```
"""
Star Photometry Utilities
=========================

A collection of functions for astronomical photometry analysis

Author: Dyson Lewis
Date: 2025

Usage:
    from star_photometry_utils import find_star_center_processed, detect_and_exclude_star_trai
ls

Disclaimer:
    no where in this code do I promise that it works correctly, quickly, or without errors.
    hopefully you find it useful
"""

import numpy as np
import matplotlib.pyplot as plt
import pylab
from scipy.ndimage import uniform_filter

def find_star_center(image, search_radius=50, box_size=250, min_radius=5):
    """
    Find the center of the brightest star in an already sky-subtracted image.

    This function locates the star by smoothing the image to find the brightest
    region, then performs intensity-weighted centroiding on the bright pixels.
    It also estimates the star's radius by analyzing the radial brightness profile.

    Parameters
    ----------
    image : numpy.ndarray
        The sky-subtracted image data (2D array)
    search_radius : int, optional
        Radius to define what counts as a "clump" of bright pixels (default: 50)
    box_size : int, optional
        Size of box around brightest region to use for centroiding (default: 250)
    min_radius : float, optional
        Minimum expected star radius in pixels (default: 5)

    Returns
    -------
    x_center : float
        X-coordinate of the star center (pixels)
    y_center : float
        Y-coordinate of the star center (pixels)
    star_radius : float
        Estimated radius of the star (pixels)

    Examples
    --------
    >>> from astropy.io import fits
    >>> image = fits.getdata('processed_star_image.fits')
    >>> x, y, radius = find_star_center_processed(image)
    >>> print(f"Star at ({x:.2f}, {y:.2f}) with radius {radius:.2f} pixels")
    """
    # For sky-subtracted images, estimate the noise level from the image edges
    edge_pixels = np.concatenate([
        image[0, :],      # top edge
        image[-1, :],     # bottom edge
        image[:, 0],      # left edge
```

```
        image[:, -1]      # right edge
    ])
    sky_noise = np.std(edge_pixels)

    # Smooth the image to find clumps
    smoothed = uniform_filter(image, size=search_radius)

    # Find the brightest clump in the smoothed image
    max_idx = np.argmax(smoothed)
    y_max, x_max = np.unravel_index(max_idx, smoothed.shape)

    # Extract a larger box around the brightest clump
    half_box = box_size // 2
    y_min = max(0, y_max - half_box)
    y_max_box = min(image.shape[0], y_max + half_box)
    x_min = max(0, x_max - half_box)
    x_max_box = min(image.shape[1], x_max + half_box)

    box = image[y_min:y_max_box, x_min:x_max_box]

    # Create coordinate grids for the box
    y_indices, x_indices = np.mgrid[y_min:y_max_box, x_min:x_max_box]

    # First pass: get rough center using bright pixels
    threshold_bright = np.percentile(box[box > 0], 90) if np.any(box > 0) else 0
    mask_bright = box > threshold_bright

    if np.sum(mask_bright) < 10:
        return float(x_max), float(y_max), min_radius

    # Calculate intensity-weighted centroid from bright pixels
    total_intensity = np.sum(box[mask_bright])
    x_center = np.sum(x_indices[mask_bright] * box[mask_bright]) / total_intensity
    y_center = np.sum(y_indices[mask_bright] * box[mask_bright]) / total_intensity

    # Now create a radial profile to find where star flux drops to noise level
    distances = np.sqrt((x_indices - x_center)**2 + (y_indices - y_center)**2)

    # Bin the radial profile
    max_radius = min(half_box, 100)  # Don't go too far out
    radial_bins = np.arange(0, max_radius, 1)
    radial_profile = []

    for r in radial_bins:
        # Get pixels in this radial bin
        mask = (distances >= r) & (distances < r + 1)
        if np.sum(mask) > 0:
            # Use median to be robust against cosmic rays
            radial_profile.append(np.median(box[mask]))
        else:
            radial_profile.append(0)

    radial_profile = np.array(radial_profile)

    # Find where the radial profile drops to ~3*noise level
    # This is where the star signal becomes indistinguishable from noise
    noise_threshold = 2 * sky_noise

    # Find the first radius where profile drops below noise threshold
    # and stays there for at least 3 consecutive bins
    star_radius = min_radius
    for i in range(len(radial_profile) - 3):
        if np.all(radial_profile[i:i+3] < noise_threshold):
```

```
            star_radius = radial_bins[i]
            break
    else:
        # If we never drop to noise, use where we drop to 10% of peak
        peak_value = np.max(radial_profile)
        for i, val in enumerate(radial_profile):
            if val < 0.1 * peak_value:
                star_radius = radial_bins[i]
                break
        else:
            print('fall back L')
            star_radius = max_radius * 0.5  # fallback

    # Make sure we have a reasonable minimum
    star_radius = max(star_radius, min_radius)

    return x_center, y_center, star_radius

def detect_trails(image_files, positions, photometry, photometry_err,
                                    time_offsets, outlier_threshold=4.0, plot=True):
    """
    Detect images where the star has trailed using Modified Z-Score (MAD-based).

