DroneDetect V2 - Frequency Domain Exploration (V5)

Motivation

Why focus on frequency domain analysis?

Swinney & Woods (2021) demonstrate in their paper "The Effect of Real-World Interference on CNN Feature Extraction and Machine Learning Classification of Unmanned Aerial Systems" (Aerospace 2021, 8, 179) that:

> **"Overall, frequency domain features extracted from a CNN were shown to be more robust than time domain features in the presence of interference."**

Quantitative results (Table from paper):

Representation	Detection (2 classes)	Type (8 classes)	Flight Mode (21 classes)
PSD + LR	100%	98.1% (±0.4%)	95.4% (±0.3%)
Spectrogram + LR	96.7%	90.5%	87.3%

Conclusion: Frequency domain features (PSD) are **7-8% more robust** to WiFi/Bluetooth interference.

Additional confirmations:

- Swinney & Woods (2022): Low-cost Raspberry Pi implementation confirms PSD robustness
- URSI 2024: "PSD has proven to be more reliable than simple spectrograms"
- Glüge et al. (2023): Independent confirmation of frequency domain advantage

---

This notebook focuses specifically on:

- **Power Spectral Density (PSD) analysis** using Welch's method
- Spectral signatures comparison across drone models

- Interference impact analysis

Note: Spectrograms (time-frequency representations) will be explored in a separate notebook dedicated to time-frequency analysis.

Context: RF signals from drones at 2.4 GHz (OcuSync, Lightbridge, WiFi) captured with BladeRF SDR.

1. Setup

In [1]:
```
# Install requirements (uncomment if needed)
# !pip install plotly scipy numpy pandas seaborn scikit-learn -q

# Mount Google Drive (optional - uncomment if dataset is on Drive)
# from google.colab import drive
# drive.mount('/content/drive')
```

In [2]:
```
# Import plotly for interactive visualizations
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
import re

print("Setup complete!")
```

```
Setup complete!
```

In [3]:
```
# Colab specific - clone repository and add to path
# !git clone https://github.com/tryph0n/mldrone.git
# %cd mldrone

# import sys
# sys.path.insert(0, '/content/mldrone/src')
```

In [4]:
```python
# Import local modules
from dronedetect import config, data_loader, preprocessing, features
```

In [5]:
```python
# Constants
SEGMENT_MS = 20  # Duration (ms) of each signal segment for analysis
N_SEGMENTS_PER_FILE = 5  # Number of segments extracted per file for analysis
RANDOM_STATE = 42

# Setup figure saving
NOTEBOOK_NAME = "01c_exploration_frequentiel_v5"
FIGURES_DIR = Path("../figures") / NOTEBOOK_NAME

# ============================================================================
# Utility Functions
# ============================================================================

def save_figure(fig, title: str):
    """
    Save plotly figure to PNG file.
    Filename is derived from the figure title.

    Args:
        fig: Plotly figure object
        title: Figure title (used for both display and filename)
    """
    FIGURES_DIR.mkdir(parents=True, exist_ok=True)
    # Sanitize title for filename
    filename = re.sub(r'[^\w\s-]', '', title).strip()
    filename = re.sub(r'[\s-]+', '_', filename)
    filepath = FIGURES_DIR / f"{filename}.png"
    fig.write_image(str(filepath), width=1200, height=800)
    print(f"Saved: {filepath}")


def compute_psd_db(segment_norm, epsilon=1e-10):
    """
    Compute PSD and convert to dB scale.

    Args:
        segment_norm: Normalized IQ segment
        epsilon: Small value to avoid log(0). Default 1e-10 sets -100 dB noise floor.

    Returns:
        freqs: Frequency array (Hz)
        psd_db: PSD in dB scale
    """
    freqs, psd_linear = features.compute_psd(segment_norm)
    psd_db = 10 * np.log10(np.maximum(psd_linear, epsilon))
    return freqs, psd_db
```

