RaptorHab

High Altitude Balloon Image & Telemetry System

Comprehensive User Documentation

Version 1.5

January 2025

Code Owner: Stephen Packer

Table of Contents 2

1. Introduction and Background
2
1.1 Project Overview 2

1.2 Key Features 2

1.3 Intended Use Cases 2
2
2. System Architecture
2
2.1 High-Level Overview 2

2.2 Protocol Structure2

3. Hardware Requirements
2
3.1 Airborne Payload Components 2
2
3.2 Ground Station Components 2
2
3.3 GPIO Pin Configuration 2
2
4. Software Requirements
2
4.1 Operating System2
2
4.2 System Dependencies 2

4.3 Python Dependencies 2

5. Software Installation
2
5.1 Prepare the Raspberry Pi 2
2
5.2 Install System Dependencies 2
2
5.3 Configure SPI Pins 2
2
5.4 Deploy RaptorHab Code 2
2
5.6 Configure GPS udev Rule (Optional) 2
2
5.7 Install Systemd Services 2
2
6. Operation 2

6.1 Starting the Airborne Payload 2
2
6.2 Starting the Ground Station 2
2

6.3 Using Systemd Services 3
3
6.4 Web Interface 3
3
7. Configuration
3
7.1 Airborne Environment Variables 3

7.2 Camera Settings 3
3
7.3 Ground Station Environment Variables 3

8. Troubleshooting
3
8.1 Radio Issues 3
3
8.2 GPS Issues 3
3
8.3 Camera Issues 3
3
8.4 Image Reception Issues 3
3
9. Regulatory Compliance 3

9.1 FCC Part 15.247 Compliance 3

9.2 High-Altitude Balloon Regulations 3
3
10. License and Legal Information 3

10.1 Code Ownership 3
3
10.2 Usage Restrictions 3
3
10.3 Permitted Uses 3

10.4 Prohibited Uses
3
10.5 Disclaimer 3

11.1 Project Directory Structure 3
3
11.2 Telemetry Data Format 3
3
11.3 Fountain Code Overview 3

11.4 Contact and Support 3
3

1. Introduction and Background

1.1 Project Overview

RaptorHab is a complete high-altitude balloon (HAB) telemetry and imagery downlink system designed for amateur radio enthusiasts and near-space exploration. The system consists of two primary components: an airborne payload that captures images and telemetry from altitude, and a ground station that receives, decodes, and displays the transmitted data in real-time.

The system is built around the Raspberry Pi Zero 2W platform and utilizes LoRa radio technology operating in the 915 MHz ISM band. It employs advanced fountain codes (RaptorQ or Luby Transform) for robust error correction, enabling reliable image transmission even in challenging RF conditions with significant packet loss.

1.2 Key Features

- **915 MHz FSK Radio Link:** 96 kbps data rate (adjustable), unlicensed operation in the ISM band in the US.
- **Fountain Codes:** Robust rateless error correction using RaptorQ (preferred) or LT codes as fallback, enabling image recovery with only ~75% of transmitted packets
- **WebP Image Compression:** Efficient image transmission with typical file sizes of ~50KB per image
- **GPS Telemetry:** Real-time position tracking with airborne dynamic model support for accurate high-altitude positioning
- **Web Interface:** Real-time dashboard with map tracking, image gallery, and telemetry display
- **SondeHub Integration:** Optional telemetry upload to the amateur balloon tracking network - future progress and update needed for this.
- **Hardware SPI:** Optimized radio driver

1.3 Intended Use Cases

RaptorHab is designed for amateur high-altitude balloon flights where operators need to receive live imagery and telemetry from their payload. Typical applications include near-space photography, atmospheric research, STEM education projects, and amateur radio experimentation. The system provides position tracking essential for payload recovery after landing.

2. System Architecture

2.1 High-Level Overview

RaptorHab operates as a unidirectional telemetry system with the airborne payload continuously transmitting data to the ground station. The system uses a simple state machine architecture for reliability.

Airborne Payload State Machine

The airborne unit cycles through transmit periods: INITIALIZING → TX_ACTIVE ↔ TX_PAUSED → ERROR_STATE (with auto-reboot). During TX_ACTIVE, the payload transmits telemetry packets interleaved with image data. If a TX pause is configured, the system enters TX_PAUSED briefly before resuming transmission.

2.2 Protocol Structure

All packets share a common structure with a 4-byte sync word ("RAPT"), packet type, sequence number, flags, variable payload, and CRC-32 checksum for integrity verification.

