Gimbal Protocol Specification for RoCam

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

January 12, 2026

1 Revision History

Date	Version	Notes
Jan. 12, 2026	Rev 0	Initial Draft

2 Symbols, Abbreviations and Acronyms

Term	Definition
Host	The controlling computer that sends commands to the gimbal
Gimbal	The motorized camera mount that receives commands and executes them
CRC	Cyclic Redundancy Check—a method for detecting errors in transmitted data
LED	Light Emitting Diode—a small indicator light
LE	Little-Endian—a byte ordering where the least significant byte comes first
IEEE-754	International standard for floating-point arithmetic
UART	Universal Asynchronous Receiver-Transmitter—a serial communication interface

Contents

3 Introduction

This document describes the serial communication protocol used between the host computer and the gimbal device in RoCam. The gimbal is a motorized mount that controls the orientation of a camera, allowing it to tilt (move up and down) and pan (move left and right).

The protocol is designed to be simple, reliable, and easy to implement. It uses a serial connection (UART) to exchange small packets of data between the host and the gimbal. Each packet includes error-checking information to ensure data integrity.

This specification is intended for developers who need to implement or understand the communication between the host software and the gimbal firmware.

4 Physical Layer

The protocol operates over a standard serial (UART) connection with the following parameters:

Parameter	**Value**
Baud Rate	115200 bits per second
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None

Table 1: Serial Port Configuration

The host computer connects to the gimbal using a serial port, typically named `/dev/ttyUSB0`, `/dev/ttyACM0`, or `/dev/ttyTHS1` on Linux systems.

5 Communication Model

5.1 Request-Response Structure

This protocol uses a strict **request-response** communication model. This means:

1. **All communication is initiated by the host.** The gimbal never sends data on its own—it only responds when asked.
2. **Every request must receive a response.** When the host sends a command, it must wait for the gimbal to reply before sending another command.
3. **One request, one response.** Each request from the host results in exactly one response from the gimbal.

This model simplifies the protocol because neither side needs to handle unexpected incoming data. The host always knows when to expect a response, and the gimbal always knows that incoming data is a new command.

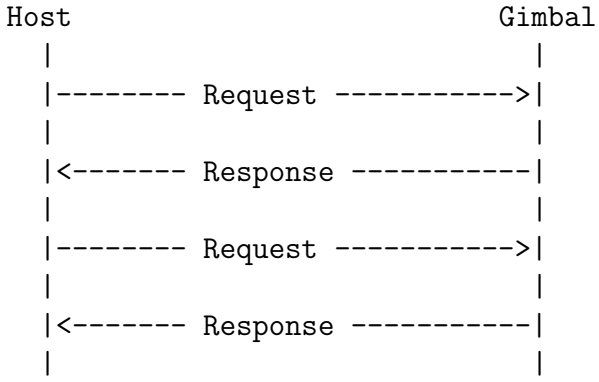

Figure 1: Request-Response Communication Flow

5.2 Timing

The host should wait for a response within a reasonable timeout period (typically 500 milliseconds). If no response is received within this time, the host should assume the command failed and may retry.

6 Packet Structure

All data exchanged between the host and gimbal is organized into **packets**. A packet is a sequence of bytes with a defined structure. Both request packets (from host to gimbal) and response packets (from gimbal to host) follow specific formats.

6.1 Byte Numbering Convention

Throughout this document, bytes within a packet are numbered starting from 0. For example, in a 3-byte packet:

- Byte 0 is the first byte
- Byte 1 is the second byte
- Byte 2 is the third byte

When describing ranges, we use the notation `[start..end]`, which includes both the start and end bytes. For example, `[2..5]` means bytes 2, 3, 4, and 5 (4 bytes total).

6.2 Request Packet Format

Request packets are sent from the host to the gimbal. Every request packet has the following structure:

Byte Position	Size	Description
0	1 byte	CRC-8 checksum (covers bytes 1 through end)
1	1 byte	Command ID (identifies which command to execute)
2..N	variable	Payload (command-specific data, may be empty)

Table 2: Request Packet Structure

Key points:

- The minimum request packet size is 2 bytes (CRC + Command ID, with no payload).
- The CRC byte is calculated over all bytes *after* itself (bytes 1 through the end of the packet).
- Different commands have different payload sizes, including zero (no payload).