In [6]:
```
# Load dataset metadata
if config.DATA_DIR.exists():
    df = data_loader.get_dataset_metadata(config.DATA_DIR)
    print(f"Total files: {len(df)}")
    print(f"Drones: {df['drone_code'].unique()}")
    print(f"States: {df['state'].unique()}")
    print(f"Interference: {df['interference'].unique()}")
else:
    df = None
    print("Dataset directory not found")
```

```
Total files: 195
Drones: ['AIR' 'DIS' 'INS' 'MIN' 'MA1' 'MAV' 'PHA']
States: ['FY' 'HO' 'ON']
Interference: ['BOTH' 'CLEAN']
```

2. Power Spectral Density Analysis

PSD Parameters

Parameter	Value	Justification
NFFT	1024 points	FFT length for Welch's method. Frequency resolution: FS/nfft = 58.6 kHz bins.
Overlap	512 samples	50% overlap for Welch's method (standard practice for variance reduction).
Window	Hamming	Reduces spectral leakage. Standard choice for PSD estimation with good side-lobe suppression.
Noise floor percentile	10th percentile	**Arbitrary choice** for noise floor estimation (commonly used heuristic).

Reference: Welch, P.D. (1967) - "The use of Fast Fourier Transform for the estimation of power spectra", IEEE Transactions on Audio and Electroacoustics, DOI: 10.1109/TAU.1967.1161901

2.1 Spectral Occupancy Validation (BW_99%)

Verify that the 60 MHz sampling rate preserves signal information.

Criterion: If BW_99% < 28 MHz (capture bandwidth), downsampling is safe.

In [7]:
```python
from scipy.signal import welch

def compute_bw_99(iq_signal, fs, nperseg=4096):
    """Compute bandwidth containing 99% of signal energy."""
    freqs, psd = welch(iq_signal, fs=fs, nperseg=nperseg, return_onesided=False)
    freqs = np.fft.fftshift(freqs)
    psd = np.fft.fftshift(psd)

    # Cumulative energy from highest power bins
    psd_sorted_idx = np.argsort(psd)[::-1]
    psd_sorted = psd[psd_sorted_idx]
    cumsum = np.cumsum(psd_sorted) / np.sum(psd_sorted)
    n_bins_99 = np.searchsorted(cumsum, 0.99)

    # Convert to bandwidth
    freq_resolution = fs / nperseg
    bw_99_hz = n_bins_99 * freq_resolution

    return bw_99_hz / 1e6  # Return in MHz

# Compute BW_99% for each drone
bw_results = []
for drone in df['drone_code'].unique():
    files = df[(df['drone_code'] == drone) &
               (df['state'] == 'ON') &
               (df['interference'] == 'CLEAN')]['file_path']
    if len(files) == 0:
        files = df[df['drone_code'] == drone]['file_path']
    file_path = files.iloc[0]
    iq = data_loader.load_raw_iq(file_path)
    bw = compute_bw_99(iq[:int(config.FS)], config.FS)
    bw_results.append({'drone': drone, 'bw_99_mhz': bw})

bw_df = pd.DataFrame(bw_results).sort_values('bw_99_mhz', ascending=False)
print("BW_99% by drone (MHz):")
print(bw_df.to_string(index=False))

max_bw = bw_df['bw_99_mhz'].max()
print(f"\nMax BW_99%: {max_bw:.2f} MHz")
print(f"Capture BW: 28 MHz")
print(f"Downsampling safe: {'YES' if max_bw < 28 else 'NO - REVIEW NEEDED'}")
```

```
BW_99% by drone (MHz):
drone  bw_99_mhz
  MAV  25.415039
  INS  23.100586
  MA1  20.786133
  PHA  20.097656
  AIR  18.706055
  DIS  17.475586
  MIN  17.373047

Max BW_99%: 25.42 MHz
Capture BW: 28 MHz
Downsampling safe: YES
```

