In [1]:
```
# Dataset parameters (imported from config)
# FS is now config.FS
SEGMENT_MS = 20  # Segment duration for feature extraction (ms)
VIZ_MS = 5  # Visualization window (ms)
N_FILES_PER_COMBO = 2  # Files per (drone, state, interference) combination
N_SEGMENTS_PER_FILE = 3  # Segments per file

# Computation constants
EPSILON_AMP = 1e-10  # Numerical stability for amplitude
EPSILON_POWER = 1e-12  # Numerical stability for power
RANDOM_STATE = 42
```

In [2]:
```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
from pathlib import Path
import warnings
warnings.filterwarnings('ignore')

# Import local modules
from dronedetect import config, data_loader, preprocessing

# Figure output directory
FIGURE_DIR = Path("figures/01b_exploration_temporal_v7")
FIGURE_DIR.mkdir(parents=True, exist_ok=True)
```

Definitions

IQ Signal: Complex representation of RF signal. I (real) and Q (imaginary) components encode amplitude and phase.

ZCR (Zero Crossing Rate): Rate of sign changes in signal. Indicator of modulation complexity.

Autocorrelation: Similarity of signal with time-shifted version of itself. Reveals repetitive patterns.

Instantaneous Phase/Frequency: Phase angle and its derivative at each time sample. Captures FM/FSK modulation.

I/Q Imbalance: Gain and phase mismatch between I and Q channels due to hardware imperfections. Device-specific signature.

Temporal Domain Analysis of Drone RF Signals

Exploration of time-domain features for drone classification.

Dataset: 195 files, 7 drones, 3 flight states, 2 interference conditions
Objective: Extract discriminative temporal features for drone model classification (AIR vs PHA vs MAV, etc.)

Key temporal features (validated by literature):

- Zero Crossing Rate (ZCR)
- Phase variance and instantaneous frequency
- I/Q imbalance (hardware fingerprinting)
- Multi-lag autocorrelation
- Envelope variance

Suppressions from v6 (not validated for drones):

- PAPR and Crest Factor (OFDM-specific metrics)
- Burst analysis (contaminated by WiFi/Bluetooth interference)

In [3]:
```python
def detrend_phase(phase_unwrapped):
    """Remove linear trend from unwrapped phase (carrier frequency offset)."""
    x = np.arange(len(phase_unwrapped))
    coeffs = np.polyfit(x, phase_unwrapped, deg=1)
    trend = np.polyval(coeffs, x)
    return phase_unwrapped - trend
```

In [4]:
```python
def load_iq_segment(file_path, start_sample=0, duration_ms=SEGMENT_MS):
    """Load IQ samples from binary file for given duration."""
    n_samples = int(config.FS * duration_ms / 1000)
```

```
    iq_full = data_loader.load_raw_iq(file_path)
    start_idx = start_sample
    end_idx = start_idx + n_samples

    return iq_full[start_idx:end_idx]


def load_segment_for_viz(file_path, start_sample=0, duration_ms=VIZ_MS):
    """Load segment for visualization with time axis."""
    iq_viz = load_iq_segment(file_path, start_sample, duration_ms)
    n_samples = len(iq_viz)
    time_axis = np.arange(n_samples) / config.FS * 1e6  # microseconds

    return iq_viz, time_axis


def save_figure(fig, filename):
    """Save plotly figure as PNG."""
    output_path = FIGURE_DIR / f"{filename}.png"
    fig.write_image(output_path, width=1400, height=800)
    print(f"Saved: {output_path}")
```

3. Utility Functions

Helper functions for data loading and visualization.

