Snapping Shrimp Acoustic Analysis Guide: Detection and Multi-Site Comparison

Margherita Silvestri

October 22, 2025

Contents

1 Introduction

1.1 File Information

- **Detection Script:** `Detect_and_plot_with_filter.py`
- **Analysis Script:** `temporal_analysis.py`
- **Author:** Margherita Silvestri
- **Date:** October 7, 2025 (Detector), October 8, 2025 (Analysis)

1.2 Workflow Overview

This comprehensive guide documents a complete acoustic analysis pipeline for snapping shrimp sound detections and visualizations:

1. **Detection Phase:** Identify individual clicks in underwater recordings using dual-threshold envelope detection
2. **Preprocessing:** Normalize audio, remove noise, and compute signal envelope
3. **Temporal Analysis:** Compute activity rates and aggregate results across multiple recording sites
4. **Visualization:** Generate interactive plots showing diurnal patterns and spatial differences

Part I
Phase 1: Click Detection

2 Workflow Summary

2.1 Overview

This script implements an automated acoustic detection pipeline for identifying snapping shrimp clicks in underwater recordings using a dual-threshold method combining envelope and derivative detection. The workflow begins by reading WAV files, converting stereo recordings to mono, and normalizing the amplitude to the $[-1, 1]$ range to standardize signals across different recording gains and hydrophone sensitivities.

2.2 Key Processing Steps

2.2.1 DC Offset Removal

DC offset (constant electrical bias from analog-to-digital converters) is removed by subtracting the mean amplitude, which improves subsequent filter performance and envelope calculation accuracy.

2.2.2 High-Pass Filtering

A Butterworth high-pass filter with 2 kHz cutoff removes low-frequency anthropogenic and environmental noise while preserving the characteristic high-frequency energy of snapping shrimp clicks, which exhibit peak frequencies between 2–5 kHz with broadband energy extending beyond 100 kHz (Song et al., 2023).

2.2.3 Envelope and Derivative Detection

The snapping shrimp detector operates in two main stages to isolate impulsive clicks from surrounding noise. First, it computes the envelope of the acoustic signal using the Hilbert transform, which represents the instantaneous amplitude or loudness of the waveform over time:

$$\text{Envelope}(t) = \sqrt{x(t)^2 + \mathcal{H}\{x(t)\}^2}$$

where $x(t)$ is the raw audio signal and $\mathcal{H}\{x(t)\}$ is its Hilbert transform. The resulting envelope highlights sharp intensity peaks corresponding to shrimp snaps.

Next, the detector measures how rapidly the envelope changes by taking its temporal derivative:

$$\frac{d\text{Envelope}}{dt}$$

The derivative emphasizes sudden amplitude increases characteristic of impulsive biological sounds. Real snapping shrimp clicks exhibit both high envelope amplitude and steep

onset slopes, while background sounds such as wave noise or engine tend to vary more gradually and show lower derivative values.

To minimize false positives, a candidate event is marked as a click only when both the envelope and its derivative exceed their respective thresholds simultaneously. Each threshold is calculated adaptively according to the signal statistics:

$$\text{Threshold} = \text{mean} + 2.5 \times \text{standard deviation}$$

where $k = 2.5$ for both envelope and derivative. This value was selected based on two complementary approaches. First, 2.5 standard deviations is a classical choice in signal processing, corresponding to approximately the 99.4th percentile of a normal distribution and effectively excluding transient noise fluctuations. Second, this choise was validated by testing various threshold combinations (ranging from 2.0 to 3.0 standard deviations) on acoustic recordings from multiple deployment sites. This formulation allows the detector to automatically adjust to varying acoustic environments, maintaining sensitivity to shrimp clicks under different noise conditions.

2.2.4 Implementation Details

In practice, the algorithm derives the Hilbert transform envelope to capture instantaneous amplitude fluctuations (Adam 2006; Au and Banks 1998Beslin et al., 201; Bohnenstiehlet al., 2016; Chang and Wang 2003) and computes its first derivative to quantify onset sharpness. Both measures are evaluated against adaptive statistical thresholds defined as:

$$\text{Threshold} = \text{mean} + (2.5 \times \text{standard deviation})$$

A click is detected only when the envelope and its derivative concurrently exceed their thresholds, ensuring robust detection. This dual-threshold framework provides:

- **Biological specificity:** True shrimp snaps present short-duration, high-derivative amplitude spikes.
- **Noise robustness:** Non-impulsive sources such as waves and distant vessels show sustained amplitude with lower derivative magnitude.
- **Automatic adaptation:** Thresholds self-scale to local noise statistics, reduce the need for manual calibration or site-specific tuning.

2.2.5 Refractory Period

A 20 ms minimum inter-click interval (refractory period) was porposed to prevent multiple detections of the same snap event and suppress surface-reflected echoes, which occur within 20 ms in shallow water environments (Bibikov and Makushevich, 2020). Here 30 ms will be used after manual revision of the results obtained with different intervals.

3 Output Files

Once finished, detection will be saved in one output folder including:

3.1 Visualization Plots

Plots for first visualizations of some parameters will include:

