DroneDetect V2 - General Data Exploration

Dataset: DroneDetect V2 (Swinney & Woods, 2021)
Paper: The Effect of Real-World Interference on CNN Feature Extraction and ML Classification of UAS
Dataset: IEEE DataPort - DroneDetect V2

This notebook analyzes the dataset structure and metadata:

- File inventory and metadata validation
- Class distribution analysis (drones, states, interference)
- Identification of missing combinations
- Data quality validation

> **Note**: For detailed data collection methodology (hardware, flight conditions, drone specs), see docs/methodology.md

DroneDetect V1 vs V2: Key Differences

The original paper describes V1 with 500 samples per class across 4 interference conditions (CLEAN, WIFI, BLUETOOTH, BOTH).

DroneDetect V2, publicly available on IEEE DataPort, differs: it contains **5 replicas** (index 0-4) per valid combination of drone/state/interference.

Only **CLEAN** and **BOTH** interference conditions were retained for simplicity.

The result is:

- **195 recording files** (2-second raw IQ each)
- **Expected combinations**: 7 drones x 3 states x 2 interferences = 42
- **Actual combinations**: 39 (3 missing)
- **Replicas per combination**: 5 (files indexed 0-4)

This notebook analyzes a reduced version of DroneDetect V2 dataset.

1. Setup

In [1]:

```python
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

import numpy as np
import pandas as pd
from pathlib import Path
from itertools import product
import re
import ipynbname

# Import local modules
from dronedetect import config, data_loader

# Constants
# SAMPLE_STRIDE: Downsampling factor for memory-efficient analysis
# - Physical: Reduces 120M samples to ~1M samples (120M / 120 = 1M)
# - Impact: Reduces sampling rate from 60 MHz to 500 kHz (60 MHz / 120 = 0.5 MHz)
# - Temporal resolution: Increases sample interval from 16.67 ns to 2 µs
# - Use case: Sufficient for amplitude/power statistics while reducing memory footprint 120x
# - Limitation: May miss fast transients < 2 µs; not suitable for high-frequency spectral analysis
SAMPLE_STRIDE = 120

print("Setup complete!")
```

```
Setup complete!
```

In [2]:

```python
# Setup figure saving
NOTEBOOK_NAME = ipynbname.name()
FIGURES_DIR = Path("../figures") / NOTEBOOK_NAME

def save_figure(fig) -> None:
    """Save plotly figure to PNG file using the figure's title as filename."""
    FIGURES_DIR.mkdir(parents=True, exist_ok=True)
    title = fig.layout.title.text if fig.layout.title.text else "untitled"
```

```python
    filename = re.sub(r'[^\w\s-]', '', title).strip()
    filename = re.sub(r'[\s-]+', '_', filename)
    filepath = FIGURES_DIR / f"{filename}.png"
    try:
        fig.write_image(str(filepath), width=1200, height=800)
        print(f"Saved: {filepath}")
    except Exception as e:
        print(f"Warning: Could not save figure (kaleido required): {e}")
```

2. Load Sample File

Verify dataset access and inspect raw IQ data format.

In [3]:
```python
# Load a sample file to verify dataset access
sample_file = config.DATA_DIR / "CLEAN" / "AIR_ON" / "AIR_0000_00.dat"

if sample_file.exists():
    iq_data = data_loader.load_raw_iq(sample_file)

    print(f"""Loading: {sample_file}

=== IQ Data Inspection ===
Shape: {iq_data.shape}
Dtype: {iq_data.dtype}
Total samples: {len(iq_data):,}
Duration: {len(iq_data) / config.FS:.2f} seconds

=== Value Range ===
Real: [{iq_data.real.min():.4f}, {iq_data.real.max():.4f}]
Imag: [{iq_data.imag.min():.4f}, {iq_data.imag.max():.4f}]
NaN count: {np.isnan(iq_data).sum()}

=== Data Format Explanation ===
File storage: {config.RAW_SAMPLE_COUNT} float32 values (interleaved I/Q)
Memory representation: {config.COMPLEX_SAMPLE_COUNT} complex64 samples
Transformation: np.fromfile(dtype=float32, count=240M).view(complex64)
  -> Each complex64 = 2 consecutive float32 (I + jQ)

Sampling rate: {config.FS / 1e6:.0f} MHz
```

```
Duration: {len(iq_data) / config.FS:.2f} seconds
Total time-domain samples: {len(iq_data):,} complex""")
else:
    print(f"""File not found: {sample_file}
Please verify the dataset path in config.py""")
```

```
Loading: /home/sambot/win_downloads/DATASETS/drones/DroneDetect_V2/CLEAN/AIR_ON/AIR_0000_00.dat

=== IQ Data Inspection ===
Shape: (120000000,)
Dtype: complex64
Total samples: 120,000,000
Duration: 2.00 seconds

=== Value Range ===
Real: [-0.9907, 0.9951]
Imag: [-0.9893, 0.9941]
NaN count: 0

=== Data Format Explanation ===
File storage: 240000000 float32 values (interleaved I/Q)
Memory representation: 120000000 complex64 samples
Transformation: np.fromfile(dtype=float32, count=240M).view(complex64)
  -> Each complex64 = 2 consecutive float32 (I + jQ)

Sampling rate: 60 MHz
Duration: 2.00 seconds
Total time-domain samples: 120,000,000 complex
```

