RESEARCH ARTICLE

JENER Journal of Empirical and Non-Empirical Research
Volume 2, Issue 10, Page 1748-1764 | Article No. 178
DOI: 10.4898/jenerv2i10.a178
ISSN: 7895-9216
Journal homepage: https://jenerjournal.com
Journal email: editor@jenerjournal.com

JENER Journal
JENER Journal of Empirical and Non-Empirical Research

Design and Development of a Low-Cost Ground Control Station for Real-Time Monitoring and Control of Unmanned Aerial Vehicles

Solomon Saiki[1], Chiemele Omukpai[2], Ikeru Oluchi Olukayode[3], Danjuma Amos Olobo[4], Samuel Agebe Ajeh[5] & Ekezoka Dada Evieshoyan[6]

[1-6]Bola Ahmed Tinubu Centre for Space Transport and Propulsion / NASRDA Lagos Nigeria

ARTICLE INFORMATION

Article history:
Published: October 2026

Keywords:
Ground Control Station
Unmanned Aerial Vehicle
MAVLink
Telemetry
Raspberry Pi
Link Budget
Age of Information
Low-Cost Design
Simulation

ABSTRACT

Ground Control Stations (GCSs) are the operator interface for monitoring and commanding Unmanned Aerial Vehicles (UAVs), yet commercial and laptop-centred solutions are costly and difficult to instrument for communication research in resource-constrained settings. This paper presents the design, implementation, and simulation-based evaluation of a low-cost, modular GCS for a Raspberry Pi 4 class computer, a SiK-class 100 mW telemetry radio, and a Python/MAVLink v2 software stack with a browser interface. The back end (about 200 lines of Python) was tested end to end through an emulated radio link against a MAVLink vehicle emulator, and analytical models of the link, queueing delay, information freshness, watchdog, command retransmission, and multi-UAV channel access were evaluated by simulation with parameters cross-checked against published measurements. The standard seven-message telemetry plan uses 30 % of the usable 3200 B/s link capacity (40 % with MAVLink signing), and the modelled end-to-end latency is 58 ms. On the testbed, the displayed position error was 1.5 m at 5 Hz on a clean channel but 5.5 m under bursty loss (20 % mean loss, 2 s fades), which exceeds the 2 m design target and shows that independent-loss models are optimistic. Exponential-backoff retries raise five-attempt command success under bursty loss from 93.5 % to 99.1 % in simulation, and a 3 s heartbeat watchdog flagged a 25 s outage after 2.7 s. On the same 57 messages/s stream the back end used about 13 times less CPU time per message than MAVProxy on the development host, although MAVProxy offers far more functions. The ground-segment hardware costs about USD 164. No ArduPilot software-in-the-loop, flight hardware, or radio-frequency measurements were available; the projected Raspberry Pi CPU load of 10 to 21 % of one core is at or above the 10 % target, and field validation is the necessary next stage.

1. Introduction

Unmanned Aerial Vehicles (UAVs) are now routine tools for precision agriculture, infrastructure inspection, environmental monitoring, disaster response, and wireless network provisioning [1], [2]. As airframes and autopilots have become inexpensive, the ground infrastructure that supervises them has become a limiting factor. The Ground Control Station (GCS) receives telemetry, presents vehicle state, uploads missions, issues commands, and logs data; its latency, reliability, and usability directly affect flight safety [3]. Mature GCS applications exist for the open-source ArduPilot and PX4 autopilots, notably Mission Planner, QGroundControl, and APM Planner 2 [4]–[6]. They are, however, desktop applications that assume a laptop, which is costly, fragile in bright sun and dust, dependent on mains charging, and awkward to instrument for communication research. Researchers who study medium-access control, routing, or latency in UAV networks need a ground segment whose communication layer is transparent and scriptable. Whether a small, inexpensive computer can deliver adequate situational awareness and safe control, and how much margin it has, is rarely quantified.

This paper therefore asks a measurable question: can a roughly USD 165 ground segment built from commodity parts meet explicit latency, freshness, reliability, safety, and resource targets? The contributions are:

- A modular GCS (Raspberry Pi 4 class computer, SiK-class radio, Python/MAVLink v2, browser interface) implemented and tested end to end, with a safety layer comprising a latching heartbeat watchdog, acknowledged commands with exponential-backoff retries, and geofence/battery alerts.
- Closed-form models of link budget and packet error, M/G/1 queueing delay, mean information age of the displayed state, watchdog false-alarm rate, command success probability, multi-UAV channel access, and bursty loss, each checked against Monte Carlo or discrete-event simulation.
- A reproducible testbed, comprising a radio-link emulator (capacity, transmit window, delay, independent and bursty loss, outages) and a MAVLink vehicle emulator, in which the real GCS software is exercised during a simulated survey mission, with a 60,000-frame fault-injection test showing that no corrupted frame is accepted.
- Evidence that independent-loss models are optimistic: under bursty loss the display-error target is missed and the watchdog false-alarm rate rises about fivefold, and exponential-backoff retries are needed to meet the command-reliability target.
- A cross-check of the model parameters against published field measurements and vendor and project data [7], [23], [30]–[36], a resource comparison with MAVProxy on the same traffic, quantified design guidelines, and a sourced bill of materials.

The evaluation uses emulated components and simulation, not ArduPilot software-in-the-loop, flight hardware, or radio-frequency measurements; Section 8 states which conclusions this limits, and the supplementary runner can execute the same tests against ArduPilot SITL. Section 2 reviews related work, Section 3 sets requirements, Section 4 describes the design, Section 5 develops the models, Section 6 describes the implementation and testbed, Section 7 reports results, and Sections 8 to 10 discuss, outline limitations, and conclude. The evaluation is simulation-based and analytical, with parameters and results cross-checked against published measurements; hardware-in-the-loop and field-flight validation remain future work, and Section 8 states which assumptions this affects. Section 2 reviews related work, Section 3 sets requirements, Section 4 describes the design, Section 5 develops the models, Section 6 describes the implementation and simulation method, Section 7 reports results, and Sections 8 to 10 discuss, outline limitations, and conclude.

2. Related Work

2.1 UAV communication and MAVLink

UAV–GCS links are typically point-to-point radio channels carrying a compact serialised protocol. MAVLink has become the de facto standard for small UAVs, with v2 adding packet signing, extension fields, and a larger message-ID space [7]. ArduPilot and PX4 implement it natively [8], [9], and the Pixhawk family made these autopilots widely available [10]. Surveys identify link reliability, bandwidth, and latency as central challenges for UAV networks [1], [2], [11].

2.2 Existing GCS software

Table 1 compares representative GCS options. Extensions of QGroundControl have targeted automated mission planning [12], and reviews of open-source flight controllers show that the airborne stack is mature and reproducible on low-cost hardware [13]. Comparatively little work quantifies the ground segment as a low-cost, embedded, instrumentable platform, and none of the tools in Table 1 is designed to expose the communication layer for protocol experiments.

Table 1. Comparison of representative GCS solutions with the proposed system.

GCS	Platform	License	Typical host	Extensibility for research
Mission Planner	Windows (Linux via Mono)	Open source	Laptop / PC	Moderate (C#, scripting)
QGroundControl	Windows, Linux, macOS, Android, iOS	Open source	Laptop / tablet	Moderate (C++/Qt)
APM Planner 2	Windows, Linux, macOS	Open source	Laptop / PC	Moderate
MAVProxy	Cross-platform (CLI)	Open source	Any Python host	High, no graphical HMI
UgCS	Windows, Linux, macOS	Commercial	Laptop / PC	Limited
Proposed GCS	Raspberry Pi + any web browser	Open source (planned)	Single-board computer	High (Python, REST/WebSocket)

2.3 Information freshness, queueing, and channel access

Three bodies of theory underpin the evaluation. First, the freshness of a remotely displayed state is captured by the age of information, whose mean for periodic updates over a lossy channel follows from renewal theory [14], [15]. Second, the delay of a shared serial or radio link under mixed packet sizes is described by M/G/1 queueing theory [16]. Third, multi-UAV sharing of a single radio channel is a classical random-access versus scheduled-access problem, with ALOHA and slotted ALOHA as reference schemes [17], [18]. The contribution here is to apply these tools to the specific MAVLink traffic mix and GCS safety logic and to validate them with simulation, giving designers closed-form guidance instead of trial and error.

3. System Requirements

Functional requirements are: FR1, decode MAVLink v2 telemetry in real time; FR2, display state on a map and instrument panel; FR3, send arm/disarm, mode, takeoff, land and return-to-launch commands; FR4, upload and download waypoint missions; FR5, detect link loss, low battery, GPS degradation, and geofence breach; FR6, log all telemetry for post-flight analysis. Non-functional requirements are expressed as measurable targets (Table 2) fixed before the evaluation so that the results in Section 7 can be judged against them.

Table 2. Quantitative design targets.

ID	Requirement	Target	Basis
T1	End-to-end telemetry latency (UAV sensor to browser)	≤ 100 ms	Order of the latency of a 3DR-radio control chain [31]
T2	Link utilisation with message signing enabled	$\rho \leq 0.5$	Queueing rule of thumb: delay grows steeply above about 0.5 [16]
T3	Mean displayed-position error at 5 Hz, UAV at about 8 m/s, packet loss ≤ 20 %	≤ 2 m	Design choice (comparable to consumer GPS error)
T4	Command success probability at 20 % packet loss (critical commands)	≥ 99 %	Design choice

ID	Requirement	Target	Basis
T5	Operator warning lead time before onboard link-loss failsafe	≥ 30 s	Design choice
T6	GCS software load on the SBC (one core)	≤ 10 %	Design choice (leaves headroom)
T7	Autonomy of the ground segment on a power bank	≥ 8 h	One working day
T8	Ground-segment hardware cost, excluding optional display	≤ USD 200	Design choice

4. System Architecture and Design

4.1 Overall architecture

The system has two segments (Fig. 1). The UAV segment holds the flight controller (ArduPilot or PX4), its sensors, and an airborne telemetry radio. The GCS segment comprises a ground radio connected by USB-serial to a single-board computer (SBC) that runs the software modules and serves the operator interface over its own Wi-Fi access point. The segments exchange MAVLink v2 frames over the radio link.