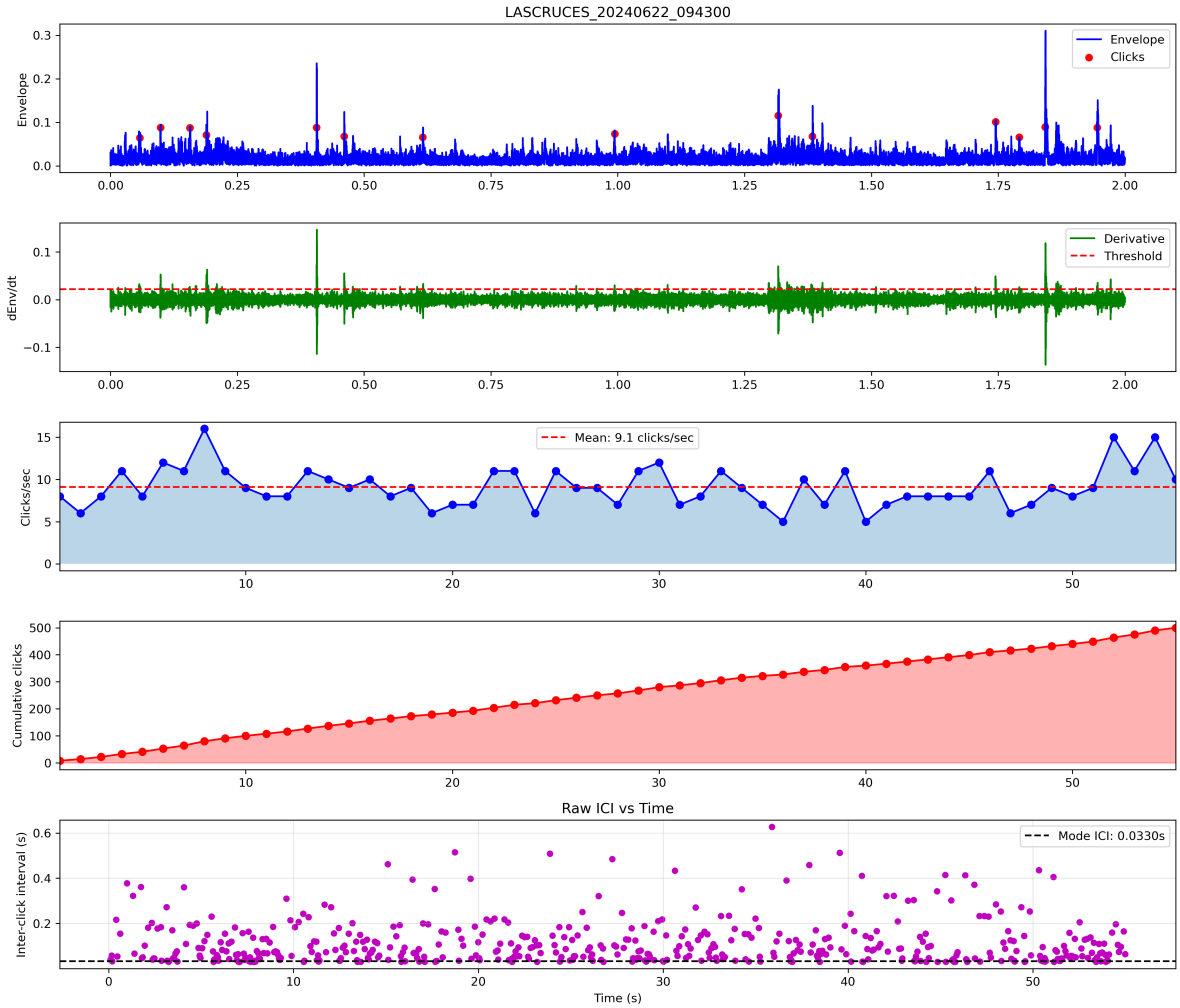

Figure 1: Five-panel diagnostic output showing: Panel 1 (top) - Envelope + detected clicks for first 2 seconds, Panel 2 - Derivative + threshold line, Panel 3 - Clicks per second, Panel 4 - Cumulative clicks over time, Panel 5 (bottom) - Inter-click interval (ICI) scatter plot with mode line indicating the most common inter-click interval duration.

3.1.1 Panel Descriptions

1. **Panel 1:** Envelope + detected clicks (first 2 seconds) - Blue line shows the Hilbert envelope of the filtered signal, with detected clicks marked as red dots

2. **Panel 2:** Derivative + threshold line - Green line shows the derivative of the envelope, red dashed line indicates the detection threshold

3. **Panel 3:** Clicks per second - Temporal distribution showing click rate across the entire recording

4. **Panel 4:** Cumulative clicks - Running sum of total detections showing activity progression over time

5. **Panel 5:** ICI scatter plot + mode line - Scatter plot showing inter-click intervals over time, with dashed line indicating the mode (most frequent interval)

3.2 Detection Files

A .txt file with all detections including:

- All detected clicks with their label (e.g., click_001, click_002, etc.)
- Start and End time of each detected click
- The score of each detected click (signal amplitude when click has been detected)

The score can be used together with manual annotations to generate an ROC curve to quantitatively test detector quality.

4 Visualization in Audacity

To visualize and check detected clicks, follow these steps:

1. Open the free software Audacity to visualize recordings in WAV format
2. Load audio file in the following way: File → Import → Audio, or directly drag the WAV file
3. Use multiview to visualize both waveform and spectrogram as follows (this will allow better click visualization to check detections)