2.1 Signal Amplitude Analysis by Drone (CLEAN/ON only)

In [4]:

```
# Load one file per drone (CLEAN/ON state)
drone_codes = ['AIR', 'DIS', 'INS', 'MA1', 'MAV', 'MIN', 'PHA']
amplitude_stats = []

for drone in drone_codes:
    file_path = config.DATA_DIR / "BOTH" / f"{drone}_FY" / f"{drone}_1110_00.dat"
    if file_path.exists():
        iq = data_loader.load_raw_iq(file_path)
        # Sample 1M points for stats (faster)
```

```
        sample = iq[::SAMPLE_STRIDE]  # 1M samples
        amplitude = np.abs(sample)

        amplitude_stats.append({
            'drone': drone,
            'mean_amplitude': amplitude.mean(),
            'std_amplitude': amplitude.std(),
            'min': amplitude.min(),
            'max': amplitude.max(),
            'power_dbm': 10 * np.log10(np.mean(np.abs(sample)**2))
        })

# Visualize
stats_df = pd.DataFrame(amplitude_stats)
print("""Signal amplitude statistics by drone (BOTH/FY):""")
print(stats_df.to_string(index=False))

fig = px.bar(stats_df, x='drone', y='mean_amplitude', error_y='std_amplitude',
             title='Mean Signal Amplitude by Drone (BOTH/FY)')
fig.update_yaxes(title_text="Mean Amplitude")
save_figure(fig)
fig.show()
```

```
Signal amplitude statistics by drone (BOTH/FY):
drone  mean_amplitude  std_amplitude      min      max  power_dbm
  AIR        0.180474       0.229678 0.001381 1.443216 -10.689342
  DIS        0.159872       0.210283 0.000000 1.443216 -11.562829
  INS        0.012852       0.013012 0.000000 0.268899 -34.756264
  MIN        0.008942       0.006448 0.000000 0.120909 -39.152660
  PHA        0.029694       0.030579 0.000000 0.841820 -27.406900
Saved: ../figures/001_exploration_general/Mean_Signal_Amplitude_by_Drone_BOTHFY.png
```

3. Dataset Statistics

In [5]:
```
if config.DATA_DIR.exists():
    df = data_loader.get_cached_metadata(force_refresh=True)
    drone_counts = df['drone_code'].value_counts()
    interference_counts = df['interference'].value_counts()
    state_counts = df['state'].value_counts()
    print(f"Loaded {len(df)} files from cache: {config.METADATA_CACHE}")
```

```python
else:
    df = None
    drone_counts = interference_counts = state_counts = None

# Note: complete_df will be created in section 3.2 for combination-level analysis
print(f"""
============================================================
METADATA LOADED
============================================================
Shape: {df.shape}
Columns: {df.columns.tolist()}
Purpose: One row per .dat file with full metadata (file_path, drone_code, state, interference, index)
""")
```

```
Loaded 195 files from cache: data/metadata_cache.parquet

============================================================
METADATA LOADED
============================================================
Shape: (195, 9)
Columns: ['drone_code', 'drone_folder', 'wifi', 'bluetooth', 'interference', 'state', 'index', 'file_path', 'interference_folde
r']
Purpose: One row per .dat file with full metadata (file_path, drone_code, state, interference, index)
```

In [6]:

```python
# Display basic metadata information
print(f"""============================================================
DATASET OVERVIEW
============================================================
Total files: {len(df)}

Drone codes: {sorted(df['drone_code'].unique())}
States: {sorted(df['state'].unique())}
Interference types: {sorted(df['interference'].unique())}

============================================================
DISTRIBUTION COUNTS
============================================================

By Drone:
{drone_counts.to_dict()}
```

```
By State:
{state_counts.to_dict()}

By Interference:
{interference_counts.to_dict()}""")
```

```
============================================================
DATASET OVERVIEW
============================================================
Total files: 195

Drone codes: ['AIR', 'DIS', 'INS', 'MA1', 'MAV', 'MIN', 'PHA']
States: ['FY', 'HO', 'ON']
Interference types: ['BOTH', 'CLEAN']

============================================================
DISTRIBUTION COUNTS
============================================================

By Drone:
{'AIR': 30, 'INS': 30, 'MIN': 30, 'MAV': 30, 'MA1': 30, 'PHA': 25, 'DIS': 20}

By State:
{'ON': 70, 'FY': 65, 'HO': 60}

By Interference:
{'BOTH': 100, 'CLEAN': 95}
```