In [5]:
```
# Load dataset metadata using data_loader
try:
    assert config.DATA_DIR.exists(), f"DATA_DIR not found: {config.DATA_DIR}"
    df = data_loader.get_cached_metadata(force_refresh=True)
    assert len(df) > 0, "Empty metadata"
    print(f"Loaded {len(df)} files")
except Exception as e:
    print(f"**FATAL ERROR**: {e}")
    raise

# Display dataset statistics
print(f"\nDrones ({len(df['drone_code'].unique())}): {sorted(df['drone_code'].unique())}")
```

```
print(f"States ({len(df['state'].unique())}): {sorted(df['state'].unique())}")
print(f"Interferences ({len(df['interference'].unique())}): {sorted(df['interference'].unique())}")

# Create DRONES list for iteration
DRONES = sorted(df['drone_code'].unique())
STATES = sorted(df['state'].unique())
INTERFERENCES = sorted(df['interference'].unique())
```

```
Loaded 195 files

Drones (7): ['AIR', 'DIS', 'INS', 'MA1', 'MAV', 'MIN', 'PHA']
States (3): ['FY', 'HO', 'ON']
Interferences (2): ['BOTH', 'CLEAN']
```

2. Load Dataset Metadata

Load metadata for 195 RF recording files.

Parameters:

- Sampling frequency: 60 MHz
- Segment duration: 20 ms (1.2M samples) for feature extraction
- Visualization: 5 ms (300K samples) for plots
- Full recording: ~2 seconds (120M samples)

4. IQ Signal Visualization

Visualize IQ signal components and complex plane trajectory to understand temporal signal structure.

IQ signals encode amplitude (magnitude) and phase information. Phase changes reveal frequency modulation and Doppler effects.

In [6]:
```
# Select sample file (AIR drone, FY state, CLEAN interference)
sample_files = df[(df['drone_code'] == 'AIR') &
                  (df['state'] == 'FY') &
                  (df['interference'] == 'CLEAN')]
```

```
file_path = Path(sample_files.iloc[0]['file_path'])
iq_viz, time_axis = load_segment_for_viz(file_path)

# Compute components
real_part = np.real(iq_viz)
imag_part = np.imag(iq_viz)
magnitude = np.abs(iq_viz)
phase = np.angle(iq_viz)
phase_unwrapped = np.unwrap(phase)
phase_detrended = detrend_phase(phase_unwrapped)

# Create 4-panel plot
fig = make_subplots(
    rows=4, cols=1,
    subplot_titles=('Real Component (I)', 'Imaginary Component (Q)',
                    'Magnitude (Amplitude)', 'Phase (Detrended)'),
    vertical_spacing=0.08
)

# Real
fig.add_trace(go.Scatter(x=time_axis, y=real_part, mode='lines',
                         line=dict(color='blue', width=0.5), name='Real'),
              row=1, col=1)

# Imaginary
fig.add_trace(go.Scatter(x=time_axis, y=imag_part, mode='lines',
                         line=dict(color='orange', width=0.5), name='Imaginary'),
              row=2, col=1)

# Magnitude
fig.add_trace(go.Scatter(x=time_axis, y=magnitude, mode='lines',
                         line=dict(color='green', width=0.5), name='Magnitude'),
              row=3, col=1)

# Phase detrended
fig.add_trace(go.Scatter(x=time_axis, y=phase_detrended, mode='lines',
                         line=dict(color='red', width=0.5), name='Phase (detrended)'),
              row=4, col=1)

fig.update_xaxes(title_text="Time (µs)", row=4, col=1)
fig.update_yaxes(title_text="Amplitude", row=1, col=1)
```

```
fig.update_yaxes(title_text="Amplitude", row=2, col=1)
fig.update_yaxes(title_text="Amplitude", row=3, col=1)
fig.update_yaxes(title_text="Radians", row=4, col=1)

fig.update_layout(
    title=f"IQ Signal Components - {VIZ_MS} ms window",
    height=1000,
    showlegend=False
)

save_figure(fig, "IQ_Signal_Components_5_ms_window")
fig.show()
```

```
Saved: figures/01b_exploration_temporal_v7/IQ_Signal_Components_5_ms_window.png
```