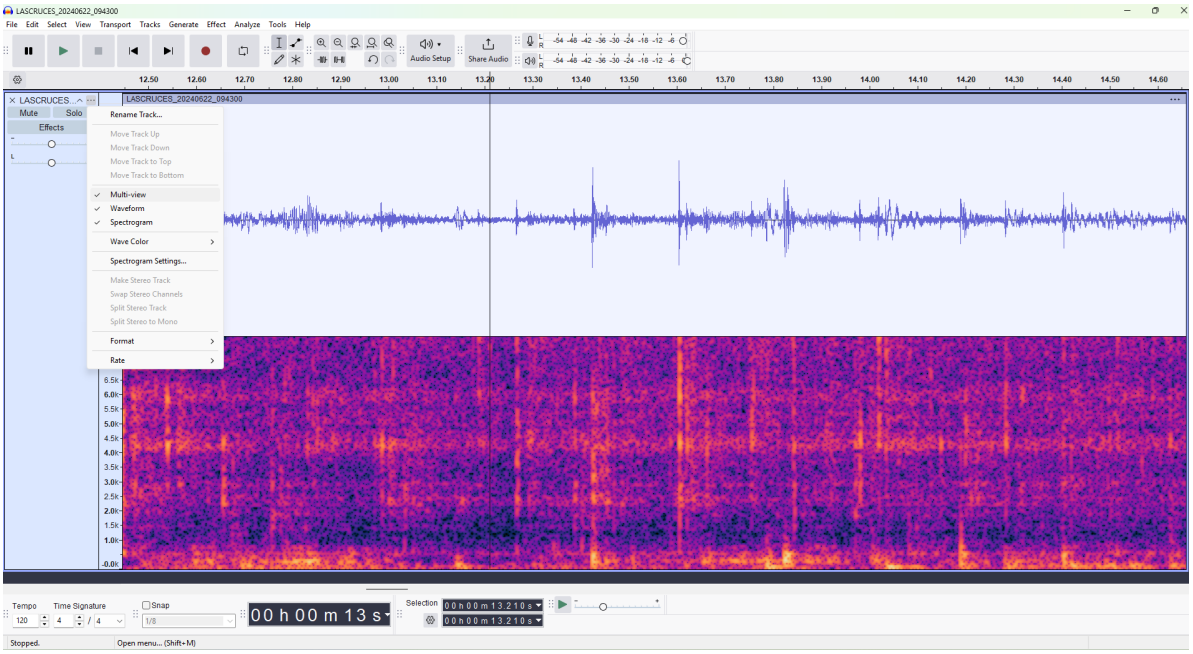

Figure 2: Audacity multiview showing simultaneous waveform (top) and spectrogram (bottom) visualization. This dual-view approach enables better recording visualization. The spectrogram shows the high-frequency, impulsive nature of snapping shrimp clicks as bright, narrow vertical lines , while the waveform provides temporal alignment for precise timing analysis.

4. Open .txt file as labels in the following way: File → Import → Labels → look for the corresponding file in the output folder

5 Parameter Comparison

You can upload multiple .txt files to compare different detectors or the effect that changing some parameters has on the detector. Below is one example that show differences in detections between using inter-click interval of 30 ms, 20 ms, or not using interval at all. Manual checks provided that the first approach (final version of the detector) avoids detecting reflections and echoes.

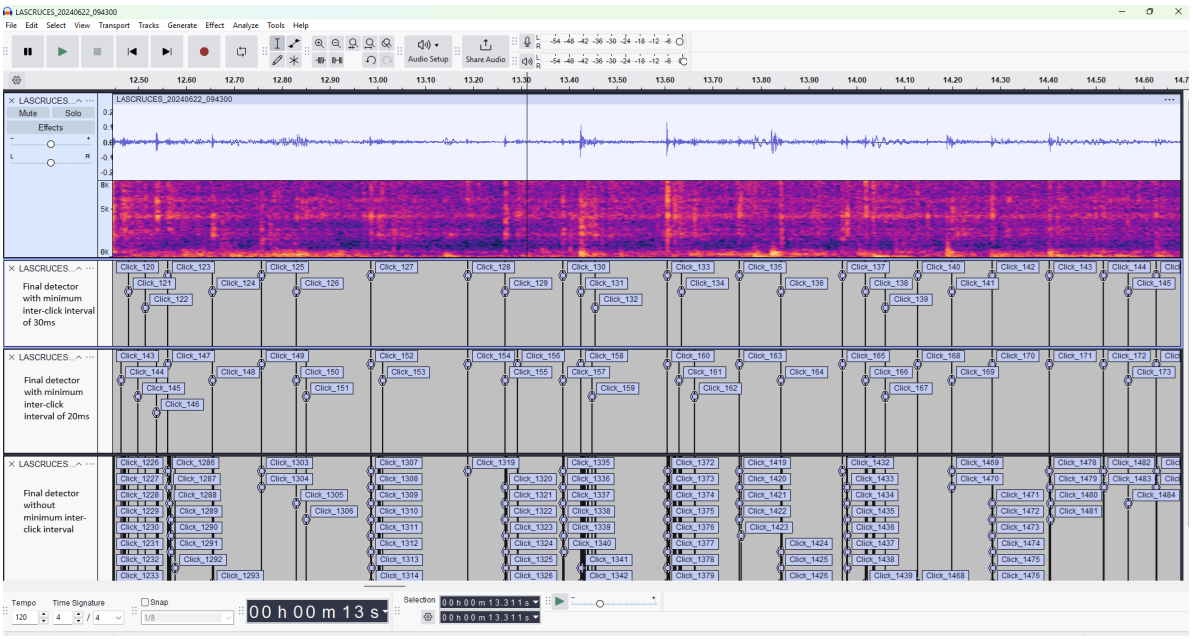

Figure 3: Comparison of detector output using different minimum inter-click interval parameters. Three detection scenarios are compared: (1) Minimum interval = 30 ms (recommended), (2) Minimum interval = 20 ms, and (3) No refractory period, allowing all detections. The 30 ms refractory period effectively suppresses false positives from surface reflections and multiple detections of single snap events while maintaining sensitivity to true click events. Manual verification confirms that the 30 (and also 40 ms) approach best balances sensitivity and specificity, eliminating reflection-induced false positives.