Field	Size	Description
SYNC	4 bytes	Sync word: "RAPT" (0x52415054)
TYPE	1 byte	Packet type identifier
SEQ	2 bytes	Sequence number (0-65535)
FLAGS	1 byte	Packet flags (urgent, retransmit, etc.)
PAYLOAD	Variable	Packet payload (max 243 bytes)
CRC32	4 bytes	CRC-32 checksum for integrity

Packet Types

Type	Name	Direction	Description
0x00	TELEMETRY	Air → Ground	GPS position, altitude, status
0x01	IMAGE_META	Air → Ground	Image metadata header
0x02	IMAGE_DATA	Air → Ground	Fountain-coded image data
0x03	TEXT_MSG	Air → Ground	Text message payload
0x10	CMD_ACK	Air → Ground	Command acknowledgment

3. Hardware Requirements

3.1 Airborne Payload Components

The airborne payload requires the following hardware components:

Component	Specification
Computer	Raspberry Pi Zero 2W
Radio Module	Waveshare SX1262 868M/915M LoRa HAT
Camera	Sony IMX219 Camera (Pi Camera Module v2)
GPS Module	L76K on UART - built into the Waveshare SX1262 Hat

Power Supply	3.7V LiPo battery with 5V regulator, or 3x AA lithium
Storage	MicroSD card (16GB+ recommended)
Antenna	915 MHz quarter-wave whip or dipole

3.2 Ground Station Components

The ground station has similar requirements but can use more powerful hardware:

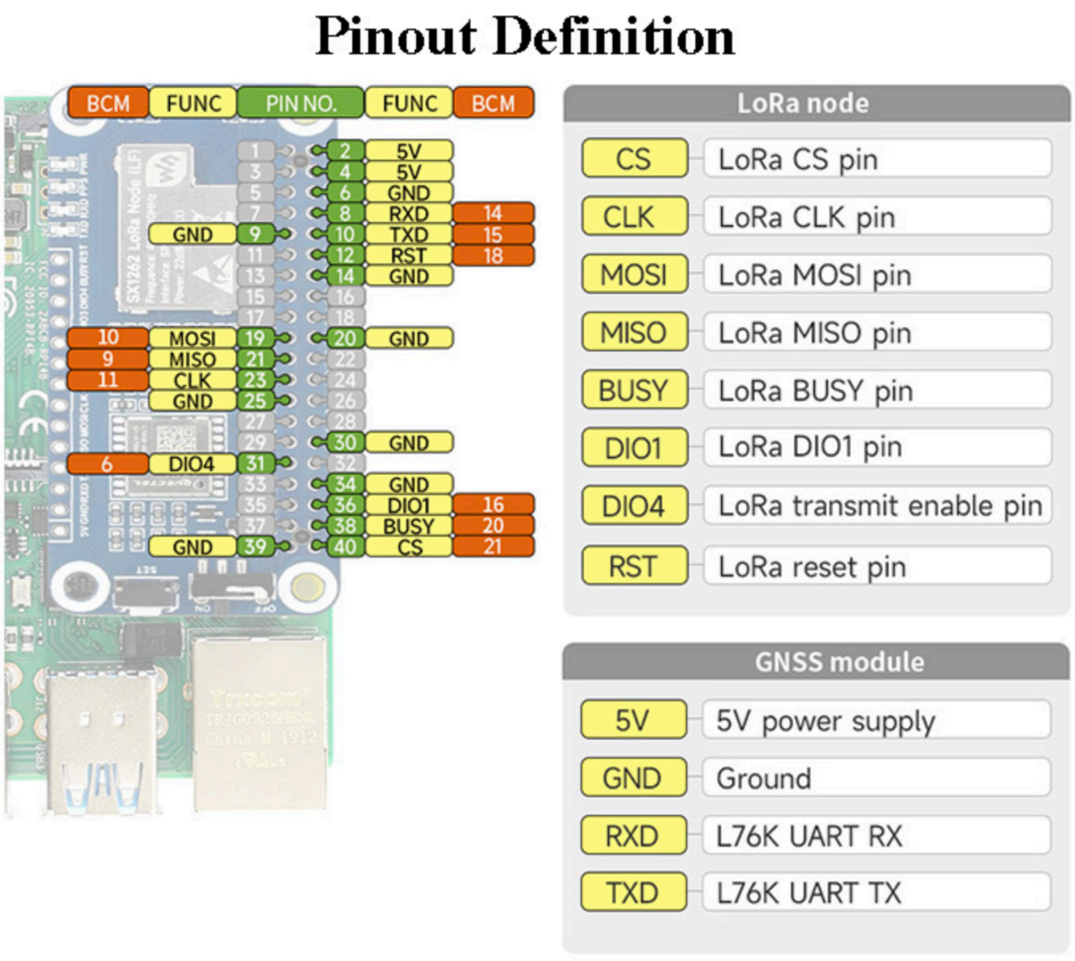
- **Computer:** Raspberry Pi Zero 2W (or any Raspberry Pi model)
- **Radio Module:** Waveshare SX1262 868M/915M LoRa HAT (identical to airborne)
- **GPS Module:** Optional L76K on Waveshare Hat
- **Antenna:** 915 MHz directional antenna (Yagi recommended) for improved range
- **Display:** Web browser on connected device for the dashboard UI

