

DOI: 10.37943/ZUOX7374**Azamat Baibussinov**

Master's Degree, Department of Technical Support
a.baibussinov@astanait.edu.kz; orcid.org/0000-0002-9714-278X
National Defense University of the Republic of Kazakhstan, Kazakhstan

Kaisarbek Yesbergenov

PhD, Associate Professor, Head of the Doctoral Program
ekb72@mail.ru, orcid.org/0009-0009-3648-5568
National Defense University of the Republic of Kazakhstan, Kazakhstan

Abdykassym Altynbay

Candidate of Technical Sciences
altinbai1955@gmail.com, orcid.org/0009-0008-3009-5190
“Digital Heritage of Eurasia” LLP, Kazakhstan

Gul Jussupova

Candidate of Technical Sciences, Department of Digitalization and Public Services
g.jussupova@sci.gov.kz, orcid.org/0009-0009-7205-3666
Ministry of Science and Higher Education of the Republic of Kazakhstan,
Kazakhstan

Daulet Toibazarov

PhD, Associate Professor, Senior Researcher,
Department of Scientific Information and Innovations
aldar-kosel@yandex.ru, orcid.org/0009-0000-9135-7360
National Defense University of the Republic of Kazakhstan, Kazakhstan

Murat Zhakenov

Candidate of Technical Sciences
muratzhakenov@outlook.com, orcid.org/0009-0005-9672-4365
“Digital Heritage of Eurasia” LLP, Kazakhstan

**RECORD-LEVEL VALIDATION OF OPEN-SOURCE UAV FLIGHT LOG PARSERS
ON THE NIST CFREDS CORPUS**

Abstract: Examiners read drone flight logs with open-source parsers, and published comparisons of these parsers count the rows a tool writes or note whether it crashed. We tested whether such counts say anything about the evidence. Six parsers (DatCon, DROP, DRDP, Fodogu, pymavlink and GRYPHON) were run three times on each of the 72 files of four NIST CFReDS drone datasets, 1,296 runs with pinned versions and hashed inputs and outputs. Every coordinate a tool wrote was compared with two reference decoders: one accepts a DJI packet only when its checksum verifies, the other reproduces the Mission Planner exports shipped with the ArduPilot logs. On the DJI Phantom 3 files, DatCon and DROP wrote no unsupported position and agreed exactly wherever both reported one; 729,793 reference records (82.4%) are reproduced by both. The count differences earlier read as complementary recovery come from repeated rows and from two undocumented display rules: DROP blanks positions logged with two or fewer satellites, and DatCon writes nothing before a flag set by the first record with non-zero position and height. At the rate used here, 98.3% of DatCon's GPS-receiver rows repeat an earlier value (97.9% at its 30 Hz default). Runs that ended with status zero and no coordinates from a file holding them made up 21 of 126 runs on formats a tool claims to support and 61 of 120 on other formats. On the ArduPilot logs, 545 of 927 positions that pass a plain latitude and longitude range test were logged without a three-dimensional fix. All 431 completed run triples were bit-identical. The procedure is stated as a formal model, so that each reported quantity has a definition.

Keywords: drone forensics; flight log parsing; tool validation; known-answer testing; record-level comparison; DJI DAT; ArduPilot DataFlash; NIST CFReDS; repeatability; silent failure

Introduction (Literature review)

A drone log holds the flight path, the altitude, the attitude of the airframe and the commands the pilot sent. In investigations these files answer the question of where the aircraft was and when [1], and law enforcement practitioners report relying on them for exactly this purpose [2]. Recent reviews of UAV forensics treat flight logs as a central source of evidence [3], [4]. Manufacturers do not publish the formats, a gap that early studies of the forensic challenges posed by drones already pointed out [5-6]. Examiners therefore work with parsers that other researchers wrote by reverse engineering the binary layout, and most of those parsers began life as research prototypes rather than as validated instruments [7].

Clark and co-authors reverse engineered the on-board DAT format of the DJI Phantom III and released DROP, deriving the packet structure from the DatCon application [7-8]; Lee and co-authors showed that the DJI remote controller keeps records that identify the pilot and the flight [9]. Fodogu reuses the Python decoder of DROP and adds decryption for the DJI Mini 2 [10]. Zhao and co-authors published DRDP for the Mavic 2 Pro [11-12]. For the ArduPilot ecosystem, Mantas and Patsakis built GRYPHON on top of pymavlink [13-14]. Salamh and co-authors compared commercial suites on DJI evidence and found that no single product recovered everything [15].

How are such tools compared? Usually by how much output they produce. A tool writes more rows, or more rows that contain coordinates, and is called more complete [7], [15]; recent work on the reconstruction of UAV traces reports its results in the same way [16]. Our own earlier preprint of this work did the same and concluded that DatCon and DROP recover complementary GPS data, because their per-file counts differed by as much as a factor of eighteen. It recommended running both tools on every DJI file and merging the results. A reviewer of that manuscript pointed out the flaw, and we agree with it: a difference in counts does not show that either tool recovered anything the other missed. The extra rows may be duplicates, resampled values, a different record type, or wrong decodings, and a coordinate that lies inside the valid range of latitude and longitude need not be a place the aircraft ever was.

Tool testing in digital forensics has long drawn the line between a program that runs and a program whose results can be relied on. The NIST Computer Forensics Tool Testing programme works from written requirements, test assertions and data sets with known answers [17]. Horsman argues that any claim about reliability needs defined test data, expected results and documented behaviour on error [18]. ISO/IEC 27037 and ISO/IEC 17025 ask for documented and validated methods [19-20]. Published comparisons of UAV parsers do not report agreement with a known answer at the level of individual records.

This paper reports such a comparison. We ask three questions. First, when a parser writes a position, does that position correspond to data that is actually in the file? Second, when two parsers return different amounts of position data from the same file, what causes the difference? Third, how often does a parser fail in a way the examiner cannot see, and are its outputs stable when the same file is parsed again? We also state the procedure as a formal model, so that the quantities we report have definitions rather than names.

Methods and Materials

Four NIST CFReDS drone datasets were used, all produced by the VTO Labs Drone Forensics Program [21]: DF001 (DJI Phantom 3), DF056 (Sky Viper Journey, ArduPilot), DF057 (a do-it-yourself ArduPilot airframe) and DF014 (Yuneec Typhoon H). The archives were downloaded from the locations the CFReDS portal links to, and their MD5 hashes were compared with the values printed in the NIST data sheets. All four matched. The DataFlash logs of DF056 were extracted read-only from the E01 card image with libewf [22] and The Sleuth Kit [23]. Every file was hashed with SHA-256, and its format was labelled from the header signature alone, never from the dataset it came from or from its extension (Table 1).