In [8]:
```
from tqdm import tqdm

# Compute average PSD power with STRATIFIED sampling
if df is not None:
    power_data = []

    # Stratified sampling: iterate over all combinations
    drones = df['drone_code'].unique()
    states = df['state'].unique()
    interferences = df['interference'].unique()

    n_files_per_combo = 3  # Files per combination
```

```python
    # Calculate total number of combinations
    total_combos = sum(
        1 for drone in drones
        for state in states
        for interference in interferences
        if len(df[(df['drone_code'] == drone) &
                  (df['state'] == state) &
                  (df['interference'] == interference)]) > 0
    )

    pbar = tqdm(total=total_combos, desc="Processing combinations")

    for drone in drones:
        for state in states:
            for interference in interferences:
                combo_files = df[
                    (df['drone_code'] == drone) &
                    (df['state'] == state) &
                    (df['interference'] == interference)
                ]

                if len(combo_files) == 0:
                    continue

                # Sample files from this combination
                sample_files = combo_files.sample(
                    n=min(n_files_per_combo, len(combo_files)),
                    random_state=RANDOM_STATE
                )

                for _, row in sample_files.iterrows():
                    try:
                        iq = data_loader.load_raw_iq(Path(row['file_path']))
                        segments = preprocessing.segment_signal(iq, segment_ms=SEGMENT_MS)
                        del iq

                        for seg in segments[:N_SEGMENTS_PER_FILE]:
                            seg_norm = preprocessing.normalize(seg)
                            freqs, psd_db = compute_psd_db(seg_norm)

                            power_data.append({
                                'drone_code': row['drone_code'],
                                'state': row['state'],
                                'interference': row['interference'],
                                'avg_power_db': np.mean(psd_db)
                            })
                    except Exception as e:
                        print(f"Error: {row['file_path']}: {e}")

                pbar.update(1)

    pbar.close()

    power_df = pd.DataFrame(power_data)
    print(f"\nProcessed {len(power_df)} segments")
    print(f"Combinations covered: {power_df.groupby(['drone_code', 'state', 'interference']).ngroups}")
else:
    power_df = pd.DataFrame()
```

```
Processing combinations:   0%|          | 0/39 [00:00<?, ?it/s]
Processing combinations: 100%|██████████| 39/39 [11:02<00:00, 16.98s/it]
Processed 585 segments
Combinations covered: 39
```

In [9]:

```
# Boxplots: power distribution by flight mode and drone
if len(power_df) > 0:
    # Boxplot: power vs flight mode
    title = "Average PSD Power by Flight Mode"
    fig = px.box(
        power_df,
        x='state',
        y='avg_power_db',
        color='state',
        title=title,
        labels={'state': 'Flight Mode', 'avg_power_db': 'Average Power (dB)'},
        height=500
    )
    save_figure(fig, title)
    fig.show()

    # Boxplot: power vs drone
```

```python
    title = "Average PSD Power by Drone"
    fig = px.box(
        power_df,
        x='drone_code',
        y='avg_power_db',
        color='drone_code',
        title=title,
        labels={'drone_code': 'Drone Code', 'avg_power_db': 'Average Power (dB)'},
        height=500
    )
    save_figure(fig, title)
    fig.show()
```

```
Saved: ../figures/01c_exploration_frequentiel_v5/Average_PSD_Power_by_Flight_Mode.png
Saved: ../figures/01c_exploration_frequentiel_v5/Average_PSD_Power_by_Drone.png
```

In [10]:
```python
# Heatmap: drone x state with average power (dB scale)
if len(power_df) > 0:
    pivot_power = power_df.pivot_table(
        values='avg_power_db',
        index='drone_code',
        columns='state',
        aggfunc='mean'
    )

    title = "Average PSD Power Drone x Flight Mode (dB)"
    fig = px.imshow(
        pivot_power,
        labels=dict(x="Flight Mode", y="Drone Code", color="Avg Power (dB)"),
        title=title,
        color_continuous_scale='Viridis',
        text_auto='.1f',
        aspect="auto"
    )
    save_figure(fig, title)
    fig.show()
```

```
Saved: ../figures/01c_exploration_frequentiel_v5/Average_PSD_Power_Drone_x_Flight_Mode_dB.png
```