3.3 GPIO Pin Configuration

The Waveshare SX1262 HAT uses the following GPIO pins (BCM numbering):

SX1262 HAT Pin	Raspberry Pi GPIO	Function
CS (NSS)	GPIO 21	SPI Chip Select
CLK (SCK)	GPIO 11	SPI Clock
MOSI	GPIO 10	SPI Data Out
MISO	GPIO 9	SPI Data In
BUSY	GPIO 20	Radio Busy Indicator
DIO1	GPIO 16	Interrupt Line
TXEN (DIO4)	GPIO 6	TX Enable Control
RST (RESET)	GPIO 18	Radio Reset

⚠ Important: The non-standard CS pin (GPIO 21) requires manual chip select control. The hardware SPI driver handles this automatically.


Pinout Definition

BCM	FUNC	PIN NO.	PIN NO.	FUNC	BCM
		1	2	5V	
		3	4	5V	
		5	6	GND	
		7	8	RXD	14
	GND	9	10	TXD	15
		11	12	RST	18
		13	14	GND	
		15	16		
		17	18		
10	MOSI	19	20	GND	
9	MISO	21	22		
11	CLK	23	24		
	GND	25	26		
		27	28		
		29	30	GND	
6	DIO4	31	32		
		33	34	GND	
		35	36	DIO1	16
		37	38	BUSY	20
	GND	39	40	CS	21

LoRa node

CS	LoRa CS pin
CLK	LoRa CLK pin
MOSI	LoRa MOSI pin
MISO	LoRa MISO pin
BUSY	LoRa BUSY pin
DIO1	LoRa DIO1 pin
DIO4	LoRa transmit enable pin
RST	LoRa reset pin

GNSS module

5V	5V power supply
GND	Ground
RXD	L76K UART RX
TXD	L76K UART TX

4. Software Requirements

4.1 Operating System

RaptorHab requires Raspberry Pi OS (Bullseye or newer) with Python 3.9 or higher. The system has been tested on Raspberry Pi OS Lite (64-bit).

4.2 System Dependencies

The following system packages must be installed:

- python3-dev, python3-pip, python3-venv
- python3-numpy, python3-serial, python3-spidev, python3-rpi.gpio
- python3-picamera2 (for airborne camera)
- libwebp-dev (WebP image encoding)
- imagemagick, fonts-dejavu (for image overlays)

4.3 Python Dependencies

The Python dependencies are listed in requirements.txt and include:

Common (Both Systems)

- **RPi.GPIO:** GPIO pin control
- **spidev:** Hardware SPI interface
- **pyserial:** Serial port communication for GPS
- **crcmod:** CRC-32 checksum implementation
- **numpy:** Numerical operations
- **raptorq:** RaptorQ fountain codes (optional but recommended)

Airborne Only

- **picamera2:** Raspberry Pi camera interface
- **Pillow:** Image processing and WebP encoding

Ground Station Only

- **Flask:** Web framework
- **Flask-SocketIO:** Real-time WebSocket support
- **eventlet:** Async support for SocketIO
- **requests:** HTTP client for SondeHub upload

5. Software Installation

******** It is highly recommended for your sanity to use the pre-build raspberry Pi image by flashing it to your SD card using Raspberry Pi Imager, there are many pit-falls with the installation of this code from the ground-up, it can be done, but you have been warned ********

Download the pre-build image here: https://mega.nz/file/2UMISB7D#ha0phYPNnOOB5M-HUbzGgcdGGwu9TnQAL7PhNJQBP-w

Using the prebuilt Pi image:
SSH Login User name “raptorqhab” password “raptorqhab”
WiFi AP Connection SSID “RaptorQHAB” password “RaptorQHAB”

Do not power your Pi with the Waveshare Hat installed without the antenna.