Table 1 – Files used, with the header signature that assigned the format label

Dataset	Content	Signature	Files	MB
DF001	DJI Phantom 3, on-board .DAT	BUILD at offset 16	22	1,551.0
DF056	ArduPilot DataFlash .BIN	A3 95	4	0.6
DF057	ArduPilot DataFlash .BIN	A3 95	15	0.7
DF057	Mission Planner text export	FMT,	15	1.1
DF014	Yuneec Sensor log	BC 1A or version string	11	4.7
DF014	Yuneec, zero length	empty	4	0.0
DF014	Yuneec, 11-byte text stub	none	1	0.0
Total			72	1,558.1

The DF057 archive holds, next to each DataFlash binary, a text log that Mission Planner produced from it. The earlier preprint counted those 15 exports as separate evidence files. Here they are marked as derived exports: they are parsed and reported, and they serve as a known answer for the binaries, but they are excluded from evidence totals. The DJI V3, LOGH, DFLY and TXT files of the preprint were not re-examined, so this paper says nothing about them.

Table 2 lists the tools. Each one was fixed to a git commit or an exact package version. Only DROP has a command line interface. For the rest we wrote the smallest possible headless entry point; each calls the function that the tool's own graphical interface or command line calls, and no tool code was changed. The DatCon driver follows the conversion path of the DatCon window. DatCon lets the user choose the CSV sample rate: its default is 30 Hz, and the window offers rates from 1 to 200 Hz and a maximum setting equal to the 600 Hz tick rate of the V1 format. The main runs used 600 Hz, so that every tick carrying data produces a row. Because this choice affects row counts, all 22 DJI files were also converted three times at the default 30 Hz (66 additional runs); both sets of results are reported. For each tool we also recorded which formats it declares: in its documentation, or by its own format check in the code. DatCon and DROP declare DJI V1; Fodogu and DRDP accept V1 files by their own format check although their documentation names newer aircraft; pymavlink and GRYPHON declare DataFlash binaries and text logs. The wrappers are included in the artefact archive, and their hashes are stored with every run.

Table 2 – Parsers, versions and how they were invoked

Tool	Version or commit	Invocation
DatCon [8]	3.5.0, commit 2a51562	DatConCLI.java, headless conversion path, 600 Hz (maximum) and 30 Hz (default)
DROP [7]	commit 80dac9d	DROP.py <input> -o <output.csv>, unmodified
DRDP [11], [12]	commit 110934b	drdp_single.py, runs the unmodified main.py -d
Fodogu [10]	commit ddeb4b7	fodogu cli.py, calls decode.checkType
pymavlink [14]	2.4.49	pymavlink_dump.py, mavutil.mavlink_connection
GRYPHON [13]	d42bbfd, MAVProxy 1.8.74	gryphon_headless.py, calls get_MAVmsgs

Runs were executed on Windows 11 with CPython 3.11.9 in a dedicated virtual environment and Java 24.0.2, three runs at a time. Each tool was run on each file, whatever format the tool claims to support, and each pair was repeated three times, which gives 1,296 runs with a timeout of 3,600 s. For every run we stored the command, the exit status, the wall-clock time and the SHA-256 of the standard output and of every file the tool wrote.

Mathematical model of the validation procedure.

The procedure is stated here as a model, so that the quantities reported below have definitions rather than names. Throughout, $\#S$ is the number of elements of a set S , and $[C]$ equals one when the condition C holds and zero otherwise. A log file is a finite string of bytes

$$f = (b(0), b(1), \dots, b(L-1)), \quad b(i) \in \{0, 1, \dots, 255\}, \quad (1)$$

and a parser g maps f to a finite sequence of output rows. The DJI V1 format frames its data in packets, and a candidate packet starting at offset o carries its length, its record type and its tick number in fixed positions:

$$\ell(o) = b(o+1), \quad \tau(o) = b(o+4) + 256 \cdot b(o+5), \quad t(o) = \sum_{i=0 \dots 3} b(o+6+i) \cdot 256^i. \quad (2)$$

Its payload is masked with the low byte of the tick number, so that the plain byte at position i is

$$p(o, i) = b(o+10+i) \oplus (t(o) \bmod 256), \quad 0 \leq i < \ell(o) - 12. \quad (3)$$

Whether the candidate is a packet at all is decided by the cyclic redundancy check of the format, obtained from the recurrence

$$c(0) = 3692_{16}, \quad c(i+1) = [c(i) / 256] \oplus T[(b(o+i) \oplus c(i)) \bmod 256], \quad (4)$$

in which T is the table of the reflected polynomial 8408_{16} . The candidate is accepted when the computed value equals the two bytes that close it:

$$\begin{aligned} \chi(o) = 1 &\Leftrightarrow b(o) = 55_{16} \wedge b(o+2) = 0 \wedge \\ c(\ell(o)-2) &= b(o+\ell(o)-2) + 256 \cdot b(o+\ell(o)-1). \end{aligned} \quad (5)$$

The decoder walks the file once, stepping over a packet it accepts and one byte forward when the check fails, so that a corrupt region costs at most a resynchronisation:

$$P(f) = \{o : \chi(o) = 1\}, \quad o \leftarrow o + \ell(o) \text{ if } \chi(o) = 1, \quad o \leftarrow o + 1 \text{ otherwise}. \quad (6)$$

Two record types carry a position in these files. Type 1 is the fused navigation solution of the flight controller and stores longitude and latitude as 64-bit floating-point radians; type 5 is the solution of the GPS receiver and stores them as 32-bit integers in units of 10^{-7} degrees, together with the UTC date and time and the number of satellites $n(p)$. Coordinates are converted to degrees on decoding. Writing dec for the decoding of the plain payload, the reference set of a file is

$$R(\tau, f) = \{(t(o), \varphi, \lambda) : o \in P(f), \tau(o) = \tau, (\varphi, \lambda) = \text{dec}(\tau, p(o)) \neq (0, 0)\}. \quad (7)$$

A record enters R only after its checksum verifies, so a reference record is what the aircraft wrote and not merely something that parses. The output of a parser is a sequence of rows

$$O(g, f) = (r(1), \dots, r(m)), \quad r(j) = (k(j), \varphi(j), \lambda(j), \dots), \quad (8)$$

where $k(j)$ is the key of the row. For DataFlash logs the log timestamp alone is not a key, since it stops advancing while the receiver has no fix, so the key is the pair of the timestamp and the occurrence index among rows that share it, ordered lexicographically:

$$k(j) = (t(j), v(j)), \quad v(j) = \#\{j' < j : t(j') = t(j)\}. \quad (9)$$

For DJI files t is the tick number and $v = 0$ throughout, since no tick repeats among accepted packets of the same type in this corpus. In the 34 DataFlash series the largest v is 294 and distinct

timestamps are at least 181 ms apart. A row counts as a position row when both coordinates are present, finite, in range and not both zero:

$$\pi(r) = 1 \Leftrightarrow \varphi, \lambda \in \mathbb{R} \wedge |\varphi| \leq 90 \wedge |\lambda| \leq 180 \wedge (\varphi, \lambda) \neq (0, 0). \quad (10)$$