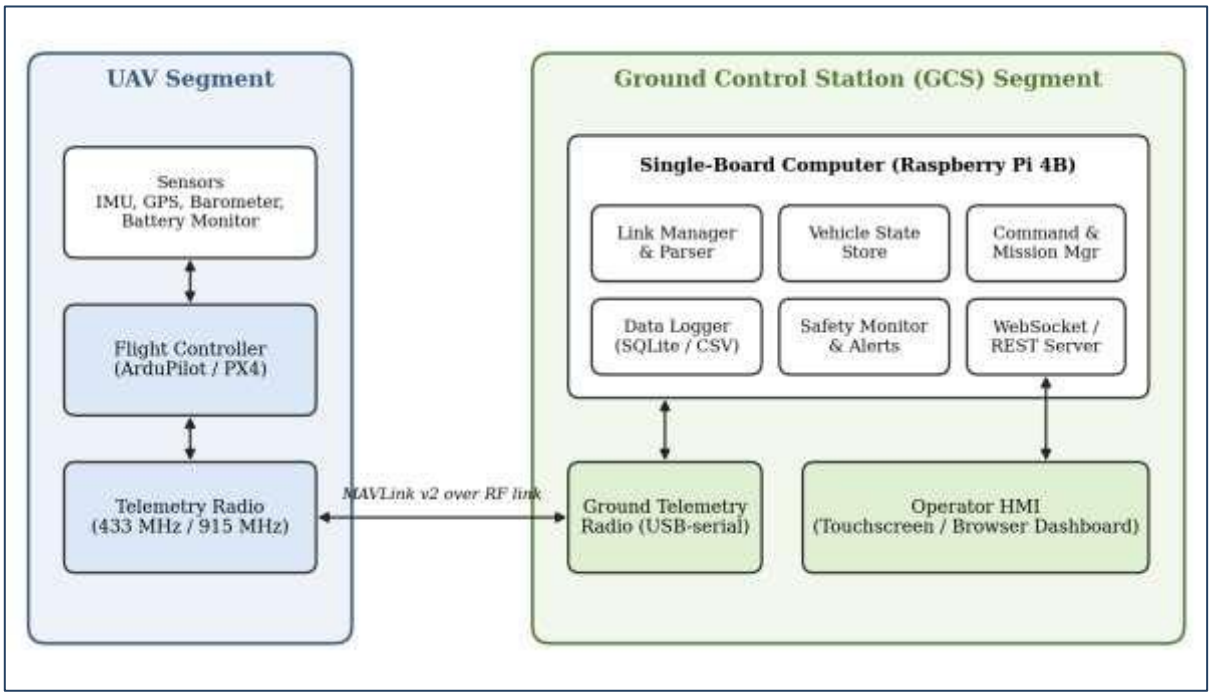

Fig. 1. Architecture of the proposed low-cost ground control station.

4.2 Hardware and cost

The SBC is a Raspberry Pi 4 Model B. Because the measured software footprint is small (Section 7.8), the 2 GB variant is specified, and the 1 GB variant (USD 35) is a viable saving; this matters because memory-driven price rises during 2026 have pushed the 4 GB board to USD 100 [24]. The ground radio is a 100 mW SiK-firmware module, interchangeable with the airborne unit, whose vendor quotes a receive sensitivity of −117 dBm and Golay-based error correction [23]. An optional 7-inch touch display lists at USD 60 [27]. Table 3 gives the bill of materials.

Table 3. Bill of materials and cost of the ground segment (USD).

Component	Specification	Price	Price basis
Single-board computer	Raspberry Pi 4 Model B, 2 GB	67.50	Official list price from 1 Oct 2026 [25]
Telemetry radio pair (air + ground)	SiK-firmware radio V3, 100 mW, 433/915 MHz	58.99	Vendor list price [23]
microSD card	32 GB, Class 10 / A1	8.00	Typical retail estimate†
Power bank	10,000 mAh, 5 V USB	15.00	Typical retail estimate†
Antenna, cables, enclosure	Spare whip/patch antenna, USB lead, case	15.00	Typical retail estimate†
Total (without display)		164.49	
Optional 7-inch touch display	Raspberry Pi Touch Display 2	60.00	Official launch price [27]
Total with display		224.49	

Component	Specification	Price	Price basis
Total with 1 GB Pi, no display		131.99	1 GB board at USD 35 [24]

† Typical retail estimates, not quoted prices; replace with local purchase prices. Prices change frequently and should be re-checked at purchase.

4.3 Communication protocol

MAVLink v2 frames carry a 10-byte header, the payload, and a 2-byte CRC; optional signing appends 13 bytes [7], [19]. The GCS uses HEARTBEAT, SYS_STATUS, GPS_RAW_INT, GLOBAL_POSITION_INT, ATTITUDE, VFR_HUD, and BATTERY_STATUS downlink, and COMMAND_LONG, SET_MODE, and the MISSION_* handshake uplink. Stream rates are requested from the autopilot so that attitude and position are sent faster than battery and GPS status (Table 5).

4.4 Software architecture

The Python back end has six loosely coupled modules (Fig. 1): a link manager and parser (pymavlink [20]), a thread-safe vehicle state store with per-field timestamps, a command and mission manager with acknowledgement and bounded retries, a SQLite/CSV data logger, a safety monitor, and a WebSocket/REST server feeding a single-page web interface with live map, artificial horizon, instrument readouts, link-status banner, command panel, and waypoint editor. Separating protocol handling from presentation allows the same back end to be scripted for experiments, for example by inserting artificial loss or delay.

4.5 Safety design

- Latching heartbeat watchdog: the link is declared lost if no HEARTBEAT arrives within T_o; the alarm clears only after three consecutive heartbeats (hysteresis), preventing the alarm flapping observed in Section 7.5.
- Two-step confirmation and a pre-arm checklist (GPS fix, battery, mode) for arming, takeoff, mode change, and return-to-launch.
- Acknowledged commands: every command is retried until COMMAND_ACK or a bounded retry count, with exponential backoff (Eq. (25)) so that retries span a fade, and failure is shown to the operator.
- Geofence and battery alerts with configurable thresholds.
- Autonomous failsafe remains onboard (return-to-launch on loss of the GCS heartbeat), so safety does not depend on the ground station alone.

4.6 Security considerations

MAVLink 2 signing authenticates frames with a truncated HMAC at a cost of 13 bytes per frame [19], which Section 7.1 shows is affordable (ρ rises from 0.30 to 0.40). The browser interface is served over the SBC's own Wi-Fi access point, which must use WPA2 or WPA3. The prototype web server has no user authentication, no transport encryption, and no protection against replayed commands beyond the two-step confirmation; these are limitations (Section 9) and must be addressed before any use beyond a closed laboratory network.

5. Analytical Models

5.1 Link budget and packet error

Free-space path loss at distance d (km) and frequency f (MHz) is [21]

$$\mathrm{FSPL}(d,f) = 20\log_{10}(d_{\mathrm{km}}) + 20\log_{10}(f_{\mathrm{MHz}}) + 32.44 \quad (1)$$

Real environments are modelled with the log-distance law with log-normal shadowing [22]:

$$PL(d) = \mathrm{FSPL}(d_0, f) + 10\,n\log_{10}\left(\frac{d}{d_0}\right) + X_\sigma \quad (2)$$

where n is the path-loss exponent, d_0 = 100 m, and X_σ is zero-mean Gaussian (σ = 5 dB). The received power and thermal noise are

$$P_r = P_t + G_t + G_r - L_m - PL(d) \quad (3)$$

$$N_0 = -174 + 10\log_{10}(B) + NF \quad (4)$$

$$\left(\frac{E_b}{N_0}\right)_{\mathrm{dB}} = P_r + X_\sigma - N_0 - L_i + 10\log_{10}\left(\frac{B}{R_b}\right) \quad (5)$$

where B is the receiver noise bandwidth, NF the noise figure, L_i the implementation loss, and R_b the air bit rate (Eq. (5) in dB). With $\gamma = E_b/N_0$ in linear units, the bit error probability of non-coherent binary FSK is [28]

$$BER = \tfrac{1}{2}\exp\left(-\tfrac{\gamma}{2}\right) \quad (6)$$

SiK firmware protects data with an extended Golay (24,12) code that corrects up to three bit errors per 24-bit codeword [23], [29]. A frame of L bytes spans $n_{cw} = \lceil 8L/12 \rceil$ codewords, so the codeword failure and frame error probabilities are

$$P_{cw} = 1 - \sum_{i=0}^{3}\binom{24}{i} BER^{\,i}(1-BER)^{24-i} \quad (7)$$

$$PER = 1 - (1 - P_{cw})^{n_{cw}}, \quad n_{cw} = \lceil 8L/12 \rceil \quad (8)$$

5.2 Channel utilisation and queueing delay

Let stream i send frames of L_i bytes at r_i Hz, L_s = 13 bytes of signature overhead per frame when signing is enabled, and C the usable payload capacity of the link (bytes/s). The utilisation is

$$\rho = \frac{1}{C}\sum_i r_i (L_i + L_s) \tag{9}$$

Frames are served in first-come order with service time $S_i = L_i/C$. With aggregate frame rate λ, the Pollaczek–Khinchine formula for an M/G/1 queue gives the mean delay [16]

$$E[T] = E[S] + \frac{\lambda E[S^2]}{2(1-\rho)} \tag{10}$$

5.3 Freshness of the displayed state

Suppose a quantity (for example position) is sent every T_u seconds and each frame is lost independently with probability p. The interval between successive received updates is $K \cdot T_u$ with K geometric, so $E[K] = 1/(1-p)$ and $E[K^2] = (1+p)/(1-p)^2$. For a renewal process the mean age of the latest received sample is $E[X^2]/(2E[X])$ [14], [15], which gives

$$\bar{\Delta} = T_u \frac{1+p}{2(1-p)} \tag{11}$$

For a vehicle with mean speed $\bar{v}$ and a fixed pipeline latency L_0 (Section 7.3), the mean displayed-position error is approximately

$$\bar{e} \approx \bar{v}\,(L_0 + \bar{\Delta}) \tag{12}$$

5.4 Watchdog and command reliability

Heartbeats are sent at $f_{hb} = 1$ Hz. A watchdog with timeout T_o raises an alarm when $m = \lfloor T_o \cdot f_{hb} \rfloor$ consecutive heartbeats are lost. For independent loss p, a run of at least m losses begins after a received heartbeat at rate $(1-p)\cdot p^m$ per heartbeat, so the false-alarm rate per hour is

$$\Lambda_{fa} = 3600 f_{hb} (1-p) p^m, \quad m = \lfloor T_o f_{hb} \rfloor \tag{13}$$

while the detection delay for a genuine link loss is approximately To. A command succeeds in one attempt if both the command and its acknowledgement survive, with probability $q = (1-p)^2$. With at most N attempts spaced by the acknowledgement timeout t_{to},

$$P_s = 1 - \left[1 - (1-p)^2\right]^N \tag{14}$$

$$E[CRTT \mid \text{success}] = t_{rt} + t_{to} \frac{\sum_{k=0}^{N-1} k(1-q)^k q}{\sum_{k=0}^{N-1} (1-q)^k q}, \quad q = (1-p)^2 \tag{15}$$

where t_{rt} is the round-trip time of a single successful exchange.