In [7]:

```
# IQ trajectory in complex plane
n_points = len(iq_viz)
time_colors = np.arange(n_points)

fig = go.Figure()

fig.add_trace(go.Scatter(
    x=np.real(iq_viz),
    y=np.imag(iq_viz),
    mode='markers',
    marker=dict(
        size=2,
        color=time_colors,
        colorscale='Viridis',
        showscale=True,
        colorbar=dict(title="Time (sample)")
    ),
    name='IQ Trajectory'
))

fig.update_layout(
    title=f"IQ Trajectory in Complex Plane ({VIZ_MS} ms)",
    xaxis_title="I (Real)",
    yaxis_title="Q (Imaginary)",
    width=800,
    height=800,
```

```
    yaxis_scaleanchor="x"
)

save_figure(fig, "IQ_Trajectory_in_Complex_Plane")
fig.show()
```

```
Saved: figures/01b_exploration_temporal_v7/IQ_Trajectory_in_Complex_Plane.png
```

5. Temporal Feature Extraction

Extract discriminative temporal features for drone classification.

Justification Scientifique (Literature-Validated Features)

Suppression de la Burst Analysis

Contamination par interférences co-canal : les bursts Bluetooth/WiFi dans la bande 2.4 GHz sont indistinguables des bursts drone avec un seuil adaptatif simple.

Source : Detection of UAVs in Presence of Wi-Fi and Bluetooth Interference (IEEE 2019)

Suppression du PAPR et Crest Factor

Métriques conçues pour systèmes OFDM multi-porteuses. Les drones utilisent FHSS/DSSS pour lesquels ces métriques ne sont pas discriminantes.

Source : RF-Based Drone Detection: Survey (arXiv 2021)

Features Temporelles Validées

- **ZCR** : Indicateur de complexité de modulation (Drones Detection Using RF Features, MDPI 2024)
- **Phase Variance** : Capture instabilités oscillateur (Physical-Layer Identification Using RF-DNA, IEEE 2017)
- **Instantaneous Frequency** : Révèle patterns de modulation FM/FHSS (Automatic Modulation Classification, IEEE 2013)
- **I/Q Imbalance** : Signatures hardware uniques (I/Q Imbalance RF Fingerprinting, IEEE 2020)
- **Autocorrélation Multi-Lags** : Détecte périodicités protocolaires (Cyclostationary Feature Detection, IEEE 2005)

In [8]:
```python
def compute_snr_percentile(amplitude, noise_percentile=10):
    """Estimate SNR using noise floor percentile method."""
    noise_floor = np.percentile(amplitude, noise_percentile)
    signal_power = np.mean(amplitude ** 2)
    noise_power = noise_floor ** 2
    snr = signal_power / (noise_power + EPSILON_POWER)
    snr_db = 10 * np.log10(snr)
    return snr_db


def compute_autocorr_multilags(amplitude, lags=[1, 5, 10, 50, 100]):
    """Multi-lag autocorrelation features.

    Detects repetitive temporal patterns and protocol periodicities.
    Reference: IEEE 2005 Cyclostationary Feature Detection
    """
    if np.std(amplitude) < EPSILON_AMP:
        return {f'autocorr_lag{lag}': 1.0 for lag in lags}

    autocorr = {}
    for lag in lags:
        if len(amplitude) > lag:
            corr_coef = np.corrcoef(amplitude[:-lag], amplitude[lag:])[0, 1]
            autocorr[f'autocorr_lag{lag}'] = corr_coef
        else:
            autocorr[f'autocorr_lag{lag}'] = np.nan

    return autocorr
```