3.1 Class Distributions

In [7]:

```
# Visualize class distributions
fig = make_subplots(
    rows=1, cols=3,
    subplot_titles=('Drone Distribution', 'Interference Distribution', 'State Distribution')
)

fig.add_trace(
    go.Bar(x=drone_counts.index, y=drone_counts.values, name='Drone', marker_color='steelblue'),
    row=1, col=1
```

```python
)
fig.add_trace(
    go.Bar(x=interference_counts.index, y=interference_counts.values, name='Interference', marker_color='coral'),
    row=1, col=2
)
fig.add_trace(
    go.Bar(x=state_counts.index, y=state_counts.values, name='State', marker_color='seagreen'),
    row=1, col=3
)

fig.update_xaxes(title_text="Drone Code", row=1, col=1)
fig.update_xaxes(title_text="Interference Type", row=1, col=2)
fig.update_xaxes(title_text="State", row=1, col=3)
fig.update_yaxes(title_text="Count", row=1, col=1)

fig.update_layout(title="Class Distributions", height=400, showlegend=False)
save_figure(fig)
fig.show()
```

```
Saved: ../figures/001_exploration_general/Class_Distributions.png
```

3.1.1 File Distribution: CLEAN vs BOTH

In [8]:

```python
# Compare CLEAN vs BOTH distribution
interference_comparison = df.groupby(['drone_code', 'interference']).size().reset_index(name='count')

fig = px.bar(interference_comparison, x='drone_code', y='count', color='interference',
             barmode='group', title='File Count by Drone and Interference Type')
fig.update_yaxes(title_text="File Count")
save_figure(fig)
fig.show()

print(f"""
=== Summary ===
CLEAN files: {len(df[df['interference'] == 'CLEAN'])}
BOTH files: {len(df[df['interference'] == 'BOTH'])}

Note: Slight imbalance due to missing combinations (DIS/HO/*, PHA/FY/CLEAN)""")
```

```
Saved: ../figures/001_exploration_general/File_Count_by_Drone_and_Interference_Type.png
```

```
=== Summary ===
CLEAN files: 95
BOTH files: 100

Note: Slight imbalance due to missing combinations (DIS/HO/*, PHA/FY/CLEAN)
```

3.2 Missing Data Combinations

In [9]:

```python
# CREATE complete_df: combination-level DataFrame
# Purpose: Analyze drone/state/interference combinations with file counts
# Structure: One row per combination (not per file like df)

# Identify missing combinations of drone/state/interference
# Count files per combination
pivot_data = df.groupby(['drone_code', 'state', 'interference']).size().reset_index(name='count')

# Get unique values from actual data
drones = sorted(df['drone_code'].unique())
states = sorted(df['state'].unique())
interferences = sorted(df['interference'].unique())

# Generate all expected combinations
all_combinations = list(product(drones, states, interferences))
expected_df = pd.DataFrame(all_combinations, columns=['drone_code', 'state', 'interference'])

# Merge with actual data
complete_df = expected_df.merge(pivot_data, on=['drone_code', 'state', 'interference'], how='left')
complete_df['count'] = complete_df['count'].fillna(0).astype(int)

# Identify missing combinations
missing_combinations = complete_df[complete_df['count'] == 0]

print(f"""
============================================================
complete_df CREATED
============================================================
Shape: {complete_df.shape}
Columns: {complete_df.columns.tolist()}
Purpose: One row per drone/state/interference combination with file count
```

```
  - Combinations with files: {len(complete_df[complete_df['count'] > 0])}
  - Missing combinations: {len(complete_df[complete_df['count'] == 0])}""")
```

```
============================================================
complete_df CREATED
============================================================
Shape: (42, 4)
Columns: ['drone_code', 'state', 'interference', 'count']
Purpose: One row per drone/state/interference combination with file count
  - Combinations with files: 39
  - Missing combinations: 3
```

