Module Interface Specification 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. 19, 2025	Rev 0	Team Work
Jan. 21, 2025	Rev 0	Team Review

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at:
https://github.com/ZifanSi/vision-guided-tracker/blob/main/docs/SRS/SRS.pdf

Abbreviation	Definition
API	Application Programming Interface. A set of protocols and tools for building software applications, used here for communication between the UI and backend services.
CV	Computer Vision. The field of computer science that focuses on enabling computers to interpret and understand visual information from the world.
FPS	Frames Per Second. A measure of video or animation performance indicating how many frames are displayed per second.
GPS	Global Positioning System. A satellite-based navigation system that provides location and time information.
HDMI	High-Definition Multimedia Interface. A digital interface standard for transmitting uncompressed video and audio data.
HTTP	Hypertext Transfer Protocol. A protocol used for transmitting data over the internet, commonly used for web APIs.
IPC	Inter-Process Communication. A mechanism for exchanging data between separate processes, typically using message passing or shared memory.
LED	Light Emitting Diode. A semiconductor light source used for status indicators and displays.
MIS	Module Interface Specification. This document type that specifies the interfaces and behaviors of software modules.
OSD	On-Screen Display. Textual or graphical information overlaid on video frames, such as telemetry data.
SDK	Software Development Kit. A collection of software tools and libraries provided by a vendor to facilitate application development.
SRS	System Requirements Specification. A document that describes the functional and non-functional requirements of a system.
UI	User Interface. The visual and interactive components through which users interact with the system.

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **ii**

3 Introduction **1**

4 Notation **1**

5 Data Structures **2**
- 5.1 BoundingBox . . . 2
- 5.2 CVData . . . 3
- 5.3 OSDData . . . 3
- 5.4 StopRecording . . . 4
- 5.5 RecordingInfo . . . 4
- 5.6 SystemStatus . . . 4

6 Module Decomposition **4**

7 MIS of CV Process Module (M2) **6**
- 7.1 Module . . . 6
- 7.2 Uses . . . 6
- 7.3 Syntax . . . 6
  - 7.3.1 Exported Constants . . . 6
  - 7.3.2 Exported Access Programs . . . 6
- 7.4 Semantics . . . 6
  - 7.4.1 State Variables . . . 6
  - 7.4.2 Environment Variables . . . 6
  - 7.4.3 Assumptions . . . 6
  - 7.4.4 Access Routine Semantics . . . 7
  - 7.4.5 Local Functions . . . 7

8 MIS of Live Video Process Module (M3) **8**
- 8.1 Module . . . 8
- 8.2 Uses . . . 8
- 8.3 Syntax . . . 8
  - 8.3.1 Exported Constants . . . 8
  - 8.3.2 Exported Access Programs . . . 8
- 8.4 Semantics . . . 8
  - 8.4.1 State Variables . . . 8
  - 8.4.2 Environment Variables . . . 8
  - 8.4.3 Assumptions . . . 8
  - 8.4.4 Access Routine Semantics . . . 8
  - 8.4.5 Local Functions . . . 9

9 MIS of Transcode Process Module (M4) 9
9.1 Module . . . 9
9.2 Uses . . . 9
9.3 Syntax . . . 9
9.3.1 Exported Constants . . . 9
9.3.2 Exported Access Programs . . . 9
9.4 Semantics . . . 9
9.4.1 State Variables . . . 9
9.4.2 Environment Variables . . . 9
9.4.3 Assumptions . . . 9
9.4.4 Access Routine Semantics . . . 10
9.4.5 Local Functions . . . 10

10 MIS of Video Stream Abstraction Module (M5) 10

11 MIS of CV Process Management Module (M7) 10
11.1 Module . . . 10
11.2 Uses . . . 10
11.3 Syntax . . . 10
11.3.1 Exported Constants . . . 10
11.3.2 Exported Access Programs . . . 11
11.4 Semantics . . . 11
11.4.1 State Variables . . . 11
11.4.2 Environment Variables . . . 11
11.4.3 Assumptions . . . 11
11.4.4 Access Routine Semantics . . . 11
11.4.5 Local Functions . . . 12

12 MIS of Live Video Process Management Module (M8) 12
12.1 Module . . . 12
12.2 Uses . . . 12
12.3 Syntax . . . 13
12.3.1 Exported Constants . . . 13
12.3.2 Exported Access Programs . . . 13
12.4 Semantics . . . 13
12.4.1 State Variables . . . 13
12.4.2 Environment Variables . . . 13
12.4.3 Assumptions . . . 13
12.4.4 Access Routine Semantics . . . 13
12.4.5 Local Functions . . . 13

13 MIS of Transcode Process Management Module (M9) 14
13.1 Module . . . 14
13.2 Uses . . . 14
13.3 Syntax . . . 14

13.3.1 Exported Constants . . . 14
13.3.2 Exported Access Programs . . . 14
13.4 Semantics . . . 14
13.4.1 State Variables . . . 14
13.4.2 Environment Variables . . . 14
13.4.3 Assumptions . . . 14
13.4.4 Access Routine Semantics . . . 14
13.4.5 Local Functions . . . 15

14 MIS of Tracking Module (M10) 15
14.1 Module . . . 15
14.2 Uses . . . 15
14.3 Syntax . . . 15
14.3.1 Exported Constants . . . 15
14.3.2 Exported Access Programs . . . 15
14.4 Semantics . . . 15
14.4.1 State Variables . . . 15
14.4.2 Formal Specification (LO_SpecMath) . . . 15
14.4.3 Environment Variables . . . 16
14.4.4 Assumptions . . . 16
14.4.5 Access Routine Semantics . . . 16
14.4.6 Local Functions . . . 17

15 MIS of Gimbal Abstraction Module (M11) 17
15.1 Module . . . 17
15.2 Uses . . . 17
15.3 Syntax . . . 17
15.3.1 Exported Constants . . . 17
15.3.2 Exported Access Programs . . . 17
15.4 Semantics . . . 18
15.4.1 State Variables . . . 18
15.4.2 Environment Variables . . . 18
15.4.3 Assumptions . . . 18
15.4.4 Access Routine Semantics . . . 18
15.4.5 Local Functions . . . 19

16 MIS of Serial Abstraction Module (M12) 19
16.1 Module . . . 19
16.2 Uses . . . 19
16.3 Syntax . . . 20
16.3.1 Exported Constants . . . 20
16.3.2 Exported Access Programs . . . 20
16.4 Semantics . . . 20
16.4.1 State Variables . . . 20
16.4.2 Environment Variables . . . 20

16.4.3 Assumptions . . . 20
16.4.4 Access Routine Semantics . . . 21
16.4.5 Local Functions . . . 21

17 MIS of System Status Module (M13) 21
17.1 Module . . . 21
17.2 Uses . . . 21
17.3 Syntax . . . 21
17.3.1 Exported Constants . . . 21
17.3.2 Exported Access Programs . . . 22
17.4 Semantics . . . 22
17.4.1 State Variables . . . 22
17.4.2 Environment Variables . . . 22
17.4.3 Assumptions . . . 22
17.4.4 Access Routine Semantics . . . 22
17.4.5 Local Functions . . . 23

18 MIS of Recording Database Module (M14) 23
18.1 Module . . . 23
18.2 Uses . . . 23
18.3 Syntax . . . 23
18.3.1 Exported Constants . . . 23
18.3.2 Exported Access Programs . . . 23
18.4 Semantics . . . 24
18.4.1 State Variables . . . 24
18.4.2 Formal Specification (LO_SpecMath) . . . 24
18.4.3 Environment Variables . . . 25
18.4.4 Assumptions . . . 25
18.4.5 Access Routine Semantics . . . 25
18.4.6 Local Functions . . . 28

19 MIS of State Management Module (M15) 28
19.1 Module . . . 28
19.2 Uses . . . 28
19.3 Syntax . . . 28
19.3.1 Exported Constants . . . 28
19.3.2 Exported Access Programs . . . 29
19.4 Semantics . . . 29
19.4.1 State Variables . . . 29
19.4.2 Formal Specification (LO_SpecMath) . . . 29
19.4.3 Environment Variables . . . 30
19.4.4 Assumptions . . . 31
19.4.5 Access Routine Semantics . . . 31
19.4.6 Local Functions . . . 34

20 MIS of API Gateway Module (M16) **35**
20.1 Module . . . 35
20.2 Uses . . . 36
20.3 Syntax . . . 36
20.3.1 Exported Constants . . . 36
20.3.2 Exported Access Programs . . . 36
20.4 Semantics . . . 36
20.4.1 State Variables . . . 36
20.4.2 Environment Variables . . . 36
20.4.3 Assumptions . . . 36
20.4.4 Access Routine Semantics . . . 36
20.4.5 API Endpoint Semantics . . . 37
20.4.6 Local Functions . . . 39

21 MIS of Preview Module (M18) **39**
21.1 Module . . . 39
21.2 Uses . . . 39
21.3 Syntax . . . 39
21.3.1 Exported Constants . . . 39
21.3.2 Exported Access Programs . . . 39
21.4 Semantics . . . 39
21.4.1 State Variables . . . 39
21.4.2 Environment Variables . . . 39
21.4.3 Assumptions . . . 40
21.4.4 Access Routine Semantics . . . 40
21.4.5 Local Functions . . . 41

22 MIS of Manual Control Module (M19) **41**
22.1 Module . . . 41
22.2 Uses . . . 41
22.3 Syntax . . . 41
22.3.1 Exported Constants . . . 41
22.3.2 Exported Access Programs . . . 41
22.4 Semantics . . . 41
22.4.1 State Variables . . . 41
22.4.2 Environment Variables . . . 42
22.4.3 Assumptions . . . 42
22.4.4 Access Routine Semantics . . . 42
22.4.5 Local Functions . . . 43