The Airborne unit requires the camera to be installed properly.

5.1 Prepare the Raspberry Pi

Start with a fresh installation of Raspberry Pi OS, install Raspberry Pi OS Lite x64 onto your SD card using Raspberry Pi Imager. Customize the Pi install to connect to your network AP and enable SSH (password login).

In the Pi's SSH session - Update the system packages:

```
sudo apt update && sudo apt upgrade -y
```

Create a temporary upload directory on the Pi:

```
cd /; sudo mkdir upload; sudo chmod 777 upload
```

In another terminal SCP the RaptorQHAB.zip to the Pi:

```
scp /local_folder_where_zip_is/RaptorQHAB.zip username@pi_ip_address:/
upload
```

Back in the SSH window for the Pi:

```
sudo mv /upload/RaptorQHAB.zip /; sudo unzip RaptorQHAB.zip; sudo chmod -R
777 RaptorQHAB; sudo rm RaptorQHAB.zip; cd RaptorQHAB
```

5.2 Install System Dependencies

Install all required system packages:

```
sudo apt install -y python3-dev python3-pip python3-venv
sudo apt install -y python3-numpy python3-serial python3-spidev python3-
rpi.gpio
sudo apt install -y python3-picamera2 libwebp-dev imagemagick fonts-dejavu
```

5.3 Configure SPI Pins and Camera

For hardware SPI support (required), add these lines to /boot/firmware/config.txt:

In the Pi SSH session:

```
sudo nano /boot/firmware/config.txt
```

Verify that "camera_auto_detect=1" is set to 1 and not a 0

Add the following to the Botton of the /boot/firmware/config.txt file

```
gpio=9=a0
gpio=10=a0
gpio=11=a0
dtoverlay=imx219
```

5.4 SPI and Serial enable

In the Raspberry Pi Ssh terminal:

```
sudo raspi-config
```

Go to Interface Options -> Serial Port -> DISABLE login shell serial, ENABLE serial port hardware
Go to Interface Options -> SPI -> ENABLE

Reboot the Pi:

```
sudo reboot now
```

5.5 Install Python Dependencies

Reconnect to the Pi through SSH:

```
cd /RaptorQHAB
pip3 install -r requirements.txt --break-system-packages
```

5.6 Installing RaptorQ

Installing the raptorq package on a Pi Zero 2W is painful slow and may not work at all. I recommend using the pre-build Pi image. If your Pi system is using python 3.13 you can use the prebuilt raptorq wheel in the raptorq folder of this project. You can also cross compile raptorq on a M series Mac processor in docker or on another faster Pi such as the Pi 5.

5.6.1 Installing RaptorQ - directly on the Pi Zero 2W

In the Pi SSH session - this will take a LONG time if it works at all:

```
sudo pip3 install raptorq —break-system-packages
```

5.6.1 Installing RaptorQ - directly on another Pi (Pi 4 or 5)

Install the SD card from this process into your Pi 5/4 and boot, connect to that Pi through SSH and run:

```
sudo pip3 install raptorq —break-system-packages
```

5.6.1 Installing RaptorQ - Building in Docker on a M processor Mac

This is possible, but you are on your own here

5.7 Install Systemd Services

For Airborne Payload:

```
sudo cp systemd/raptorhab-airborne.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable raptorhab-airborne
```

For Ground Station:

```
sudo cp systemd/raptorhab-ground.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable raptorhab-ground
```

6. Operation

6.1 Starting the Airborne Payload

To start the airborne payload manually:

```
cd /RaptorQHAB; python3 -m airborne.main
```

For debug/simulation mode (no hardware required):

```
cd /RaptorQHAB; python3 -m airborne.main —debug
```

With custom settings:

```
cd /RaptorQHAB; python3 -m airborne.main --callsign MYBALLOON --frequency
915.5 --power 20
```

6.2 Starting the Ground Station

To start the ground station manually:

```
cd /RaptorQHAB; python3 -m ground.main
```

For simulation mode (generates fake telemetry for testing):

```
cd /RaptorQHAB; python3 -m ground.main —simulate
```

With custom port and data path:

```
cd /RaptorQHAB; python3 -m ground.main --web-port 8080 --data-path /mnt/data
```

6.3 Using Systemd Services

For automatic startup on boot, use the systemd services:

```
# Start the service
sudo systemctl start raptorhab-airborne
sudo systemctl start raptorhab-ground

# Check status
sudo systemctl status raptorhab-airborne
sudo systemctl status raptorhab-ground

# View logs
journalctl -u raptorhab-airborne -f
journalctl -u raptorhab-ground -f
```