In [9]:
```python
def compute_phase_features(iq_segment):
    """Phase dynamics features.

    Captures oscillator instabilities and modulation characteristics.
    Reference: IEEE 2017 Physical-Layer Identification Using RF-DNA
    """
    phase = np.angle(iq_segment)
    phase_unwrapped = np.unwrap(phase)
    phase_derivative = np.diff(phase_unwrapped)

    return {
```

```
        'phase_variance': np.var(phase_unwrapped),
        'phase_derivative_std': np.std(phase_derivative),
        'phase_range': np.ptp(phase_unwrapped)
    }


def compute_instantaneous_freq(iq_segment):
    """Instantaneous frequency statistics from phase derivative.

    Reveals frequency modulation and hopping patterns.
    Reference: IEEE 2013 Automatic Modulation Classification
    """
    phase_unwrapped = np.unwrap(np.angle(iq_segment))
    inst_freq = np.diff(phase_unwrapped) * config.FS / (2 * np.pi)

    return {
        'inst_freq_mean': np.mean(inst_freq),
        'inst_freq_std': np.std(inst_freq),
        'inst_freq_range': np.ptp(inst_freq)
    }
```

In [10]:

```
def compute_iq_imbalance(iq_segment):
    """I/Q imbalance features for hardware fingerprinting.

    Measures gain and phase imbalances unique to each transmitter.
    Reference: IEEE 2020 I/Q Imbalance RF Fingerprinting
    """
    i_component = np.real(iq_segment)
    q_component = np.imag(iq_segment)

    # Gain imbalance (dB)
    gain_imbalance = 20 * np.log10(
        (np.std(i_component) + EPSILON_AMP) / (np.std(q_component) + EPSILON_AMP)
    )

    # Mean amplitude imbalance
    mean_imbalance = np.mean(np.abs(i_component)) - np.mean(np.abs(q_component))

    # DC offsets
    dc_offset_i = np.mean(i_component)
```

```
    dc_offset_q = np.mean(q_component)

    return {
        'iq_gain_imbalance_db': gain_imbalance,
        'iq_mean_imbalance': mean_imbalance,
        'dc_offset_i': dc_offset_i,
        'dc_offset_q': dc_offset_q
    }


def compute_envelope_variance(iq_segment):
    """Amplitude envelope variance.

    Wi-Fi bursts → high variance, Lightbridge continuous → low variance.
    """
    amplitude = np.abs(iq_segment)

    return {
        'envelope_variance': np.var(amplitude),
        'envelope_cv': np.std(amplitude) / (np.mean(amplitude) + EPSILON_AMP)
    }
```

In [11]:

```
def compute_temporal_features(iq_segment):
    """Extract validated temporal features for drone classification."""
    amplitude = np.abs(iq_segment)

    # Zero-crossing rates
    zcr_real = np.sum(np.diff(np.sign(np.real(iq_segment))) != 0) / len(iq_segment)
    zcr_imag = np.sum(np.diff(np.sign(np.imag(iq_segment))) != 0) / len(iq_segment)

    # SNR estimation
    snr_db = compute_snr_percentile(amplitude)

    # Phase dynamics
    phase_feats = compute_phase_features(iq_segment)

    # Instantaneous frequency
    inst_freq_feats = compute_instantaneous_freq(iq_segment)

    # I/Q imbalance
```

```
    iq_imbalance_feats = compute_iq_imbalance(iq_segment)

    # Multi-lag autocorrelation
    autocorr_feats = compute_autocorr_multilags(amplitude)

    # Envelope variance
    envelope_feats = compute_envelope_variance(iq_segment)

    return {
        'zcr_real': zcr_real,
        'zcr_imag': zcr_imag,
        'snr_db': snr_db,
        **phase_feats,
        **inst_freq_feats,
        **iq_imbalance_feats,
        **autocorr_feats,
        **envelope_feats
    }


# Define KEY_FEATURES list
KEY_FEATURES = [
    # Baseline temporal
    'zcr_real', 'zcr_imag', 'snr_db',

    # Phase dynamics
    'phase_variance', 'phase_derivative_std', 'phase_range',

    # Instantaneous frequency
    'inst_freq_mean', 'inst_freq_std', 'inst_freq_range',

    # I/Q imbalance
    'iq_gain_imbalance_db', 'iq_mean_imbalance', 'dc_offset_i', 'dc_offset_q',

    # Autocorrelation
    'autocorr_lag1', 'autocorr_lag5', 'autocorr_lag10', 'autocorr_lag50', 'autocorr_lag100',

    # Envelope
    'envelope_variance', 'envelope_cv'
]
```

```
print(f"Total features: {len(KEY_FEATURES)}")
```

```
Total features: 20
```