23 MIS of Recording Management Module (M20) **44**
23.1 Module . . . 44
23.2 Uses . . . 44
23.3 Syntax . . . 44
23.3.1 Exported Constants . . . 44

23.3.2 Exported Access Programs . . . 44
23.4 Semantics . . . 44
23.4.1 State Variables . . . 44
23.4.2 Environment Variables . . . 44
23.4.3 Assumptions . . . 45
23.4.4 Access Routine Semantics . . . 45
23.4.5 Local Functions . . . 47

24 MIS of Configuration Module (M21) 47
24.1 Module . . . 47
24.2 Uses . . . 48
24.3 Syntax . . . 48
24.3.1 Exported Constants . . . 48
24.3.2 Exported Access Programs . . . 48
24.4 Semantics . . . 48
24.4.1 State Variables . . . 48
24.4.2 Formal Specification (LO_SpecMath) . . . 48
24.4.3 Environment Variables . . . 49
24.4.4 Assumptions . . . 49
24.4.5 Access Routine Semantics . . . 50
24.4.6 Local Functions . . . 50

25 Appendix 53

3 Introduction

The following document details the Module Interface Specifications for Rocam: High Performance Vision-Guided Rocket Tracker.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at:

https://github.com/ZifanSi/vision-guided-tracker

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by RoCam.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of RoCam uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, RoCam uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

The following table summarizes the programming language type notations used in this specification:

Type Notation	Mathematical Type	Description
int	$\mathbb{Z}$	32-bit signed integer.
float	$\mathbb{R}$	32-bit single-precision floating point (IEEE 754).
f32	$\mathbb{R}$	32-bit single-precision floating point (IEEE 754).
f64	$\mathbb{R}$	64-bit double-precision floating point (IEEE 754).
u64	$\mathbb{N}$	64-bit unsigned integer.
bool	{true, false}	Boolean value (true or false).
str	sequence of char	String (sequence of characters).
bytes	sequence of byte	Sequence of bytes (8-bit unsigned integers).
null | Type	Type $\cup$ {null}	Optional/nullable type (union with null).
'literal1' | 'literal2'	{'literal1', 'literal2'}	Union of string literals (enum-like type).
Type[]	sequence of Type	Array or sequence of elements of type Type.
Optional[Type]	Type $\cup$ {null}	Optional type (Python-style notation).
Tuple[Type1, Type2]	Type1 $\times$ Type2	Tuple containing values of Type1 and Type2.
(Type1, Type2) $\to$ ReturnType	Type1 $\times$ Type2 $\to$ ReturnType	Function type taking parameters of Type1 and Type2, returning ReturnType.

5 Data Structures

This section defines all data structures used across the modules in the system. These structures are shared between modules and are defined here for reference.

5.1 BoundingBox

Coordinates are normalized (0.0 to 1.0).

Field Name	Type	Description
confidence	float	Confidence score of the detection, 0.0 to 1.0
left	float	Left coordinate (normalized)
top	float	Top coordinate (normalized)
width	float	Width (normalized)
height	float	Height (normalized)

5.2 CVData

Field Name	Type	Description
frame_number	int	Frame number of the current frame, every frame is numbered sequentially
average_fps	float	Average frames per second over the last 60 frames
preview	bytes	JPEG encoded preview of the current frame for the frontend. The frame is rotated 90 degrees counter-clockwise, the frontend needs to rotate it 90 degrees clockwise to display it correctly
bbox	null | `BoundingBox`	Bounding box of the rocket in the current frame, null if no rocket is found

5.3 OSDData

Field Name	Type	Description
frame_number	int	The frame number this OSD data is for
translate_x	int	Transformation data used to digitally stabilize the frame
translate_y	int	Transformation data used to digitally stabilize the frame
scale	float	Transformation data used to digitally stabilize the frame
average_fps	float	Text to be rendered on the frame
gimbal_tilt_deg	float	Text to be rendered on the frame
gimbal_pan_deg	float	Text to be rendered on the frame
gimbal_focal_length_mm	float	Text to be rendered on the frame
device_ip_addresses	str[]	Text to be rendered on the frame
timestamp_ms	int	Text to be rendered on the frame
tracking_state	'idle' | 'armed' | 'tracking'	Text to be rendered on the frame
longitude	float	Longitude coordinate
latitude	float	Latitude coordinate

5.4 StopRecording

Empty structure with no fields.

5.5 RecordingInfo

Field Name	Type	Description
id	str	Recording identifier
name	str	Recording name
start_time	str | null	Start time of the recording
duration_seconds	int | null	Duration of the recording in seconds
video_path	str	Path to the video file
log_path	str	Path to the log file

5.6 SystemStatus

Field Name	Type	Description
average_fps	float	Average frames per second
gimbal_tilt_deg	float	Gimbal tilt angle in degrees
gimbal_pan_deg	float	Gimbal pan angle in degrees
gimbal_focal_length_mm	float	Gimbal focal length in millimeters
tracking_state	'idle' | 'armed' | 'tracking'	Current tracking state
preview	bytes	Preview image data
osd_data	`OSDData` | null	On-screen display data, null if not available
timestamp_ms	int	System time in milliseconds
longitude	float | null	Longitude coordinate, null if not available
latitude	float | null	Latitude coordinate, null if not available
gpu_utilization	float	GPU utilization percentage (0.0 to 100.0)
cpu_utilization	float	CPU utilization percentage (0.0 to 100.0)
core_temperature_celsius	float	Core temperature in Celsius
memory_usage_bytes	{used: int, total: int}	Memory usage in bytes
disk_usage_bytes	{used: int, total: int}	Disk usage in bytes
in_progress_recording_id	str | null	ID of the recording currently in progress, null if no recording

6 Module Decomposition

The following list is taken directly from the Module Guide document for this project. 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

7 MIS of CV Process Module (M2)

7.1 Module

cv_process

7.2 Uses

- Video Stream Abstraction Module (M5)

7.3 Syntax

7.3.1 Exported Constants

- WIDTH: Width of the video feed (in pixels).
- HEIGHT: Height of the video feed (in pixels).
- SHARED_MEMORY_PATH: Path to the shared memory file.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	Runs indefinitely	No expected exceptions. If unexpected exception occurs, log the error to stderr and exit.

7.4 Semantics

7.4.1 State Variables

- IPC connection to the CV Process Management Module (M7).
- recording: null | `RecordingInfo` (recording information, null if not recording).
- osd_data: `OSDData` (OSD data for frame transformation and text rendering).

7.4.2 Environment Variables

- External camera sensor.

7.4.3 Assumptions

Camera is assumed to be connected and working properly.

7.4.4 Access Routine Semantics

main():

- transition:
  - Connect to the IPC server of the CV Process Management Module (M7).
  - Start `pipeline()` in a background thread.
  - For each message received from CV Process Management Module (M7).
    - (if `RecordingInfo`): Set state variable `recording` to the recording info.
    - (if `StopRecording`): Set state variable `recording` to null.
    - (if `OSDData`): Set state variable `osd_data` to the OSD data.
- exception: None

7.4.5 Local Functions

pipeline():

- transition:
  - For each frame from the camera.
    - Detect the rocket location in the frame.
    - (always):
      - Rotate the frame 90 degrees clockwise.
      - Translate the frame by `osd_data.translate_x` and `osd_data.translate_y` pixels.
      - Scale the frame by `osd_data.scale`.
      - Render the text from `osd_data` over the frame.
      - Send the frame to the Live Video Process Module (M3) via shared memory.
    - (if `recording`):
      - Append `osd_data` to `recording.log_path`.
      - Encode the frame and save to `recording.video_path`.
    - (skip every 1 frame):
      - Downsample to 480p.
      - Encode the frame to JPEG.
      - Send `CVData` to the CV Process Management Module (M7) via IPC with the following fields:
        - `frame_number`: The frame number.
        - `average_fps`: The average FPS of the video.
        - `bbox`: The detected rocket location.
        - `preview`: encoded preview JPEG image.
- exception: None

8 MIS of Live Video Process Module (M3)

8.1 Module

live_video_process

8.2 Uses

- Video Stream Abstraction Module (M5)

8.3 Syntax

8.3.1 Exported Constants

This module does not have any exported constants.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	Runs indefinitely	No expected exceptions. If unexpected exception occurs, log the error to stderr and exit.

8.4 Semantics

8.4.1 State Variables

This module does not have any state variables.

8.4.2 Environment Variables

- HDMI monitor connected to the Jetson (for displaying frames).

8.4.3 Assumptions

HDMI monitor is assumed to be connected and working properly.

8.4.4 Access Routine Semantics

main():

- transition:
    - For each frame from the CV Process Module (M2).
        - Output the frame to the HDMI monitor.

- exception: None

8.4.5 Local Functions

This module does not have any local functions.

9 MIS of Transcode Process Module (M4)

9.1 Module

transcode_process

9.2 Uses

- Video Stream Abstraction Module (M5)

9.3 Syntax

9.3.1 Exported Constants

This module does not have any exported constants.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	recording: `RecordingInfo` output_video_path: str	None	No expected exceptions. If unexpected exception occurs, log the error to stderr and exit.

9.4 Semantics

9.4.1 State Variables

This module does not have any state variables.

9.4.2 Environment Variables

This module does not have any environment variables.

9.4.3 Assumptions

- The provided `RecordingInfo` is valid.
- The provided `output_video_path` is writable.

9.4.4 Access Routine Semantics

main(recording: `RecordingInfo`, output_video_path: str):

- transition:
  - Read the recorded video from `recording.video_path`.
  - Read the log from `recording.log_path`.
  - For each frame in the recorded video:
    - Find the corresponding `OSDData` from the log.
    - Rotate the frame 90 degrees clockwise.
    - Translate the frame by `osd_data.translate_x` and `osd_data.translate_y` pixels.
    - Scale the frame by `osd_data.scale`.
    - Render the text from `osd_data` over the frame.
    - Encode the frame in a compressed video format and save to `output_video_path`.
- exception: None

9.4.5 Local Functions

This module does not have any local functions.

