoder/pipeline choices.
- **Rationale:** Treating HDMI output as an explicit interface ensures compatibility with adjacent systems. Encapsulation makes it easier to swap pipeline configurations without changing higher-level logic.
- **Modules:** M5 (abstraction), M3 (output process), M1 (platform)
- **SRS Trace:** INT-1; SLR-3

DD-9: Auditable operator actions and system events

- **Decision:** Operator actions and key system events (state transitions, record start/stop, file deletion/download) are logged with timestamps and relevant metadata.
- **Rationale:** Audit logs support post-flight analysis, troubleshooting, and accountability. Centralizing logging through state/record management ensures consistency across features.
- **Modules:** M15 (state/action events), M14 (recording events/files), M16 (request context)
- **SRS Trace:** AUR-1

6.2 Traceability Conventions

In the traceability tables, each SRS requirement identifier (e.g., FR, SLR, SCR, IR, AUR, INT, RFR, PAR, AR/SR, EZ/PI/UPR/LR, EPE, RR, CR, AS) is mapped to the module(s) responsible for implementing and supporting that requirement.

The conventions used are:

- **Implementation responsibility:** A requirement is traced to the *leaf module(s)* that implement the required behavior. Container modules (e.g., M1, M6, M17) may appear only when the requirement is inherently system-level or spans multiple leaf modules.
- **Primary vs. supporting modules:** When one module is the main owner of the behavior (e.g., M15 for mode transitions), it is listed first. Additional modules are included when they provide essential support (e.g., UI initiation, API mediation, hardware actuation, storage).
- **Cross-cutting requirements:** Requirements that affect multiple parts of the system (e.g., SCR-3 remote operation, RFR-1 error reporting, auditability AUR-1) are intentionally traced to both: (i) the *interface modules* that expose the behavior (UI/API), and (ii) the *backend modules* that realize the underlying functionality.
- **Verification intent:** Traced modules are expected to provide the artifacts needed to verify the requirement (e.g., unit tests for pure logic modules, integration tests for API/UI interactions, and end-to-end tests for performance and operational requirements).

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *RoCam* means the module will be implemented by the RoCam software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown,this means that the module is not a leaf and will not have to be implemented.

7.1 Jetson Module (M1)

The jetson module contains all the code that runs on the Nvidia Jetson.

7.2 CV Process Module (M2)

Secrets:

- How to retrieve and process the video feed from camera
- Computer vision algorithms
- How to start/stop recording

Services:

- Detects rocket location from the camera feed
- Record camera feed to disk
- Generate preview video feed
- Generate live stream video feed

Implemented By: RoCam

Type of Module: Process

7.3 Live Video Process Module (M3)

Secrets:

- How to stabilize and overlay text to the live video feed
- How to handle CV process crashes
- How to output video to HDMI

Services:

- Stablize and overlay text to the live stream video feed
- Output the live stream video feed to HDMI

Implemented By: RoCam

Type of Module: Process

7.4 Transcode Process Module (M4)

Secrets:

- How to encode recorded video to a more efficient format
- How to stablize and overlay text to the recorded video

Services: Transcode a recording with optional stablization and text overlay

Implemented By: RoCam

Type of Module: Process

7.5 Video Stream Abstraction Module (M5)

Secrets: How to efficiently handle video stream data

Services: Video stream retrieval, routing, processing, and output.

Implemented By: Nvidia Deepstream SDK

Type of Module: Library

7.6 Control Process Module (M6)

The control process module orchestrates all the processes and manages the system state.

7.7 CV Process Management Module (M7)

Secrets:

- How to start the CV process
- How to communicate with the CV process

Services: Allows the Control Process Module (M6) to access service provided by the CV Process Module (M2)

Implemented By: RoCam

Type of Module: Abstract Object

7.8 Live Video Process Management Module (M8)

Secrets:

- How to start the Live Video process
- How to communicate with the Live Video process

Services: Allows the Control Process Module (M6) to access service provided by the Live Video Process Module (M3)

Implemented By: RoCam

Type of Module: Abstract Object

7.9 Transcode Process Management Module (M9)

Secrets:

- How to start the Transcode process
- How to communicate with the Transcode process

Services: Allows the Control Process Module (M6) to access service provided by the Transcode Process Module (M4)

Implemented By: RoCam

Type of Module: Abstract Object

7.10 Tracking Module (M10)

Secrets: The tracking algorithm

Services: Calculates the angle of the gimbal required to keep the rocket in the frame.