6 Required Packages

To run this script, create a conda environment and install the following packages:

```
conda create -n snaprate python=3.9 numpy scipy matplotlib
conda activate snaprate
```

6.1 Import Statements

```
import os
import glob
import numpy as np
import scipy.io.wavfile as wav
from scipy.signal import hilbert, butter, filtfilt
import matplotlib.pyplot as plt
```

7 Function Descriptions

7.1 Function 1: Safe WAV Reading with Error Handling

```
def safe_read_wav(path):
    try:
        return wav.read(path)
    except ValueError as e:
        print(f"Warning: Skipping unsupported file: {os.path.
           basename(path)}")
        return None, None
    except Exception as e:
        print(f"Error reading {os.path.basename(path)}: {e}")
        return None, None
```

7.2 Function 2: High-Pass Filter (Butterworth, Zero-Phase)

```
def highpass_filter(sig, sr, cutoff=2000, order=4):
    nyq = 0.5 * sr
    norm_cutoff = cutoff / nyq
    b, a = butter(order, norm_cutoff, btype='high')
    return filtfilt(b, a, sig)
```

7.3 Function 3: Main Click Detection and Plotting

7.3.1 Function Signature

```
def detect_and_plot_with_ici(wav_path, output_dir=None,
   env_thresh_factor=2.5,
                             der_thresh_factor=2.5, t_min=0.03):
```

7.3.2 Processing Pipeline

1. Read and preprocess audio (mono conversion, normalization, DC removal)
2. High-pass filter (> 2 kHz) to remove low-frequency noise
3. Compute Hilbert envelope and derivative to track amplitude changes
4. Set adaptive thresholds based on signal statistics
5. Detect clicks using dual thresholds + refractory period
6. Save results (labels, scores) and generate 5-panel diagnostic figure

7.3.3 Parameters

- **wav_path** (str): Full path to input WAV file
- **output_dir** (str, optional): Output directory (default: current working directory)
- **env_thresh_factor** (float, optional): Envelope threshold multiplier (default: 2.5)
  - Formula: threshold = mean + factor × std
- **der_thresh_factor** (float, optional): Derivative threshold multiplier (default: 2.5)
- **t_min** (float, optional): Minimum inter-click interval in seconds (default: 0.04 = 40 ms)

7.3.4 Processing Steps

```
out = output_dir or os.getcwd()
os.makedirs(out, exist_ok=True)
base = os.path.splitext(os.path.basename(wav_path))[0]

labels_file = os.path.join(out, f"{base}.labels.txt")
scores_file = os.path.join(out, f"{base}.scores.txt")
fig_file = os.path.join(out, f"{base}_overview.png")
```

```
sr, data = safe_read_wav(wav_path)
if data is None:
    return
```

```
if data.ndim > 1:
    data = data[:, 0]
```

```
data = data.astype(np.float32)
max_amplitude = np.max(np.abs(data))
data /= (max_amplitude + 1e-10)
```

Purpose: Standardize signal amplitude across recordings with different recording gains, hydrophone sensitivities, source distances, and bit depths (16-bit, 24-bit, 32-bit).

```
dc_offset = np.mean(data)
sig = data - dc_offset
```

Purpose: Eliminate constant bias introduced by ADC/preamp electronics, sensor drift, and environmental electrical interference. Effect: Improves filter performance and envelope calculation accuracy.

```
sig = highpass_filter(sig, sr, cutoff=2000, order=4)
```

Purpose: Remove low-frequency noise sources (ocean waves, vessel engines, wind/flow noise). Rationale: Shrimp clicks have peak energy at 2–20 kHz.

```
env = np.abs(hilbert(sig))
denv = np.diff(env, prepend=env[0])
```

Hilbert Transform Method creates analytic signal $z(t) = x(t) + j \cdot H\{x(t)\}$, enabling envelope $= |z(t)|$ to track instantaneous amplitude and derivative $= d|z(t)|/dt$ to highlight rapid amplitude rises.

```
e_th = env.mean() + env.std() * env_thresh_factor
d_th = denv.mean() + denv.std() * der_thresh_factor
```

Adaptive Method uses threshold = mean + factor × std to automatically scale thresholds to local noise statistics.

```
min_samp = int(t_min * sr)
```

Refractory Period prevents detecting the same click multiple times.

```
clicks = []
clicks_scores = []
last_click_idx = -min_samp

for i in range(len(denv)):
    if (env[i] > e_th and
        denv[i] > d_th and
        (i - last_click_idx >= min_samp)):
        clicks.append(i / sr)
        clicks_scores.append(env[i])
        last_click_idx = i

```

```
clicks = np.array(clicks)
clicks_scores = np.array(clicks_scores)
```

Detection Logic: (1) Envelope exceeds threshold, (2) Derivative exceeds threshold, (3) Sufficient time since last detection.

```
with open(labels_file, 'w') as f, open(scores_file, 'w') as sf:
    f.write("# start\tend\tlabel\tscore\n")
    sf.write("# time\tscore\tlabel\n")

    for idx, (t, score) in enumerate(zip(clicks, clicks_scores), 1):
        f.write(f"{t:.6f}\t{t+0.001:.6f}\tClick_{idx}\t{score:.6f}\n
            ")
        sf.write(f"{t:.6f}\t{score:.6f}\tClick_{idx}\n")
```

Output Format: labels.txt (Audacity-compatible) and scores.txt (ROC analysis format).

Step 12: Generate Diagnostic Plots The script creates a figure with five panels displaying envelope analysis, derivative response, temporal click distribution, cumulative count, and inter-click interval statistics.

8 Main Execution Block

8.1 Batch Processing Workflow

```
if __name__ == "__main__":
    input_folder = r"C:\Users\margh\Desktop\RECORDINGSMARFUTURA\
        RECORDINGS\LASCRUCES"

    output_folder = os.path.join(input_folder, "output_all_8")
    os.makedirs(output_folder, exist_ok=True)

    wav_files = glob.glob(os.path.join(input_folder, "*.wav"))

    for wav_path in wav_files:
        base = os.path.splitext(os.path.basename(wav_path))[0]

        out_dir = os.path.join(output_folder, base)
        os.makedirs(out_dir, exist_ok=True)

        detect_and_plot_with_ici(
            wav_path=wav_path,
            output_dir=out_dir,
            env_thresh_factor=2.5,
```

```
            der_thresh_factor=2.5,
            t_min=0.03
        )
```

8.2 Execution Notes

This block scans input folder for all WAV files, creates output subdirectories for each file, and runs detection on each file with specified parameters.