10 MIS of Video Stream Abstraction Module (M5)

Nvidia Deepstream SDK is too complex to specify in this MIS, even if we only include the functionalities we need. Its full documentation is available at https://docs.nvidia.com/metropolis/deepstream/7.1/index.html.

11 MIS of CV Process Management Module (M7)

11.1 Module

cv_process_management

11.2 Uses

- CV Process Module (M2)

11.3 Syntax

11.3.1 Exported Constants

This module does not have any exported constants.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
start_process	on_cv_data: (cv_data: `CVData`) → None on_process_crash: (error: str) → None	-	
start_recording	recording_info: `RecordingInfo`	-	IPCError
stop_recording	-	-	IPCError
set_osd_data	osd_data: `OSDData`	-	IPCError

11.4 Semantics

11.4.1 State Variables

- IPC connection to the CV Process Module (M2).

11.4.2 Environment Variables

This module does not have any environment variables.

11.4.3 Assumptions

This module does not have any assumptions.

11.4.4 Access Routine Semantics

start_process(on_cv_data: (cv_data: `CVData`) → None, on_process_crash: (error: str) → None):

- transition:
  - Start an IPC server.
  - Start the CV Process Module (M2) in a separate process.
  - Wait for the CV Process Module (M2) to connect to the IPC server.
  - Start `receive_cv_data_loop()` in a background thread.
- output: None
- exception: IPCError if the CV Process Module (M2) does not connect to the IPC server within a timeout period.

start_recording(recording_info: `RecordingInfo`):

- transition: Send the recording info to the CV Process Module (M2) through the IPC connection.
- output: None

- exception: IPCError if failed to send the value through the IPC connection.

stop_recording():

- transition: Send a stop recording message to the CV Process Module (M2) through the IPC connection.
- output: None
- exception: IPCError if failed to send the value through the IPC connection.

set_osd_data(osd_data: `OSDData`):

- transition: Send the OSD data to the CV Process Module (M2) through the IPC connection.
- output: None
- exception: IPCError if failed to send the value through the IPC connection.

11.4.5 Local Functions

receive_cv_data_loop():

- transition:
  - Loop forever.
    - Receive a message from the IPC connection.
    - Call the `on_cv_data` callback with the `CVData` received.
    - If the IPC connection is closed or error occurs, exit the loop.
- output: None
- exception: None

12 MIS of Live Video Process Management Module (M8)

12.1 Module

live_video_process_management

12.2 Uses

- Live Video Process Module (M3)

12.3 Syntax

12.3.1 Exported Constants

This module does not have any exported constants.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
start_process	-	-	-

12.4 Semantics

12.4.1 State Variables

This module does not have any state variables.

12.4.2 Environment Variables

This module does not have any environment variables.

12.4.3 Assumptions

This module does not have any assumptions.

12.4.4 Access Routine Semantics

start_process():

- transition: Run `run_live_video_process()` in a background thread.
- output: None
- exception: processStartError

12.4.5 Local Functions

run_live_video_process():

- transition:
    - Loop forever.
        - Start the Live Video Process Module (M3) in a separate process.
        - Wait for the Live Video Process Module (M3) to exit.
- output: None
- exception: None

13 MIS of Transcode Process Management Module (M9)

13.1 Module

transcode_process_management

13.2 Uses

- Transcode Process Module (M4)

13.3 Syntax

13.3.1 Exported Constants

This module does not have any exported constants.

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
start_transcode	recording: `RecordingInfo`	output_video_stream: str	-

13.4 Semantics

13.4.1 State Variables

This module does not have any state variables.

13.4.2 Environment Variables

This module does not have any environment variables.

13.4.3 Assumptions

- The provided `RecordingInfo` is valid.

13.4.4 Access Routine Semantics

start_transcode(recording: `RecordingInfo`):

- transition:
  - Create a temporary output stream and save the path to `output_video_stream`.
  - Start the Transcode Process Module (M4) in a separate process with the given recording and `output_video_stream`.
- output: output_video_stream: str
- exception: None

13.4.5 Local Functions

This module does not have any local functions.

14 MIS of Tracking Module (M10)

14.1 Module

tracking

14.2 Uses

- Gimbal Abstraction Module (M11)

14.3 Syntax

14.3.1 Exported Constants

This module does not have any exported constants.

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	gimbal: Gimbal Abstraction Module (M11)	-	GimbalError
step	rocket_bbox: `BoundingBox`	-	-

14.4 Semantics

14.4.1 State Variables

- gimbal: Gimbal Abstraction Module (M11)

14.4.2 Formal Specification (LO_SpecMath)

Abstract State. Let the abstract state be:

$$S \triangleq (g)$$

where

$$g \in \mathit{GimbalHandle} \cup \{\bot\}.$$

State Invariant.

$$I_1 : \; g = \bot \;\Leftrightarrow\; \text{module not initialized}$$

Auxiliary Predicates and Functions.
Bounding boxes are normalized:

$\text{validBBox}(b) \triangleq (0 \le b.\text{left} \le 1) \land (0 \le b.\text{top} \le 1) \land (0 \le b.\text{width} \le 1) \land (0 \le b.\text{height} \le 1) \land (b.\text{left}+b.\text{widt}$

Define the bbox center and height:

$$cx(b) \triangleq b.\text{left} + \frac{b.\text{width}}{2}, \qquad cy(b) \triangleq b.\text{top} + \frac{b.\text{height}}{2}, \qquad h(b) \triangleq b.\text{height}.$$

The desired visual objective is:

$$cx(b) = 0.5, \quad cy(b) = 0.5, \quad h(b) = 0.5.$$

We abstract the control law that converts a bbox and current gimbal state into a command (using θ for tilt and ϕ for pan):

$$\text{controlLaw}(b, \theta, \phi, f) = (\theta_d, \phi_d, f_d)$$

with the intent that applying (θ_d, ϕ_d, f_d) reduces the error in $(cx(b), cy(b), h(b))$ toward $(0.5, 0.5, 0.5)$.

Transition Schemas (Pre/Post).

`init(gimbal)`:

$$\{\, g = \bot \,\}\ \text{init}(gimbal)\ \{\, g' = gimbal \,\}$$

`step(b)`:

$$\{\, g \neq \bot \wedge \text{validBBox}(b) \,\}\ \text{step}(b)\ \{\ \begin{array}{l} (\theta, \phi) = g.\text{measure\_deg}() \ \wedge\ f = g.\text{get\_focal\_length}() \ \wedge \\ (\theta_d, \phi_d, f_d) = \text{controlLaw}(b, \theta, \phi, f) \ \wedge \\ g.\text{move\_deg}(\theta_d, \phi_d) \ \wedge\ g.\text{set\_focal\_length}(f_d) \end{array}\ \}$$

If any gimbal call fails, `GimbalError` is raised.

14.4.3 Environment Variables

This module does not have any environment variables.

14.4.4 Assumptions

This module does not have any assumptions.

14.4.5 Access Routine Semantics

init(gimbal: Gimbal Abstraction Module (M11)):

- transition: Set the `gimbal` state variable to the given gimbal.
- output: None
- exception: -

step(rocket_bbox: `BoundingBox`):

- transition:
  1. Get the current gimbal state from `gimbal.measure_deg()` and `gimbal.get_focal_length()`.

2. Calculate the desired gimbal state from `rocket_bbox` so that the rocket is centered and the height of the rocket is 50% of the frame height.
3. Set the desired gimbal state using `gimbal.move_deg()` and `gimbal.set_focal_length()`.

- output: None
- exception: GimbalError if the functions in `gimbal` return an error.

14.4.6 Local Functions

This module does not have any local functions.

15 MIS of Gimbal Abstraction Module (M11)

15.1 Module

gimbal

15.2 Uses

- Serial Abstraction Module (M12)

15.3 Syntax

15.3.1 Exported Constants

This module does not have any exported constants.

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
connect	port: str baudrate: int	-	SerialError
move_deg	tilt: f32 pan: f32	-	GimbalError
measure_deg	-	tilt: f32, pan: f32	GimbalError
control_arm_led	enabled: bool	-	GimbalError
control_status_led	enabled: bool	-	GimbalError
get_gps_data	-	(coordinates: Optional[Tuple[float, float]], timestamp: Optional[int])	GimbalError
set_focal_length	focal_length_mm: f32	-	GimbalError
get_focal_length	-	focal_length_mm: f32	GimbalError

15.4 Semantics

15.4.1 State Variables

- Opened hardware serial port to the gimbal.

15.4.2 Environment Variables

- This module interacts with an external gimbal device.

15.4.3 Assumptions

This module does not have any assumptions.

15.4.4 Access Routine Semantics

connect(port: str, baudrate: int):

- transition: Opens the hardware serial port to the gimbal with the given port and baudrate.
- output: None
- exception: serialOpenError

move_deg(tilt: f32, pan: f32):

- transition: Sends the tilt and pan angles to the gimbal through the Serial Abstraction Module (M12) (see the Gimbal Protocol doc for details).
- output: None
- exception: GimbalError

measure_deg():

- transition: Retrieves the tilt and pan angle measurements from the gimbal through the Serial Abstraction Module (M12) (see the Gimbal Protocol doc for details).
- output: (tilt: f32, pan: f32)
- exception: GimbalError

control_arm_led(enabled: bool):

- transition: Controls the arm LED on the gimbal through the Serial Abstraction Module (M12) (see the Gimbal Protocol doc for details).
- output: None
- exception: GimbalError

control_status_led(enabled: bool):

- transition: Controls the status LED on the gimbal through the Serial Abstraction Module (M12) (see the Gimbal Protocol doc for details).
- output: None
- exception: GimbalError

get_gps_data():

- transition: Retrieves GPS data from the gimbal through the Serial Abstraction Module (M12) (see the Gimbal Protocol doc for details).
- output: (coordinates: Optional[Tuple[float, float]], timestamp: Optional[int])
  - If coordinates are unknown, coordinates is None.
  - If timestamp is unknown, timestamp is None.
- exception: GimbalError

set_focal_length(focal_length_mm: f32):

- transition: Sets the camera focal length on the gimbal through the Serial Abstraction Module (M12) (see the Gimbal Protocol doc for details).
- output: None
- exception: GimbalError

get_focal_length():

- transition: Retrieves the current camera focal length from the gimbal through the Serial Abstraction Module (M12) (see the Gimbal Protocol doc for details).
- output: focal_length_mm: f32
- exception: GimbalError

15.4.5 Local Functions

16 MIS of Serial Abstraction Module (M12)

16.1 Module

serial

16.2 Uses

- pyserial library module.