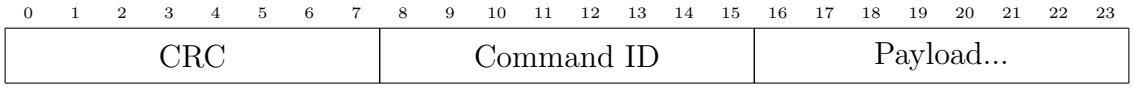

Figure 2: Request Packet Layout (sizes in bits, 8 bits = 1 byte)

6.3 Response Packet Format

Response packets are sent from the gimbal to the host. Every response packet has the following structure:

Byte Position	Size	Description
0..N-1	variable	Data (response-specific, may be empty)
N	1 byte	CRC-8 checksum (covers bytes 0 through N-1)

Table 3: Response Packet Structure

Key points:

- The minimum response packet size is 1 byte (just the CRC, with no data).
- The CRC byte is the *last* byte and is calculated over all preceding bytes.
- For commands that only need to confirm success (acknowledgment), the response contains 0 data bytes. In this case, the CRC is calculated over an empty sequence, which always equals `0x00`.

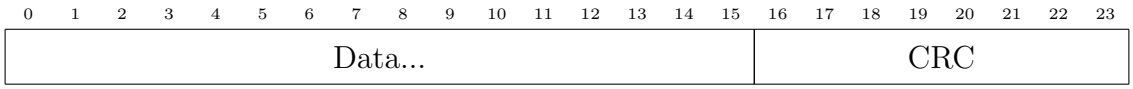

Figure 3: Response Packet Layout (sizes in bits, 8 bits = 1 byte)

6.4 Acknowledgment Responses

Many commands only need to indicate success or failure. For these commands, the gimbal responds with a single byte: the CRC calculated over zero data bytes.

Since the CRC-8/SMBUS algorithm with no input data produces `0x00`, a successful acknowledgment is simply:

Byte 0	Meaning
`0x00`	Success (CRC of empty data)
Any other value	Error

Table 4: Acknowledgment Response

7 Error Detection: CRC-8/SMBUS

To detect transmission errors (such as electrical noise corrupting data), both request and response packets include a **CRC** (Cyclic Redundancy Check) byte. This protocol uses the **CRC-8/SMBUS** algorithm.

7.1 What is a CRC?

A CRC is a small number calculated from a sequence of bytes using a mathematical formula. The sender calculates the CRC and includes it in the packet. The receiver recalculates the CRC from the received data and compares it to the received CRC. If they match, the data is likely correct. If they differ, an error occurred during transmission.

7.2 CRC-8/SMBUS Parameters

The CRC-8/SMBUS algorithm uses the following parameters:

Parameter	Value
Width	8 bits (produces a 1-byte result)
Polynomial	`0x07`
Initial Value	`0x00`
Reflect Input	No
Reflect Output	No
XOR Output	`0x00`
Check Value	`0xF4` (CRC of ASCII string "123456789")

Table 5: CRC-8/SMBUS Algorithm Parameters

7.3 CRC Calculation Example

Here is a step-by-step description of the CRC calculation:

1. Start with the CRC value set to `0x00`.
2. For each byte in the data:
   (a) XOR the current CRC value with the byte.
   (b) For each of the 8 bits in the byte:
   - If the highest bit of the CRC is 1, shift the CRC left by 1 and XOR with the polynomial (`0x07`).
   - Otherwise, just shift the CRC left by 1.

   (c) Keep only the lowest 8 bits of the result.
3. The final CRC value is the checksum.

Special case: When calculating the CRC over zero bytes (empty data), the result is `0x00` because the initial value is `0x00` and no processing occurs.

8 Data Types and Byte Order

8.1 Integer Values

Single-byte integer values (such as Command ID or LED state) are unsigned 8-bit integers with values from 0 to 255.

8.2 Floating-Point Values

Floating-point numbers are encoded using the **IEEE-754** format:

- **Single-precision** (float32): 4 bytes (32 bits), used for angles
- **Double-precision** (float64): 8 bytes (64 bits), used for GPS coordinates

- Byte order: **Little-endian** (least significant byte first)

Little-endian means that when a multi-byte number is transmitted, the byte containing the smallest part of the value is sent first. For example, the number 12.5 as float32 is stored as the bytes `0x00 0x00 0x48 0x41` in memory on a little-endian system, and transmitted in that order.