6.4 Web Interface

Access the ground station web interface at http://localhost:5000 (or the Pi's IP address on port 5000). The interface provides:

- **Dashboard:** Real-time telemetry display including position, altitude, speed, battery, and temperatures
- **Map View:** Full-screen flight track visualization with multiple map layers
- **Image Gallery:** Received images with filtering, full-size viewer, and download capability
- **Signal Quality:** RSSI indicators and packet statistics

7. Configuration

7.1 Airborne Environment Variables

Configuration can be set via environment variables:

Variable	Description
`RAPTORHAB_CALLSIGN`	Payload callsign (default: RPHAB1)
`RAPTORHAB_FREQUENCY`	RF frequency in MHz (default: 915.0)
`RAPTORHAB_TX_POWER`	TX power in dBm (0-22, default: 22)
`RAPTORHAB_GPS_DEVICE`	GPS serial device path
`RAPTORHAB_DEBUG`	Enable debug mode (1/true/yes)
`RAPTORHAB_CAPTURE_INTERVAL`	Seconds between auto captures (default: 150)
`RAPTORHAB_WEBP_QUALITY`	WebP image quality (0-100, default: 75)

7.2 Camera Settings

Camera parameters can be adjusted via environment variables (scale 0-200 where 100 is neutral):

- `RAPTORHAB_CAMERA_BRIGHTNESS:` Image brightness
- `RAPTORHAB_CAMERA_CONTRAST:` Image contrast

- `RAPTORHAB_CAMERA_SATURATION:` Color saturation
- `RAPTORHAB_CAMERA_SHARPNESS:` Image sharpness
- `RAPTORHAB_CAMERA_RED_GAIN:` Red channel gain (lower to fix red tint)
- `RAPTORHAB_CAMERA_BLUE_GAIN:` Blue channel gain

7.3 Ground Station Environment Variables

- `RAPTORHAB_GND_CALLSIGN:` Station callsign
- `RAPTORHAB_GND_FREQUENCY:` RF frequency in MHz
- `RAPTORHAB_GND_DATA_PATH:` Data storage path
- `RAPTORHAB_GND_IMAGE_PATH:` Image storage path
- `RAPTORHAB_GND_WEB_PORT:` Web interface port (default: 5000)
- `RAPTORHAB_GND_DEBUG:` Enable debug mode
- `RAPTORHAB_GND_SIMULATE:` Enable simulation mode

8. Troubleshooting

8.1 Radio Issues

Hardware SPI Failed, Falling Back to Bit-Bang

This message indicates the SPI pins are not configured correctly. Verify configuration:

```
pinctrl get 9,10,11
```

Should show: a0 for all three pins
If not showing a0, ensure /boot/firmware/config.txt has the gpio= lines and reboot (see 5.3 above).

spidev Not Available

Install the spidev package:

```
pip install spidev
```

Radio Not Responding

1. Check that the LoRa HAT is properly seated on the GPIO header
2. Verify the BUSY pin goes LOW after reset
3. Try power cycling the Raspberry Pi
4. Check that no other software is using the SPI bus

8.2 GPS Issues

GPS Not Acquiring Fix

- Ensure the GPS has a clear view of the sky
- Wait up to 5 minutes for cold start acquisition
- Check the GPS antenna connection
- Verify the serial port is correct (/dev/serial0 or /dev/ttyAMA0)

GPS Serial Port Errors

If using the UART GPS, ensure the serial console is disabled:

```
sudo raspi-config
# Interface Options -> Serial Port
# Would you like a login shell to be accessible over serial? -> No
# Would you like the serial port hardware to be enabled? -> Yes
```

8.3 Camera Issues

Camera Not Detected

- Check the ribbon cable connection to both the Pi and camera
- Ensure the camera is enabled - see 5.3 above

Images Have Red/Pink Tint

Adjust the color gain settings:

- Set RAPTORHAB_CAMERA_RED_GAIN to 80 (reduce red)
- Set RAPTORHAB_CAMERA_BLUE_GAIN to 120 (increase blue)

8.4 Image Reception Issues

Images Not Completing

- Ensure both systems are on the same frequency
- Check signal strength (RSSI) - should be better than -110 dBm
- Use a directional antenna on the ground station
- Reduce image quality to decrease file size

9. Regulatory Compliance

9.1 FCC Part 15.247 Compliance

RaptorHab is designed for operation under FCC Part 15.247 rules for unlicensed digital modulation in the 902-928 MHz ISM band:

Parameter	Value
Frequency Band	902-928 MHz ISM
Modulation	2-GFSK, BT=0.5
6dB Bandwidth	>500 kHz (±125 kHz deviation at 200 kbps)
Max TX Power	30 dBm (1 Watt) conducted
Default TX Power	22 dBm (~160 mW)

⚠ Important: Users are responsible for ensuring that antenna gain + TX power does not exceed regulatory limits for their region. Always verify local regulations before operating.