5.5 Multi-UAV channel access

With M UAVs, each sending λ frames/s of mean airtime τ, the offered load is $G = M\lambda\tau$ in units of frame times. For random access the normalised throughput is [17], [18]

$$S_{pure} = G e^{-2G}, \qquad S_{slot} = G e^{-G} \tag{16}$$

and the per-frame delivery ratio of a given UAV, with η the efficiency of a scheduled (TDMA) frame after guard times, is

$$PDR_{slot} = e^{-(M-1)\lambda\tau}, \; PDR_{pure} = e^{-2(M-1)\lambda\tau}, \; PDR_{TDMA} = \min\left(1, \frac{\eta}{M\lambda\tau}\right) \tag{17}$$

5.6 Vehicle, battery, and platform models used in simulation

The simulated UAV is a point mass whose horizontal acceleration tracks a desired velocity vd with time constant τ_v and saturation a_{max}; roll and pitch follow from the acceleration components along the right-hand (r) and forward (h) body axes:

$$a = \mathrm{sat}\left(\frac{v_d - v}{\tau_v}, a_{max}\right), \quad \phi = \mathrm{atan2}(a \cdot r, g), \quad \theta = -\mathrm{atan2}(a \cdot h, g) \tag{18}$$

Battery power is $P = P_h + c_v \cdot v^2$ + terms for climb and manoeuvre; with state of charge s, capacity Q (Ah), cell voltage 3.3 + 0.9s, four cells in series, and internal resistance R_{int},

$$V_{pack} = 4(3.3 + 0.9\,s) - I R_{int}, \qquad \frac{ds}{dt} = -\frac{I}{3600\,Q} \tag{19}$$

The CPU utilisation of the SBC is modelled from measured per-operation costs on the development host, scaled by a conservative slowdown factor κ for the Raspberry Pi 4; electrical power and autonomy follow from the idle and full-load power of the board [26]:

$$u = \kappa\left[\lambda_m (t_{dec} + t_{log}) + f_{ui}\, t_{json}\right] \tag{20}$$

$$P_{sys} = P_{idle} + (P_{max} - P_{idle})\frac{u}{4} + P_{radio} \tag{21}$$

$$t_{run} = \frac{\eta E_{bat}}{P_{sys}} \tag{22}$$

5.7 Bursty (time-correlated) loss

Real radio loss is clustered in fades. It is modelled as a two-state Gilbert–Elliott channel [37], [38] with exponentially distributed good and bad sojourns of mean Dg and Db seconds, with all frames lost in the bad state. The mean loss is

$$p = \frac{D_b}{D_g + D_b} \tag{23}$$

For heartbeats at 1 Hz, a run of m = ⌊To⌋ consecutive losses requires a fade longer than about m − 1 seconds, so the false-alarm rate per hour becomes (valid for Db of 2 s or more)

$$\Lambda_{GE} = \frac{3600}{D_g + D_b} D_b (1 - e^{-1/D_b}) e^{-(m-1)/D_b} \tag{24}$$

which exceeds the independent-loss rate of Eq. (13) when Db is of the order of To or longer. Retransmissions spaced evenly by tto are all lost when a fade outlasts the retry window, so a spacing that grows exponentially, with cap 4 s, is used instead:

$$\Delta t_k = \min(t_{to}\, 2^{k-1},\ 4\ \mathrm{s}), \quad k = 1, \ldots, N \tag{25}$$

6. Implementation and Simulation Method

6.1 Software implementation

The back end is Python 3 using pymavlink [20] for MAVLink v2 encoding and decoding, an asynchronous web framework for the WebSocket/REST endpoints, and SQLite for logging. The same code runs on a laptop and on the Raspberry Pi, which simplifies testing. Listing 1 shows the latching link monitor that implements the safety rule of Section 4.5.

Listing 1. Latching heartbeat watchdog with hysteresis (Python).

```
import time
from pymavlink import mavutil


class LinkMonitor:
    def __init__(self, port, baud=57600, timeout=3.0, reacquire=3):
        self.conn = mavutil.mavlink_connection(port, baud=baud)
        self.timeout, self.reacquire = timeout, reacquire
        self.last_hb, self.good_run, self.lost = time.time(), 0, False


    def poll(self, state):
        msg = self.conn.recv_match(blocking=False)
        while msg is not None:
            t = msg.get_type()
            if t == 'HEARTBEAT':
                self.last_hb = time.time()
                self.good_run += 1 if self.lost else 0
            if t != 'BAD_DATA':          # CRC failures are dropped
                state.update(t, msg.to_dict(), time.time())
            msg = self.conn.recv_match(blocking=False)
        if not self.lost and time.time() - self.last_hb > self.timeout:
            self.lost, self.good_run = True, 0
            state.raise_event('LINK_LOST')
        elif self.lost and self.good_run >= self.reacquire:
            self.lost = False
            state.raise_event('LINK_RESTORED')
        return not self.lost
```

Commands are sent as COMMAND_LONG and counted successful only on a matching COMMAND_ACK; otherwise they are retried up to N times at intervals of t_{to}. Missions use the standard MAVLink mission handshake with per-item acknowledgement.

6.2 Simulation framework

The study uses four complementary tools, all implemented in Python and supplied as supplementary material. (i) A codec benchmark builds genuine MAVLink v2 frames with pymavlink, records their sizes with and without signing, and times decoding of 42,000 frames. (ii) A link model implements Eqs. (1) to (8) with Monte Carlo shadowing. (iii) A discrete-event queue simulator serves frames of the measured sizes at the usable capacity C, with Poisson or periodic arrivals. (iv) A flight simulator flies the UAV model of Eq. (18) and (19) around a lawnmower survey mission (eight waypoints, 100 m east of home, 8 m/s, 30 m altitude); at 50 Hz it generates the seven MAVLink messages of Table 5, packs them with pymavlink, drops them according to the distance-dependent packet error, delays them using the latency model, and feeds the survivors to the GCS parser. Decoded fields are compared with the transmitted values, and the displayed state is compared with the true state. Watchdog, command, and multi-UAV studies use Monte Carlo simulation of the corresponding protocols. Table 4 lists the parameters.

Table 4. Simulation parameters (all assumptions are stated here).

Parameter	Value	Basis
Transmit power, antenna gains	20 dBm (100 mW), 2 dBi each end	SiK radio datasheet class [7], [23]

Parameter	Value	Basis
Misc. loss / implementation loss / noise figure	6 dB / 3 dB / 8 dB	Assumed
Noise bandwidth, air rate	100 kHz, 64 kbps	Assumed for 64 kbps GFSK
Usable payload fraction of air rate	0.40, so C = 3,200 B/s	TDM, half-duplex, Golay overhead (assumed)
Path-loss exponent n (rural / suburban / urban)	2.5 / 3.5 / 4.5 (+25 dB clutter in urban)	Urban calibrated to 500 m typical range [7]; elevated-UAV exponents of 1.2–2.2 measured in [30] are used in Section 7.10
Log-normal shadowing, reference distance	σ = 5 dB, d0 = 100 m	Assumed
UART baud rate (both ends)	57,600 baud, 8N1	Radio default [32]
TDM window access wait	mean 16.5 ms (33 ms window)	Default Max Window of SiK firmware [32]
Heartbeat rate, watchdog timeout, reacquire count	1 Hz, 3 s, 3 heartbeats	Design choice
Onboard GCS-heartbeat failsafe timeout	5 s, then return-to-launch	Typical autopilot default
Command acknowledgement timeout	0.6 s	Design choice
UAV speed limit, acceleration limit, altitude	8 m/s, 3 m/s², 30 m	Assumed
Battery	4S, 5 Ah, 3.3 + 0.9 s V per cell	Assumed
SBC slowdown factor κ (Pi 4 vs host)	10	Conservative assumption
Pi 4 idle / full-load power; radio power	2.4 W / 6.8 W; 0.6 W	Board power from [26]; radio assumed

6.3 Implementation and testbed

The GCS back end comprises 200 lines of Python (link manager, state store, command and mission manager, safety monitor, SQLite logger, and WebSocket server) and a 33-line browser dashboard with a link-status banner, artificial horizon, track map, instrument readouts, and confirmed command buttons. For testing, a 92-line radio emulator sits between the autopilot (TCP) and the GCS (UDP). It models a first-in-first-out link of 3200 B/s, a uniformly distributed transmit-window wait of up to 33 ms, UART serialisation at 57,600 baud, independent or two-state bursty loss, and scripted outages. A 146-line vehicle emulator runs the quadcopter model of Section 5.6 in real time and speaks MAVLink 2 over TCP: it honours per-message stream intervals, acknowledges commands, refuses to arm before a 20 s pre-arm delay, implements the mission upload handshake with re-requests, supports AUTO, GUIDED, RTL and LAND modes, and switches to return-to-launch if no heartbeat from the GCS system identifier arrives for 5 s. A separate reference connection logs the true position at 20 Hz over an ideal link, and a 88-line runner automates the experiments. The vehicle emulator is the author's own model and is not ArduPilot; the runner can launch ArduPilot SITL instead, but this could not be done in the available environment because compiling it was impractically slow. All timings are wall-clock on a Linux x86 development host (2.1 GHz Intel Xeon) with the autopilot, radio emulator, and GCS on the same machine.

7. Results and Discussion

7.1 Frame sizes and bandwidth budget

Table 5 gives the measured MAVLink v2 frame sizes for the telemetry plan. The plan occupies 963 B/s (25 frames/s), a utilisation of ρ = 0.30 of the 3200 B/s usable capacity. Enabling signing adds 13 bytes to each frame, raising the load to 1288 B/s (ρ = 0.40, an increase of 34 %), still within target T2 (Fig. 2). Decoding 42,000 frames took 8.4 µs per message on the development host (119 thousand messages/s), and encoding a COMMAND_LONG took 4.0 µs.

Table 5. Telemetry plan and measured MAVLink v2 frame sizes.