8.3 64-bit Integer Values

64-bit unsigned integers (u64) are used for timestamps:

- Size: 8 bytes (64 bits)
- Byte order: **Little-endian** (least significant byte first)
- Range: 0 to $2^{64} - 1$

9 Command Reference

This section describes each command supported by the protocol. For each command, we specify:

- The Command ID (a number identifying the command)
- The request format (what the host sends)
- The response format (what the gimbal replies)
- The purpose of the command

9.1 Command Summary

Command ID	Name	Description
`0x00`	Set ARM LED	Turn the ARM indicator LED on or off
`0x01`	Set Status LED	Turn the Status indicator LED on or off
`0x02`	Move	Move the gimbal to a specified tilt and pan angle
`0x03`	Measure	Read the current tilt and pan angles
`0x04`	Get GPS Data	Read the current GPS coordinates and timestamp
`0x05`	Set Focal Length	Set the camera focal length in millimeters
`0x06`	Get Focal Length	Read the current camera focal length in millimeters

Table 6: Command Summary

9.2 Set ARM LED (Command ID: 0x00)

9.2.1 Purpose

This command controls the ARM indicator LED on the gimbal. The ARM LED typically indicates whether the gimbal is armed (ready to move) or disarmed (safe to handle).

9.2.2 Request Format

Byte	Size	Value	Description
0	1	(calculated)	CRC-8 over bytes [1..2]
1	1	0x00	Command ID
2	1	0x00 or 0x01	LED state: 0 = off, 1 = on

Table 7: Set ARM LED Request (3 bytes total)

9.2.3 Response Format

Byte	Size	Value	Description
0	1	0x00	CRC-8 over 0 bytes (acknowledgment)

Table 8: Set ARM LED Response (1 byte total)

9.2.4 Example

To turn the ARM LED on:

- Payload: 0x01 (on)
- Bytes to CRC: 0x00 0x01 (Command ID + payload)
- CRC result: 0x07
- Complete request: 0x07 0x00 0x01
- Expected response: 0x00

9.3 Set Status LED (Command ID: 0x01)

9.3.1 Purpose

This command controls the Status indicator LED on the gimbal. The Status LED can be used to indicate various system states, such as connection status or error conditions.

9.3.2 Request Format

Byte	Size	Value	Description
0	1	(calculated)	CRC-8 over bytes [1..2]
1	1	`0x01`	Command ID
2	1	`0x00` or `0x01`	LED state: 0 = off, 1 = on

Table 9: Set Status LED Request (3 bytes total)

9.3.3 Response Format

Byte	Size	Value	Description
0	1	`0x00`	CRC-8 over 0 bytes (acknowledgment)

Table 10: Set Status LED Response (1 byte total)

9.3.4 Example

To turn the Status LED off:

- Payload: `0x00` (off)
- Bytes to CRC: `0x01 0x00` (Command ID + payload)
- CRC result: `0x07`
- Complete request: `0x07 0x01 0x00`
- Expected response: `0x00`

9.4 Move (Command ID: `0x02`)

9.4.1 Purpose

This command instructs the gimbal to move to a specified orientation. The orientation is given as two angles:

- **Tilt**: The vertical angle (up/down rotation)
- **Pan**: The horizontal angle (left/right rotation)

Both angles are specified in degrees as floating-point numbers.

9.4.2 Request Format

Byte	Size	Value	Description
0	1	(calculated)	CRC-8 over bytes [1..9]
1	1	`0x02`	Command ID
2..5	4	(float32 LE)	Tilt angle in degrees
6..9	4	(float32 LE)	Pan angle in degrees

Table 11: Move Request (10 bytes total)

9.4.3 Response Format

Byte	Size	Value	Description
0	1	`0x00`	CRC-8 over 0 bytes (acknowledgment)

Table 12: Move Response (1 byte total)

9.4.4 Example

To move to tilt = 0.0° and pan = 0.0° (center position):

- Tilt (0.0 as float32 LE): `0x00 0x00 0x00 0x00`
- Pan (0.0 as float32 LE): `0x00 0x00 0x00 0x00`
- Bytes to CRC: `0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00`
- CRC result: `0x0E`
- Complete request: `0x0E 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00`
- Expected response: `0x00`

9.5 Measure (Command ID: `0x03`)

9.5.1 Purpose

This command requests the current orientation of the gimbal. The gimbal responds with its current tilt and pan angles in degrees.