In [10]:

```python
# Visualize heatmaps for each state with consistent color scale
# Define consistent color scale range for all heatmaps (0 to 5 files per combination)
color_min = 0
color_max = 5

for state in states:
    state_df = complete_df[complete_df['state'] == state]
    pivot_table = state_df.pivot(index='drone_code', columns='interference', values='count')

    fig = px.imshow(
        pivot_table,
        labels=dict(x="Interference", y="Drone Code", color="File Count"),
        title=f"File Count Heatmap - State {state}",
        color_continuous_scale='RdYlGn',
        zmin=color_min,
        zmax=color_max,
        text_auto=True,
        aspect="auto"
    )
    save_figure(fig)
    fig.show()

# Summary
missing_summary = "\n".join([f"  - {row['drone_code']} / {row['state']} / {row['interference']}"
                             for _, row in missing_combinations.iterrows()]) if len(missing_combinations) > 0 else ""

print(f"""
==================================================
MISSING COMBINATIONS SUMMARY
```

```
==================================================
Expected: {len(all_combinations)} combinations ({len(drones)} drones x {len(states)} states x {len(interferences)} interferenc
Found: {len(pivot_data)} unique combinations
Missing: {len(missing_combinations)} combinations

Missing combinations:
{missing_summary}""")
```

```
Saved: ../figures/001_exploration_general/File_Count_Heatmap_State_FY.png
Saved: ../figures/001_exploration_general/File_Count_Heatmap_State_HO.png
Saved: ../figures/001_exploration_general/File_Count_Heatmap_State_ON.png
==================================================
MISSING COMBINATIONS SUMMARY
==================================================
Expected: 42 combinations (7 drones x 3 states x 2 interferences)
Found: 39 unique combinations
Missing: 3 combinations

Missing combinations:
  - DIS / HO / BOTH
  - DIS / HO / CLEAN
  - PHA / FY / CLEAN
```

Note on Paper Documentation

> **Note**: The DrondetectV1 paper does not discuss:
>
> - Why certain combinations are missing (DIS/HO, PHA/FY/CLEAN)
> - IQ value normalization or out-of-range handling
> - File duration variations
>
> The paper focuses on interference impact on CNN classification, not dataset quality details.

3.3 Data Quality Validation

This section validates the integrity and consistency of the DroneDetect V2 dataset:

- **File integrity**: Verify sample counts, duration, and IQ value ranges

- **Metadata consistency**: Ensure filenames match directory structure
- **Index distribution**: Confirm all combinations have exactly 5 replicas
- **IQ distributions**: Analyze signal characteristics and statistical properties