Each position row is compared with the most recent reference record of the same type at or before its key, that is with $\rho(r) = \arg \max \{ k(\rho) : \rho \in R, k(\rho) \leq k(r) \}$. The row is supported when both coordinates agree to the tolerance $\varepsilon = 10^{-6}$ degrees, about 0.1 m on the ground:

$$s(r) = 1 \Leftrightarrow \max (| \varphi(r) - \varphi(\rho(r)) |, | \lambda(r) - \lambda(\rho(r)) |) \leq \varepsilon. \quad (11)$$

Rows are read in output order. A supported row is fresh, $e(r) = 1$, when it sits at the key of its own reference record and no earlier row has already reproduced that record; every other supported row is a repeat, whether it carries a value forward from an earlier key or writes the same key and value a second time. The three counts

$$e(r(j)) = s \cdot [k(j) = k(\rho)] \cdot [\nexists j' < j : e(r(j')) = 1 \wedge \rho(r(j')) = \rho(r(j))], \quad (12)$$

$$\Phi = \sum \pi \cdot e, \quad \Psi = \sum \pi \cdot s \cdot (1 - e), \quad U = \sum \pi \cdot (1 - s)$$

partition the position rows of a run, since every position row falls into exactly one of them:

$$N(g, f) = \sum \pi(r) = \Phi + \Psi + U. \quad (13)$$

U counts the rows a reference decoding does not account for, and it is the quantity a benchmark should report first. Two further ratios describe an output: the share of its position rows that the reference supports, and the share of reference records the tool reproduces at their own key,

$$\sigma = (\Phi + \Psi) / N, \quad \varrho = \# \{ \rho \in R : \exists r \in O, k(r) = k(\rho) \wedge s(r) = 1 \} / \# R. \quad (14)$$

Each run is then placed in exactly one outcome class, tested in the order given. With T = 3,600 s the timeout, x the exit status, a the presence of the primary output file and m the number of data records it holds,

$$\Omega = \text{timeout [stopped at T]}; \text{ error [x} \neq 0 \text{ or uncaught exception]}; \quad (15)$$

$$\text{no output [a = 0]}; \text{ empty [m = 0]}; \text{ no position [N = 0]};$$

$$\text{verified [R exists} \wedge \sigma \geq 0.99\text{]}; \text{ unverified otherwise.}$$

A run is called silent when the tool gives no sign of failure although the file does contain position data. The indicator is defined only for files with a reference decoder, that is DJI V1 and ArduPilot DataFlash, and it is split by whether the format lies within the declared scope of the tool, $I(g, f) = 1$, or outside it:

$$S(g, f) = 1 \Leftrightarrow x = 0 \wedge \Omega \in \{ \text{no output, empty, no position} \} \wedge \# R(f) > 0, \quad (16)$$

$$S_{in} = \sum S \cdot I, \quad S_{out} = \sum S \cdot (1 - I).$$

What a parser shows is not what it decodes. Both DJI parsers apply a display rule to a decoded record before writing it out. DROP converts the coordinates to degrees and blanks them if any one of five conditions holds, so a record is shown only when all of their negations hold:

$$\gamma(D, \rho) = [3 \leq n(\rho) \leq 31] \cdot [\varphi(\rho) \neq 0] \cdot [\lambda(\rho) \neq 0] \cdot [|\varphi(\rho)| > \delta] \cdot [|\lambda(\rho)| > \delta],$$

$$\delta = 0.0175^\circ. \quad (17)$$

DatCon writes no coordinate until a flag latches, and the flag never resets. It is set by the first on-screen-display record (type 12, set Q) in which latitude, longitude and the height above the launch home point h (int16, units of 0.1 m) are all non-zero; in a file without such records it is set by the first navigation record with non-zero latitude and longitude:

$$t^* = \min \{ t(q) : q \in Q, \varphi(q) \neq 0 \wedge \lambda(q) \neq 0 \wedge h(q) \neq 0 \},$$

$$\gamma(A, \rho) = [t(\rho) \geq t^*]. \quad (18)$$

The records a parser makes visible, and the number of position rows it writes, are then

$$V(g) = \{ \rho \in R : \exists r \in O(g), e(r) = 1, \rho(r) = \rho \},$$

$$\Phi(g) = \# V(g), \quad N(g) = \# V(g) + \Psi(g) \text{ when } U(g) = 0. \quad (19)$$

This is enough to say what a comparison of counts can show. Let $K(g)$ be the set of keys at which g writes a supported position row. Let A and B be parsers with $U(A) = U(B) = 0$ whose outputs agree on every shared key,

$$K(g) = \{ k(r) : \pi(r) \cdot s(r) = 1 \},$$

$$\forall k \in K(A) \cap K(B) : \max (|\varphi(A, k) - \varphi(B, k)|, |\lambda(A, k) - \lambda(B, k)|) \leq \varepsilon. \quad (20)$$

Their ratio of position rows is then

$$D = N(A) / N(B) = (\# V(A) + \Psi(A)) / (\# V(B) + \Psi(B)), \quad (21)$$

which depends only on the display rules and on how often each tool repeats a value. It says nothing about decoding. The records one parser shows and the other does not are bounded by the rules themselves,

$$V(A) \setminus V(B) \subseteq \{ \rho : \gamma(B, \rho) = 0 \}, \quad V(B) \setminus V(A) \subseteq \{ \rho : \gamma(A, \rho) = 0 \}, \quad (22)$$

so an examiner who merges the two outputs adds records that one tool suppressed on purpose, not evidence the other failed to decode. The evidence a parser leaves behind on a file, and over a corpus C of files, is

$$L(g, f) = \# R(f) - \# V(g, f), \quad L(g) = \sum (f \in C) L(g, f). \quad (23)$$

Coverage of a parser over a group of N files is the share k/N it verifies. It is reported below with a 95% Wilson score interval [24], which behaves better than the normal approximation when the share is near zero or one, and the same interval is given for the shares of silent runs:

$$\hat{c} = k / N, \quad \hat{c} \pm [\hat{c} + z^2 / (2N) \pm z \cdot \sqrt{ \hat{c}(1-\hat{c}) / N + z^2 / (4N^2) }] / (1 + z^2 / N),$$

$$z = 1.96. \quad (24)$$

Repeatability is decided on the hashes of the outputs. With H the SHA-256 function, and $O(i)$ and $x(i)$ the output and the exit status of the i -th run of the same pair,

$$\Delta(g, f) = 1 \Leftrightarrow H(O(1)) = H(O(2)) = H(O(3)) \wedge x(1) = x(2) = x(3). \quad (25)$$