16.3 Syntax

16.3.1 Exported Constants

- DEFAULT_BAUDRATE: Default baudrate used when the caller does not provide one.
- READ_TIMEOUT_MS: Maximum time to wait for incoming bytes before a read is considered failed.
- MAX_FRAME_SIZE_BYTES: Upper bound on a single read/write transaction size to prevent unbounded buffering.
- IO_RETRY_COUNT: Maximum number of retries for transient I/O failures before raising an exception.

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
open_port	port: str baudrate: int	-	SerialError
send_bytes	data: bytes	-	SerialError
receive_bytes	num_bytes: int	data: bytes	SerialError

16.4 Semantics

16.4.1 State Variables

- port_handle: PortHandle | null (null iff no port is open)
- port_name: str | null
- baudrate: int | null

16.4.2 Environment Variables

- Hardware serial port.

16.4.3 Assumptions

- The given `port` refers to a valid OS serial device path/name and the device is physically connected.
- The caller invokes `open_port` successfully before calling `send_bytes` or `receive_bytes`.

16.4.4 Access Routine Semantics

open_port(port: str, baudrate: int):

- transition: Opens the hardware serial port with the given port and baudrate.
- output: None
- exception: serialOpenError

send_bytes(data: bytes):

- transition: Sends the given bytes to the hardware serial port.
- output: None
- exception: serialWriteError

receive_bytes(num_bytes: int):

- transition: Receives the given number of bytes from the hardware serial port.
- output: data: bytes
- exception: serialReadError

16.4.5 Local Functions

This module does not have any local functions.

17 MIS of System Status Module (M13)

17.1 Module

jetson-stats

17.2 Uses

- jetson-stats library module.

17.3 Syntax

17.3.1 Exported Constants

This module does not have any exported constants.

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_gpu_utilization	-	utilization: float (0.0 to 100.0)	statusError
get_cpu_utilization	-	utilization: float (0.0 to 100.0)	statusError
get_memory_usage_bytes	-	usage: {used: int, total: int} (in bytes)	statusError
get_core_temperature_celsius	-	temperature: float (in Celsius)	statusError

17.4 Semantics

17.4.1 State Variables

This module does not have any state variables.

17.4.2 Environment Variables

- The Jetson system.

17.4.3 Assumptions

This module does not have any assumptions.

17.4.4 Access Routine Semantics

get_gpu_utilization():

- transition: Retrieves the GPU utilization from the Jetson system.
- output: utilization: float (0.0 to 100.0)
- exception: statusError

get_cpu_utilization():

- transition: Retrieves the CPU utilization from the Jetson system.
- output: utilization: float (0.0 to 100.0)
- exception: statusError

get_memory_usage_bytes():

- transition: Retrieves the memory usage from the Jetson system.
- output: usage: {used: int, total: int} (in bytes)
- exception: statusError

get_core_temperature_celsius():

- transition: Retrieves the core temperature from the Jetson system.
- output: temperature: float (in Celsius)
- exception: statusError

17.4.5 Local Functions

This module does not have any local functions.

18 MIS of Recording Database Module (M14)

18.1 Module

recording

18.2 Uses

This module does not use any other modules.

18.3 Syntax

18.3.1 Exported Constants

This module does not have any exported constants.

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	base_path: str	-	-
allocate_recording	-	recording_info: `RecordingInfo`	recordingError
read_log_by_id	recording_id: str	log: list[`OSDData`] | null	-
get_recording_by_id	recording_id: str	recording_info: `RecordingInfo` | null	-
list_all_recordings	-	recordings: list[`RecordingInfo`]	-
rename_recording	recording_id: str, new_name: str	-	recordingNotFoundError
delete_recording	recording_id: str	-	recordingNotFoundError
space_usage_bytes	-	{used: int, total: int} (in bytes)	-

18.4 Semantics

18.4.1 State Variables

- `base_path`: Base path for recording video and log files.

18.4.2 Formal Specification (LO_SpecMath)

Abstract State. Let the abstract state be:

$$S \triangleq (bp, recs, logs)$$

where

$$bp \in Str \cup \{\bot\}, \quad recs \in (RecId \rightharpoonup RecMeta), \quad logs \in (RecId \rightharpoonup seq(OSDData)).$$

Here, $\rightharpoonup$ is a partial function (finite map).
RecMeta structure.

$$RecMeta \triangleq (name, videoPath, logPath)$$

with

$$name \in Str, \quad videoPath \in Str, \quad logPath \in Str.$$

State Invariants.

$$I_1 : \ bp = \bot \Rightarrow dom(recs) = \emptyset \wedge dom(logs) = \emptyset$$

$$I_2 : \ dom(logs) \subseteq dom(recs)$$

$$I_3 : \ \forall id \in dom(recs) : \ recs(id).videoPath = bp +\!\!+ \text{“}/\text{”} +\!\!+ id +\!\!+ \text{“}/video.avi\text{”}$$

$$I_4 : \ \forall id \in dom(recs) : \ recs(id).logPath = bp +\!\!+ \text{“}/\text{”} +\!\!+ id +\!\!+ \text{“}/log.txt\text{”}$$

(++ denotes string concatenation.)
Auxiliary Functions.

$$freshId(recs) \triangleq \text{an } id \in RecId \text{ such that } id \notin dom(recs).$$

$$defaultName(t) \triangleq \text{“}Recording\ \text{”} +\!\!+ t$$

where t is a timestamp string.
Transition Schemas (Pre/Post).
`init(base_path)`:

$$\{\, true \,\}\ init(p)\ \{\, bp' = p \wedge recs' = \emptyset \wedge logs' = \emptyset \,\}$$

`allocate_recording()` (may raise `recordingError`):

$$\{\, bp \neq \bot \,\}\ allocate\_recording()\ \{\ \begin{array}{l} id = freshId(recs) \ \wedge \\ recs' = recs \oplus \{id \mapsto (defaultName(now),\ bp + +\text{“}/\text{”} + +id + +\text{“}/vide \\ logs' = logs \oplus \{id \mapsto \langle\rangle\} \end{array}$$

If $bp = \perp$, `recordingError` may be raised.
`read_log_by_id(id)` (observation):

$$read\_log\_by\_id(id) = \begin{cases} logs(id) & \text{if } id \in dom(logs) \\ \perp & \text{otherwise} \end{cases}$$

`get_recording_by_id(id)` (observation):

$$get\_recording\_by\_id(id) = \begin{cases} \textit{RecordingInfo from } recs(id) \text{ and } logs(id) & \text{if } id \in dom(recs) \\ \perp & \text{otherwise} \end{cases}$$

`list_all_recordings()` (observation):

$$list\_all\_recordings() = [\ get\_recording\_by\_id(id) \mid id \in dom(recs)\ ]$$

`rename_recording(id,newName)`:

$$\{\, id \in dom(recs) \,\}\ rename\_recording(id, n)\ \{\, recs'(id).name = n \wedge \forall j \neq id :\ recs'(j) = recs(j) \,\}$$

If $id \notin dom(recs)$, raise `recordingNotFoundError`.
`delete_recording(id)`:

$$\{\, id \in dom(recs) \,\}\ delete\_recording(id)\ \{\, recs' = recs \setminus \{id\} \wedge logs' = logs \setminus \{id\} \,\}$$

If $id \notin dom(recs)$, raise `recordingNotFoundError`.
`space_usage_bytes()` (observation):

$$space\_usage\_bytes() = (used, total)$$

where $used, total \in \mathbb{N}$ summarize storage usage under bp.

18.4.3 Environment Variables

This module does not have any environment variables.

18.4.4 Assumptions

This module does not have any assumptions.

18.4.5 Access Routine Semantics

init(base_path: str):

- transition: Sets the `base_path` state variable to the given base path.
- output: None
- exception: -

allocate_recording():

- transition:
  - Generate a new random ID `recording_id` for the recording.
  - Generate a name `Recording {YYYY-MM-DD HH:MM:SS}`.
  - Create a folder under `base_path` with the ID as the folder name.
  - Create a file {`base_path`}/{`recording_id`}`/meta.json` with the name in the json.
  - Return recording info:
    - id: The generated ID.
    - name: The generated name.
    - start_time: null.
    - duration_seconds: null.
    - video_path: {`base_path`}/{`recording_id`}`/video.avi`.
    - log_path: {`base_path`}/{`recording_id`}`/log.txt`.
- output: recording_info: `RecordingInfo`
- exception: recordingError

read_log_by_id(recording_id: str):

- transition:
  - Check if file at {`base_path`}/{`recording_id`}`/log.txt` exists.
  - If it does:
    - Create an empty list `logs`.
    - For each line in the file.
      - Parse the line as json, as a `OSDData` object.
      - Append the `OSDData` object to the list.
    - Return `logs`.
  - If it does not, return null.
- output: log: list[`OSDData`] | null
- exception: -

get_recording_by_id(recording_id: str):

- transition:
  - Check if folder at {`base_path`}/{`recording_id`} exists.
  - If it does, and both `video.avi`, `log.txt` and `meta.json` exist:
    - `logs` = `read_log_by_id(recording_id)`.