3.3.1 File Integrity

In [11]:
```
# Validate file integrity and consistency for ALL files
print("""============================================================
FILE INTEGRITY VALIDATION (ALL FILES)
============================================================""")

# Expected sample count per file
EXPECTED_SAMPLES = 120_000_000  # 2 seconds at 60 MHz
BATCH_SIZE = 5

# Check all files
corrupted_files = []
duration_issues = []
sample_count_issues = []
range_violations = []

# Validate ALL files
total_files = len(df)

for idx, row in df.iterrows():
    if (idx + 1) % BATCH_SIZE == 0:
        print(f"  Validating file {idx + 1}/{total_files}...")
    try:
        iq = data_loader.load_raw_iq(row['file_path'])
        sample_count = len(iq)
        i_min, i_max = iq.real.min(), iq.real.max()
        q_min, q_max = iq.imag.min(), iq.imag.max()
        duration = sample_count / config.FS
        del iq

        # Check sample count (should be exactly 120M)
        if sample_count != EXPECTED_SAMPLES:
            sample_count_issues.append((row['file_path'], sample_count))
```

```
        # Check duration (should be ~2 seconds)
        if abs(duration - 2.0) > 0.01:
            duration_issues.append((row['file_path'], duration))

        # Validate IQ value ranges (must be in [-1, 1] for normalized RF signals)
        if i_min < -1.0 or i_max > 1.0 or q_min < -1.0 or q_max > 1.0:
            range_violations.append({
                'file': row['file_path'],
                'i_range': (i_min, i_max),
                'q_range': (q_min, q_max)
            })

    except Exception as e:
        corrupted_files.append((row['file_path'], str(e)))

print(f"""
Validated {total_files} files (100%)

=== Results ===
Corrupted files: {len(corrupted_files)}
Sample count issues (!= {EXPECTED_SAMPLES:,}): {len(sample_count_issues)}
Duration issues (!= 2.0s): {len(duration_issues)}""")

if corrupted_files:
    corrupted_list = "\n".join([f"  - {f}: {err}" for f, err in corrupted_files])
    print(f"\nCorrupted files:\n{corrupted_list}")

if sample_count_issues:
    sample_issues_list = "\n".join([f"  - {f}: {count:,} samples" for f, count in sample_count_issues])
    print(f"\nSample count issues:\n{sample_issues_list}")

if duration_issues:
    duration_list = "\n".join([f"  - {f}: {dur:.3f}s" for f, dur in duration_issues])
    print(f"""\nDuration issues:
{duration_list}

⚠️ Note: Files with duration < 2s will be handled in the preprocessing notebook.
   Options: padding, exclusion, or variable-length windowing.""")

if range_violations:
    violations_list = "\n".join([f"  - {Path(v['file']).name}\n    I: [{v['i_range'][0]:.4f}, {v['i_range'][1]:.4f}]\n    Q: [
```

```python
                                    for v in range_violations[:10]])
    more_msg = f"  ... and {len(range_violations) - 10} more files" if len(range_violations) > 10 else ""
    print(f"""\nFiles with IQ values outside [-1, 1]: {len(range_violations)}

Files violating [-1, 1] range:
{violations_list}
{more_msg}""")
else:
    print("-> All files have IQ values within expected range [-1, 1]")


# Store validation results for conclusion
validation_results = {
    'total_files': total_files,
    'corrupted': len(corrupted_files),
    'sample_issues': len(sample_count_issues),
    'duration_issues': len(duration_issues)
}
```

```
============================================================
FILE INTEGRITY VALIDATION (ALL FILES)
============================================================
  Validating file 5/195...
  Validating file 10/195...
  Validating file 15/195...
  Validating file 20/195...
  Validating file 25/195...
  Validating file 30/195...
  Validating file 35/195...
  Validating file 40/195...
  Validating file 45/195...
  Validating file 50/195...
  Validating file 55/195...
  Validating file 60/195...
  Validating file 65/195...
  Validating file 70/195...
  Validating file 75/195...
  Validating file 80/195...
  Validating file 85/195...
  Validating file 90/195...
  Validating file 95/195...
  Validating file 100/195...
  Validating file 105/195...
  Validating file 110/195...
  Validating file 115/195...
  Validating file 120/195...
  Validating file 125/195...
  Validating file 130/195...
  Validating file 135/195...
  Validating file 140/195...
  Validating file 145/195...
  Validating file 150/195...
  Validating file 155/195...
  Validating file 160/195...
  Validating file 165/195...
  Validating file 170/195...
  Validating file 175/195...
  Validating file 180/195...
  Validating file 185/195...
  Validating file 190/195...
```

```
  Validating file 195/195...

Validated 195 files (100%)

=== Results ===
Corrupted files: 0
Sample count issues (!= 120,000,000): 2
Duration issues (!= 2.0s): 2

Sample count issues:
  - /home/sambot/win_downloads/DATASETS/drones/DroneDetect_V2/BOTH/INS_FY/INS_1110_00.dat: 109,240,315 samples
  - /home/sambot/win_downloads/DATASETS/drones/DroneDetect_V2/CLEAN/INS_FY/INS_0010_00.dat: 105,340,926 samples

Duration issues:
  - /home/sambot/win_downloads/DATASETS/drones/DroneDetect_V2/BOTH/INS_FY/INS_1110_00.dat: 1.821s
  - /home/sambot/win_downloads/DATASETS/drones/DroneDetect_V2/CLEAN/INS_FY/INS_0010_00.dat: 1.756s

⚠️ Note: Files with duration < 2s will be handled in the preprocessing notebook.
   Options: padding, exclusion, or variable-length windowing.

Files with IQ values outside [-1, 1]: 52

Files violating [-1, 1] range:
  - AIR_1110_00.dat
    I: [-0.9795, 1.0200]
    Q: [-0.9785, 1.0210]
  - AIR_1110_01.dat
    I: [-1.0044, 0.9951]
    Q: [-1.0054, 0.9941]
  - AIR_1110_02.dat
    I: [-0.9795, 1.0200]
    Q: [-0.9785, 1.0210]
  - AIR_1110_03.dat
    I: [-1.0044, 0.9951]
    Q: [-1.0054, 0.9941]
  - AIR_1110_04.dat
    I: [-0.9795, 1.0200]
    Q: [-0.9785, 1.0210]
  - AIR_1101_00.dat
    I: [-1.0044, 1.0200]
    Q: [-1.0054, 0.9941]
```

```
- AIR_1101_01.dat
  I: [-1.0044, 1.0200]
  Q: [-1.0054, 1.0210]
- AIR_1101_02.dat
  I: [-0.9795, 1.0200]
  Q: [-0.9785, 1.0210]
- AIR_1101_04.dat
  I: [-0.9795, 1.0200]
  Q: [-0.9785, 1.0210]
- AIR_1100_01.dat
  I: [-1.0044, 0.9951]
  Q: [-1.0054, 0.9941]
... and 42 more files
```