9.2 High-Altitude Balloon Regulations

When launching a high-altitude balloon, ensure compliance with FAA regulations (14 CFR Part 101) or equivalent regulations in your country. This typically includes notifying the FAA/aviation authority before launch, radar reflector requirements for larger payloads, payload weight limits, and flight termination systems for certain flights.

10. License and Legal Information

10.1 Code Ownership

Code Owner: Stephen Packer
All source code, documentation, and associated materials in the RaptorHab project are the intellectual property of Stephen Packer.

10.2 Usage Restrictions

NOT FOR COMMERCIAL USE

This software and documentation are provided for personal, educational, and non-commercial use only. Commercial use, redistribution for profit, or incorporation into commercial products is strictly prohibited without explicit written permission from the code owner.

10.3 Permitted Uses

- Personal high-altitude balloon projects
- Educational and academic research
- Amateur radio experimentation
- Non-profit STEM education programs
- Modification for personal use

10.4 Prohibited Uses

- Selling the software or hardware kits based on this project
- Including in commercial products or services
- Providing as a paid service to others
- Removing or modifying copyright notices

10.5 Disclaimer

THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. THE AUTHOR SHALL NOT BE LIABLE FOR ANY DAMAGES ARISING FROM THE USE OF THIS SOFTWARE. Users assume all responsibility for ensuring safe operation and compliance with all applicable laws and regulations.

11. Appendix

11.1 Project Directory Structure

```
raptorhab/
├── airborne/          # Airborne payload code
│   ├── main.py         # Entry point, state machine
│   ├── config.py       # Configuration
│   ├── camera.py       # IMX219 capture
│   ├── gps.py          # uBlox GPS interface
│   ├── fountain.py     # RaptorQ/LT encoder
│   ├── packets.py      # Packet scheduling
│   ├── telemetry.py    # Sensor collection
│   └── utils.py        # Utilities
├── ground/            # Ground station code
│   ├── main.py         # Entry point
│   ├── config.py       # Configuration
│   ├── receiver.py     # Packet reception
│   ├── decoder.py      # Fountain decoder
│   ├── telemetry.py    # Telemetry processing
│   ├── storage.py      # Image/data storage
│   ├── sondehub.py     # SondeHub integration
│   ├── web.py          # Flask web interface
│   └── templates/      # HTML templates
├── common/            # Shared modules
│   ├── protocol.py     # Packet structures
│   ├── constants.py    # Shared constants
│   ├── crc.py          # CRC-32 implementation
│   └── radio.py        # SX1262 FSK driver
├── systemd/           # Service files
├── requirements.txt   # Python dependencies
└── README.md          # Quick start guide
```

11.2 Telemetry Data Format

The telemetry packet contains 36 bytes of data including: latitude and longitude (as signed 32-bit integers scaled by 10^7), altitude (in millimeters), speed (in cm/s), heading (in centidegrees), satellite count, fix type, GPS timestamp, battery voltage (mV), CPU temperature, radio temperature, current image ID and progress, and RSSI.

11.3 Fountain Code Overview

RaptorHab uses fountain codes (also known as rateless codes) for image transmission. Unlike traditional error correction where a fixed amount of redundancy is added, fountain codes generate an unlimited stream of encoded symbols. The receiver can recover the original data from any subset of symbols slightly larger than the original data size.

The system supports two encoder implementations: RaptorQ (preferred, requires the raptorq package) provides near-optimal performance requiring only about 2% overhead for reliable decoding; LT Codes (Luby Transform) is the fallback implementation using a Robust Soliton Distribution, requiring approximately 5-10% overhead.
For practical purposes, this means images can be recovered even with 20-25% packet loss, making the system highly resilient to the variable RF conditions encountered in balloon flights.

11.4 Contact and Support

For questions, bug reports, or feature requests regarding this project, please contact the code owner. When reporting issues, include the full error log from journalctl, your hardware configuration, and steps to reproduce the problem.