Implemented By: RoCam

Type of Module: Abstract Object

7.11 Gimbal Abstraction Module (M11)

Secrets: The communication protocol of the gimbal

Services:

- Command the gimbal to go to a specified angle
- Measure the current angle of the gimbal

Implemented By: RoCam

Type of Module: Abstract Object

7.12 Serial Abstraction Module (M12)

Secrets: How to interface with the hardware serial ports.

Services: Send or receive bytes through the serial port of the hardware.

Implemented By: pip package pyserial

Type of Module: Library

7.13 System Status Module (M13)

Secrets: How to retrieve various system status

Services: Provide various system status (e.g. GPU utilization, core temperature)

Implemented By: pip package jetson-stats

Type of Module: Library

7.14 Recording Database Module (M14)

Secrets: How to track all the recordings and logs

Services: Track all the files that store recordings and logs

Implemented By: RoCam

Type of Module: Abstract Object

7.15 State Management Module (M15)

Secrets:

- The data structure to store the state of the system
- State transition logic

Services:

- Manages the state transitions of the system
- Aggregate data for the API Gateway

Implemented By: RoCam

Type of Module: Abstract Object

7.16 API Gateway Module (M16)

Secrets: The API endpoints and their corresponding logic.

Services: Provides the API endpoints for the UI.

Implemented By: RoCam

Type of Module: Abstract Object

7.17 UI Module (M17)

The UI module contains all the code that the operator uses to interact with the system via the web interface.

7.18 Preview Module (M18)

Secrets: How to retrieve and display the video feed from the API Gateway.

Services: Displays preview of the video feed to the user.

Implemented By: RoCam

Type of Module: Library

7.19 Manual Control Module (M19)

Secrets: Intuitive user interface for manual control, and how to send manual control commands to the API Gateway.

Services: Allows the operator to control the gimbal manually via a user interface.

Implemented By: RoCam

Type of Module: Library

7.20 Recording Management Module (M20)

Secrets: How to list, delete, and download the recorded videos and log files.

Services: Allows the operator to list, delete, and download the recorded videos and log files.

Implemented By: RoCam

Type of Module: Library

7.21 Configuration Module (M21)

Secrets: The UI for configuring the system.

Services: Allows the operator to configure the system via a user interface.

Implemented By: RoCam

Type of Module: Library

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
FR-1	M1, M3, M2, M6, M7, M8
FR-2	M11, M10, M19, M16, M6
FR-3	M2, M1, M6, M7
FR-4	M15, M11, M10, M2, M6, M7
FR-5	M15, M2, M10, M6
FR-6	M19, M16, M11, M15, M6
FR-7	M10, M2, M11, M15, M6
FR-8	M1, M2, M3, M6, M7, M8
FR-9	M3, M11, M6, M8
FR-10	M18, M17, M15, M16, M2, M13
FR-11	M2, M14, M20, M16, M17, M6, M4, M9
FR-12	M20, M16, M14, M17, M6, M4, M9

Table 1: Trace Between Functional Requirements and Modules

Req.	Modules
AR-1	M17, M20, M21
AR-2	M17, M20
SR-1	M17, M21

Table 2: Trace Between Appearance and Style Requirements and Modules

Req.	Modules
EZ-1	M17
EZ-2	M17, M16
EZ-3	M17, M16, M21, M13
EZ-4	M17, M21, M16, M15
PI-1	M17, M21, M18, M20
PI-2	M17, M21
LR-1	M17
UPR-1	M17
UPR-2	M17, M16
UPR-3	M17
AR-1	M17, M15, M16

Table 3: Trace Usability and Humanity Requirements and Modules

Req.	Modules
SLR-1	M1, M2, M10, M11, M15, M6, M7
SLR-2	M1, M2, M3, M10, M6, M7
SLR-3	M3, M1, M16, M6, M8
SCR-1	M17, M19, M15, M16, M11, M6
SCR-2	M10, M15, M2, M11, M1, M6
SCR-3	M17, M16, M15, M1, M6
SCR-4	M17, M15, M16, M11, M6
PAR-1	M10, M2, M11, M3, M16, M1, M6
RFR-1	M17, M16, M15, M6, M13
RFR-2	M15, M10, M11, M16, M6
RFR-3	M1, M16, M17, M15, M6
RFR-4	M10, M2, M15, M11, M1, M6
RFR-5	M10, M2, M11, M15, M6
CR-1	M14, M20, M1, M6, M4, M9