3.3.2 Metadata Consistency

In [12]:

```
# Validate metadata consistency: filename codes vs directory hierarchy
print("""
============================================================
METADATA CONSISTENCY VALIDATION
============================================================""")

inconsistencies = []

for _, row in df.iterrows():
    file_path = Path(row['file_path'])

    # Extract info from directory structure
    # Expected: .../INTERFERENCE/DRONE_STATE/DRONE_XXXX_YY.dat
    dir_interference = file_path.parent.parent.name  # CLEAN or BOTH
    dir_drone_state = file_path.parent.name  # e.g., AIR_ON

    # Parse drone and state from directory name
    dir_parts = dir_drone_state.split('_')
    if len(dir_parts) >= 2:
        dir_drone = dir_parts[0]
        dir_state = dir_parts[1]
    else:
        dir_drone = dir_parts[0]
        dir_state = 'UNKNOWN'
```

```
    # Compare with extracted metadata
    if row['drone_code'] != dir_drone:
        inconsistencies.append((row['file_path'], 'drone_code', row['drone_code'], dir_drone))
    if row['state'] != dir_state:
        inconsistencies.append((row['file_path'], 'state', row['state'], dir_state))
    if row['interference'] != dir_interference:
        inconsistencies.append((row['file_path'], 'interference', row['interference'], dir_interference))

if inconsistencies:
    inconsist_list = "\n".join([f"  - {Path(f).name}\n    {field}: extracted='{extracted}' vs directory='{from_dir}'"
                                for f, field, extracted, from_dir in inconsistencies[:5]])
    more_inconsist = f"  ... and {len(inconsistencies) - 5} more inconsistencies" if len(inconsistencies) > 5 else ""
    print(f"""
Files checked: {len(df)}
Inconsistencies found: {len(inconsistencies)}

Inconsistencies (showing first 5):
{inconsist_list}
{more_inconsist}

============================================================
CONVENTION ADOPTED
============================================================
This code uses filenames (MA1/MAV) as ground truth, not directory names (MP1/MP2).
Reason: Filenames match DJI official model codes (Mavic Air, Mavic Pro).
The paper aerospace-08-00179-v2.pdf uses full names, not abbreviated codes.""")
else:
    print(f"""
Files checked: {len(df)}
Inconsistencies found: {len(inconsistencies)}

-> All metadata extracted from filenames matches directory hierarchy""")
```

```
============================================================
METADATA CONSISTENCY VALIDATION
============================================================

Files checked: 195
Inconsistencies found: 60

Inconsistencies (showing first 5):
  - MA1_1110_00.dat
    drone_code: extracted='MA1' vs directory='MP1'
  - MA1_1110_01.dat
    drone_code: extracted='MA1' vs directory='MP1'
  - MA1_1110_02.dat
    drone_code: extracted='MA1' vs directory='MP1'
  - MA1_1110_03.dat
    drone_code: extracted='MA1' vs directory='MP1'
  - MA1_1110_04.dat
    drone_code: extracted='MA1' vs directory='MP1'
  ... and 55 more inconsistencies

============================================================
CONVENTION ADOPTED
============================================================
This code uses filenames (MA1/MAV) as ground truth, not directory names (MP1/MP2).
Reason: Filenames match DJI official model codes (Mavic Air, Mavic Pro).
The paper aerospace-08-00179-v2.pdf uses full names, not abbreviated codes.
```

3.3.3 Index Distribution

In [13]:

```python
# Analyze index distribution
index_counts = df['index'].value_counts().sort_index()
expected_indices = set(range(5))  # 0-4
actual_indices = set(df['index'].unique())
missing_indices = expected_indices - actual_indices

missing_idx_msg = f"\nWarning: Missing indices: {missing_indices}" if missing_indices else "\n-> All expected indices (0-4) ar

# Verify all valid combinations have exactly 5 replicas
valid_combinations = complete_df[complete_df['count'] > 0]
all_have_5 = (valid_combinations['count'] == 5).all()
```

```python
non_5_msg = ""
if not all_have_5:
    non_5 = valid_combinations[valid_combinations['count'] != 5]
    non_5_msg = "\nCombinations with != 5 replicas:\n" + "\n".join(
        [f"  - {row['drone_code']}/{row['state']}/{row['interference']}: {row['count']} files"
         for _, row in non_5.iterrows()])

print(f"""
============================================================
INDEX DISTRIBUTION
============================================================

Index distribution (expected: 5 files per combination):
{index_counts.to_dict()}
{missing_idx_msg}

=== REPLICA UNIFORMITY ===
All valid combinations have exactly 5 replicas: {all_have_5}{non_5_msg}""")
```

```
============================================================
INDEX DISTRIBUTION
============================================================

Index distribution (expected: 5 files per combination):
{0: 39, 1: 39, 2: 39, 3: 39, 4: 39}

-> All expected indices (0-4) are present

=== REPLICA UNIFORMITY ===
All valid combinations have exactly 5 replicas: True
```