Provenance is checked against the NIST data sheet, which gives the polygon Π of the documented flights and the set W of their dates. For a decoded record we use the indicator of the polygon, the distance to its boundary when the record lies outside, and the agreement of the recorded date. Distances are computed in a local equiarectangular projection, $x = 111,320 \cdot \lambda \cdot \cos \varphi(\rho)$ m and $y = 111,320 \cdot \varphi$ m:

$$\begin{aligned} \iota(\rho) &= [(\varphi(\rho), \lambda(\rho)) \in \Pi], \\ d(\rho) &= \min (q \in \partial \Pi) \| (x, y)(\rho) - (x, y)(q) \|, \\ w(\rho) &= [date(\rho) \in W]. \end{aligned} \quad (26)$$

Within 150 km of the polygon the projection differs from the great-circle distance by at most 0.14%; for the distances of thousands of kilometres quoted below we give great-circle values. For DataFlash logs the reference records also carry the fix status of the receiver, and positions logged with a status below three are counted separately, because a range test of the kind used in earlier comparisons accepts them while the receiver had no three-dimensional fix. None of these checks establishes positional accuracy; they establish that a decoded value is what the device wrote, and whether it is consistent with what is known about the acquisition.

The reference decoders were written from the published framing and not from the code of any tool. The DataFlash decoder was checked against the Mission Planner text exports that NIST distributes with the DF057 binaries, which are a known answer produced by different software. For the Yuneec Sensor format no public decoder exists, so values decoded from those files could not be verified at all, and Yuneec files are left out of the silent-failure counts of equation (16); they are discussed separately. The protocol, including the classes of equation (15) and the threshold 0.99, was fixed before the full run; the threshold is a protocol choice and is not derived from the data. Two corrections were made afterwards to the analysis code, not to any tool run, and both are reported here: the signature rule for Yuneec files was widened once nine of them turned out to begin with the packet sync rather than the version string, and the key of equation (9) was extended by the occurrence index after DataFlash timestamps were found to repeat while the receiver had no fix.

Results

The DJI decoder accepted 26,247,078 packets from the 22 DF001 files and rejected 187 candidates whose checksum failed. It found 885,843 navigation positions in 13 files and 25,022 receiver positions in 16 files. Six files hold neither. No file contains a record of type 2096. The DataFlash decoder reproduced the Mission Planner export message for message in 13 of the 15 DF057 logs; in the other 2 the export has one more final GPS message and every earlier message is identical, which is what a truncated last record looks like when one program keeps it and the other does not.

Table 3 gives the outcome of every tool on every file group, with each class of equation (15) in its own column. Three parsers verified positions on the DJI files and two on the ArduPilot binaries. None extracted a position from a Yuneec Sensor log. No run fell into the unverified class: all 66 runs that reported positions had $\sigma = 1$, so any threshold between 0.50 and 1.00 gives the same classes. Verified shares, with 95% Wilson intervals, are 16 of 22, 72.7% [51.8%; 86.9%] for DatCon on the DJI files, 13 of 22, 59.1% [38.7%; 76.7%] for DROP and for Fodogu, and 5 of 19, 26.3% [11.8%; 48.8%] for pymavlink and for GRYPHON on the DataFlash binaries.

What the table does show is loss, and it matters whether the tool claims the format. On formats within the declared scope, 21 of 126, 16.7% [11.2%; 24.1%] of evidence runs were silent (21 of the 90 runs on files that do hold positions): three by DROP, three by Fodogu and 15 by DRDP, all on DJI files. Outside the declared scope the share was 61 of 120, 50.8% [42.0%; 59.6%]. If only the documentation defines the scope, so that Fodogu and DRDP no longer claim V1, the

in-scope count falls to 3 of 82, 3.7% [1.2%; 10.2%]. DatCon is the only tool that never failed silently; it refuses a file it cannot read with an exception. pymavlink and GRYPHON were never silent within their scope.

Table 3 – Outcome of the first repetition by parser and file group. The unverified class is empty for every row. Silent runs are counted only where a reference decoder exists; for derived Mission Planner exports they are given in brackets and excluded from evidence totals

Tool	File group	In scope	Files	Verified	No pos.	Empty	No output	Error	Timeout	Silent
DatCon	DJI V1 .DAT	yes	22	16	6	0	0	0	0	0
	DataFlash .BIN	no	19	0	0	0	0	19	0	0
	Yuneec Sensor	no	16	0	0	0	0	16	0	–
	MP text export	no	15	0	0	0	0	15	0	(0)
DROP	DJI V1 .DAT	yes	22	13	9	0	0	0	0	3
	DataFlash .BIN	no	19	0	0	0	19	0	0	13
	Yuneec Sensor	no	16	0	0	0	16	0	0	–
	MP text export	no	15	0	0	0	15	0	0	(9)
Fodogu	DJI V1 .DAT	yes	22	13	9	0	0	0	0	3
	DataFlash .BIN	no	19	0	0	0	3	16	0	3
	Yuneec Sensor	no	16	0	0	0	12	4	0	–
	MP text export	no	15	0	0	0	14	1	0	(8)
DRDP	DJI V1 .DAT	yes	22	0	0	21	0	0	1	15
	DataFlash .BIN	no	19	0	0	19	0	0	0	13
	Yuneec Sensor	no	16	0	0	16	0	0	0	–
	MP text export	no	15	0	0	15	0	0	0	(9)
pymavlink	DJI V1 .DAT	no	22	0	0	22	0	0	0	16
	DataFlash .BIN	yes	19	5	0	0	0	14	0	0
	Yuneec Sensor	no	16	0	0	3	0	13	0	–
	MP text export	yes	15	7	6	0	0	2	0	(0)
GRYPHON	DJI V1 .DAT	no	22	0	0	22	0	0	0	16
	DataFlash .BIN	yes	19	5	0	0	0	14	0	0
	Yuneec Sensor	no	16	0	0	3	0	13	0	–
	MP text export	yes	15	7	5	0	0	3	0	(0)