9 Parameter Recommendations

The minimum inter-click interval (t_min) should be adjusted based on recording conditions:

- **Rocky substrate and shallow waters (depth $>$ 15 meters):** Use 30 ms or 40 ms to avoid detecting reflections
- **Low noise environments:** May use 20 ms to detect very rapid clicks
- **High noise environments:** Increase to 50 ms or higher to suppress reflections and echo-induced false positives

Part II
Phase 2: Temporal Analysis and Multi-Site Comparison

10 Overview

After generating recording detection results, the next step is to analyze temporal patterns of snapping shrimp activity across multiple recording sites. This section covers Python code for computing clicks per minute from detector output and creating interactive visualizations to identify daily patterns, diurnal rhythms, and spatial differences in shrimp activity.

10.1 Scientific Rationale

Snapping shrimp exhibit strong temporal patterns influenced by tidal cycles, diel (day-night) behavior, and environmental factors (Song et al., 2023; Jeong & Paeng 2022; Lee et al., 2021; Bibikov & Makushevich 2020; Butler et al., 2017; Bohnenstiehl et al., 2016). By aggregating clicks per minute across recordings and comparing across sites, researchers can detect diurnal behavioral patterns, identify seasonal or tidal trends, compare acoustic activity between habitat types and locations, validate detector performance across different environmental conditions, and generate time-series for long-term monitoring assessments.

11 Multi-Site Click Rate Analysis

11.1 Required Packages for Temporal Analysis

Install the required packages in your conda environment:

```
conda install pandas plotly
```

Import statements for the temporal analysis workflow:

```
import os
import glob
import pandas as pd
import numpy as np
import plotly.graph_objects as go
import re
```

11.2 Configuration Section

The first step in temporal analysis is to define your recording sites, assign consistent colors for visualization, and specify file paths.

11.2.1 Step 1: Define Sites and Colors

```
# Define recording sites and assign consistent visualization colors
SITES = ["LASCRUCES", "VENTANAS", "ZAPALLAR"]
SITE_COLORS = {
    "LASCRUCES": "crimson",
    "VENTANAS": "seagreen",
    "ZAPALLAR": "royalblue"
}
```

Using consistent colors across all plots improves figure interpretation and makes it easy to identify sites at a glance.

11.2.2 Step 2: Set Base and Output Directories

```
# Set base directory containing all recordings
ROOT_BASE = r"C:\Users\margh\Desktop\RECORDINGSMARFUTURA\RECORDINGS"

# Create output directory for interactive plots
OUTPUT_DIR = r"C:\Users\margh\Desktop\RECORDINGSMARFUTURA\plots\
   interactive\ALL_SITES_HOURLY_ROLLING"
os.makedirs(OUTPUT_DIR, exist_ok=True)
```

11.2.3 Step 3: Set Global Y-Axis Limits

```
# Global y-axis limits ensure consistency across all plots
GLOBAL_Y_MIN = 50
GLOBAL_Y_MAX = 800
```

Setting consistent axis limits enables direct visual comparison of activity levels across sites and time periods. Adjust these values based on your data distribution.

11.3 Load and Aggregate Detection Results

The workflow loads per-recording detection files and computes clicks per minute for each recording.

11.3.1 Processing Workflow

```
# Initialize empty list to store records
records = []
for site in SITES:
    folder_root = os.path.join(ROOT_BASE, site, "output_all_3")
    folders = glob.glob(os.path.join(folder_root, "*"))
    for folder in folders:
        base = os.path.basename(folder)
        parts = base.split("_")
```

```
        if len(parts) < 3:
            continue
        date_str = parts[1]
        time_str = re.sub(r'\D', '', parts[2])
        if len(time_str) != 6:
            continue
        try:
            dt = pd.to_datetime(date_str + time_str, format="%Y%m%d%H%M%S")
        except Exception:
            continue

        labels_file = os.path.join(folder, f"{base}.labels.txt")
        if not os.path.exists(labels_file):
            continue

        starts = []
        with open(labels_file) as f:
            for line in f:
                if line.startswith("#") or not line.strip():
                    continue
                try:
                    starts.append(float(line.split("\t")[0]))
                except Exception:
                    pass

        cpm = len(starts) / ((starts[-1] - starts[0]) / 60) if len(starts) > 1 else 0
        records.append({
            "site": site,
            "datetime": dt,
            "clicks_per_min": cpm,
            "recording": base
        })

# Create pandas DataFrame
df = pd.DataFrame(records)
if df.empty:
    raise RuntimeError("No data loaded for any site.")
```

11.3.2 Key Concepts

- **Clicks per minute (CPM):** Normalized activity metric computed as total clicks divided by recording duration in minutes
- **DateTime parsing:** Folder names encode recording timestamps; parsing enables temporal analysis

- **DataFrame structure:** Each row represents one recording with site, timestamp, activity rate, and metadata
- **Regular expressions:** Pattern matching removes non-numeric characters from time strings for robust parsing

12 Interactive Full Time-Series Visualization

12.1 Creating the Interactive Time-Series Plot

The full time-series plot displays clicks per minute over the entire monitoring period for all sites simultaneously, with nighttime shading to highlight diel behavioral patterns.

```
# Create new Plotly figure
fig = go.Figure()

# Add one Scatter trace per site
for site, grp in df.groupby("site"):
    fig.add_trace(go.Scatter(
        x=grp["datetime"],
        y=grp["clicks_per_min"],
        mode="markers",
        marker=dict(color=SITE_COLORS[site], size=6, opacity=0.7),
        name=site,
        text=grp["recording"],
        hovertemplate=(
            f"<b>{site}</b><br>%{{x}}<br>Clicks/min: %{{y:.1f}}"
            "<br>Recording: %{{text}}<extra></extra>"
        )
    ))

# Add nighttime shading (20:00-06:00)
shapes = []
for d in df["datetime"].dt.normalize().unique():
    # Night period 1: 00:00-06:00
    shapes.append(dict(
        type="rect", xref="x", yref="paper",
        x0=d, x1=d + pd.Timedelta(hours=6),
        y0=0, y1=1, fillcolor="navy",
        opacity=0.1, layer="below", line_width=0
    ))
    # Night period 2: 20:00-24:00
    shapes.append(dict(
        type="rect", xref="x", yref="paper",
        x0=d + pd.Timedelta(hours=20),
        x1=d + pd.Timedelta(days=1),
        y0=0, y1=1, fillcolor="navy",
```

```
            opacity=0.1, layer="below", line_width=0
        ))

# Update layout with titles, axes, and styling
fig.update_layout(
    title="Snapping Shrimp Clicks: All Sites Full Time Series",
    xaxis_title="Date & Time",
    yaxis_title="Clicks per minute",
    yaxis=dict(range=[GLOBAL_Y_MIN, GLOBAL_Y_MAX]),
    shapes=shapes,
    width=1400,
    height=700
)

# Save interactive plot as HTML
fig.write_html(os.path.join(OUTPUT_DIR, "all_sites_full_ts.html"))
print("Saved all-sites full time series with site colors.")
```