3.3.3.1 Replica Reproducibility Check

In [14]:

```python
# Reproducibility check: Load 5 replicas of a single combination (BOTH/FY)
# We use the first available drone with BOTH/FY combination
test_combination = complete_df[(complete_df['interference'] == 'BOTH') &
                               (complete_df['state'] == 'FY') &
                               (complete_df['count'] > 0)].iloc[0]

test_drone = test_combination['drone_code']
```

```python
print(f"""============================================================
REPLICA REPRODUCIBILITY: {test_drone} / FY / BOTH
============================================================""")

replica_stats = []
for idx in range(5):
    files = df[(df['drone_code'] == test_drone) &
               (df['state'] == 'FY') &
               (df['interference'] == 'BOTH') &
               (df['index'] == idx)]

    if len(files) > 0:
        file_path = files.iloc[0]['file_path']
        iq = data_loader.load_raw_iq(file_path)
        # Sample 1M points
        sample = iq[::SAMPLE_STRIDE]

        replica_stats.append({
            'replica': idx,
            'mean_real': sample.real.mean(),
            'std_real': sample.real.std(),
            'mean_imag': sample.imag.mean(),
            'std_imag': sample.imag.std(),
            'range_real': (sample.real.min(), sample.real.max()),
            'range_imag': (sample.imag.min(), sample.imag.max())
        })

replica_df = pd.DataFrame(replica_stats)
intra_std = replica_df[['mean_real', 'std_real', 'mean_imag', 'std_imag']].std()

print(f"""
Replica statistics (sampled 1M points each):
{replica_df.to_string(index=False)}

=== INTRA-REPLICA VARIABILITY ===
Standard deviation of means across replicas:
  Real: {intra_std['mean_real']:.6f}
  Imag: {intra_std['mean_imag']:.6f}

Interpretation: Low variability indicates consistent recording conditions across replicas.""")
```

```
============================================================
REPLICA REPRODUCIBILITY: AIR / FY / BOTH
============================================================

Replica statistics (sampled 1M points each):
 replica  mean_real  std_real  mean_imag  std_imag             range_real             range_imag
       0   0.048748  0.193431   0.087943  0.194415 (-0.9794922, 1.0200195) (-0.9785156, 1.0209961)
       1   0.009135  0.062286   0.015609  0.063444 (-1.0043945, 0.9951172) (-1.0053711, 0.9941406)
       2   0.048308  0.203140   0.086878  0.204177 (-0.9794922, 1.0200195) (-0.9785156, 1.0209961)
       3   0.009203  0.073582   0.015537  0.074639 (-1.0043945, 0.9951172) (-1.0053711, 0.9941406)
       4   0.047592  0.203286   0.086620  0.204342 (-0.9794922, 1.0200195) (-0.9785156, 1.0209961)

=== INTRA-REPLICA VARIABILITY ===
Standard deviation of means across replicas:
  Real: 0.021391
  Imag: 0.039206

Interpretation: Low variability indicates consistent recording conditions across replicas.
```

In [15]:

```
# Reproducibility check: Load 5 replicas of a single combination (CLEAN/FY)
# We use the first available drone with CLEAN/FY combination
test_combination = complete_df[(complete_df['interference'] == 'CLEAN') &
                                (complete_df['state'] == 'FY') &
                                (complete_df['count'] > 0)].iloc[0]

test_drone = test_combination['drone_code']
print(f"""============================================================
REPLICA REPRODUCIBILITY: {test_drone} / FY / CLEAN
============================================================""")

replica_stats = []
for idx in range(5):
    files = df[(df['drone_code'] == test_drone) &
               (df['state'] == 'FY') &
               (df['interference'] == 'CLEAN') &
               (df['index'] == idx)]

    if len(files) > 0:
        file_path = files.iloc[0]['file_path']
        iq = data_loader.load_raw_iq(file_path)
        # Sample 1M points
```

```
        sample = iq[::SAMPLE_STRIDE]

        replica_stats.append({
            'replica': idx,
            'mean_real': sample.real.mean(),
            'std_real': sample.real.std(),
            'mean_imag': sample.imag.mean(),
            'std_imag': sample.imag.std(),
            'range_real': (sample.real.min(), sample.real.max()),
            'range_imag': (sample.imag.min(), sample.imag.max())
        })

replica_df = pd.DataFrame(replica_stats)
intra_std = replica_df[['mean_real', 'std_real', 'mean_imag', 'std_imag']].std()

print(f"""
Replica statistics (sampled 1M points each):
{replica_df.to_string(index=False)}

=== INTRA-REPLICA VARIABILITY ===
Standard deviation of means across replicas:
  Real: {intra_std['mean_real']:.6f}
  Imag: {intra_std['mean_imag']:.6f}

Interpretation: Low variability indicates consistent recording conditions across replicas.""")
```

```
============================================================
REPLICA REPRODUCIBILITY: AIR / FY / CLEAN
============================================================

Replica statistics (sampled 1M points each):
 replica  mean_real  std_real  mean_imag  std_imag                range_real                range_imag
       0   0.044444  0.180218   0.081819  0.181130  (-0.9794922, 1.0200195)  (-0.9785156, 1.0209961)
       1   0.008779  0.039203   0.014864  0.039553 (-0.97998047, 0.7451172)  (-0.8623047, 0.9941406)
       2   0.000792  0.005051   0.001742  0.005968 (-0.14990234, 0.2734375)   (-0.15625, 0.25195312)
       3   0.009016  0.019241   0.015171  0.019439 (-0.42578125, 0.7451172) (-0.47460938, 0.7182617)
       4   0.041672  0.063895   0.074735  0.067648  (-0.9794922, 1.0200195)  (-0.9785156, 1.0209961)

=== INTRA-REPLICA VARIABILITY ===
Standard deviation of means across replicas:
  Real: 0.020483
  Imag: 0.037550

Interpretation: Low variability indicates consistent recording conditions across replicas.
```