In [11]:
```python
# Compare spectral signatures across drone models (CLEAN, ON state)
if config.DATA_DIR.exists() and df is not None:
    drones_to_plot = ['AIR', 'DIS', 'INS', 'MIN', 'MA1', 'MAV']  # PHA may not have CLEAN data

    fig = go.Figure()

    for drone in drones_to_plot:
        drone_files = df[(df['drone_code'] == drone) &
                         (df['state'] == 'ON') &
                         (df['interference'] == 'CLEAN')]

        if len(drone_files) > 0:
            file_path = Path(drone_files.iloc[0]['file_path'])
            iq = data_loader.load_raw_iq(file_path)

            segments = preprocessing.segment_signal(iq, segment_ms=SEGMENT_MS)
            segment_norm = preprocessing.normalize(segments[0])

            # PSD in dB
            freqs, psd_db = compute_psd_db(segment_norm)

            fig.add_trace(go.Scatter(
                x=freqs / 1e6,
                y=psd_db,
                mode='lines',
                name=drone,
                line=dict(width=2)
            ))

    fig.update_layout(
        title="Spectral Signatures by Drone Model (CLEAN & ON mode)",
        xaxis_title="Frequency Offset (MHz)",
        yaxis_title="Power Spectral Density (dB)",
        height=600,
        hovermode='x unified'
    )
    save_figure(fig, "Spectral Signatures by Drone Model CLEAN ON dB")
    fig.show()
```

```
Saved: ../figures/01c_exploration_frequentiel_v5/Spectral_Signatures_by_Drone_Model_CLEAN_ON_dB.png
```

In [12]:
```
# Compare CLEAN vs BOTH interference for selected drones
if config.DATA_DIR.exists() and df is not None:
```

```python
test_drones = ['AIR', 'MA1', 'MIN']

fig = make_subplots(
    rows=len(test_drones), cols=1,
    subplot_titles=[f'{d}: CLEAN vs BOTH Interference' for d in test_drones],
    vertical_spacing=0.08
)

colors = {'CLEAN': 'blue', 'BOTH': 'red'}

for row_idx, drone in enumerate(test_drones, start=1):
    for interference in ['CLEAN', 'BOTH']:
        drone_files = df[(df['drone_code'] == drone) &
                         (df['state'] == 'ON') &
                         (df['interference'] == interference)]

        if len(drone_files) > 0:
            file_path = Path(drone_files.iloc[0]['file_path'])
            iq = data_loader.load_raw_iq(file_path)

            segments = preprocessing.segment_signal(iq, segment_ms=SEGMENT_MS)
            segment_norm = preprocessing.normalize(segments[0])
            freqs, psd_db = compute_psd_db(segment_norm)

            fig.add_trace(
                go.Scatter(
                    x=freqs / 1e6,
                    y=psd_db,
                    mode='lines',
                    name=interference if row_idx == 1 else None,
                    legendgroup=interference,
                    showlegend=(row_idx == 1),
                    line=dict(color=colors[interference], width=2)
                ),
                row=row_idx, col=1
            )

fig.update_xaxes(title_text="Frequency Offset (MHz)", row=len(test_drones), col=1)
fig.update_yaxes(title_text="PSD (dB)")
fig.update_layout(
    height=300 * len(test_drones),
    title_text="Interference Impact on Spectrum (CLEAN vs BOTH)",
    hovermode='x unified'
)
save_figure(fig, "Interference Impact CLEAN vs BOTH")
fig.show()
```

```
Saved: ../figures/01c_exploration_frequentiel_v5/Interference_Impact_CLEAN_vs_BOTH.png
```

3. Drone Discrimination Analysis (MA1/MAV)

MA1 (OcuSync 1.0) and MAV (OcuSync 2.0) account for 42% of SVM classification errors.

This section extracts higher-order spectral statistics to improve discrimination:

- **Spectral kurtosis**: Oscillator frequency stability
- **Spectral skewness**: Antenna radiation asymmetry
- **BW_90**: Front-end filtering characteristics