12.1.1 Plot Interpretation

- **Marker colors:** Different colors represent different recording sites for easy discrimination
- **Marker density:** Clustered high-density points indicate consistent high activity periods
- **Nighttime shading:** Navy blue rectangular overlays show 20:00–06:00 night periods
- **Interactive features:** Hover over points to display exact timestamp, site name, click rate, and recording identifier

12.1.2 Advantages of Interactive HTML Output

Interactive Plotly plots provide zoom and pan capabilities, ability to toggle sites via legend, hover tooltips with detailed information, and export options to save as PNG directly from browser.

13 Daily Summary Analysis

13.1 Computing Daily Mean Activity

After generating the full time-series, computing daily means reduces noise and highlights longer-term trends.

```
# Add 'date' column to DataFrame
df["date"] = df["datetime"].dt.date

```

```
# Compute daily mean clicks per minute for each site
daily = df.groupby(["site", "date"])["clicks_per_min"].mean().
    reset_index()
```

13.2 Creating the Daily Summary Plot

```
# Create new Plotly figure for daily summary
fig2 = go.Figure()

# Add one Scatter trace per site (markers + lines)
for site, grp in daily.groupby("site"):
    fig2.add_trace(go.Scatter(
        x=grp["date"],
        y=grp["clicks_per_min"],
        mode="markers+lines",
        marker=dict(color=SITE_COLORS[site], size=8),
        name=site,
        hovertemplate=(
            f"<b>{site}</b><br>%{{x}}<br>"
            "Mean clicks/min: %{y:.1f}<extra></extra>"
        )
    ))

# Update layout and save
fig2.update_layout(
    title="Snapping Shrimp Clicks: All Sites Daily Means",
    xaxis_title="Date",
    yaxis_title="Mean clicks per minute",
    yaxis=dict(range=[GLOBAL_Y_MIN, GLOBAL_Y_MAX]),
    width=1000,
    height=500
)
fig2.write_html(os.path.join(OUTPUT_DIR, "all_sites_daily_summary.
    html"))
print("Saved all-sites daily summary.")
```

13.2.1 Daily Summary Interpretation

The daily summary plot provides several insights: (1) Trend lines reveal temporal trends across weeks or months, (2) Site comparison shows similar or different environmental conditions, (3) Reduced variance smooths out fine-scale fluctuations, (4) Anomaly detection identifies unusually high or low activity days.

14 Hourly Rolling Mean Analysis

14.1 Purpose and Rationale

The hourly rolling mean provides an intermediate temporal scale between individual recordings and daily aggregation, revealing tidal cycles, circadian rhythms, and weather-driven changes that operate at sub-daily time-scales.

```
# Compute hourly averages
df["hour"] = df["datetime"].dt.floor("H")
hourly = df.groupby(["site", "hour"]).agg({
    "clicks_per_min": "mean"
}).reset_index()

# Compute rolling mean (window = 3 hours)
hourly["rolling_cpm"] = hourly.groupby("site")["clicks_per_min"].
   transform(
    lambda x: x.rolling(window=3, center=True, min_periods=1).mean()
)
```

14.2 Creating the Hourly Rolling Mean Plot

```
# Create figure with hourly rolling mean
fig3 = go.Figure()

for site, grp in hourly.groupby("site"):
    fig3.add_trace(go.Scatter(
        x=grp["hour"],
        y=grp["rolling_cpm"],
        mode="lines+markers",
        marker=dict(color=SITE_COLORS[site], size=6, opacity=0.7),
        line=dict(width=2),
        name=f"{site} (3-hour rolling mean)",
        hovertemplate=(
            f"<b>{site}</b><br>%{{x}}<br>"
            "Rolling mean CPM: %{y:.1f}<extra></extra>"
        )
    ))

# Add nighttime shading
shapes_rolling = []
for d in hourly["hour"].dt.normalize().unique():
    shapes_rolling += [
        dict(type="rect", xref="x", yref="paper",
             x0=d, x1=d + pd.Timedelta(hours=6), y0=0, y1=1,
             fillcolor="navy", opacity=0.1, layer="below",
                line_width=0),
```

```
        dict(type="rect", xref="x", yref="paper",
             x0=d + pd.Timedelta(hours=20), x1=d + pd.Timedelta(days=1),
             y0=0, y1=1, fillcolor="navy", opacity=0.1, layer="below", line_width=0)
    ]

# Update layout
fig3.update_layout(
    title="Snapping Shrimp Clicks: Hourly Rolling Mean Across Sites (3-hour window)",
    xaxis_title="Date & Hour",
    yaxis_title="Rolling mean clicks per minute",
    yaxis=dict(range=[GLOBAL_Y_MIN, GLOBAL_Y_MAX]),
    shapes=shapes_rolling,
    width=1400,
    height=700
)

# Save interactive plot
fig3.write_html(os.path.join(OUTPUT_DIR, "all_sites_hourly_mean_rolling.html"))
print("Saved all-sites hourly rolling mean plot.")
```