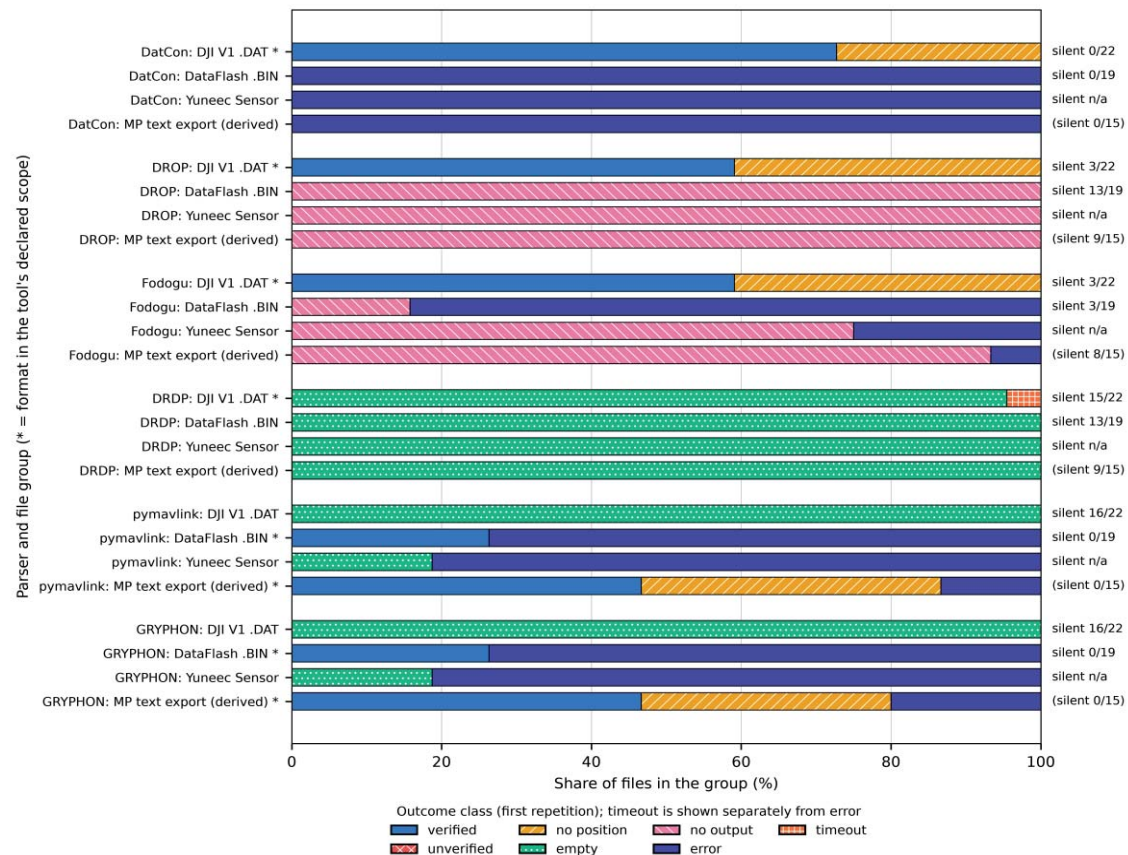


Figure 1 – Outcome of every parser on every file group (first repetition). Bars give the share of files in each class; timeout is shown separately from error; an asterisk marks a format within the tool's declared scope. Silent counts are given only where a reference decoder exists (n/a for Yuneec; derived exports in brackets)

The Yuneec files need a separate word. Without a decoder for the Sensor format there is no reference set $R(f)$. The only indication that a Sensor file holds positions is the plaintext telemetry the ST16 ground station wrote for the flight, and our analysis paired the two by the file number alone (Sensor_NNNNN with Telemetry_NNNNN). The archive timestamps of 9 of the 11 non-empty Sensor files agree with those of their telemetry files within 2 s, but Sensor_00004.bin differs by 810 s; Sensor_00016.bin differs by 4 s. Because the pairing is not established from the data, Yuneec runs are not counted as silent. Had the ST16 telemetry been accepted as a reference, 21 of the 96 Yuneec runs would be silent, seven each for DROP, Fodogu and DRDP.

The DJI files are the only ones on which several tools produce substantial output, and they are where the earlier count comparison was made. DatCon wrote 1,199,316 navigation position rows over the 22 files and DROP wrote 1,032,053. These are the totals the preprint reported as GPS records. Under equations (12) and (13) they decompose as follows. Not one row of either tool is unsupported, $U = 0$ in every file, and the largest deviation of a supported row from its reference record is $1.4 \cdot 10^{-9}$ m, which is floating-point rounding. The two tools write a supported position at 905,580 common keys and agree on all of them bit for bit (0 disagreements). At 729,793 of these keys both reproduce the reference record itself; at the other 175,787 there is no navigation record and both repeat the previous value. No key is fresh for one tool and a repeat for the other. The output of Fodogu is byte-identical to that of DROP on all 22 files.

DatCon reproduces 793,444 navigation records and repeats a value in 405,872 rows, $\Psi/N = 33.8\%$. DROP reproduces 820,721 records and writes 211,332 repeats, $\Psi/N = 20.5\%$. Of DROP's repeats, 13,587 are a second row with the same tick and the same value. They occur where a late

packet with an earlier tick interrupts the packets of a tick: DROP closes the row of the interrupted tick and opens a new one when the tick returns (DROP.py, lines 235–237), while DatCon discards late packets. No tick repeats among the accepted reference records. For the receiver record the repetition is extreme. DatCon wrote 1,509,330 rows carrying a receiver position, of which 25,021 reproduce one of the 25,022 receiver records (they hold 19,220 distinct coordinate pairs, since a receiver at rest repeats its position) and the other 98.3% repeat an earlier value. The one receiver record DatCon does not write is the last tick of FLY021: DatCon emits the rows of a tick only when it reads the first packet of the next tick, and after the last tick of a file there is none. DROP does not decode the receiver record at all.

Row counts therefore measure the way a tool lays out its table, and they depend on the sample rate chosen by the user (Table 5). At DatCon's default of 30 Hz the navigation rows fall to 113,455 and their repeat share to 1.9%, because most ticks carrying a navigation record are no longer written: only 12.6% of reference records appear at their own key. The receiver record, logged far more slowly, keeps a repeat share of 97.9%. At either rate $U = 0$ and the tools agree on every shared key, and the latch of equation (18) holds; all 22 pairs at 30 Hz were repeatable.

Table 5 – DatCon on the 22 DJI files at 600 Hz (maximum, used in the main runs) and at 30 Hz (DatCon default)

Quantity	600 Hz	30 Hz
Navigation position rows N	1,199,316	113,455
Fresh $\Phi = \#V$ / repeats Ψ	793,444 / 405,872	111,342 / 2,113
Unsupported U	0	0
Ψ/N , navigation record	33.8%	1.9%
Recall ϕ at own key	89.6%	12.6%
Receiver rows N / Ψ/N	1,509,330 / 98.3%	143,022 / 97.9%
Shared keys with DROP / disagreements	905,580 / 0	103,004 / 0
Both / DatCon only / DROP only / neither	729,793 / 63,651 / 90,928 / 1,471	102,439 / 8,903 / 718,282 / 56,219
Repeatable pairs	22/22	22/22

The records shown by one tool only follow the rules of equations (17) and (18). Of the 885,843 reference navigation records, both tools show 729,793 (82.4%), DatCon alone shows 63,651, DROP alone shows 90,928, and 1,471 are shown by neither. Rule of equation (17), applied in degrees, predicts the 820,721 records DROP shows without a single exception. Every record DatCon omits but one, 92,398 of 92,399, lies before the latch of equation (18), and the latch predicts DatCon's first position row in 13 of 13 files; a latch on latitude and longitude alone predicts it in none. The one exception is the last tick of FLY012, for the reason given above. All 63,651 records that only DatCon shows were logged with two or fewer satellites, with a median of zero: they are positions the navigation filter produced without GPS aiding. The records that only DROP shows are positions logged before take-off. Table 4 gives the per-file figures and Figure 2 shows the decomposition.