In [12]:
```
import traceback

# Stratified sampling
sample_files = []
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:
                n_sample = min(N_FILES_PER_COMBO, len(combo_files))
                sample_files.append(
                    combo_files.sample(n=n_sample, random_state=RANDOM_STATE)
                )

sample_files = pd.concat(sample_files, ignore_index=True)
print(f"Sampled {len(sample_files)} files for feature extraction")
print(f"Expected segments: {len(sample_files) * N_SEGMENTS_PER_FILE}")
```

```
Sampled 78 files for feature extraction
Expected segments: 234
```

In [13]:
```
# Extract features from all sampled files
from tqdm.auto import tqdm

features_list = []

for row in tqdm(sample_files.itertuples(), total=len(sample_files), desc="Processing files"):
    file_path = row.file_path
    drone_code = row.drone_code
    state = row.state
    interference = row.interference

    # Extract N_SEGMENTS_PER_FILE segments from each file
```

```
    for seg_idx in range(N_SEGMENTS_PER_FILE):
        try:
            start_sample = seg_idx * int(config.FS * SEGMENT_MS / 1000)
            iq_segment = load_iq_segment(file_path, start_sample, SEGMENT_MS)

            # Compute features
            features = compute_temporal_features(iq_segment)
            features['drone_code'] = drone_code
            features['state'] = state
            features['interference'] = interference
            features['file_path'] = file_path
            features['segment_idx'] = seg_idx

            features_list.append(features)

        except Exception as e:
            print(f"Error processing {file_path} segment {seg_idx}:")
            traceback.print_exc()
            continue

# Create DataFrame
features_df = pd.DataFrame(features_list)
print(f"\nExtracted features from {len(features_df)} segments")
print(f"Features shape: {features_df.shape}")
print(f"\nFeature columns: {list(features_df.columns)}")
```

```
Processing files: 100%|██████████| 78/78 [17:55<00:00, 13.79s/it]
Extracted features from 234 segments
Features shape: (234, 25)

Feature columns: ['zcr_real', 'zcr_imag', 'snr_db', 'phase_variance', 'phase_derivative_std', 'phase_range', 'inst_freq_mean',
'inst_freq_std', 'inst_freq_range', 'iq_gain_imbalance_db', 'iq_mean_imbalance', 'dc_offset_i', 'dc_offset_q', 'autocorr_lag1',
'autocorr_lag5', 'autocorr_lag10', 'autocorr_lag50', 'autocorr_lag100', 'envelope_variance', 'envelope_cv', 'drone_code', 'stat
e', 'interference', 'file_path', 'segment_idx']
```