9.5.2 Request Format

Byte	Size	Value	Description
0	1	(calculated)	CRC-8 over bytes [1..1]
1	1	`0x03`	Command ID

Table 13: Measure Request (2 bytes total)

Since there is no payload, the CRC is calculated over just the Command ID byte (`0x03`), which results in `0x1B`.

9.5.3 Response Format

Byte	Size	Value	Description
0..3	4	(float32 LE)	Current tilt angle in degrees
4..7	4	(float32 LE)	Current pan angle in degrees
8	1	(calculated)	CRC-8 over bytes [0..7]

Table 14: Measure Response (9 bytes total)

9.5.4 Example

Request to read current angles:

- Bytes to CRC: `0x03`
- CRC result: `0x1B`
- Complete request: `0x1B 0x03`

Example response (tilt = 12.5°, pan = 3.25°):

- Tilt (12.5 as float32 LE): `0x00 0x00 0x48 0x41`
- Pan (3.25 as float32 LE): `0x00 0x00 0x50 0x40`
- Data bytes: `0x00 0x00 0x48 0x41 0x00 0x00 0x50 0x40`
- CRC over data: `0xD9`
- Complete response: `0x00 0x00 0x48 0x41 0x00 0x00 0x50 0x40 0xD9`

9.6 Get GPS Data (Command ID: 0x04)

9.6.1 Purpose

This command requests the current GPS data from the gimbal. The gimbal responds with its current longitude, latitude, and Unix timestamp in milliseconds. GPS typically acquires time before coordinates. If coordinates are unknown, they are returned as NaN (Not a Number). If the timestamp is unknown, it is returned as zero.

9.6.2 Request Format

Byte	Size	Value	Description
0	1	(calculated)	CRC-8 over bytes [1..1]
1	1	0x04	Command ID

Table 15: Get GPS Data Request (2 bytes total)

Since there is no payload, the CRC is calculated over just the Command ID byte (0x04), which results in 0x0C.

9.6.3 Response Format

Byte	Size	Value	Description
0..7	8	(float64 LE)	Longitude in degrees
8..15	8	(float64 LE)	Latitude in degrees
16..23	8	(u64 LE)	Unix timestamp in milliseconds
24	1	(calculated)	CRC-8 over bytes [0..23]

Table 16: Get GPS Data Response (25 bytes total)

Note: The GPS may provide data in stages:

- If coordinates are unknown, longitude and latitude are returned as NaN (IEEE-754 NaN value).
- If the timestamp is unknown, it is returned as zero.
- GPS typically acquires time before coordinates, so it is possible to have a valid timestamp with NaN coordinates.

9.6.4 Example

Request to read GPS data:

- Bytes to CRC: 0x04

- CRC result: `0x0C`
- Complete request: `0x0C 0x04`

Example response with full GPS lock (longitude = -79.9167°, latitude = 43.2567°, timestamp = 1705123456789 ms):

- Longitude (-79.9167 as float64 LE): `0xCD 0xCC 0xCC 0xCC 0xCC 0x9C 0x4F 0xC0`
- Latitude (43.2567 as float64 LE): `0x33 0x33 0x33 0x33 0x33 0x33 0x4D 0x40`
- Timestamp (1705123456789 as u64 LE): `0x15 0xDE 0x29 0x09 0x8E 0x01 0x00 0x00`
- Data bytes: `0xCD 0xCC 0xCC 0xCC 0xCC 0x9C 0x4F 0xC0 0x33 0x33 0x33 0x33 0x33 0x33 0x4D 0x40 0x15 0xDE 0x29 0x09 0x8E 0x01 0x00 0x00`
- CRC over data: (calculated)
- Complete response: (data bytes + CRC byte)

Example response with time only (timestamp = 1705123456789 ms, coordinates unknown):