        - Return recording info:
            - id: The given ID.
            - name: The name in the `meta.json`.
            - start_time: Convert the first `OSDData.timestamp_ms` in `logs` to a string representation.
            - duration_seconds: The difference between the last and first `OSDData.timestamp_ms` in `logs` (in seconds).
            - video_path: {`base_path`}/{`recording_id`}/`video.avi`.
            - log_path: {`base_path`}/{`recording_id`}/`log.txt`.
        - If it does not, return null.
    - output: recording_info: `RecordingInfo` | null
    - exception: recordingNotFoundError

list_all_recordings():

- transition:
    - Create an empty list `recordings`.
    - For each folder under `base_path`:
        - `recording_info` = `get_recording_by_id(folder_name)`.
        - If `recording_info` is not null, append it to `recordings`.
    - Return `recordings`.
- output: recordings: list[`RecordingInfo`]
- exception: -

rename_recording(recording_id: str, new_name: str):

- transition:
    - Check if file at {`base_path`}/{`recording_id`}/`meta.json` exists.
    - If it does, update the `name` field to the given new name.
    - If it does not, return recordingNotFoundError.
- output: None
- exception: recordingNotFoundError

delete_recording(recording_id: str):

- transition: Delete the folder at {`base_path`}/{`recording_id`}.
- output: None

- exception: recordingNotFoundError

space_usage_bytes():

- transition:
  - return:
    - used: the total space used under `base_path`
    - total: the total space under `base_path`
- output: {used: int, total: int} (in bytes)
- exception: -

18.4.6 Local Functions

This module does not have any local functions.

19 MIS of State Management Module (M15)

19.1 Module

stateManagement

19.2 Uses

- CV Process Management Module (M7)
- Live Video Process Management Module (M8)
- Transcode Process Management Module (M9)
- Recording Database Module (M14)
- Tracking Module (M10)
- Gimbal Abstraction Module (M11)
- System Status Module (M13)

19.3 Syntax

19.3.1 Exported Constants

This module does not have any exported constants.

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
arm	-	-	-
disarm	-	-	-
status	-	`SystemStatus`	-
manual_move	direction: str	-	-
manual_move_to	tilt: f32, pan: f32	-	-
start_recording	-	-	-
stop_recording	-	-	-
download_recording	recording_id: str	output_video_stream: str	recordingNotFoundError

19.4 Semantics

19.4.1 State Variables

- cv_process_mng: CV Process Management Module (M7)
- live_video_process_mng: Live Video Process Management Module (M8)
- transcode_process_mng: Transcode Process Management Module (M9)
- recording_db: Recording Database Module (M14)
- gimbal: Gimbal Abstraction Module (M11)
- tracking: Tracking Module (M10)
- system_status: System Status Module (M13)
- is_armed: bool
- cv_data: `CVData` | null
- osd_data: `OSDData` | null
- in_progress_recording_id: str | null (if not null, a recording is in progress)

19.4.2 Formal Specification (LO_SpecMath)

Abstract State. Let the abstract state be:

$$S \triangleq (armed, cv, osd, recId)$$

where

$$armed \in \mathbb{B}, \quad cv \in CVData \cup \{\bot\}, \quad osd \in OSDData \cup \{\bot\}, \quad recId \in RecId \cup \{\bot\}.$$

State Invariants.

$$I_1 : \ (cv = \bot) \ \Leftrightarrow \ (osd = \bot)$$

$$I_2 : \ recId \neq \bot \Rightarrow armed = \texttt{true}$$

(Recording implies the system is armed.)
Derived Function (Tracking State).

$$trackingState(armed, cv) \triangleq \begin{cases} \texttt{'idle'} & \text{if } armed = \texttt{false} \\ \texttt{'armed'} & \text{if } armed = \texttt{true} \wedge (cv = \bot \vee cv.bbox = \bot) \\ \texttt{'tracking'} & \text{if } armed = \texttt{true} \wedge cv \neq \bot \wedge cv.bbox \neq \bot \end{cases}$$

Auxiliary Functions.

$$createOSD(cv) \triangleq \text{the OSDData computed from cv and current device telemetry}$$

$$freshRec(recDb) \triangleq \text{a new recording id allocated by } recording\_db.allocate\_recording()$$

Transition Schemas (Pre/Post).
arm:

$$\{\, true \,\}\ arm\ \{\, armed' = \texttt{true} \wedge cv' = cv \wedge osd' = osd \wedge recId' = recId \,\}$$

disarm:

$$\{\, recId = \bot \,\}\ disarm\ \{\, armed' = \texttt{false} \wedge cv' = cv \wedge osd' = osd \wedge recId' = \bot \,\}$$

If $recId \neq \bot$, disarm is either disallowed or must first stop recording (your implementation chooses the policy; keep interface consistent).
start_recording (may raise recordingError):

$$\{\, recId = \bot \,\}\ start\_recording\ \{\, recId' \neq \bot \wedge armed' = \texttt{true} \,\}$$

$$\{\, recId \neq \bot \,\}\ start\_recording \text{ raises } recordingError$$

stop_recording:

$$\{\, true \,\}\ stop\_recording\ \{\, recId' = \bot \,\}$$

on_cv_data(d):

$$\{\, true \,\}\ on\_cv\_data(d)\ \{\, cv' = d \wedge osd' = createOSD(d) \wedge armed' = armed \wedge recId' = recId \,\}$$

status() (observation):

$$status(S) = SystemStatus(\ldots,\ tracking\_state := trackingState(armed, cv),\ \ldots)$$

19.4.3 Environment Variables

This module does not have any environment variables.

19.4.4 Assumptions

- Dependent modules satisfy their advertised interfaces: process management modules can start/stop their processes and report failure via their exceptions/callbacks; hardware abstraction modules either perform the requested action or raise the documented exception.
- CV data arrival is asynchronous. Before the first CV update is received, status-related fields derived from `cv_data` may be unavailable (represented by null) and must be handled by callers.
- Time synchronization from GPS may not be available at startup; the system continues to operate using local time until a valid GPS timestamp is obtained.
- At most one recording session may be active at a time; this is represented by `in_progress_recording_` being non-null.
- Manual control commands and tracking commands are not issued concurrently in a way that violates the Gimbal Abstraction Module interface (i.e., gimbal commands are serialized at the abstraction boundary).

19.4.5 Access Routine Semantics

init():

- transition:
  - Start `update_system_time()` in a background thread.
  - Call `cv_process_mng.start_process(on_cv_data, on_cv_process_crash)`.
  - Call `live_video_process_mng.start_process()`.
  - Call `recording_db.init("/mnt/data/recordings")`.
  - Call `gimbal.connect("/dev/ttyTHS1", 115200)`.
  - Call `tracking.init(gimbal)`.
  - Set `is_armed` to false.
  - Set `in_progress_recording_id` to null.
- output: None
- exception: -

arm():

- transition:
  - Set `is_armed` to true.
  - Call `gimbal.control_arm_led(true)`.

- output: None
- exception: None

disarm():

- transition:
  - Set `is_armed` to false.
  - Call `gimbal.control_arm_led(false)`.
- output: None
- exception: None

status():

- transition: Return `SystemStatus` with the following fields:
  - average_fps: `cv_data.average_fps`.
  - gimbal_tilt_deg: From `gimbal.measure_deg()`.
  - gimbal_pan_deg: From `gimbal.measure_deg()`.
  - gimbal_focal_length_mm: From `gimbal.get_focal_length()`.
  - tracking_state:
    - If `is_armed` is false, return 'idle'.
    - If `is_armed` is true, and `cv_data.bbox` is not null, return 'tracking'.
    - If `is_armed` is true, and `cv_data.bbox` is null, return 'armed'.
  - preview: `cv_data.preview`.
  - osd_data: `osd_data`.
  - timestamp_ms: Get from system time.
  - longitude: Extract from `gimbal.get_gps_data()` coordinates (first element of tuple if coordinates is not None, else null).
  - latitude: Extract from `gimbal.get_gps_data()` coordinates (second element of tuple if coordinates is not None, else null).
  - gpu_utilization: From `system_status.get_gpu_utilization()`.
  - cpu_utilization: From `system_status.get_cpu_utilization()`.
  - core_temperature_celsius: From `system_status.get_core_temperature_celsius()`.
  - memory_usage_bytes: From `system_status.get_memory_usage_bytes()`.
  - disk_usage_bytes: From `recording_db.space_usage_bytes()`.
  - in_progress_recording_id: `in_progress_recording_id`.
- output: `SystemStatus`

- exception: None

manual_move(direction: str):

- transition:
  - Get current gimbal angle from `gimbal.measure_deg()`.
  - Calculate the new gimbal angle based on the given direction.
    - If direction is "up", new tilt is `current_tilt + 10`.
    - If direction is "down", new tilt is `current_tilt - 10`.
    - If direction is "left", new pan is `current_pan - 10`.
    - If direction is "right", new pan is `current_pan + 10`.
  - Call `gimbal.move_deg(new_tilt, new_pan)`.
- output: None
- exception: None

manual_move_to(tilt: float, pan: float):

- transition: Call `gimbal.move_deg(tilt, pan)`
- output: None
- exception: None

start_recording():

- transition:
  - (if state `in_progress_recording_id` is not null): Return recordingError.
  - Let recording_info = `recording_db.allocate_recording()`.
  - Set state `in_progress_recording_id` to `recording_info.id`.
  - Call `cv_process_mng.start_recording(recording_info)`.
- output: None
- exception: recordingError

stop_recording():

- transition:
  - Call `cv_process_mng.stop_recording()`.
  - Set state `in_progress_recording_id` to null.
- output: None
- exception: None

download_recording(recording_id: str):

- transition:
  - (if state `in_progress_recording_id` is not null): Return recordingInProgressError.
  - Let recording_info = `recording_db.get_recording_by_id(recording_id)`.
  - (if recording_info is null): Return recordingNotFoundError.
  - Let output_video_stream = `transcode_process_mng.start_transcode(recording_info)`.
  - Return `output_video_stream`.
- output: output_video_stream: str
- exception: recordingNotFoundError, recordingInProgressError