In [14]:
```
# Display summary statistics by state
summary = features_df.groupby('state')[KEY_FEATURES].mean()
print("Feature means by flight state:")
print(summary.round(3))
```

```
Feature means by flight state:
       zcr_real  zcr_imag  snr_db  phase_variance  phase_derivative_std  \
state
FY        0.040     0.019   5.475    4.129291e+08                 0.272
HO        0.057     0.023   7.161    5.212698e+08                 0.307
ON        0.117     0.088   8.698    5.762061e+09                 0.548

         phase_range  inst_freq_mean  inst_freq_std  inst_freq_range  \
state
FY      39446.863281   -93499.968750     2597696.00       58718128.0
HO      43496.796875   -48499.320312     2929931.75       56259236.0
ON     156265.078125  -257108.625000     5235979.00       60109268.0

       iq_gain_imbalance_db  iq_mean_imbalance  dc_offset_i  dc_offset_q  \
state
FY                   -0.201             -0.008        0.013        0.021
HO                   -0.288             -0.007        0.011        0.019
ON                   -0.100             -0.008        0.014        0.023

       autocorr_lag1  autocorr_lag5  autocorr_lag10  autocorr_lag50  \
state
FY             0.851          0.691           0.647           0.541
HO             0.858          0.763           0.708           0.557
ON             0.750          0.534           0.510           0.441

       autocorr_lag100  envelope_variance  envelope_cv
state
FY               0.556              0.007        0.744
HO               0.571              0.004        0.918
ON               0.426              0.010        0.805
```

5b. Stationarity Analysis (ADF Test)

Validate that 20 ms segments are stationary using Augmented Dickey-Fuller test.

Decision rule: p-value < 0.05 → segment is stationary

This validates the use of mean/variance statistics in feature extraction. Reference: RF-UAVNet shows 50ms > 20ms > 10ms accuracy (89.4% vs 83.6% vs 76.9%)

In [15]:

```python
from statsmodels.tsa.stattools import adfuller

def test_stationarity_adf(iq_segment):
    """Test stationarity using ADF test on amplitude envelope."""
    amplitude = np.abs(iq_segment)
    # Subsample for speed (10k points sufficient)
    if len(amplitude) > 10000:
        amplitude = amplitude[::len(amplitude)//10000]
    result = adfuller(amplitude, autolag='AIC')
    return {
        'adf_statistic': result[0],
        'adf_pvalue': result[1],
        'is_stationary': result[1] < 0.05
    }

# Test multiple segment durations
WINDOW_SIZES_MS = [5, 10, 20, 50, 100]
stationarity_results = []

for drone in DRONES[:3]:  # Sample 3 drones
    file_path = df[(df['drone_code'] == drone) &
                   (df['state'] == 'FY') &
                   (df['interference'] == 'CLEAN')].iloc[0]['file_path']
    iq_full = data_loader.load_raw_iq(file_path)

    for window_ms in WINDOW_SIZES_MS:
        samples = int(window_ms * 1e-3 * config.FS)
        n_windows = min(50, len(iq_full) // samples)
        p_values = []

        for i in range(n_windows):
            segment = iq_full[i*samples:(i+1)*samples]
            result = test_stationarity_adf(segment)
            p_values.append(result['adf_pvalue'])

        stationarity_results.append({
            'drone': drone,
            'window_ms': window_ms,
            'mean_pvalue': np.mean(p_values),
```

```
            'pct_stationary': np.mean([p < 0.05 for p in p_values]) * 100
        })

stationarity_df = pd.DataFrame(stationarity_results)
pivot = stationarity_df.pivot_table(
    values='pct_stationary',
    index='drone',
    columns='window_ms'
)

print("% Stationary segments by window size:")
print(pivot.round(1))

# Visualize
fig = go.Figure()
for drone in pivot.index:
    fig.add_trace(go.Scatter(
        x=list(pivot.columns),
        y=pivot.loc[drone],
        name=drone,
        mode='lines+markers'
    ))
fig.add_hline(y=95, line_dash="dash", line_color="red",
              annotation_text="95% threshold")
fig.update_layout(
    title="Stationarity by Segment Duration",
    xaxis_title="Window size (ms)",
    yaxis_title="% Stationary segments",
    height=400
)
save_figure(fig, "stationarity_by_window_size")
fig.show()
```

```
% Stationary segments by window size:
window_ms    5      10     20     50     100
drone
AIR         78.0   96.0   98.0  100.0  100.0
DIS        100.0  100.0  100.0  100.0  100.0
INS         86.0   90.0   98.0  100.0  100.0
Saved: figures/01b_exploration_temporal_v7/stationarity_by_window_size.png
```