- Longitude (NaN as float64 LE): `0x00 0x00 0x00 0x00 0x00 0x00 0xF8 0x7F`
- Latitude (NaN as float64 LE): `0x00 0x00 0x00 0x00 0x00 0x00 0xF8 0x7F`
- Timestamp (1705123456789 as u64 LE): `0x15 0xDE 0x29 0x09 0x8E 0x01 0x00 0x00`
- Data bytes: `0x00 0x00 0x00 0x00 0x00 0x00 0xF8 0x7F 0x00 0x00 0x00 0x00 0x00 0x00 0xF8 0x7F 0x15 0xDE 0x29 0x09 0x8E 0x01 0x00 0x00`
- CRC over data: (calculated)
- Complete response: (data bytes + CRC byte)

Example response without GPS lock (no time, no coordinates):

- Longitude (NaN as float64 LE): `0x00 0x00 0x00 0x00 0x00 0x00 0xF8 0x7F`
- Latitude (NaN as float64 LE): `0x00 0x00 0x00 0x00 0x00 0x00 0xF8 0x7F`
- Timestamp (0 as u64 LE): `0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00`
- Data bytes: `0x00 0x00 0x00 0x00 0x00 0x00 0xF8 0x7F 0x00 0x00 0x00 0x00 0x00 0x00 0xF8 0x7F 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00`
- CRC over data: (calculated)
- Complete response: (data bytes + CRC byte)

9.7 Set Focal Length (Command ID: 0x05)

9.7.1 Purpose

This command sets the camera focal length on the gimbal. The focal length is specified in millimeters as a floating-point number.

9.7.2 Request Format

Byte	Size	Value	Description
0	1	(calculated)	CRC-8 over bytes [1..5]
1	1	0x05	Command ID
2..5	4	(float32 LE)	Focal length in mm

Table 17: Set Focal Length Request (6 bytes total)

9.7.3 Response Format

Byte	Size	Value	Description
0	1	0x00	CRC-8 over 0 bytes (acknowledgment)

Table 18: Set Focal Length Response (1 byte total)

9.7.4 Example

To set focal length to 50.0 mm:

- Focal length (50.0 as float32 LE): 0x00 0x00 0x48 0x42
- Bytes to CRC: 0x05 0x00 0x00 0x48 0x42 (Command ID + payload)
- CRC result: (calculated)
- Complete request: (CRC byte + Command ID + payload)
- Expected response: 0x00

9.8 Get Focal Length (Command ID: 0x06)

9.8.1 Purpose

This command requests the current camera focal length from the gimbal. The focal length is returned in millimeters as a floating-point number.

9.8.2 Request Format

Byte	Size	Value	Description
0	1	(calculated)	CRC-8 over bytes [1..1]
1	1	`0x06`	Command ID

Table 19: Get Focal Length Request (2 bytes total)

Since there is no payload, the CRC is calculated over just the Command ID byte (`0x06`), which results in `0x0D`.

9.8.3 Response Format

Byte	Size	Value	Description
0..3	4	(float32 LE)	Current focal length in mm
4	1	(calculated)	CRC-8 over bytes [0..3]

Table 20: Get Focal Length Response (5 bytes total)

9.8.4 Example

Request to read current focal length:

- Bytes to CRC: `0x06`
- CRC result: `0x0D`
- Complete request: `0x0D 0x06`

Example response (focal length = 50.0 mm):

- Focal length (50.0 as float32 LE): `0x00 0x00 0x48 0x42`
- Data bytes: `0x00 0x00 0x48 0x42`
- CRC over data: `0x4B`
- Complete response: `0x00 0x00 0x48 0x42 0x4B`

10 Error Handling

10.1 CRC Mismatch

If the receiver calculates a CRC that does not match the received CRC byte, the packet should be considered corrupted. The host should:

1. Discard the corrupted response
2. Optionally retry the command
3. Report an error if retries are exhausted

10.2 Timeout

If the host does not receive a response within the timeout period (typically 500 ms), it should:

1. Assume the command failed
2. Optionally retry the command
3. Report an error if retries are exhausted

10.3 Invalid Acknowledgment

For commands expecting a single-byte acknowledgment (`0x00`), any other value indicates an error. The host should treat this as a command failure.