19.4.6 Local Functions

create_osd_data(cv_data: `CVData`):

- transition: Return the following `OSDData` object:
  - frame_number: `cv_data.frame_number`.
  - translate_x, translate_y, scale: Calculated from `cv_data.bbox` (if `cv_data.bbox` is not null, such that when applied to the frame, the rocket is centered and the height of the rocket is 75% of the frame height; if `cv_data.bbox` is null, use default values: translate_x = 0, translate_y = 0, scale = 1.0).
  - average_fps: `cv_data.average_fps`.
  - gimbal_tilt_deg: From `gimbal.measure_deg()`.
  - gimbal_pan_deg: From `gimbal.measure_deg()`.
  - gimbal_focal_length_mm: From `gimbal.get_focal_length()`.
  - device_ip_addresses: Get from system network interfaces.
  - timestamp_ms: Get from system time.
  - tracking_state:
    - If `is_armed` is false, return 'idle'.
    - If `is_armed` is true, and `cv_data.bbox` is not null, return 'tracking'.
    - If `is_armed` is true, and `cv_data.bbox` is null, return 'armed'.
  - longitude: Extract from `gimbal.get_gps_data()` coordinates (first element of tuple if coordinates is not None, else 0.0).
  - latitude: Extract from `gimbal.get_gps_data()` coordinates (second element of tuple if coordinates is not None, else 0.0).
- output: `OSDData`

- exception: None

on_cv_data(cv_data: `CVData`):

- transition:
  - Set state variable `cv_data` to the given `CVData`.
  - Set state variable `osd_data` to `create_osd_data(cv_data)`.
  - Run `cv_process_mng.set_osd_data(osd_data)`.
  - (if `is_armed` is true, and `cv_data.bbox` is not null): Call `tracking.step(cv_data.bbox)`.
- output: None
- exception: None

update_system_time():

- transition:
  - Loop every 5 seconds.
    - Call `gimbal.get_gps_data()`.
    - If the timestamp is not None, set the system time to the timestamp and break the loop.

    .
- output: None
- exception: None

on_cv_process_crash(error: str):

- transition:
  - Log the error to stderr.
  - Call `cv_process_mng.start_process(on_cv_data, on_cv_process_crash)`.
  - (if state `in_progress_recording_id` is not null):
    - Set state `in_progress_recording_id` to null.
    - Call `start_recording()`.
- output: None
- exception: None

20 MIS of API Gateway Module (M16)

20.1 Module

apiGateway

20.2 Uses

- State Management Module (M15)

20.3 Syntax

20.3.1 Exported Constants

This module does not have any exported constants.

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
start_server	host: str, port: int	-	serverStartError

20.4 Semantics

20.4.1 State Variables

- state_management: State Management Module (M15)

20.4.2 Environment Variables

This module does not have any environment variables.

20.4.3 Assumptions

This module does not have any assumptions.

20.4.4 Access Routine Semantics

(This is the entry point of the Jetson Module)
start_server(host: str, port: int):

- transition:
  - `state_management.init()`.
  - Start a HTTP server.
  - Respond to all HTTP requests forever.
- output: None
- exception: None

20.4.5 API Endpoint Semantics

POST /api/status:

- transition: Return `state_management.status()`.
- output: StatusResponse payload
- exception: None

POST /api/manual_move:

- transition:
  - Parse request body to get `direction` (must be one of: "up", "down", "left", "right").
  - Call `state_management.manual_move(direction)`.
- output: EmptyResponse payload
- exception: None

POST /api/manual_move_to:

- transition:
  - Parse request body to get `tilt` and `pan` (both numbers).
  - Call `state_management.manual_move_to(tilt, pan)`.
- output: EmptyResponse payload
- exception: None

POST /api/gimbal/home:

- transition: Call `state_management.manual_move_to(0.0, 0.0)`
- output: EmptyResponse payload
- exception: None

POST /api/arm:

- transition: Call `state_management.arm()`.
- output: EmptyResponse payload
- exception: None

POST /api/disarm:

- transition: Call `state_management.disarm()`

- output: EmptyResponse payload
- exception: None

POST /api/recordings/start:

- transition: Call `state_management.start_recording()`.
- output: RecordingStatusResponse payload
- exception: recordingError (if recording already in progress)

POST /api/recordings/stop:

- transition: Call `state_management.stop_recording()`
- output: RecordingStatusResponse payload
- exception: None

GET /api/recordings:

- transition: Return `state_management.recording_db.list_all_recordings()`.
- output: RecordingListResponse payload
- exception: None

DELETE /api/recordings/{recordingId}:

- transition: Call `state_management.recording_db.delete_recording(recordingId)`.
- output: EmptyResponse payload
- exception: recordingNotFoundError (returns 404 HTTP status)

PATCH /api/recordings/{recordingId}:

- transition:
  - Parse request body to get `new_name` (string).
  - Call `state_management.recording_db.rename_recording(recordingId, new_name)`.
- output: EmptyResponse payload
- exception: recordingNotFoundError (returns 404 HTTP status)

GET /api/recordings/{recordingId}/download:

- transition:
  - Let output_video_stream = `state_management.download_recording(recordingId)`.
  - Stream the file at `output_video_stream` as binary data with Content-Type: video/mp4.
- output: Binary video file stream
- exception: recordingNotFoundError (returns 404 HTTP status)

20.4.6 Local Functions

21 MIS of Preview Module (M18)

21.1 Module

preview

21.2 Uses

- API Gateway Module (M16)

21.3 Syntax

21.3.1 Exported Constants

This module does not have any exported constants.

21.3.2 Exported Access Programs

This module does not have any exported access programs.

21.4 Semantics

21.4.1 State Variables

- last_preview_jpeg: bytes | null (most recent preview image returned by the backend; null if no preview has been received yet)
- last_status: `SystemStatus` | null (most recent decoded status payload; null if none)
- refresh_interval_ms: int (UI polling interval for requesting status/preview)

State Invariant:

- last_status = null $\Leftrightarrow$ last_preview_jpeg = null

21.4.2 Environment Variables

- Screen (for showing the preview).
- Network connection to the API Gateway Module (M16).

21.4.3 Assumptions

- The API Gateway Module (M16) is reachable at the configured host/port.
- The API endpoint `POST /api/status` (see M16) returns a `SystemStatus` object whose `preview` field is a JPEG-encoded image (bytes) compatible with the UI runtime.
- The UI runtime can display a JPEG-encoded byte sequence on the screen and can execute periodic refresh events.

21.4.4 Access Routine Semantics

init(refresh_interval_ms: int):

- transition:
  - Set `last_preview_jpeg` to null and `last_status` to null.
  - Set `refresh_interval_ms` to the given value.
  - Begin periodic execution of `refresh()` every `refresh_interval_ms`.
- output: None
- exception: None

refresh():

- transition:
  - Call the API Gateway endpoint `POST /api/status` (see M16).
  - If the request succeeds:
    - Set `last_status` to the returned `SystemStatus`.
    - Set `last_preview_jpeg` to `last_status.preview`.
  - If the request fails (timeout/connection error):
    - Keep `last_status` and `last_preview_jpeg` unchanged.
    - Indicate in the UI that the preview may be stale.
- output: None
- exception: None

render():

- transition:
  - If `last_preview_jpeg` is not null, display the image on screen.
  - If `last_preview_jpeg` is null, display a placeholder (e.g., "Waiting for preview").
  - If `last_status` is not null, display key telemetry fields (e.g., FPS, tracking state, gimbal angles) alongside the preview.
- output: None
- exception: None

21.4.5 Local Functions

This module does not have any local functions.

22 MIS of Manual Control Module (M19)

22.1 Module

manualControl

22.2 Uses

- API Gateway Module (M16)

22.3 Syntax

22.3.1 Exported Constants

This module does not have any exported constants.

22.3.2 Exported Access Programs

This module does not have any exported access programs.

22.4 Semantics

22.4.1 State Variables

- selected_direction: 'up' | 'down' | 'left' | 'right' | null (currently selected step move direction; null if none)
- step_size_deg: float (degrees to move per step command; default 10.0)
- target_tilt_deg: float (latest user-entered target tilt angle for absolute move)
- target_pan_deg: float (latest user-entered target pan angle for absolute move)
- last_command_time_ms: int | null (timestamp of last command issued; null if none)
- last_command_status: 'ok' | 'failed' | null (result of last command; null if none)

State Invariant:

- last_command_status = null $\Leftrightarrow$ last_command_time_ms = null
- step_size_deg ¿ 0

22.4.2 Environment Variables

- Screen (for displaying manual control UI elements).
- User input device (mouse/keyboard/touch) for issuing commands.
- Network connection to the API Gateway Module (M16).

22.4.3 Assumptions

- The API Gateway Module (M16) is reachable at the configured host/port.
- The endpoint `POST /api/manual_move` (see M16) accepts a direction in {'up','down','left','right'} and causes a corresponding step movement.
- The endpoint `POST /api/manual_move_to` (see M16) accepts numeric tilt/pan angles and causes an absolute movement.
- Commands are user-driven; the UI does not automatically issue movement commands without a user action.