Table 4 – DJI Phantom 3 files with position records: reference navigation records, fresh and repeated rows (Φ/Ψ , DatCon at 600 Hz), and the records each tool makes visible. Φ equals the number of records reproduced; unsupported rows (U) are zero throughout

File	GPS date	Reference	DatCon Φ/Ψ	DROP Φ/Ψ	Both	DatCon only	DROP only
FLY005	2018-06-07	0	0/0	0/0	0	0	0
FLY006	2018-06-07	1,115	0/0	915/222	0	0	915
FLY007	2018-06-07	15,988	11,329/5,952	14,948/3,753	10,400	929	4,548
FLY008	2018-06-19	287,880	279,805/142,162	226,118/57,608	218,043	61,762	8,075
FLY009	2018-06-20	7,073	0/0	7,073/1,834	0	0	7,073
FLY010	2018-06-20	0	0/0	0/0	0	0	0
FLY011	2018-06-20	13,953	13,143/6,671	13,953/3,658	13,143	0	810

File	GPS date	Reference	DatCon Φ/Ψ	DROP Φ/Ψ	Both	DatCon only	DROP only
FLY012	2018-06-20	21,598	5,140/2,685	21,598/5,827	5,140	0	16,458
FLY013	2018-06-20	56,416	50,719/25,950	56,416/14,623	50,719	0	5,697
FLY015	2018-06-20	75,787	72,399/37,503	75,787/19,960	72,399	0	3,388
FLY016	2018-06-21	16,092	732/368	16,092/4,031	732	0	15,360
FLY017	2018-06-21	185,649	175,292/89,645	185,609/47,754	175,292	0	10,317
FLY018	2018-06-21	16,936	0/0	16,896/4,231	0	0	16,896
FLY019	2018-06-21	187,052	184,885/94,936	185,012/47,756	183,925	960	1,087
FLY020	2018-06-21	304	0/0	304/75	0	0	304
FLY021	2018-06-21	0	0/0	0/0	0	0	0
Total		885,843	793,444/405,872	820,721/211,332	729,793	63,651	90,928

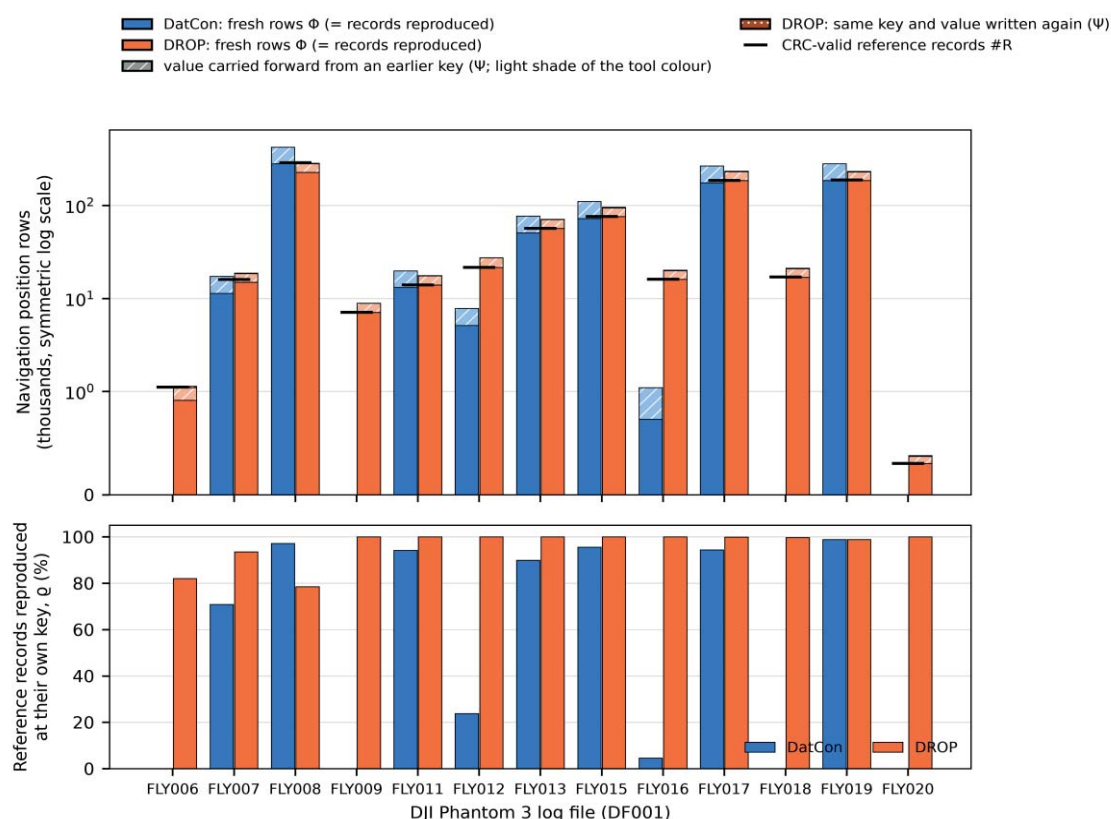


Figure 2 – Navigation position rows of DatCon (600 Hz) and DROP split into fresh and repeated rows, against the number of CRC-valid reference records (top), and the share of reference records each tool reproduces at its own key (bottom)

DRDP wrote a header and nothing else for every Phantom 3 file it finished, which places those runs in the empty class; 15 of the files contain position records. Its source explains the result. DRDP does recognise V1 files, since it tests the same BUILD marker at offset 16 that DatCon uses and verifies the same checksum, so the statement in the preprint that the marker is absent from V1 files was wrong. It extracts positions only from record type 2096, which the Mavic 2 Pro uses and the Phantom 3 does not. On FLY021, whose last byte is 0x55, DRDP does not terminate: reading the length byte raises an index error, and the recovery routine looks for the next 0x55 from the same offset and returns it again. All three repetitions reached the one-hour timeout, the only run of the corpus in the timeout class, which also explains the timeout recorded in the preprint.