Table 4: Trace Between Performance Requirements and Modules

Req.	Modules
EPE-1	M1, M11, M13
INT-1	M3, M1, M6, M8
INT-2	M11, M12, M16, M1, M6
RR-1	M14, M20, M1

Table 5: Trace Between Operational and Environmental Requirements and Modules

Req.	Modules
IR-1	M17, M19, M16, M15, M11, M1, M10
AUR-1	M14, M20, M16, M15, M17

Table 6: Trace Between Security Requirements and Modules

Req.	Modules
CR-1	M17, M18, M21
CR-2	M17, M18, M21

Table 7: Trace Between Cultural Requirements and Modules

Req.	Modules
LR-1	M17, M14, M20, M16, M1

Table 8: Trace Between Compliance Requirements and Modules

Anticipated Change	Modules
AC1	M2, M1
AC2	M17, M18, M19, M21

Table 9: Trace Between Anticipated Changes and Modules

Unlikely Change	Modules
UC1	M1, M11, M12, M5
UC2	M10, M2
UC3	M17, M16, M19

Table 10: Trace Between Unlikely Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

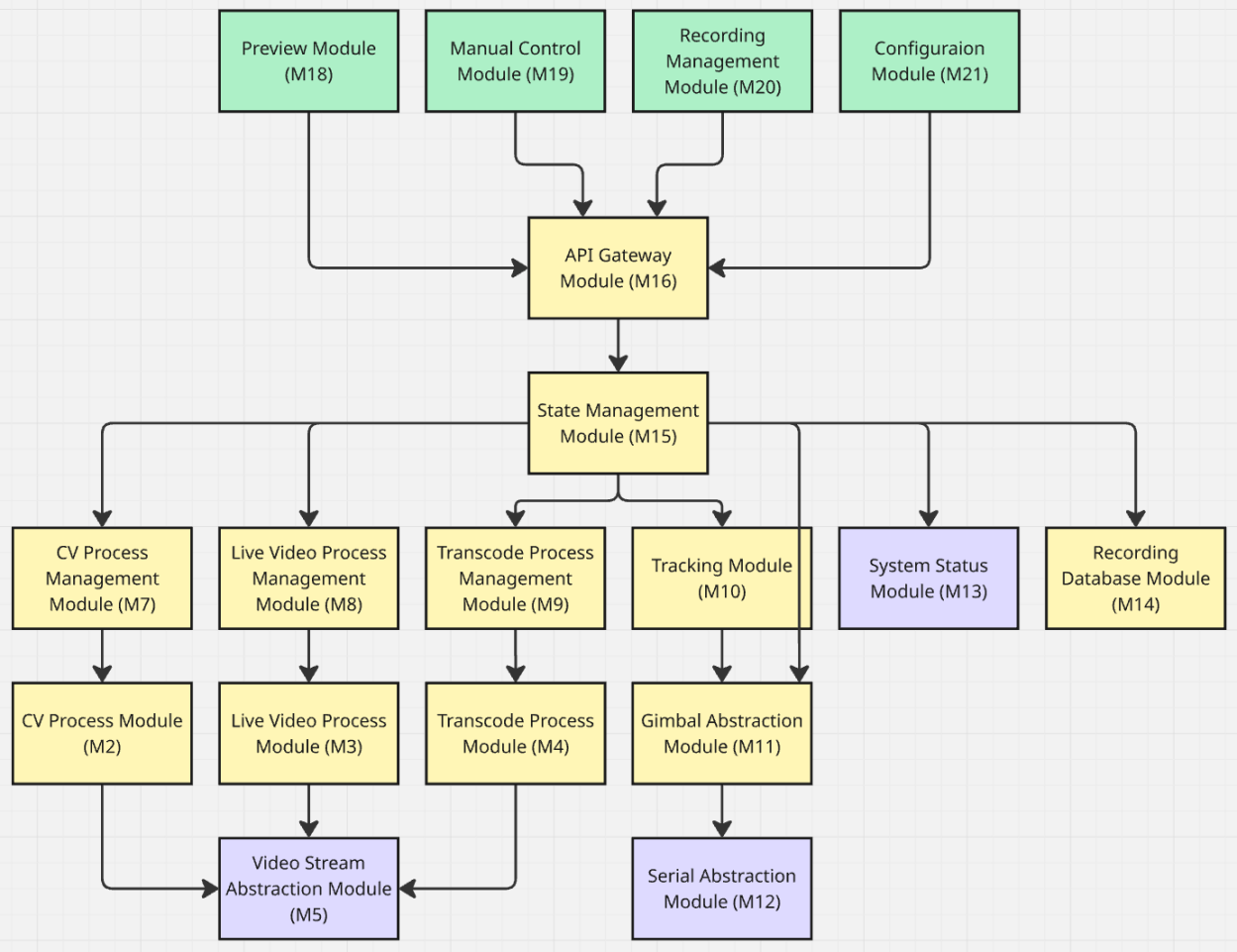

Figure 1: Use hierarchy among modules

10 User Interfaces

The system provides a single-page web interface for the operator. This web UI is the only entry point for interacting with the system, including live preview, state switching, manual gimbal control, and recording management.

10.1 Main Control Page

Figure 2 shows the main *Control* page. The left panel displays the live camera feed for situational awareness. The right panel displays the current system state (e.g., *Disarmed* or *Armed*) and real-time gimbal pose telemetry, including *Tilt* and *Pan* angles. The top-right corner includes *Fullscreen* for better field visibility and an *Emergency Stop* control for safety-critical shutdown.