22.4.4 Access Routine Semantics

This module does not provide exported access programs. Its behaviour is defined by responses to user interactions, which invoke API endpoints on M16.
on_select_direction(dir: 'up' | 'down' | 'left' | 'right'):

- transition: Set `selected_direction` to `dir`.
- output: None
- exception: None

on_set_step_size(step_size_deg: float):

- transition:
  - (if `step_size_deg` ≤ 0): keep current `step_size_deg` unchanged.
  - (otherwise): set `step_size_deg` to the given value.
- output: None
- exception: None

on_step_move():

- transition:
  - (if `selected_direction` is null): set `last_command_status` to 'failed'.
  - (otherwise):

        - Call API endpoint `POST /api/manual_move` (see M16) with `direction` = `selected_direction`.
        - If request succeeds, set `last_command_status` to 'ok'; else set to 'failed'.
      - Set `last_command_time_ms` to current system time.
  - output: None
  - exception: None

on_set_target_angles(tilt_deg: float, pan_deg: float):

  - transition:
      - Set `target_tilt_deg` to `tilt_deg`.
      - Set `target_pan_deg` to `pan_deg`.
  - output: None
  - exception: None

on_move_to():

  - transition:
      - Call API endpoint `POST /api/manual_move_to` (see M16) with `tilt` = `target_tilt_deg` and `pan` = `target_pan_deg`.
      - If request succeeds, set `last_command_status` to 'ok'; else set to 'failed'.
      - Set `last_command_time_ms` to current system time.
  - output: None
  - exception: None

render():

  - transition:
      - Display step movement controls (direction selection and step move trigger).
      - Display absolute movement controls (tilt/pan input and move trigger).
      - Display feedback for `last_command_status` and `last_command_time_ms`.
  - output: None
  - exception: None

22.4.5 Local Functions

This module does not have any local functions.

23 MIS of Recording Management Module (M20)

23.1 Module

recordingManagement

23.2 Uses

- API Gateway Module (M16)
- Recording Database Module (M14)

23.3 Syntax

23.3.1 Exported Constants

This module does not have any exported constants.

23.3.2 Exported Access Programs

This module does not have any exported access programs.

23.4 Semantics

23.4.1 State Variables

- The current set of recordings available to the user (may be empty if not yet retrieved).
- The currently selected recording (may be none).
- The status of the most recent backend interaction (success/failure).

State Invariant:

- selected_recording_id $\neq$ null $\Rightarrow \exists r \in recordings : r.id = selected\_recording\_id$

23.4.2 Environment Variables

- Screen (for displaying recording list and actions).
- User input device (mouse/keyboard/touch).
- Network connection to the API Gateway Module (M16).

23.4.3 Assumptions

- The API Gateway Module (M16) is reachable at the configured host/port.
- The endpoint `GET /api/recordings` (see M16) returns a list of `RecordingInfo`.
- The endpoints `DELETE /api/recordings/{recordingId}` and `PATCH /api/recordings/{recordin` behave as specified in M16, including returning an error for unknown recording IDs.
- Recording IDs are stable for the lifetime of a recording and uniquely identify a recording.

23.4.4 Access Routine Semantics

This module does not provide exported access programs. Its behaviour is defined by responses to user interactions, which invoke API endpoints on M16.
on_refresh():

- transition:
  - Call API endpoint `GET /api/recordings` (see M16).
  - If request succeeds:
    - Set `recordings` to the returned list.
    - Set `last_refresh_time_ms` to current system time.
    - Set `last_error` to null.
    - (if `selected_recording_id` is not present in the new list): set `selected_recording_id` to null.
  - If request fails:
    - Keep `recordings` unchanged.
    - Set `last_error` to an error string suitable for display.
- output: None
- exception: None

on_select(recording_id: str):

- transition:
  - (if $\exists r \in recordings : r.id = recording\_id$): set `selected_recording_id` to `recording_id`.
  - (otherwise): set `selected_recording_id` to null and set `last_error` to an appropriate message.
- output: None
- exception: None

on_rename(selected_recording_id: str, new_name: str):

- transition:
  - Call API endpoint `PATCH /api/recordings/{recordingId}` (see M16) with `recordingId` = `selected_recording_id` and body containing `new_name`.
  - If request succeeds:
    - Set `last_error` to null.
    - Call `on_refresh()` to update `recordings`.
  - If request fails:
    - Set `last_error` to an error string suitable for display.
- output: None
- exception: None

on_delete(selected_recording_id: str):

- transition:
  - Call API endpoint `DELETE /api/recordings/{recordingId}` (see M16) with `recordingId` = `selected_recording_id`.
  - If request succeeds:
    - Set `selected_recording_id` to null.
    - Set `last_error` to null.
    - Call `on_refresh()` to update `recordings`.
  - If request fails:
    - Set `last_error` to an error string suitable for display.
- output: None
- exception: None

on_download(selected_recording_id: str):

- transition:
  - Open a download/stream from API endpoint `GET /api/recordings/{recordingId}/download` (see M16).
  - If request succeeds, stream bytes to a user-selected save location (browser download or OS file picker).
  - If request fails, set `last_error` to an error string suitable for display.
- output: None
- exception: None

render():

- transition:
  - Display `recordings` as a selectable list (name, start time, duration).
  - Display actions for the selected recording: rename, delete, download.
  - Display `last_error` (if not null) and `last_refresh_time_ms` (if not null).
- output: None
- exception: None

23.4.5 Local Functions

fetch_recordings() → list[`RecordingInfo`] | null:

- transition: Call `GET /api/recordings` via M16 and decode the response into a list of `RecordingInfo`; return null if the request fails.
- output: list[`RecordingInfo`] | null
- exception: None

update_selection(recordings: list[`RecordingInfo`], selected_id: str | null) → str | null:

- transition:
  - If `selected_id` is null, return null.
  - If $\exists r \in recordings : r.id = selected\_id$, return `selected_id`.
  - Otherwise return null.
- output: str | null
- exception: None

set_error(msg: str):

- transition: Set state variable `last_error` to `msg`.
- output: None
- exception: None

24 MIS of Configuration Module (M21)

24.1 Module

configuration

24.2 Uses

- Preview Module (M18)
- Manual Control Module (M19)
- Recording Management Module (M20)
- API Gateway Module (M16)

24.3 Syntax

24.3.1 Exported Constants

This module does not have any exported constants.

24.3.2 Exported Access Programs

This module does not have any exported access programs.

24.4 Semantics

24.4.1 State Variables

- backend_host: str (hostname or IP address of the API Gateway backend)
- backend_port: int (TCP port of the API Gateway backend)
- refresh_interval_ms: int (polling interval used by the Preview module M18)
- move_step_deg: f32 (manual control step size in degrees used by M19)
- selected_recording_id: str | null (recording currently selected in M20)

State Invariant:

- $refresh\_interval\_ms > 0$
- $move\_step\_deg > 0$
- $1 \leq backend\_port \leq 65535$

24.4.2 Formal Specification (LO_SpecMath)

Abstract State. Let the abstract state be:

$$S \triangleq (host, port, interval, step, sel)$$

where

$$host \in Str, \quad port \in \mathbb{N}, \quad interval \in \mathbb{N}, \quad step \in \mathbb{R}, \quad sel \in RecId \cup \{\bot\}.$$

State Invariants.

$$I_1 : \ interval > 0 \qquad I_2 : \ step > 0 \qquad I_3 : \ 1 \leq port \leq 65535$$

Auxiliary Predicate.

$valid(host, port, interval, step) \triangleq (host \neq \text{“”}) \wedge (1 \leq port \leq 65535) \wedge (interval \in \mathbb{N} \wedge interval > 0) \wedge (step \in$

Abstract Persistence Store. Assume an abstract key-value store:

$$store \in (Str \rightharpoonup Str)$$

that persists across restarts.
Transition Schemas (Pre/Post).
`load_settings`:

$$\{\, true \,\}\ load\_settings\ \{\, host' = store(\text{“}backend\_host\text{”}) \text{ or default } \text{“}localhost\text{”}, \ \ldots \,\}$$

(Each setting loads from `store` if present, otherwise uses the documented default.)
`save_settings`:

$\{\, true \,\}\ save\_settings\ \{\, store' = store \oplus \{\text{“}backend\_host\text{”} \mapsto host, \ \text{“}backend\_port\text{”} \mapsto str(port), \ \text{“}refresh\_i$

`apply_settings(h,p,i,s)`:

$\{\, valid(h, p, i, s) \,\}\ apply\_settings(h, p, i, s)\ \{\, host' = h \wedge port' = p \wedge interval' = i \wedge step' = s \,\}$

If $valid(h, p, i, s)$ is false, state does not change (and UI shows error).
`set_selected_recording(id)`:

$$\{\, true \,\}\ set\_selected\_recording(id)\ \{\, sel' = id \,\}$$

where $id \in RecId \cup \{\bot\}$.

24.4.3 Environment Variables

- Screen/GUI environment (to display and edit settings).
- Local persistent storage (browser local storage or a configuration file) for saving settings across restarts.

24.4.4 Assumptions

- The UI runtime provides a mechanism to persist small key-value settings locally.
- The backend host/port entered by the user correspond to a reachable API Gateway instance (M16) when the system is used.
- Numeric settings entered by the user are validated by the UI before being applied.

24.4.5 Access Routine Semantics

This module does not export access programs. Its behaviour is defined through UI events and shared state read by other UI modules.

24.4.6 Local Functions

load_settings():

- transition:
  - Read persisted settings from local storage.
  - If a setting is missing, use a reasonable default:
    - backend_host := “localhost”
    - backend_port := 5000
    - refresh_interval_ms := 200
    - move_step_deg := 10.0
  - Set state variables to the loaded values.
- output: None
- exception: None

save_settings():

- transition: Persist current state variables to local storage.
- output: None
- exception: None

validate_settings(host: str, port: int, refresh_interval_ms: int, step_deg: f32) → bool:

- transition:
  - Return true iff:
    - host $\neq$ “”
    - port $\in \mathbb{N}$ and $1 \leq port \leq 65535$
    - refresh_interval_ms $\in \mathbb{N}$ and $refresh\_interval\_ms > 0$
    - step_deg $\in \mathbb{R}$ and $step\_deg > 0$
- output: bool
- exception: None

apply_settings(host: str, port: int, refresh_interval_ms: int, step_deg: f32):

- transition:

        - If `validate_settings(...)` is false, do nothing and display an error message.
        - Otherwise:
            - Set `backend_host`, `backend_port`, `refresh_interval_ms`, `move_step_deg`.
            - Call `save_settings()`.
            - Notify dependent UI modules that configuration has changed:
                - M18 updates its polling interval to `refresh_interval_ms`.
                - M19 uses `move_step_deg` for step-based movement.
    - output: None
    - exception: None

set_selected_recording(recording_id: str | null):

- transition:
    - Set `selected_recording_id` to the given value.
    - Persist the selection by calling `save_settings()`.
- output: None
- exception: None

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

25 Appendix

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

   What went well was that the requirements gave us a clear checklist for the Module Guide and the MIS. We could map each feature (e.g., arming/disarming, manual control, tracking, recording, downloading) to a small set of responsible modules, and then confirm the mapping again in the traceability tables. This made the document feel structured instead of random, because every section was written to support a requirement.