14.2.1 Rolling Mean Interpretation

The 3-hour rolling mean provides several advantages:

- **Noise reduction:** Smooths recording-to-recording variability while preserving sub-daily temporal structure
- **Tidal signals:** Semi-diurnal tidal forcing (approximately 12.4 hours) produces modulation visible at this time-scale
- **Circadian rhythm:** 24-hour behavioral cycles become apparent in the rolling mean amplitude
- **Event detection:** Transient increases or decreases (storms, disturbances) appear as coherent spikes across multiple sites

14.3 Customization Options

To adapt this workflow to your specific project:

- **Modify site list:** Edit the SITES list to include your recording locations
- **Adjust colors:** Change color names in SITE_COLORS dictionary

- **Update paths:** Modify ROOT_BASE and OUTPUT_DIR to match your file organization
- **Change y-axis range:** Adjust GLOBAL_Y_MIN and GLOBAL_Y_MAX based on your data
- **Modify night hours:** Edit hour values (currently 20:00–06:00) to match your study site
- **Adjust rolling window:** Change window=3 to other values (1, 7, 24) for different smoothing effects

14.4 Expected Output Files

Upon execution, the complete code generates three interactive HTML files:

- **all_sites_daily_summary.html** - Daily mean activity plot for long-term trend visualization
- **all_sites_full_ts.html** - Full time-series plot with hourly resolution and nighttime shading
- **all_sites_hourly_mean_rolling.html** - Hourly rolling mean with 3-hour smoothing window

To view these files: (1) Locate HTML files in OUTPUT_DIR, (2) Open directly in web browser, (3) Interact using zoom, pan, legend toggle, and hover features.

15 Interactive Plots as Static Images

15.1 Daily Summary Plot

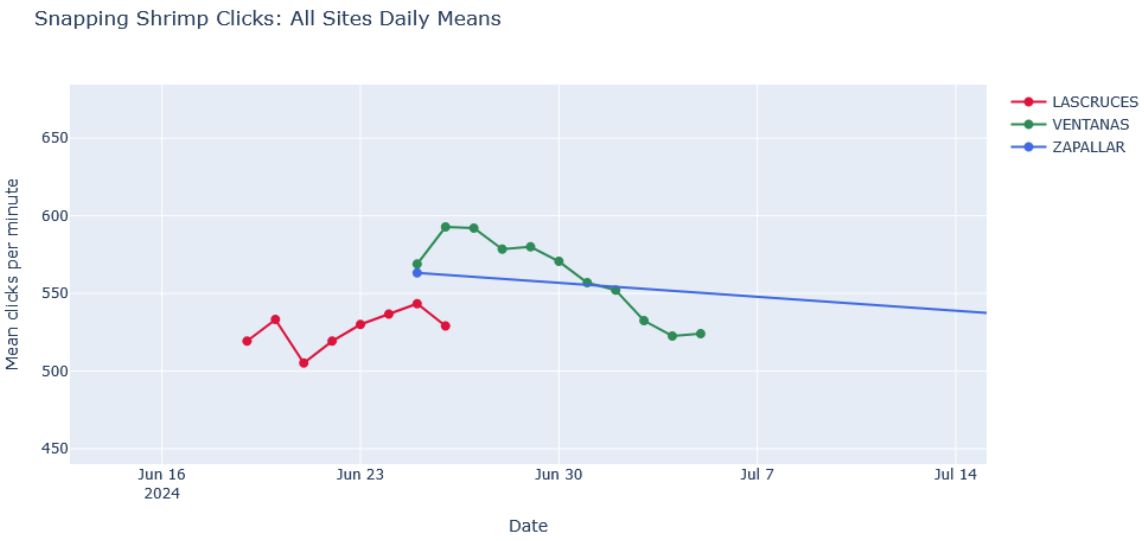

Figure 4: Daily mean click rates for each recording site (June 16 - July 14, 2024).

15.2 Full Time-Series Plot

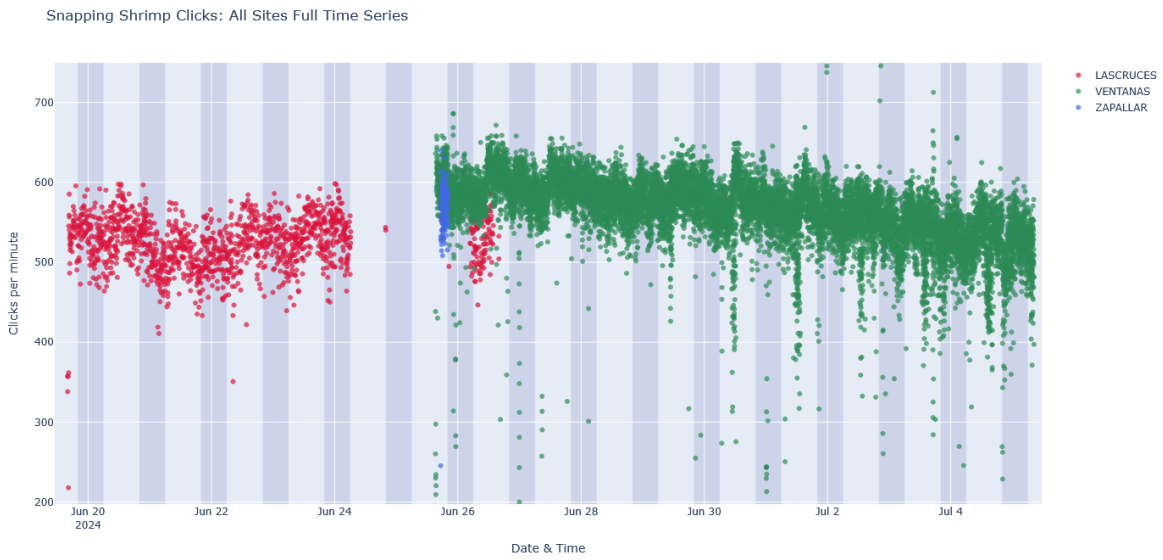

Figure 5: Full time-series plot showing clicks per minute across all three recording sites. Each marker represents one recording session. Navy blue shaded regions indicate nighttime hours (20:00–06:00 local time). Distinct site colors enable easy discrimination: LASCRUCES (crimson), VENTANAS (seagreen), ZAPALLAR (royalblue). The plot reveals temporal patterns in shrimp acoustic activity across the complete monitoring period.