3.3.4 IQ Value Distributions

Sampling strategy: We sample 100,000 points (0.08% of 120M samples) for visualization because:

1. Reduces rendering time (plotly performance)
2. Avoids overplotting (120M points would appear as solid blob)
3. Maintains statistical representation (random sampling)
4. The full signal range is [-0.99, 0.99] (verified in section 2), but most samples are concentrated near origin (weak signal baseline)
5. Extreme values (peaks) are rare and may not appear in this sample

In [16]:

```python
# Analyze IQ value distributions - AIR_0000_00.dat (CLEAN/ON)
sample_file = config.DATA_DIR / "CLEAN" / "AIR_ON" / "AIR_0000_00.dat"

if sample_file.exists():
    print(f"""
============================================================
IQ VALUE DISTRIBUTION
File: {sample_file.name} ({sample_file.parent.parent.name} / {sample_file.parent.name.split('_')[1]})
============================================================""")
```

```
    # Load IQ data for this specific file
    iq_data = data_loader.load_raw_iq(sample_file)

    # Create histograms for I and Q components
    fig = make_subplots(
        rows=1, cols=2,
        subplot_titles=('In-phase (I) Distribution', 'Quadrature (Q) Distribution')
    )

    # Sample 100k points for faster plotting
    np.random.seed(42)
    sample_indices = np.random.choice(len(iq_data), size=min(100000, len(iq_data)), replace=False)
    i_sample = iq_data.real[sample_indices]
    q_sample = iq_data.imag[sample_indices]

    fig.add_trace(
        go.Histogram(x=i_sample, nbinsx=100, name='I', marker_color='steelblue'),
        row=1, col=1
    )
    fig.add_trace(
        go.Histogram(x=q_sample, nbinsx=100, name='Q', marker_color='coral'),
        row=1, col=2
    )

    fig.update_xaxes(title_text="Amplitude", row=1, col=1)
    fig.update_xaxes(title_text="Amplitude", row=1, col=2)
    fig.update_yaxes(title_text="Count", row=1, col=1)

    fig.update_layout(
        title=f"IQ Component Distributions CLEAN ON - {sample_file.name}",
        height=400,
        showlegend=False
    )
    save_figure(fig)
    fig.show()

    # IQ scatter plot (constellation diagram)
    fig = go.Figure()
    fig.add_trace(go.Scattergl(
        x=i_sample,
        y=q_sample,
```

```python
        mode='markers',
        marker=dict(size=1, opacity=0.3, color='steelblue'),
        name='IQ Samples'
    ))

    fig.update_layout(
        title=f"IQ Constellation Diagram CLEAN ON - {sample_file.name}",
        xaxis_title="In-phase (I)",
        yaxis_title="Quadrature (Q)",
        height=600,
        width=600
    )
    save_figure(fig)
    fig.show()

    print(f"""
I component: mean={i_sample.mean():.4f}, std={i_sample.std():.4f}
Q component: mean={q_sample.mean():.4f}, std={q_sample.std():.4f}""")
```

```
============================================================
IQ VALUE DISTRIBUTION
File: AIR_0000_00.dat (CLEAN / ON)
============================================================
Saved: ../figures/001_exploration_general/IQ_Component_Distributions_CLEAN_ON_AIR_0000_00dat.png
Saved: ../figures/001_exploration_general/IQ_Constellation_Diagram_CLEAN_ON_AIR_0000_00dat.png
I component: mean=0.0094, std=0.0335
Q component: mean=0.0160, std=0.0347
```