   Another thing that went well was using the "design for change" idea. By writing down anticipated changes (e.g., vision model updates and UI updates), it became easier to decide what each module should hide. For example, the CV Process Module hides model details and video input handling, while the UI modules hide page layout and user interaction. This helped us keep the interfaces stable and avoid spreading the same decision across multiple modules.

   The prototypes also helped a lot. Because performance requirements are strict (e.g., real-time video, 1080p at 60 fps, continuous output), we could not rely only on a "paper design." Running small experiments with the NVIDIA pipeline and the Jetson hardware gave us confidence that the chosen module boundaries are realistic. These experiments also made it easier to write the "Services" of each module in simple terms, because we understood what each module can reliably provide and what information should stay hidden behind the interface.

   Finally, peer review feedback was useful for improving clarity. It pushed us to write more explicit design decisions (e.g., why we isolate worker processes and why the API Gateway is the single control surface). Overall, the deliverable went well because the requirements, traceability, and prototypes worked together: the requirements told us what to build, traceability confirmed coverage, and prototypes validated feasibility.

2. What pain points did you experience during this deliverable, and how did you resolve them?

The Video Stream Abstraction Module (provided by NVIDIA) was challenging because it is extremely complex. To make it more complicated, we also wanted to take advantage of the hardware as much as possible to achieve our performance goals (e.g., low-latency display and stable 1080p throughput). A lot of prototype experiments had to be run to make sure the module decomposition in the MIS is actually feasible and performant enough. On the bright side, the prototype code helped us understand the access routines and semantics of each module better (for example, which data needs to cross process boundaries and which details can be hidden behind the abstraction).

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

   A lot of the design decisions came from constraints imposed by the Video Stream Abstraction Module. One key design decision was to spawn separate threads/processes for video processing and communicate over IPC channels. We made this choice because the video processing logic (from NVIDIA) can crash, and we need fault isolation to keep critical parts of the system stable (e.g., gimbal control and the API control surface). Most decisions did not come directly from speaking to clients; instead, they came from engineering constraints (performance and reliability), peer feedback (review comments), and lessons learned from prototypes (e.g., which components needed isolation to avoid cascading failures).

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), if any, needed to be changed, and why?

   No other documents needed to be changed. The MIS remains consistent with the existing requirements and traceability tables (for example, each major feature has a clear module responsible for it, and the access routines map back to the required system behaviours).

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

   One limitation is our recording download pipeline. In the current implementation, when the user requests a download, the backend does not create a new output file in advance; instead, it re-renders the recording on demand (reading the raw video and log, applying the OSD transforms, and streaming the resulting MP4 to the browser in a single HTTP request). This design is simple and avoids managing persistent state or temporary files, but it introduces several drawbacks.

   First, because the output is produced as a stream, the system typically cannot provide an accurate Content-Length up front. As a result, the browser cannot show reliable download progress, and users cannot estimate whether a download will take seconds or minutes.

Second, the work is performed synchronously on the critical path of a user request. Rendering is CPU/GPU intensive and can compete with real-time tracking workloads, causing unpredictable latency and reduced responsiveness during downloads. If multiple users request downloads, the backend can become overloaded since rendering time scales with the recording duration and resolution.

Third, the current approach has limited fault tolerance and resumability. If the connection drops mid-download or the process crashes, the user must restart from the beginning, and the system cannot reuse partial work.

Since our tech stack covers both hardware and software, with unlimited resources, we would improve this by separating "render" from "download":

- (1) Asynchronous render pipeline (pre-render artifacts).
  - Software: Introduce a job queue (e.g., a lightweight worker + persistent job state) where a download request enqueues a "render MP4" job instead of rendering inline. Store the rendered MP4 as an artifact per recording (with versioning tied to OSD settings), and expose job status (queued/running/failed/done) via an API endpoint.
  - Hardware: Offload rendering to a dedicated worker (separate process and resource group), and run it when the Jetson is idle or under a defined utilization threshold (CPU/GPU/DLA). With more resources, dedicate a second compute node (another Jetson or a small edge GPU box) purely for rendering to prevent contention with real-time tracking.
- (2) Accurate progress and ETA reporting.
  - Software: Track progress as "frames processed / total frames" (or "segments completed") and compute ETA using an exponential moving average of render throughput. Stream progress to the UI via WebSocket/SSE and show clear states (rendering vs transferring).
  - Hardware: Use hardware-accelerated decode/encode (NVDEC/NVENC) when available to stabilize throughput, producing more predictable ETA. If budget allows, add faster storage (NVMe) to reduce I/O stalls that make progress estimates noisy.
- (3) File-based downloads with known size and resumability.
  - Software: Serve completed MP4 artifacts as standard file transfers with known `Content-Length`, and enable HTTP range requests for resume. Optionally split output into fixed-duration segments during rendering, then mux at the end, so partial results can be reused after failures.
  - Hardware: Increase storage capacity to keep rendered artifacts (and optional segments) without forcing aggressive cleanup. If needed, add a small NAS or external SSD for artifact storage so the Jetson's local disk is not a bottleneck.

- (4) Caching, deduplication, and concurrency controls.
  - Software: Cache artifacts keyed by (recording id, OSD configuration hash, encoder settings) so repeated requests do not re-render. Add per-user/per-system concurrency limits, priority (interactive requests first), and backpressure to protect real-time tracking. Implement cleanup policies (LRU/TTL) and metrics (queue time, render time, failures).
  - Hardware: With more resources, allocate guaranteed CPU cores/GPU time slices (or run the renderer on a separate device) so caching and job spikes never degrade tracking latency. Add more RAM to reduce swapping and improve stability under concurrent workloads.

These changes would improve user experience, scalability, and system stability while keeping real-time tracking performance predictable.

Limitation	**Impact**	**Improve**
Unknown output size	No reliable progress	Pre-render MP4 artifacts
On-demand streaming	One request does all work	Separate render vs download
Sync on critical path	Competes with tracking	Async job queue; render when idle
Load scales with length	Long videos overload CPU/GPU	Concurrency limits; caching
No resumability	Restart after drop/crash	Stored files; HTTP range/resume
Weak fault tolerance	Partial work is lost	Retries + checkpoints (by segments)

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)

   - Single-process architecture (all video, CV, API, and gimbal control in one process).
     - Benefits: simplest implementation and deployment; no IPC overhead; shared in-memory data is easy (no serialization); potentially slightly lower latency for passing frames and detections between components.
     - Tradeoffs: poor fault isolation (a crash in the NVIDIA/video pipeline can take down gimbal control and the API); harder to keep real-time control responsive under heavy video load; debugging is harder because failures cascade through the same process; reliability is constrained by the least stable subsystem.
     - Why we did not choose it: our prototypes showed the video stack is the most crash-prone/complex component, and we cannot risk losing control or the

UI when that happens. We therefore separated critical control logic from video/CV into separate processes to isolate failures and keep the system recoverable.

- Alternative process boundaries (CV + display in one worker process, or display driven directly by the CV process rather than a dedicated Live Video process).
  - Benefits: fewer processes/threads to manage; fewer IPC channels; simpler supervision logic; slightly less overhead.
  - Tradeoffs: reduced resilience (a CV crash would also stop HDMI output); tighter coupling between detection and rendering; harder to maintain continuous video output during CV restarts; less flexibility to change display behaviour without touching CV.
  - Why we did not choose it: we wanted continuous video output even if detection fails, and we wanted to be able to restart CV independently. Splitting CV and display keeps the operator feedback loop alive during CV recovery.
- Full database for recordings (e.g., SQLite/PostgreSQL) instead of a file-system-backed recording store.
  - Benefits: stronger querying and indexing (search by time, duration, metadata); transactional updates; easier concurrent access; improved integrity checks and schema evolution.
  - Tradeoffs: higher CPU/RAM/storage overhead on the Jetson; additional deployment and maintenance complexity; more failure modes (DB corruption/migrations); operational burden that is disproportionate to our simple access patterns.
  - Why we did not choose it: our workload is lightweight (list, rename, delete, read logs by id), and the file-based layout already provides a natural index (one directory per recording). The custom store meets requirements with lower resource use and less operational risk on the target hardware.
- Pre-rendered downloads (generate MP4 artifacts at record-time or as background jobs) instead of on-demand rendering and streaming.
  - Benefits: known file sizes for accurate download progress; faster downloads (simple file transfer); supports resume/range requests; less compute on the critical path of a user request.
  - Tradeoffs: higher storage consumption (extra rendered copies); longer time to finalize a recording; background job scheduling and cleanup complexity; must handle partial renders and failures.
  - Why we did not choose it: we prioritized a simpler end-to-end pipeline and minimized persistent artifacts during early iterations. The on-demand approach reduces storage and bookkeeping, and we documented it as a known limitation with a clear upgrade path if resources allow.

Option	**Benefit**	**Tradeoff**
One process	Simple, low overhead	Crash can take down all
CV + display together	Fewer processes	CV crash stops video
Full database	Better queries	Too heavy to run/maintain
Pre-render downloads	Fast + resumable	More storage + complexity