15.3 Hourly Rolling Mean Plot

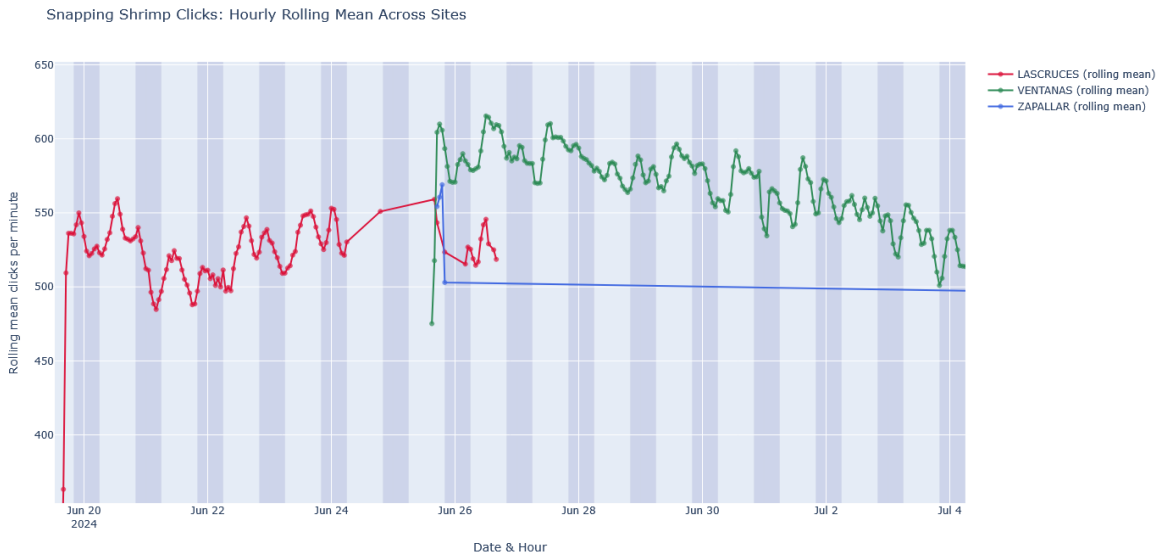

Figure 6: Hourly rolling mean (3-hour window) of clicks per minute across all sites with nighttime shading. The 3-hour moving average smooths short-term fluctuations while preserving sub-daily temporal resolution.

16 Validation and Next Steps

16.1 Detector Validation and Future Improvements

To assess detector performance, manually annotate a subset of recordings by reviewing spectrograms and waveforms in Audacity. Compare manual click identifications with detector output using the saved score files to generate receiver operating characteristic (ROC) curves. Calculate sensitivity (true positive rate) and specificity (true negative rate) across different threshold values. Adjust detector parameters (envelope threshold factor, derivative threshold factor, minimum inter-click interval) based on validation results to optimize performance for your specific recordings and acoustic environment. Consider implementing machine learning classifiers trained on labeled data to improve specificity for the project.

17 References

1. Adam, O. (2006). Advantages of the Hilbert Huang transform for marine mammals signals analysis. *The Journal of the Acoustical Society of America*, 120(5), 2965–2973.

2. Au, W. W., & Banks, K. (1998). The acoustics of the snapping shrimp *Synalpheus parneomeris* in Kaneohe Bay. *The Journal of the Acoustical Society of America*, 103(1), 41–47.

3. Beslin, W. A., Whitehead, H., & Gero, S. (2018). Automatic acoustic estimation of sperm whale size distributions achieved through machine recognition of on-axis clicks. *The Journal of the Acoustical Society of America*, 144(6), 3485–3495.

4. Bibikov, N. G., & Makushevich, I. V. (2020). Structure of the snapping shrimps' acoustical activity in the Black Sea shallow water. *The Journal of the Acoustical Society of America*, 148(4), EL388–EL393.

5. Bohnenstiehl, D. R., Lillis, A., & Eggleston, D. B. (2016). The curious acoustic behavior of estuarine snapping shrimp: temporal patterns of snapping shrimp sound in sub-tidal oyster reef habitat. *PLoS One*, 11(1), e0143691.

6. Butler, J., Butler IV, M. J., & Gaff, H. (2017). Snap, crackle, and pop: Acoustic-based model estimation of snapping shrimp populations in healthy and degraded hard-bottom habitats. *Ecological Indicators*, 77, 377-385.

7. Jeong, I., & Paeng, D. G. (2022). Circadian and tidal changes in snapping shrimp (Alpheus brevicristatus) sound observed by a moored hydrophone in the coastal Sea of Western Jeju. *Applied Sciences*, 10, 1029003.

8. Lee, D. H., Choi, J. W., Shin, S., & Song, H. C. (2021). Temporal variability in acoustic behavior of snapping shrimp in the East China Sea and its correlation with ocean environments. *Frontiers in Marine Science*, 8, 779283.

9. Song, Z., Ou, W., Su, Y., Li, H., Fan, W., Sun, S., & Zhang, Y. (2023). Sounds of snapping shrimp (Alpheidae) as important input to the soundscape in the southeast China coastal sea. *Frontiers in Marine Science*, 10, 1029003.

10. Wiggins, S. M., & Hildebrand, J. A. (2007, April). High-frequency Acoustic Recording Package (HARP) for broad-band long-term monitoring of marine mammals. In *2007 symposium on underwater technology and workshop on scientific use of submarine cables and related technologies* (pp. 551–557). IEEE.