Message	Rate (Hz)	Frame (B)	Signed (B)	Rate (B/s)	Signed (B/s)
HEARTBEAT	1	21	34	21	34
SYS_STATUS	2	43	56	86	112
GPS_RAW_INT	2	42	55	84	110
GLOBAL_POSITION_INT	5	40	53	200	265
ATTITUDE	10	40	53	400	530
VFR_HUD	4	31	44	124	176
BATTERY_STATUS	1	48	61	48	61
Total	25			963	1288

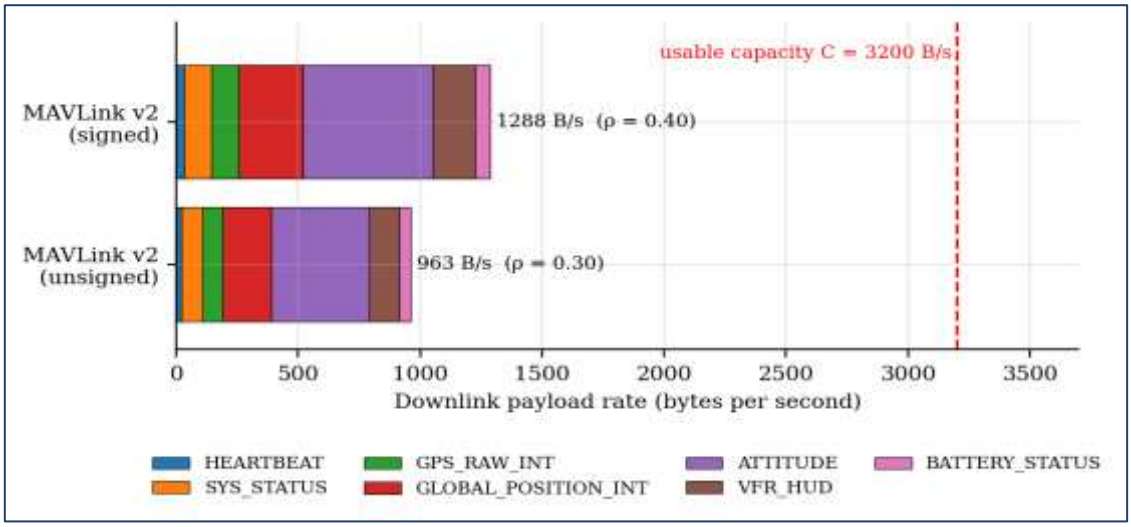

Fig. 2. Downlink bandwidth budget of the telemetry plan, with and without MAVLink v2 signing.

7.2 Link budget and range

The receiver noise floor from Eq. (4) is -116 dBm. Fig. 3 shows the mean received power and the packet loss of a 40-byte frame against distance, and Table 6 lists the range at which the loss reaches 5 % and 50 %. In the urban scenario the 433 MHz range is 425 m, consistent with the 300–500 m quoted by the radio vendor [23], which supports the calibration. For the suburban case the range is 3.2 km, and for open terrain 12.5 km, although the radio horizon and visual-line-of-sight rules, which are not modelled, limit the usable range below this. At 915 MHz the range is about 34 % shorter under the same exponent because free-space loss is 6.5 dB higher. A practical consequence is the cliff effect: in the suburban case the loss rises from 5 % to 50 % between 3.2 and 5.6 km, so the packet-loss ratio gives little advance warning. The GCS should therefore also display received signal strength and a link-margin estimate. The exponents used here describe ground-level clutter; for an elevated UAV the published measurements are more favourable (Section 7.10).

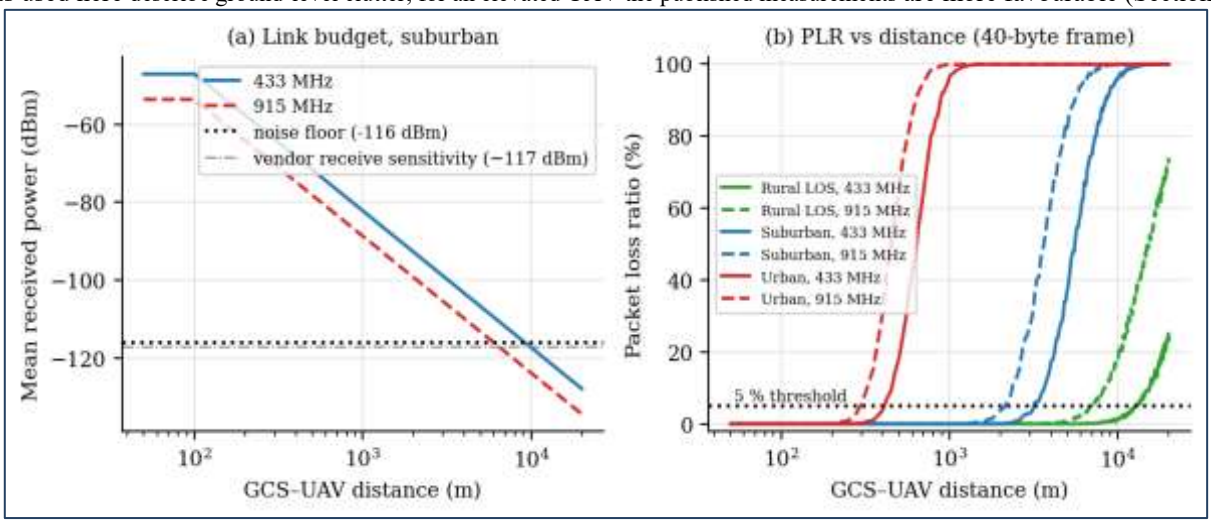

Fig. 3. (a) Mean received power versus distance (suburban). (b) Packet loss versus distance for rural, suburban, and urban scenarios at 433 and 915 MHz.

Table 6. Predicted range (m) at which packet loss reaches 5 % and 50 % (40-byte frame).

Scenario	Frequency	Range at 5 % loss (m)	Range at 50 % loss (m)
Rural LOS (n=2.5)	433 MHz	12,525	> 20,000
Rural LOS (n=2.5)	915 MHz	6,950	14,900
Suburban (n=3.5)	433 MHz	3,200	5,600
Suburban (n=3.5)	915 MHz	2,125	3,700
Urban (n=4.5, +25 dB clutter)	433 MHz	425	650
Urban (n=4.5, +25 dB clutter)	915 MHz	300	475

7.3 Latency

Fig. 4 compares the analytic M/G/1 delay of Eq. (10) with discrete-event simulation. For Poisson arrivals the two agree to a mean relative error of 0.5 % (maximum 1.3 %), validating the model. Real MAVLink streams are periodic and therefore smoother than

Poisson traffic, so their delay stays near the service time up to high load; the Poisson result is thus a conservative bound. At the operating point (ρ = 0.30) the mean queueing-plus-service delay is 13.4 ms. Table 7 sums the pipeline: the modelled end-to-end latency is 58.2 ms, meeting target T1 (≤ 100 ms). The radio (queueing plus TDM access) and the two serial hops dominate; raising the UART rate to 115,200 baud would save about 7 ms. The decoding time of the GCS is negligible.

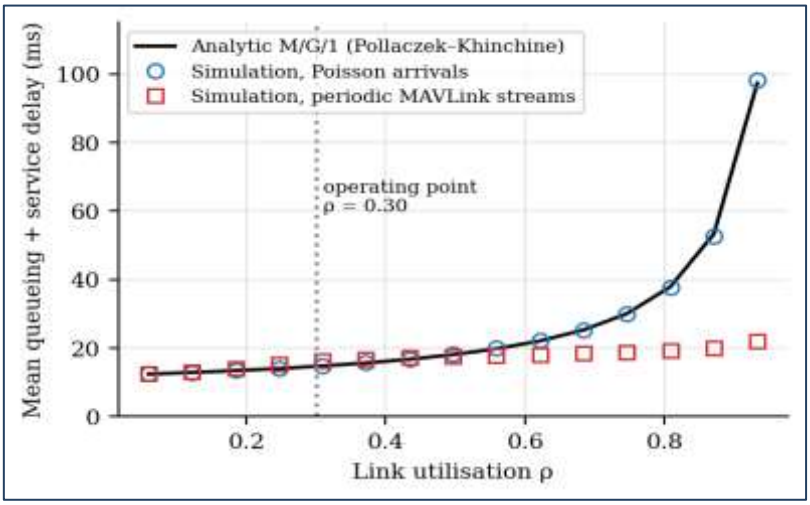

Fig. 4. Mean queueing plus service delay versus link utilisation: analytic M/G/1, and simulation with Poisson and periodic arrivals.

Table 7. End-to-end latency budget for an ATTITUDE frame at the operating point.

Component	Mean (ms)	Basis
Flight-controller UART serialisation (57,600 baud)	6.94	computed
Radio queueing + service (simulated)	13.38	simulated
TDM window access wait (33 ms window)	16.50	assumed
Ground UART/USB hand-off	7.94	computed + assumed
MAVLink decode (Pi-4 projection, ×10 host)	0.08	measured × factor
State store + WebSocket (LAN)	5.00	assumed
Display refresh (mean half-frame at 60 Hz)	8.33	computed
Total	58.2	

7.4 Flight simulation and display accuracy

The simulated survey mission lasts 425 s, covers 3070 m at a mean survey speed of 7.9 m/s, reaches 720 m from the GCS, and consumes 31 % of the battery (Figs. 5 and 6). Over the mission the UAV generated 10,710 MAVLink frames; all 10,710 were delivered (the modelled loss is negligible within 720 m) and decoded by the GCS parser with 0 field mismatches in the ATTITUDE and GLOBAL_POSITION_INT payloads, confirming correct framing and decoding. A separate fault-injection test flipped one to three random bits in each of 60,000 frames; the CRC rejected 60,000 of them and none was accepted with altered content.

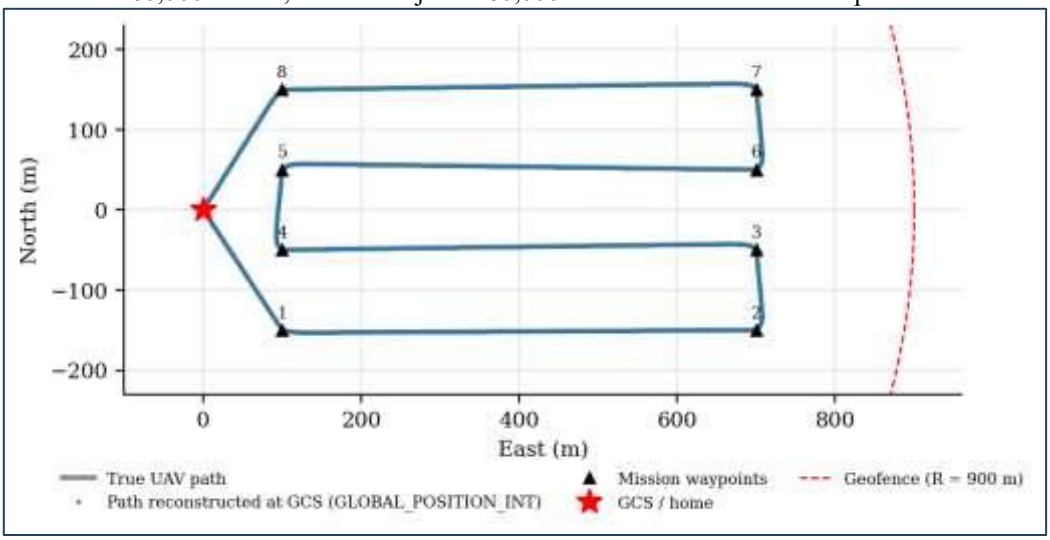

Fig. 5. Simulated survey mission: true UAV path and the path reconstructed by the GCS from received GLOBAL_POSITION_INT messages.