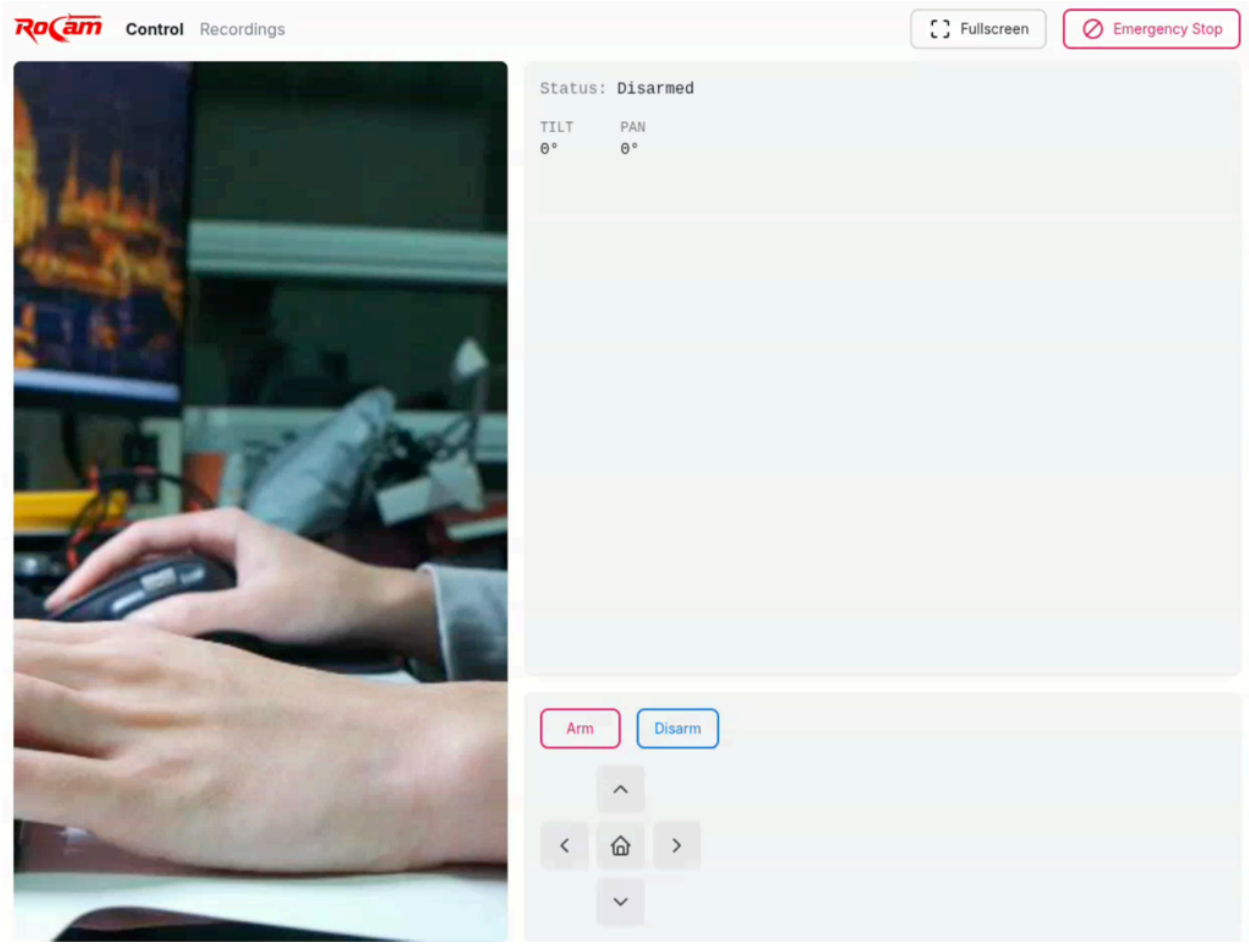

Figure 2: RoCam web interface overview (Control page).

10.2 Disarmed Mode: Manual Gimbal Control

In *Disarmed* mode, the operator can manually control the gimbal using the on-screen directional pad (up/down/left/right). This mode is used for setup and calibration (e.g., aiming at the expected launch area before liftoff) and as a fallback when manual intervention is required. Manual control commands are sent to the backend through the API Gateway, which relays them to the gimbal control logic running on the Jetson. The UI continuously updates the displayed gimbal angles so the emphasizes to the operator the immediate effect of each command.