Dates and coordinates place 13 of the 16 DJI files with positions on the documented flight days of 19 to 21 June 2018, with between 81 and 100 per cent of sampled navigation positions inside the boundary polygon and the remainder within 75 m of it. FLY005 to FLY007 were

recorded on 7 June 2018. The navigation positions of FLY006 and FLY007 lie about 94 km east of the documented area, near Broomfield in Colorado. FLY005 holds no navigation position, and its 758 receiver positions lie between 1,330 and 3,804 km away along the great circle, at latitudes between 51 and 70 degrees north. DatCon decodes those values correctly and writes them into 45,477 rows. They pass a range test and they are not places the aircraft was.

The 19 DataFlash binaries hold 2,306 GPS messages, of which 927 carry a non-zero position across 13 files. Of those, 545 were logged with a status below three, that is without a three-dimensional fix. The 97 positions of DF056 all have a fix and all lie inside the NIST polygon. None of the 830 DF057 positions does. In two logs every position, 176 records, has status 1, zero satellites and the coordinates 34.94 N, 159.91 W, in the Pacific Ocean about 4,701 km from the documented area; the remaining DF057 positions sit about 94 km from it. The data sheet of DF057 documents flights on 20 June 2018 inside the polygon, and none of the 15 logs records a position there.

pymavlink verified every position in the four DF056 logs and in one DF057 log. On the other 14 DF057 binaries it raised an exception while opening and indexing the file, before returning a single message: twelve times a struct error on a record shorter than its declared length, twice a complaint about an invalid length in a format message. 8 of those files contain 688 positions that the library cannot reach. GRYPHON, which is built on pymavlink, behaves the same way. Because the tool reports the failure, these runs are errors and not silent failures. Both tools parse most of the Mission Planner exports of the same logs, so an examiner working from the raw evidence would obtain nothing for those 14 logs while the ground-station export of the same flight parses without trouble. On the DJI files both tools exit normally with no messages, and GRYPHON prints "No GPS signal loss" and "No Alt Anomaly Detected" for all 22 of them.

Outputs were stable. Of the 432 tool–file pairs, 431 completed, and for all 431 of them the exit status and the SHA-256 of every output file were identical across the three runs, so Δ of equation (25) equals one and the outcome classes do not change between repetitions. The remaining pair, DRDP on FLY021, timed out three times and is reported as a timeout, not as a repeatable result. Counts also reproduce across operating systems: the row counts of DatCon equal those released with the preprint for all 22 files, and those of DROP and Fodogu for 20 of 22, the two exceptions being files whose results the preprint had overwritten through a name collision with the DF005 dataset. For pymavlink the comparison covers the second batch of the preprint, 35 files (4 DataFlash binaries, 15 Mission Planner exports and 16 Yuneec Sensor files): message counts equal the released values for 20 of them, and the other 15 are files on which it raised an exception in both environments.

Discussion

Our earlier preprint measured the parser and not the evidence. A row count is fixed by the output schema and by the sample rate the user picks. A count of rows with coordinates is inflated by repetition, and it also depends on which of several coordinate columns a script happens to pick. A parsed flag reports only that a process exited. Read at the level of records, the same data says something else: the two DJI parsers do not recover different evidence. They pass the same decoded records through different filters, which is what equations (21) and (22) predict once $U = 0$ and the outputs agree on shared keys.

Neither rule is written down for the user, and neither is harmless. Dropping positions logged with fewer than three satellites is a defensible choice, but on a session that never reached three satellites DROP reports nothing at all, and the examiner sees an empty column instead of a warning. DatCon suppresses everything before its latch, which can remove the positions recorded while the aircraft sat on the ground, and after the latch it prints positions computed without GPS aiding without marking them. The ArduPilot results show the same problem from the other side. A range test accepted 545 positions logged without a fix, including a cluster in the Pacific. Those records were decoded correctly. They are what the flight controller wrote, and they are not locations of the aircraft.

We therefore withdraw the recommendation of the preprint that examiners merge the outputs of DatCon and DROP with conflict resolution based on a coordinate bounding box. There is nothing to resolve, since the tools agree wherever they overlap, and a row-level merge adds repeated values together with records that one tool hides on purpose. A better procedure is to decode once at record level and keep the record type, the key and the satellite or fix status with every position, then filter explicitly. A position recovered from a log should be reported with the record it came from.

The failures that would cost an examiner something are not the ones such a benchmark reports. A parser that never terminates because a file ends with one particular byte. A library that refuses a raw log while reading the vendor export of the same flight. Parsers that finish normally and return nothing from files that hold positions: 21 of 126 runs within the tools' declared scope and 61 of 120 outside it. The second figure is less a defect of the tools than of the practice of feeding a file to whatever parser is at hand, since a tool given a format it does not know should say so and most of these do not. A table of row counts shows none of that, while the class of equation (15) and the indicator of equation (16) cost nothing to report and say what happened.

The DJI results hold for one format generation. The V1 on-board DAT of the Phantom 3 is an old format and its payload is only masked with the tick number, not encrypted, which is why an independent reference decoder could be written from public knowledge at all. Later DJI aircraft encrypt their logs, keep flight records in the mobile application or in the vendor cloud [25], and tools such as Fodogu carry model-specific decryption [10]. For those formats the same model applies, but a reference decoding depends on keys and code that are harder to verify, and the agreement found here should not be assumed.

Compared with the commercial tool assessments of Salamh and co-authors [15], the present study covers fewer tools and one ecosystem in depth, and it adds a known answer against which output can be checked. Compared with the practice of NIST CFTT [17] and the argument of Horsman [18], it is a small instance of the same idea: define the expected result first, then measure the deviation. We do not claim that any tool tested here is validated for casework in the sense of ISO/IEC 17025 [20]. Such a claim needs written requirements for an intended use, test data beyond one corpus and an estimate of the error rate.

Four limitations should be kept in mind. The reference decoder for the DJI format was written from the same public knowledge of the format that the tools embody, so a shared misunderstanding of a field would escape it; the checksum guarantees only that a reference record is what the aircraft stored. Neither the corpus nor this study provides an independent trajectory, so positional accuracy is not assessed. Five of the six tools needed a headless entry point, and although each calls the tool's own function, the sample rate chosen for DatCon changes its row counts and recall (Table 5) but not its agreement with DROP or its display rule. Finally, only one DJI format and four datasets were re-examined, and the Yuneec Sensor format has no public decoder, so for those files the result says only that no tool extracted a position; whether the files hold positions could not be established.

Conclusion

Measured record by record, the open-source DJI parsers tested here decode position data correctly. Every position they wrote corresponds to a packet whose checksum verifies, and where their outputs overlap they agree exactly. The large differences between their outputs, which we previously read as complementary recovery, come from repeated rows, from the sample rate, and from two undocumented display rules, one that hides positions before take-off and one that hides positions logged with few satellites.