In [13]:
```
from scipy.stats import kurtosis, skew

def extract_spectral_features(psd, freqs):
    """Extract 2nd-order spectral statistics for discrimination."""
    psd_norm = psd / np.sum(psd)

    # Spectral centroid and spread
    centroid = np.sum(freqs * psd_norm)
    spread = np.sqrt(np.sum((freqs - centroid)**2 * psd_norm))

    # Higher-order statistics
    kurt = kurtosis(psd)
    sk = skew(psd)

    # 90% bandwidth (number of bins containing 90% energy)
    psd_sorted = np.sort(psd)[::-1]
    cumsum = np.cumsum(psd_sorted) / np.sum(psd_sorted)
    bw_90 = np.searchsorted(cumsum, 0.9)

    return {
```

```python
        'centroid_mhz': centroid / 1e6,
        'spread_mhz': spread / 1e6,
        'kurtosis': kurt,
        'skewness': sk,
        'bw_90_bins': bw_90
    }

# Compare MA1 vs MAV
ma1_mav_features = []

for drone in ['MA1', 'MAV']:
    files = df[(df['drone_code'] == drone) &
               (df['interference'] == 'CLEAN')]['file_path'][:10]

    for file_path in files:
        iq = data_loader.load_raw_iq(file_path)
        freqs, psd = welch(iq[:int(config.FS)], fs=config.FS, nperseg=1024)

        features_dict = extract_spectral_features(psd, freqs)
        features_dict['drone'] = drone
        features_dict['file'] = Path(file_path).name
        ma1_mav_features.append(features_dict)

ma1_mav_df = pd.DataFrame(ma1_mav_features)

# Statistical comparison
print("=== MA1 vs MAV Spectral Features ===\n")
for col in ['kurtosis', 'skewness', 'bw_90_bins', 'spread_mhz']:
    ma1_vals = ma1_mav_df[ma1_mav_df['drone'] == 'MA1'][col]
    mav_vals = ma1_mav_df[ma1_mav_df['drone'] == 'MAV'][col]

    # Effect size (Cohen's d)
    pooled_std = np.sqrt((ma1_vals.std()**2 + mav_vals.std()**2) / 2)
    cohens_d = abs(ma1_vals.mean() - mav_vals.mean()) / pooled_std if pooled_std > 0 else 0

    print(f"{col}:")
    print(f"  MA1: {ma1_vals.mean():.3f} +/- {ma1_vals.std():.3f}")
    print(f"  MAV: {mav_vals.mean():.3f} +/- {mav_vals.std():.3f}")
    print(f"  Cohen's d: {cohens_d:.2f} ({'small' if cohens_d < 0.5 else 'medium' if cohens_d < 0.8 else 'LARGE'})")
    print()
```

```python
# Visualization
fig = make_subplots(rows=2, cols=2, subplot_titles=['Kurtosis', 'Skewness', 'BW_90 (bins)', 'Spread (MHz)'])

for i, col in enumerate(['kurtosis', 'skewness', 'bw_90_bins', 'spread_mhz']):
    row, col_idx = i // 2 + 1, i % 2 + 1
    for drone, color in [('MA1', 'blue'), ('MAV', 'red')]:
        vals = ma1_mav_df[ma1_mav_df['drone'] == drone][col]
        fig.add_trace(go.Box(y=vals, name=drone, marker_color=color, showlegend=(i==0)),
                      row=row, col=col_idx)

fig.update_layout(height=600, title="MA1 vs MAV Spectral Feature Comparison")
save_figure(fig, "ma1_mav_discrimination_features")
fig.show()
```

```
/home/sambot/mldrone/.venv/lib/python3.10/site-packages/scipy/signal/_spectral_py.py:652: UserWarning:

Input data is complex, switching to return_onesided=False
```

```
=== MA1 vs MAV Spectral Features ===

kurtosis:
  MA1: inf +/- nan
  MAV: 0.810 +/- 0.486
  Cohen's d: 0.00 (small)

skewness:
  MA1: 1.391 +/- 0.379
  MAV: 1.612 +/- 0.837
  Cohen's d: 0.34 (small)

bw_90_bins:
  MA1: 307.000 +/- 46.200
  MAV: 314.400 +/- 112.349
  Cohen's d: 0.09 (small)

spread_mhz:
  MA1: 10.416 +/- 0.482
  MAV: 10.819 +/- 1.005
  Cohen's d: 0.51 (medium)
```

```
/home/sambot/mldrone/.venv/lib/python3.10/site-packages/pandas/core/nanops.py:1016: RuntimeWarning:

invalid value encountered in subtract
```

```
Saved: ../figures/01c_exploration_frequentiel_v5/ma1_mav_discrimination_features.png
```