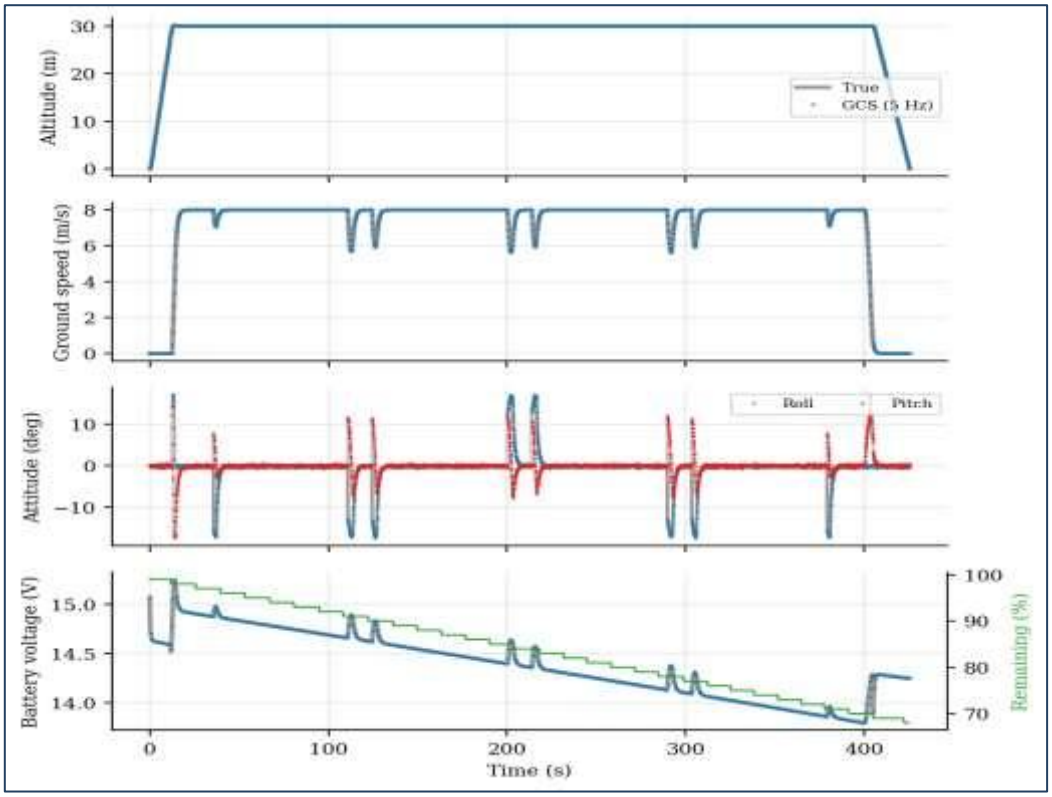

Fig. 6. True and GCS-received altitude, ground speed, attitude, and battery state during the mission.

Table 8 reports the error of the zero-order-hold display against the true state. The position RMS error during the survey phase is 1.28 m, dominated by the age of the 5 Hz position stream rather than by sensing error.

Table 8. Reconstruction error of the GCS display (zero-order hold, loss-free link).

Quantity	Source message (rate)	RMS error
Horizontal position (survey phase)	GLOBAL_POSITION_INT (5 Hz)	1.28 m
Altitude	GLOBAL_POSITION_INT (5 Hz)	0.084 m
Roll	ATTITUDE (10 Hz)	0.64°
Pitch	ATTITUDE (10 Hz)	0.56°
Ground speed	VFR_HUD (4 Hz)	0.070 m/s
Battery voltage	SYS_STATUS (2 Hz)	0.019 V

Fig. 7 shows how update rate and packet loss affect the displayed-position error. At 5 Hz the mean error is 1.26 m with no loss, 1.45 m at 10 % loss, 1.67 m at 20 %, and 2.31 m at 40 %, so target T3 ($\leq$ 2 m up to 20 % loss) is met; at 1 Hz the error rises from 4.4 m to 9.6 m. The 95th-percentile error at 5 Hz and 20 % loss is 3.6 m. Eqs. (11) and (12) predict the simulated means with a mean relative error of 1.4 % (maximum 3.3 %), so the design can be tuned analytically. Raising the position rate from 5 to 10 Hz reduces the mean error by about 0.4 m but adds 200 B/s (+6 % utilisation), so 5 Hz is a sensible default.

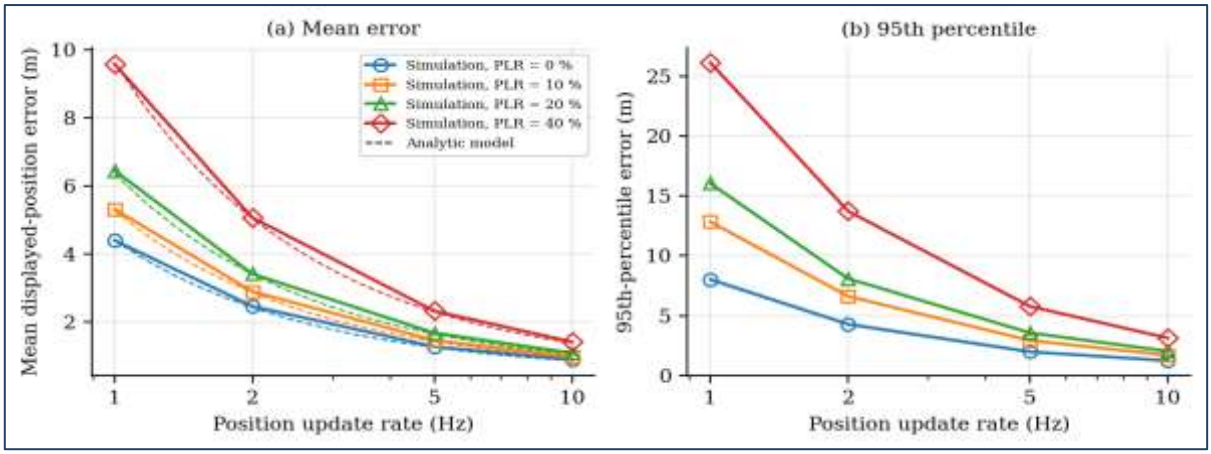

Fig. 7. Displayed-position error versus position update rate and packet loss: simulation (markers) and analytic model (dashed).

7.5 Safety: watchdog and link loss

Fig. 8a compares Eq. (13) with a 4,000,000-heartbeat simulation for each loss level. For 20 % loss, a 3 s timeout gives 23 false alarms per hour (simulated 23), a 5 s timeout gives 0.9 per hour, and at 10 % loss a 3 s timeout gives 3.2 per hour. The timeout therefore trades detection delay against nuisance alarms: since false alarms become frequent only when the link is already degraded, a 3 s timeout with latching is a reasonable default.

Fig. 8b and the following statistics come from 300 simulated outbound legs at 10 m/s, with the onboard failsafe returning the UAV to launch after 5 s without a GCS heartbeat. The GCS alarm first fired at a mean distance of 4.65 km (5th to 95th percentile 3.90 to 5.28 km), whereas the onboard return-to-launch began at 5.72 km on average. The operator was thus warned a mean 104 s before the failsafe, meeting target T5 (≥ 30 s). Without hysteresis the raw watchdog produced 7.0 alarm onsets per leg because the link fluctuates near the edge of coverage, which motivates the latch of Section 4.5. In 4 of 300 runs the failsafe fired without a preceding GCS alarm because loss was asymmetric between up- and downlink, so the GCS should also treat repeated COMMAND_ACK failures as a link-quality indicator.

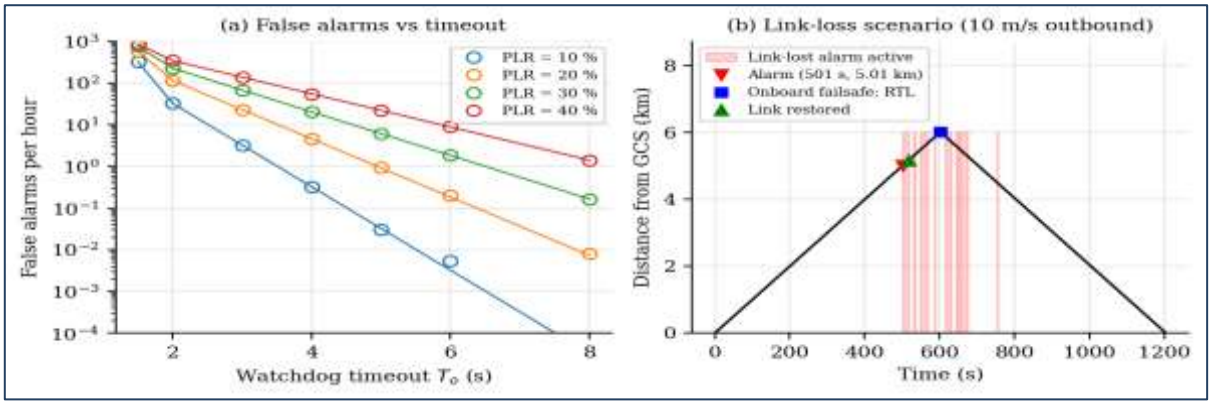

Fig. 8. (a) Watchdog false-alarm rate versus timeout: Eq. (13) (lines) and simulation (markers). (b) Link-loss scenario with alarm, onboard return-to-launch, and link restoration.

The gradual-fade result above should be distinguished from an abrupt loss. In the testbed, a 25 s total outage was injected during the survey leg: the watchdog raised the alarm 2.7 s after the outage began, the emulated autopilot began its return-to-launch 5 s after it began, and the alarm cleared 2.9 s after the link returned (three heartbeats). The warning lead before the failsafe was therefore only about 2.3 s, so target T5 is met for gradual degradation but not for abrupt loss.

7.6 Command reliability

Fig. 9 shows command success and round-trip time against packet loss for one to five attempts (t_{to} = 0.6 s, single-exchange round trip 116 ms). The simulation agrees with Eq. (14) within 0.8 percentage points. At 20 % loss, a single attempt succeeds 64 % of the time, three attempts 95.3 %, and five attempts 99.4 % with a mean round-trip time of 436 ms for successful commands (Table 9). At 40 % loss five attempts still give only 89.3 %. Target T4 (≥ 99 % at 20 % loss) is therefore met only with five attempts, which suits critical commands such as return-to-launch; three attempts suffice for routine commands.