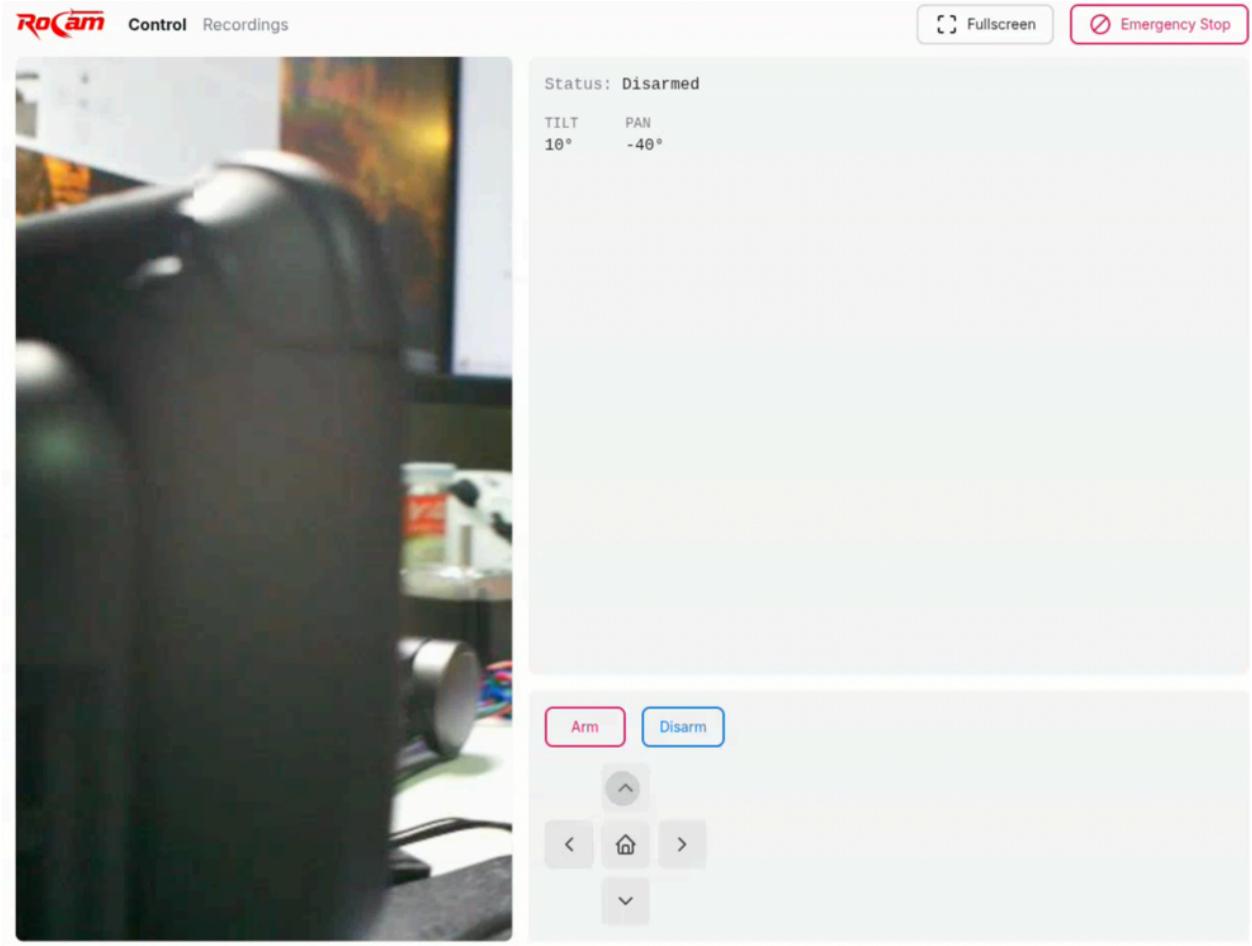

Figure 3: Disarmed mode with manual gimbal controls and real-time tilt/pan feedback.

10.3 Armed Mode: Automatic Tracking

In *Armed* mode, the system runs the rocket detection and tracking pipeline. The Computer Vision and Tracking modules estimate the target location in the frame and compute the gimbal control commands to keep the rocket centered. The operator monitors the live preview and can transition to *Disarmed* mode to take manual control if needed.

10.4 Recordings

The UI also provides a *Recordings* section for managing recorded videos (and associated logs/metadata). The operator can start/stop recordings during operation and later list, download, or delete recordings through the Recording Management functions exposed by the backend API.

11 Design of Communication Protocols

11.1 Gimbal Protocol

The Gimbal Protocol is specified at: https://github.com/SpaceY-Labs/RoCam/blob/main/docs/Design/GimbalProtocol/GimbalProtocol.pdf

11.2 Operator Interface API Protocol

The Operator Interface API Protocol is specified at: https://github.com/SpaceY-Labs/RoCam/blob/main/docs/Design/EndpointsSpecification/EndpointsSpecification.pdf

12 Timeline

This project tracks a module-level implementation schedule for Rev 0, including task ownership and lightweight verification checkpoints to build confidence in the integrated system. The authoritative and up-to-date plan (tasks, assignees, dependencies, and status) is maintained in GitHub Projects:

RoCam Rev 0 Timeline (GitHub Projects): https://github.com/orgs/SpaceY-Labs/projects/2

12.1 Rev 0 Timeline Snapshot

12.1.1 Phase 1: Core Interfaces and State Backbone

Dates	Modules	Deliverables / Checks
Jan 21–Jan 24	M12, M11	Serial I/O wrapper + gimbal command/telemetry API (pan/tilt setpoint, readback); bounds/rate checks; bench test with dummy commands.