What should worry an examiner sits elsewhere. Parsers exit normally and return nothing from files that hold positions, including files in formats they claim to read. One parser does not terminate when a file ends with a particular byte. A reference library refuses raw logs whose vendor exports it reads without difficulty. Coordinates pass a range test although the receiver never had a fix. None of this appears in a count of rows.

The model of equations (1)–(26) is what makes the difference reportable. Its quantities, the three counts of equation (12), the support rate and recall of equation (14), the classes of equation (15), the silent-failure indicator of equation (16) and the repeatability indicator of equation (25), can be computed for any parser and any format for which a reference decoding exists, and we suggest them as the minimum content of a parser evaluation.

Data and code availability

The corpus is available from NIST CFReDS [21]. An anonymised artefact archive accompanies this submission as supplementary material: both reference decoders, the tool wrappers and the headless DatCon driver, the protocol, the records of all 1,296 runs and of the 66 additional DatCon runs with the SHA-256 of every output, and the scripts that produce every number, table and figure of this paper from those records.

References

- [1]. Studiawan, H., Grispos, G., & Choo, K.-K. R. (2023). Unmanned aerial vehicle (UAV) forensics: The good, the bad, and the unaddressed. *Computers & Security*, 132, 103340. <https://doi.org/10.1016/j.cose.2023.103340>
- [2]. Thantilage, R. D., Buttner, G., & Genoe, R. (2025). Drone forensics in law enforcement: Assessing utilization, challenges, and emerging necessities. *Forensic Science International: Digital Investigation*, 55, 302003. <https://doi.org/10.1016/j.fsidi.2025.302003>
- [3]. Studiawan, H., & Choo, K.-K. R. (Eds.). (2026). *Drone and UAV forensics*. Springer. <https://doi.org/10.1007/978-3-031-93511-4>
- [4]. Sihag, V., Choudhary, G., Choudhary, P., & Dragoni, N. (2023). Cyber4Drone: A systematic review of cyber security and forensics in next-generation drones. *Drones*, 7(7), 430. <https://doi.org/10.3390/drones7070430>
- [5]. Horsman, G. (2016). Unmanned aerial vehicles: A preliminary analysis of forensic challenges. *Digital Investigation*, 16, 1–11. <https://doi.org/10.1016/j.diin.2015.11.002>
- [6]. Bouafif, H., Kamoun, F., Iqbal, F., & Marrington, A. (2018). Drone forensics: Challenges and new insights. In 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). IEEE. <https://doi.org/10.1109/NTMS.2018.8328747>
- [7]. Clark, D. R., Meffert, C., Baggili, I., & Breitingner, F. (2017). DROP (DRone Open source Parser) your drone: Forensic analysis of the DJI Phantom III. *Digital Investigation*, 22, S3–S14. <https://doi.org/10.1016/j.diin.2017.06.013>
- [8]. Walker, B. (2019). DatCon: A converter for DJI flight logs (Version 3.5.0) [Computer software]. <https://github.com/BudWalkerJava/DatCon>
- [9]. Lee, S., Seo, H., & Kim, D. (2023). Digital forensic research for analyzing drone pilot: Focusing on DJI remote controller. *Sensors*, 23(21), 8934. <https://doi.org/10.3390/s23218934>
- [10]. FOSEGU. (2024). Fodogu: DJI drone forensic tool [Computer software]. <https://github.com/FOSEGU/Fodogu>
- [11]. Zhao, Z., Wang, Y., & Liao, G. (2024). Digital forensic research for analyzing drone and mobile device: Focusing on DJI Mavic 2 Pro. *Drones*, 8(7), 281. <https://doi.org/10.3390/drones8070281>
- [12]. Zhao, Z. (2025). DRDP: Drone data parser [Computer software]. <https://github.com/EveZzy/DRDP>
- [13]. Mantas, E., & Patsakis, C. (2019). GRYPHON: Drone forensics in dataflash and telemetry logs. In N. Attrapadung & T. Yagi (Eds.), *Advances in Information and Computer Security (IWSEC 2019)*, LNCS 11689 (pp. 377–390). Springer. https://doi.org/10.1007/978-3-030-26834-3_22
- [14]. ArduPilot Development Team. (2024). pymavlink: A Python MAVLink library (Version 2.4.49) [Computer software]. <https://github.com/ArduPilot/pymavlink>

- [15]. Salamh, F. E., Mirza, M. M., & Karabiyik, U. (2021). UAV forensic analysis and software tools assessment: DJI Phantom 4 and Matrice 210 as case studies. *Electronics*, 10(6), 733. <https://doi.org/10.3390/electronics10060733>
- [16]. Baibussinov, A., Shayakhmetov, M., Rzayeva, L., & Yesbergenov, K. (2026). A hardware-software complex for the reconstruction of unmanned aerial vehicle digital traces under logical data damage using LSTM-based telemetry recovery and multi-source confidence scoring. *Journal of Cybersecurity and Privacy*, 6(4), 123. <https://doi.org/10.3390/jcp6040123>
- [17]. National Institute of Standards and Technology. (2023). Computer Forensics Tool Testing Program (CFTT). <https://www.nist.gov/itl/ssd/software-quality-group/computer-forensics-tool-testing-program-cftt>
- [18]. Horsman, G. (2019). Tool testing and reliability issues in the field of digital forensics. *Digital Investigation*, 28, 163–175. <https://doi.org/10.1016/j.diin.2019.01.009>
- [19]. International Organization for Standardization. (2012). ISO/IEC 27037:2012 Information technology — Security techniques — Guidelines for identification, collection, acquisition and preservation of digital evidence. ISO.
- [20]. International Organization for Standardization. (2017). ISO/IEC 17025:2017 General requirements for the competence of testing and calibration laboratories. ISO.
- [21]. National Institute of Standards and Technology. (2018). Computer Forensic Reference Data Sets: Drone images. <https://cfreds.nist.gov/>
- [22]. Metz, J. (2024). libewf: Library and tools to access the Expert Witness Compression Format (EWF) [Computer software]. <https://github.com/libyal/libewf>
- [23]. Carrier, B. (2024). The Sleuth Kit [Computer software]. <https://www.sleuthkit.org/sleuthkit/>
- [24]. Agresti, A. (2019). *An introduction to categorical data analysis* (3rd ed.). John Wiley & Sons. ISBN 978-1-119-40526-9
- [25]. Kim, H., We, D., An, W., & Park, M. (2026). Correlation analysis of pilots and drones using DJI cloud forensic data. *Forensic Science International: Digital Investigation*, 57, 302101. <https://doi.org/10.1016/j.fsidi.2026.302101>