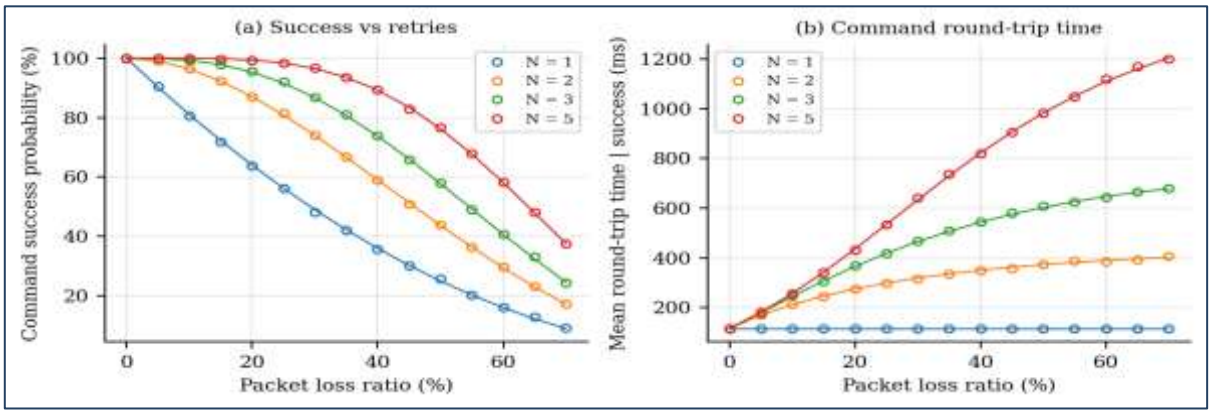

Fig. 9. Command success probability (a) and mean round-trip time of successful commands (b) versus packet loss for N attempts; lines: Eqs. (14) and (15), markers: simulation.

Table 9. Command success probability and mean round-trip time (analytic).

Attempts N	Success, 20 % loss	Round trip, 20 %	Success, 40 % loss	Round trip, 40 %
1	64.0 %	116 ms	36.0 %	116 ms
2	87.0 %	275 ms	59.0 %	351 ms
3	95.3 %	366 ms	73.8 %	544 ms
5	99.4 %	436 ms	89.3 %	822 ms

7.7 Multi-UAV channel access

The mean frame airtime at capacity C is $\tau = 12.0$ ms, so one UAV running the full telemetry plan offers a load of $G = 0.30$. Fig. 10a validates Eq. (16): peak throughput is 0.184 for pure ALOHA (at $G = 0.5$) and 0.367 for slotted ALOHA (at $G = 1$), with simulated values within 0.6 percentage points. Fig. 10b shows the consequence for a fleet sharing one radio channel: with three UAVs at full rate, the delivery ratio is 30 % (pure ALOHA), 55 % (slotted ALOHA), and 99.7 % (TDMA); with five UAVs it is 9 %, 30 %, and 60 %. Contention-based access therefore wastes most of the channel at this load, and a scheduled access scheme with per-UAV rate reduction beyond about three UAVs is required. This also identifies the communication layer as the natural place for the MAC and routing experiments the architecture is meant to support.

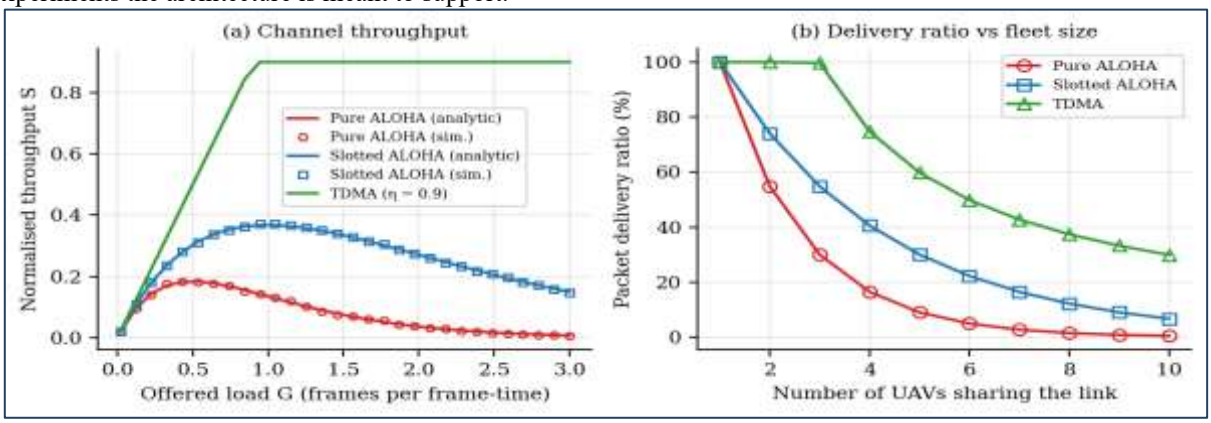

Fig. 10. (a) Channel throughput of pure ALOHA, slotted ALOHA, and TDMA ($\eta = 0.9$): analytic and simulated. (b) Per-UAV delivery ratio versus fleet size at full telemetry rate.

7.8 Resource use, power, and cost

Table 10 summarises the platform load. Summing the component costs (decode, logging, ten state pushes per second) gave only 0.38 ms of CPU per second of telemetry, but the whole back-end process measured on the testbed used 2.1 % of one host core over a 127 s run that included streaming, command traffic, logging, the web server, and a 20 Hz state sampler (1.4 % in the bursty run). A sum of component costs would therefore underestimate the real load by roughly a factor of five, because interpreter and threading overheads are not counted. Projecting the measured value to a Raspberry Pi 4 with a slowdown of $\kappa = 5$ to 10, which is an assumption and not a measurement, gives 10 to 21 % of one core (2.6 to 5.1 % of the four), which is at or above target T6 and must be verified on the board. The process occupied 58 MB of resident memory, so a 1 or 2 GB board is adequate. With the projected load the estimated power is 3.2 to 3.1 W (excluding the display), giving 9.7 to 10.1 h from a 10,000 mAh power bank. The ground segment costs USD 164.49 (Table 3), or USD 224.49 with the optional display, which exceeds target T8, so the display is best treated as optional because any phone or tablet can serve as the interface.

Table 10. Resource use, power, and autonomy (host measurements and Raspberry Pi 4 projections).

Quantity	Value	Basis
MAVLink decode cost	8.4 µs per message	Measured on host
SQLite insert / JSON state push	0.77 µs per row / 15.5 µs	Measured on host
Whole back-end process, clean-channel run	2.06 % of one host core	Measured on testbed
Whole back-end process, bursty-channel run	1.42 % of one host core	Measured on testbed
Pi 4 CPU load ($\kappa = 5$ to 10)	10 to 21 % of one core	Projection (assumed κ)
Resident memory	58 MB	Measured on testbed
Log volume	3,180 rows, 0.8 MB in 127 s	Measured on testbed
System power (excluding display)	3.2 to 3.1 W	Eq. (21), board power from [26]
Autonomy on 10,000 mAh power bank	9.7 to 10.1 h	Eq. (22), $\eta = 0.85$

7.9 Compliance with design targets

Table 11. Results against the design targets of Table 2 for independent and bursty loss.

ID	Target	Independent loss	Bursty loss (20 %, 2 s fades)	Met
T1	Latency ≤ 100 ms	58.2 ms (model)	Unchanged (delay of delivered frames)	Yes
T2	$\rho \leq 0.5$ with signing	$\rho = 0.40$	Unchanged	Yes
T3	Position error ≤ 2 m (5 Hz)	1.67 m (model); 1.50 m (testbed, no loss)	5.0 m (model); 5.5 m (testbed)	Independent loss only
T4	Command success ≥ 99 % at 20 % loss	99.4 % (N = 5)	93.5 % fixed; 99.1 % with backoff (N = 5)	Yes, with backoff

ID	Target	Independent loss	Bursty loss (20 %, 2 s fades)	Met
T5	Failsafe warning lead ≥ 30 s	104 s (gradual fade, model)	Abrupt outage: 2.3 s (testbed)	Gradual loss only
T6	CPU ≤ 10 % of one core	2.1 % on host; 10 to 21 % projected on Pi	Same	Borderline
T7	Autonomy ≥ 8 h	9.7 to 10.1 h (projected)	Same	Yes
T8	Cost ≤ USD 200 (no display)	USD 164.49 (USD 224.49 with display)	Same	Yes (display excluded)

7.10 Cross-check against published measurements

Because no flight hardware was available to the author, the model inputs and outputs were compared with measurements and specifications published by others (Table 12). The sources differ in quality: two are peer-reviewed or preprint measurement studies [30], [31], one is a thesis [33], two are vendor or manufacturer documentation [23], [32], and one is a public software project [34]; none evaluates a complete GCS, so each is used only for the parameter it measures.

Table 12. Published measurements and specifications compared with the model.

Quantity	Published value (source)	This study	Outcome
UHF air-to-ground path-loss exponent	Measured at 446 MHz, UAV at 60/70/80 m altitude: η = 2.18 / 1.24 / 1.47 [30]	n = 2.5 (rural), 3.5 (suburban), 4.5 (urban)	Model conservative for an elevated UAV; with the measured exponents the 5 % loss range exceeds 20 km at 433 MHz (limited by horizon and regulation, not by the link budget)
Packet loss of a 3DR 915 MHz link at 100 m, line of sight	2.0 % at 30 Hz bidirectional traffic [31]	About 0 % at 100 m	Real links have a loss floor of about 2 %; re-evaluated below
Typical range of a 100 mW SiK radio	500 m, sensitivity −121 dBm [7]; well over 1 km with whip antennas [36]	Urban 425 m; suburban 3.2 km (433 MHz)	Urban case was calibrated to this figure, so it is not an independent check; suburban is an upper estimate
Reported in-field ranges of other UAV links	802.15.4: 500–1,500 m; LoRa drive tests: 650–1,600 m [35]	Urban 300–425 m; suburban 2.1–3.2 km	Same order of magnitude in clutter; different radios
SiK TDM window	Default maximum window 33 ms [32]	Mean wait 16.5 ms (33 ms window)	Adopted from documentation
End-to-end latency	About 100 ms for a gimbal chain dominated by the 3DR radio and video [31]; +0.11 s extra latency on an 8-bit APM 2.5 under heavy load [33]	One-way 58 ms, round trip 116 ms	Same order of magnitude; indicative only because the systems differ
Telemetry bandwidth of the standard stream set	About 932 B/s, 32 % of a SiK link with ECC (about 2,900 B/s) [34]	963 B/s; ρ = 0.30 at C = 3,200 B/s, 0.33 at 2,900 B/s	Same order of magnitude (the project figure is a budget, not a measurement); signed ρ = 0.44 at 2,900 B/s still meets T2
Raspberry Pi 4 board power	Idle and full-load figures from the manufacturer [26]	2.4 W and 6.8 W	Adopted

The measured loss floor of 2 % [31] was then applied analytically using Eqs. (11) to (14), which the simulations validated. With p = 0.02 the mean displayed-position error at 5 Hz is 1.27 m (0.87 m at 10 Hz and 4.5 m at 1 Hz), a single command attempt succeeds 96.0 % of the time and three attempts 99.994 %, and a 3 s watchdog raises 0.03 false alarms per hour (a 2 s watchdog would raise 1.4). The independent-loss targets of Table 2 therefore remain satisfied with the measured loss floor; the stronger test is bursty loss, treated next.