Jan 22–Jan 26	M15	Define modes + valid transitions (Disarmed/Armed/Tracking/Recording); unit tests for transition validity.
Jan 23–Jan 28	M16	Implement core REST endpoints (status, state transitions, manual control, record start/stop, recordings list/download/delete); basic contract tests vs endpoint spec.

Table 11: Phase 1 schedule (snapshot). Owners and latest status are maintained in GitHub Projects.

12.1.2 Phase 2: Runtime Pipelines and UI Workflows

Dates	Modules	Deliverables / Checks
Jan 24–Jan 30	M2	CV worker: camera ingest, detection output format, preview stream output, recording start/stop hooks; smoke test on sample clip.
Jan 25–Feb 01	M10	Tracking logic: detection→gimbal angles; clamp/rate limit; integrate with M15 gating; offline replay test on saved detections.
Jan 26–Feb 02	M3, M5	Live output pipeline: overlay telemetry, HDMI output path, handle CV worker restart signaling; sustained live output demo.
Jan 27–Feb 03	M14, M4	Recording metadata DB + file indexing; transcode job (async) with optional overlay; create→transcode→playback check.
Jan 24–Feb 04	M18, M19, M20, M21	UI workflows: preview display, manual D-pad control with telemetry, recording management (list/download/delete), basic configuration UI; end-to-end UI↔API tests.

Table 12: Phase 2 schedule (snapshot). Owners and latest status are maintained in GitHub Projects.

12.1.3 Phase 3: Orchestration, Recovery, and Integrated Demo

Dates	Modules	Deliverables / Checks
Feb 02–Feb 06	M6, M7, M8, M9	Worker supervision: start/stop, health monitoring, restart, status surfaced to UI; fault-injection check (kill worker→recover).
Feb 05–Feb 08	System-level	Integrated demo script: Disarmed manual aim→Arm→Track→Record→Download/Review; final checklist run.

Table 13: Phase 3 schedule (snapshot). Owners and latest status are maintained in GitHub Projects.

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.