In [16]:
```python
# Verify NaN values in features
nan_counts = features_df[KEY_FEATURES].isna().sum()
nan_features = nan_counts[nan_counts > 0]

if len(nan_features) > 0:
    print("WARNING: NaN values detected in features:")
    print(nan_features)
    print(f"\nTotal segments with NaN: {features_df[KEY_FEATURES].isna().any(axis=1).sum()}")
else:
    print("No NaN values detected in features.")
```

```
No NaN values detected in features.
```

6. Flight Mode Discrimination

Analyze which temporal features separate ON, HO, and FY flight states.

In [17]:
```python
# Select top discriminative features for visualization
selected_features = ['zcr_real', 'phase_variance', 'inst_freq_std', 'envelope_variance']

fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=selected_features,
    vertical_spacing=0.12,
    horizontal_spacing=0.1
)

for idx, feature in enumerate(selected_features):
    row = idx // 2 + 1
    col = idx % 2 + 1

    for state in STATES:
        state_data = features_df[features_df['state'] == state][feature]

        fig.add_trace(
            go.Box(y=state_data, name=state, showlegend=(idx == 0)),
            row=row, col=col
        )
```

```
fig.update_layout(
    title="Temporal Features by Flight Mode",
    height=800,
    showlegend=True
)

save_figure(fig, "Temporal_Features_by_Flight_Mode")
fig.show()
```

```
Saved: figures/01b_exploration_temporal_v7/Temporal_Features_by_Flight_Mode.png
```

7. Cross-Drone Temporal Signatures

Compare temporal signatures across drone models.

In [18]:
```
# Amplitude envelope comparison across drones (FY state, CLEAN interference)
state = 'FY'
interference = 'CLEAN'

fig = go.Figure()

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

    if len(drone_files) > 0:
        file_path = Path(drone_files.iloc[0]['file_path'])
        iq_viz, time_axis = load_segment_for_viz(file_path)

        fig.add_trace(go.Scatter(
            x=time_axis,
            y=np.abs(iq_viz),
            mode='lines',
            name=drone,
            line=dict(width=0.8)
        ))

fig_title = f"Amplitude Envelope Comparison - State {state} - Interference {interference}"
```

```
fig.update_layout(
    title=f"{fig_title} ({VIZ_MS} ms)",
    xaxis_title="Time (µs)",
    yaxis_title="Amplitude",
    height=600,
    hovermode='x unified'
)

save_figure(fig, f"Amplitude_Envelope_Comparison_State_{state}_Interference_{interference}")
fig.show()
```

```
Saved: figures/01b_exploration_temporal_v7/Amplitude_Envelope_Comparison_State_FY_Interference_CLEAN.png
```

8. Interference Impact

Assess robustness of temporal features to Bluetooth and WiFi interference.

In [19]:
```
# Compare features between CLEAN and BOTH interference conditions
selected_features = ['zcr_real', 'snr_db', 'phase_variance', 'inst_freq_std']

fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=selected_features,
    vertical_spacing=0.12,
    horizontal_spacing=0.1
)

for idx, feature in enumerate(selected_features):
    row = idx // 2 + 1
    col = idx % 2 + 1

    for interference in INTERFERENCES:
        int_data = features_df[features_df['interference'] == interference][feature]

        fig.add_trace(
            go.Box(y=int_data, name=interference, showlegend=(idx == 0)),
            row=row, col=col
        )
```

```
fig.update_layout(
    title="Temporal Features - CLEAN vs BOTH Interference",
    height=800,
    showlegend=True
)

save_figure(fig, "Temporal_Features_CLEAN_vs_BOTH_Interference")
fig.show()
```

```
Saved: figures/01b_exploration_temporal_v7/Temporal_Features_CLEAN_vs_BOTH_Interference.png
```