7.11 Testbed results

Table 13 reports the end-to-end tests of the implemented back end. On the clean channel the achieved stream rates matched the requested ones to within 0.1 Hz (heartbeat 1.03, attitude 10.0, position 5.1 Hz), confirming that per-message stream intervals configure correctly through the emulated link. The mean displayed-position error in flight was 1.50 m (95th percentile 2.25 m) at 7.9 m/s, about 20 % above the 1.24 m predicted by Eqs. (11) and (12), the difference being attributable to the 20 Hz sampling and scheduling jitter of the testbed. Under bursty loss (realised downlink loss 18.6 %) the mean error rose to 5.5 m with a 95th percentile of 27.3 m, close to the 5.0 m and 25.5 m predicted by the bursty-loss simulation (Fig. 12). In a real-time command test under the same bursty channel with three attempts per command (40 commands per scheme, realised loss 23 %), fixed-interval retries succeeded in 80 % of cases and backoff retries in 85 %; the simulation predicts 88 % and 91 %. With only 40 commands per scheme the sampling error is about ±6 percentage points, so the test confirms the level of the simulation but cannot resolve the benefit of backoff, which rests on the simulation.

Table 13. Testbed measurements of the implemented GCS (emulated radio and vehicle, 5 Hz position stream).

Measurement	Clean channel	Bursty channel (20 %, 2 s fades)
Time from start to streams configured	0.86 s	2.80 s
Mission upload (6 items)	0.53 s	0.55 s
Arm command round trip	100 ms	71 ms
Command round trip, mean / 95th percentile	88 / 106 ms	see text
Mean / RMS / 95th-percentile position error	1.50 / 1.57 / 2.25 m	5.5 / 10.0 / 27.3 m
Realised downlink / uplink loss	0.0 % / 0.0 %	18.6 % / 10.8 %
Watchdog alarm onsets during the run	0	4
Dashboard push rate / age at client	9.9 Hz / 0.8 ms	9.9 Hz / 0.6 ms
Back-end CPU (one host core) / memory	2.1 % / 58 MB	1.4 % / 57 MB

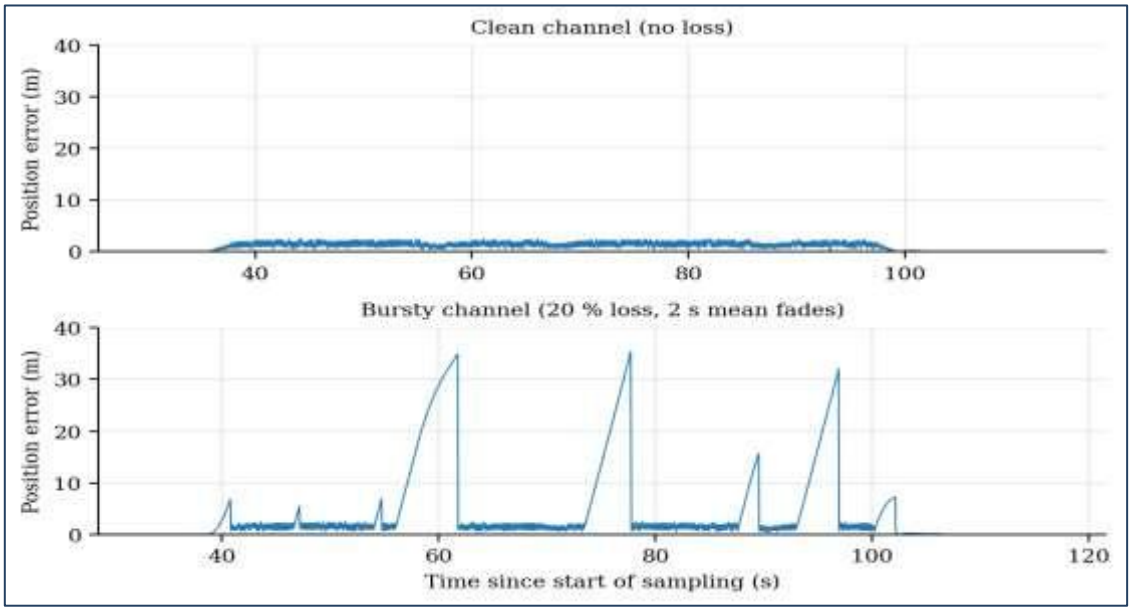

Fig. 12. Displayed-position error against ground truth during the survey leg on the testbed: clean and bursty channels.

7.12 Bursty loss

Table 14 and Fig. 11 compare independent and bursty loss at the same 20 % mean loss. As fades lengthen, the mean displayed-position error grows from 1.8 m to 5.0 m for 2 s fades and 10.3 m for 5 s fades, and the 95th percentile reaches 26 m, because a 5 Hz stream cannot help when the link is down. The false-alarm rate of a 3 s watchdog rises from 23 to 106 per hour for 2 s fades; Eq. (24) predicts 104 per hour. These alarms are largely genuine fades, not spurious events, but they show that an alarm rate computed under independence understates how often the operator will be warned. Fixed-interval retries lose effectiveness because all attempts fall in the same fade, and exponential backoff recovers it: with five attempts and 2 s fades, success rises from 93.5 % to 99.1 %, and with eight attempts to 99.98 %, at the price of a longer worst-case completion time (about 12 s for five attempts).

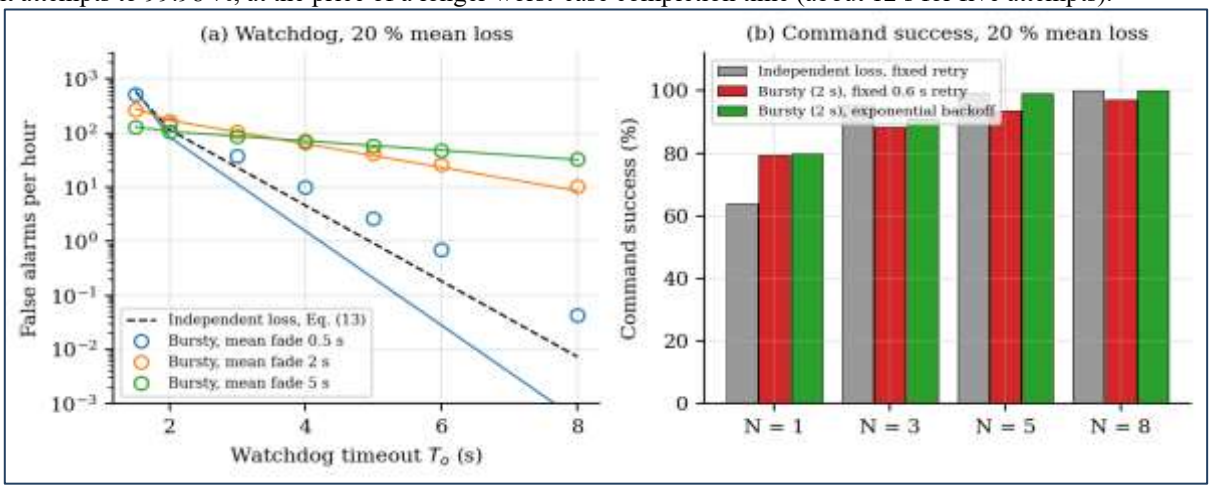

Fig. 11. (a) Watchdog false-alarm rate for independent and bursty loss (20 % mean loss): Eq. (24) (lines) and simulation (markers). (b) Command success for fixed and exponential-backoff retries.

Table 14. Independent versus bursty loss at 20 % mean loss (simulation).

Loss process	Mean position error (m)	95th percentile (m)	False alarms/h (To = 3 s)	Command success, N = 5, fixed	Command success, N = 5, backoff
Independent	1.77	3.5	23	99.4 %	99.4 %
Bursty, 0.5 s fades	2.37	7.9	38	n/a	n/a
Bursty, 2 s fades	4.96	25.5	106	93.5 %	99.1 %
Bursty, 5 s fades	10.27	62.9	86	87.3 %	95.4 %

7.13 Comparison with MAVProxy

MAVProxy, the ArduPilot project's command-line GCS [39], was run headless against the same vehicle emulator on its second connection, with the same legacy 4 Hz streams requested by both programs (about 57 messages/s). Over 80 s, the GCS back end used 0.51 s of CPU time (0.64 % of one core, 112 µs per message) and 49 MB of memory, against 6.73 s (8.4 %, 1475 µs per message) and 72 MB for MAVProxy, so MAVProxy used about 13 times more CPU per message. This comparison is limited: in this run the back end had logging and the web server switched off, MAVProxy loads many modules and writes a telemetry log, and MAVProxy supports parameters, calibration, many vehicles, and far more commands. The result shows that a purpose-built minimal back end is cheaper to run, not that it replaces MAVProxy, and Mission Planner and QGroundControl could not be run in the headless environment and were not compared.

Table 15. Resource comparison with MAVProxy on the same message stream (development host).

Metric	This GCS back end	MAVProxy
CPU time over the run	0.51 s of 80 s	6.73 s of 80 s
CPU use (one core)	0.64 %	8.4 %
CPU time per message	112 µs	1475 µs
Resident memory	49 MB	72 MB
Functions	Monitoring, commands, missions, watchdog, browser HMI	Full-featured console GCS with many modules

8. Discussion

The results support six design guidelines. First, a seven-message plan with 5 Hz position and 10 Hz attitude uses about one third of a 64 kbps SiK link, leaving room for signing, mission transfers, and a second UAV. Second, MAVLink signing [19] should be enabled because its cost (+325 B/s) is modest. Third, because packet loss rises from negligible to severe over a short distance and, in practice, arrives in fades, the watchdog (3 s timeout, three-heartbeat latch) should be complemented by signal-strength and link-margin indicators, and the interface must show the age of every displayed value so that the operator can see when the picture is stale. Fourth, retries must be spaced with exponential backoff, with three attempts for routine commands and five for critical ones. Fifth, for more than about three UAVs on one radio channel, scheduled access with reduced per-UAV rates is needed. Sixth, given memory-driven price rises [24], [25], the 1 GB or 2 GB Raspberry Pi 4 is the appropriate choice.

The main contribution beyond a design is the finding that independent-loss analysis, which is what most simple models and many quick tests assume, understates the effect of fades by a large factor: the 2 m display target that holds under independent loss is missed by a factor of 2.5 under bursty loss, in both the simulation and the testbed. A 5 Hz stream cannot compensate for a link that is down; only prediction on the display (dead reckoning with a visible uncertainty) or a better link would.

Threats to validity should be stated plainly. The vehicle is the author's emulator, so autopilot-specific behaviour (stream scheduling, arming checks, failsafe logic, message timing) reflects the author's implementation of the documented interface and not ArduPilot or PX4, and the ArduPilot SITL runs that would test this were not possible in the available environment. The radio is emulated on a single machine, so no radio-frequency effects, antenna patterns, interference, or serial-port behaviour are tested. The bursty channel has two states and exponential sojourns, which is a model, and its parameters were chosen, not measured. The CPU load was measured on an x86 development host and projected to the Raspberry Pi with an assumed factor; the projected load is at or above the 10 % target and must be measured on the board. The path-loss exponents measured in [30] are from one site at 446 MHz, and three prices in Table 3 are estimates. No human-factors study was performed, the dashboard was syntax-checked but not evaluated by operators, and functional parity with Mission Planner or QGroundControl is not claimed.

9. Limitations and Future Work

The most important next step is validation on real systems: running the supplied runner against ArduPilot software-in-the-loop on a laptop, then against a Pixhawk-class flight controller and SiK radios on the bench, and finally in field flights with a safety pilot in accordance with local aviation and spectrum regulations (the 433 MHz and 915 MHz bands are not available everywhere). These tests would replace the emulated vehicle and radio with real ones and measure the Raspberry Pi load and the loss statistics directly. Further work includes authentication and encryption for the web interface, display-side dead reckoning with a visible uncertainty to mitigate stale data, multi-vehicle support with scheduled access, coverage path planning for agricultural monitoring, alternative links (LoRa, 4G) with signing, a pluggable communication layer for MAC and routing experiments, automated post-flight analytics, and a formal usability study.

10. Conclusion

A low-cost ground control station was designed, implemented in about 200 lines of Python, and evaluated through an emulated radio and vehicle, analytical models validated by simulation, and published measurements. Under independent loss the design meets seven

of the eight targets in simulation: 58 ms modelled latency, link utilisation of 0.40 with signing, 1.7 m position error at 5 Hz and 20 % loss, 99.4 % command success with five attempts, and a hardware cost of USD 164. The CPU target is borderline once projected to the Raspberry Pi. Under bursty loss, which is more realistic, the position-error and failsafe-warning targets are missed (5.5 m measured on the testbed against 2 m), and command reliability is met only with exponential-backoff retries. The back end used about 13 times less CPU per message than MAVProxy on the same stream, with a far smaller feature set. These findings come from emulated components; validation against ArduPilot SITL, real radios, and the Raspberry Pi is the necessary next stage, and the supplied runner is intended to make that step straightforward.

Acknowledgements
The author acknowledges the open-source communities behind ArduPilot, PX4, MAVLink, pymavlink, and the Python scientific stack, whose software made this work possible.

Funding
No external funding was received for this work.

Conflict of Interest
The author declares no conflict of interest.

Data Availability
The GCS back end, radio and vehicle emulators, experiment runner, simulation code, measured benchmarks, raw run logs, and all generated results and figures are provided as supplementary material (gcs_supplementary_v4.zip) and are available from the corresponding author on request. The external measurements used for cross-checking are taken from the cited publications and documents [7], [23], [26], [30]–[36] and were not re-measured by the author.

References

[1] A. Shakhatreh et al., “Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges,” IEEE Access, vol. 7, pp. 48572–48634, 2019.
[2] M. Mozaffari, W. Saad, M. Bennis, Y.-H. Nam, and M. Debbah, “A tutorial on UAVs for wireless networks: Applications, challenges, and open problems,” IEEE Commun. Surveys Tuts., vol. 21, no. 3, pp. 2334–2360, 2019.
[3] H. Chao, Y. Cao, and Y. Chen, “Autopilots for small unmanned aerial vehicles: A survey,” Int. J. Control Autom. Syst., vol. 8, no. 1, pp. 36–44, 2010.
[4] ArduPilot Development Team, “Mission Planner ground control station,” ArduPilot Documentation. [Online]. Available: https://ardupilot.org/planner
[5] Dronecode Foundation, “QGroundControl ground control station.” [Online]. Available: https://qgroundcontrol.com
[6] ArduPilot Development Team, “APM Planner 2,” ArduPilot Documentation. [Online]. Available: https://ardupilot.org/planner2
[7] A. Koubâa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, “Micro Air Vehicle Link (MAVLink) in a nutshell: A survey,” IEEE Access, vol. 7, pp. 87658–87680, 2019.
[8] ArduPilot Development Team, “ArduPilot: Versatile, trusted, open autopilot software.” [Online]. Available: https://ardupilot.org
[9] L. Meier, D. Honegger, and M. Pollefeys, “PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms,” in Proc. IEEE Int. Conf. Robotics and Automation (ICRA), 2015, pp. 6235–6240.
[10] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, “PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision,” Autonomous Robots, vol. 33, no. 1–2, pp. 21–39, 2012.
[11] L. Gupta, R. Jain, and G. Vaszkun, “Survey of important issues in UAV communication networks,” IEEE Commun. Surveys Tuts., vol. 18, no. 2, pp. 1123–1152, 2016.
[12] M. Ramirez-Atencia and D. Camacho, “Extending QGroundControl for automated mission planning of UAVs,” Sensors, vol. 18, no. 7, Art. no. 2339, 2018.
[13] A. Ebeid, M. Skriver, and J. Jensen, “A survey of open-source UAV flight controllers and flight simulators,” Microprocessors and Microsystems, vol. 61, pp. 11–20, 2018.
[14] S. Kaul, R. Yates, and M. Gruteser, “Real-time status: How often should one update?” in Proc. IEEE INFOCOM, 2012, pp. 2731–2735.
[15] R. D. Yates and S. K. Kaul, “The age of information: Real-time status updating by multiple sources,” IEEE Trans. Inf. Theory, vol. 65, no. 3, pp. 1807–1827, 2019.
[16] L. Kleinrock, Queueing Systems, Volume 1: Theory. New York, NY, USA: Wiley, 1975.
[17] N. Abramson, “The ALOHA system: Another alternative for computer communications,” in Proc. AFIPS Fall Joint Computer Conf., 1970, pp. 281–285.
[18] L. G. Roberts, “ALOHA packet system with and without slots and capture,” ACM SIGCOMM Comput. Commun. Rev., vol. 5, no. 2, pp. 28–42, 1975.
[19] MAVLink Developer Guide, “Message signing (MAVLink 2).” [Online]. Available: https://mavlink.io/en/guide/message_signing.html
[20] ArduPilot, “pymavlink: Python MAVLink interface and utilities.” [Online]. Available: https://github.com/ArduPilot/pymavlink
[21] H. T. Friis, “A note on a simple transmission formula,” Proc. IRE, vol. 34, no. 5, pp. 254–256, 1946.
[22] T. S. Rappaport, Wireless Communications: Principles and Practice, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2002.
[23] Holybro, “SiK Telemetry Radio V3,” product page. [Online]. Available: https://holybro.com/products/sik-telemetry-radio-v3 (accessed Oct. 2026).
[24] Raspberry Pi Ltd, “A new 3GB Raspberry Pi 4 for $83.75, and more memory-driven price increases,” Apr. 2026. [Online]. Available: https://www.raspberrypi.com/news/a-new-3gb-raspberry-pi-4-for-83-75-and-more-memory-driven-price-increases/

[25] OMG! Ubuntu, "Raspberry Pi 2GB price increase from 1 October 2026," Oct. 2026. [Online]. Available: https://www.omgubuntu.co.uk/2026/10/raspberry-pi-2gb-costs-how-much-now
[26] Raspberry Pi Ltd, "Thermal testing Raspberry Pi 4," Raspberry Pi News. [Online]. Available: https://www.raspberrypi.com/news/thermal-testing-raspberry-pi-4/
[27] Elektronik, "Raspberry Pi releases a new Touch Display after nine years." [Online]. Available: https://www.elektroniknet.de/international/raspberry-pi-releases-a-new-touch-display-after-nine-years.221559.html
[28] J. G. Proakis and M. Salehi, Digital Communications, 5th ed. New York, NY, USA: McGraw-Hill, 2008.
[29] M. J. E. Golay, "Notes on digital coding," Proc. IRE, vol. 37, no. 6, p. 657, 1949.
[30] B. Ede, S. Yarkan, A. R. Ekti, T. Baykas, H. A. Cirpan, and A. Gorcin, "Measurement based statistical channel characterization of air-to-ground path loss model at 446 MHz for narrow-band signals in low altitude UAVs," in Proc. IEEE 91st Veh. Technol. Conf. (VTC-Spring), 2020; arXiv:1912.10750.
[31] A. Cherpillod, S. Mintchev, and D. Floreano, "Embodied flight with a drone," arXiv:1707.01788, 2017.
[32] ArduPilot Development Team, "Configuring a telemetry radio using Mission Planner," Copter Documentation. [Online]. Available: https://ardupilot.org/copter/docs/common-configuring-a-telemetry-radio-using-mission-planner.html
[33] J. A. Marty, "Vulnerability analysis of the MAVLink protocol for command and control of unmanned aircraft," M.S. thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, USA, 2014, AFIT-ENG-14-M-50. [Online]. Available: https://scholar.afit.edu/etd/613
[34] 0winDev, "android-ground-control-station, pull request #46: Request the telemetry streams the HUD needs," GitHub. [Online]. Available: https://github.com/0winDev/android-ground-control-station/pull/46
[35] L. Shi, N. J. Hernández Marcano, and R. H. Jacobsen, "A review on communication protocols for autonomous unmanned aerial vehicles for inspection application," arXiv:2111.06714, Aarhus University.
[36] ThunderFly, "TFSIK01: High-reliable wireless telemetry data link," documentation. [Online]. Available: https://docs.thunderfly.cz/avionics/TFSIK01/
[37] E. N. Gilbert, "Capacity of a burst-noise channel," Bell Syst. Tech. J., vol. 39, no. 5, pp. 1253–1265, 1960.
[38] E. O. Elliott, "Estimates of error rates for codes on burst-noise channels," Bell Syst. Tech. J., vol. 42, no. 5, pp. 1977–1997, 1963.
[39] ArduPilot, "MAVProxy: A UAV ground station software package for MAVLink based systems." [Online]. Available: https://github.com/ArduPilot/MAVProxy