    This function identifies telescope tracking errors that cause stars to trail
    during exposure by detecting significant flux drops compared to the median.
    It uses the Modified Z-Score method based on Median Absolute Deviation (MAD),
    which is robust to outliers and only flags values significantly BELOW the median.

    Parameters
    ----------
    image_files : list of str
        List of image filenames
    positions : list of tuple
        List of (x, y) star positions for each image
    photometry : numpy.ndarray
        Measured photometry values in DN/s (1D array)
    photometry_err : numpy.ndarray
        Photometry uncertainties in DN/s (1D array)
    time_offsets : numpy.ndarray
        Time of each observation in seconds (1D array)
    outlier_threshold : float, optional
        Number of modified z-score sigma for outlier detection (default: 4.0)
        Typical values: 3.5-4.5 (lower = more sensitive to trails)
    plot : bool, optional
        Whether to plot diagnostic figures (default: True)

    Returns
    -------
    valid_indices : numpy.ndarray
        Indices of images without trails (can be used to filter arrays)

    Notes
    -----
    The Modified Z-Score is calculated as:
        M_i = 0.6745 * (x_i - median) / MAD
    where MAD is the Median Absolute Deviation of negative deviations.
    The factor 0.6745 makes MAD comparable to standard deviation for normal data.

    Only images with flux significantly BELOW the median are flagged, as trailing
    stars spread light over a larger area, reducing measured flux.

    Examples
    --------
```

```
    >>> # Basic usage
    >>> valid_idx = detect_and_exclude_star_trails(
    ...     filenames, positions, flux, flux_err, times,
    ...     outlier_threshold=4.0, plot=True
    ... )
    >>> # Filter your data arrays
    >>> clean_flux = flux[valid_idx]
    >>> clean_times = times[valid_idx]

    >>> # More sensitive detection
    >>> valid_idx = detect_and_exclude_star_trails(
    ...     filenames, positions, flux, flux_err, times,
    ...     outlier_threshold=3.5, plot=False
    ... )
    """
    valid_indices = []
    trailed_indices = []

    print("=" * 60)
    print("STAR TRAIL DETECTION (Modified Z-Score)")
    print("=" * 60)
    print(f"Threshold: {outlier_threshold} modified z-score (below median only)")

    # Calculate median flux
    median_flux = np.median(photometry)
    deviations = photometry - median_flux

    # Only use negative deviations for MAD calculation
    negative_devs = deviations[deviations < 0]
    mad = np.median(np.abs(negative_devs)) if len(negative_devs) > 0 else np.median(np.abs(dev
iations))

    # Modified z-score: 0.6745 * (x - median) / MAD
    # The 0.6745 factor makes MAD comparable to standard deviation for normal data
    modified_zscores = 0.6745 * deviations / mad if mad > 0 else np.zeros_like(photometry)

    print(f"Median flux: {median_flux:.1f} DN/s")
    print(f"MAD (negative deviations): {mad:.1f} DN/s")
    print(f"Equivalent sigma: {mad / 0.6745:.1f} DN/s")

    # Plot diagnostic figure
    if plot:
        fig, (ax1, ax2, ax3) = pylab.subplots(3, 1, figsize=(12, 12))

        # Determine which points are good vs bad
        good_mask = modified_zscores >= -outlier_threshold
        bad_mask = modified_zscores < -outlier_threshold

        # Extract band prefix for labeling
        band_prefix = image_files[0].split('_')[0] if '_' in image_files[0] else 'IMG'

        # Top panel: Light curve with flagged points
        if np.any(good_mask):
            ax1.errorbar(time_offsets[good_mask], photometry[good_mask],
                         photometry_err[good_mask], fmt='o', alpha=0.5,
                         color='blue', label='Valid')
        if np.any(bad_mask):
            ax1.errorbar(time_offsets[bad_mask], photometry[bad_mask],
                         photometry_err[bad_mask], fmt='x', markersize=10,
                         color='red', label='Rejected (trails)', linewidth=2)

        # Add index labels to each point
        for i in range(len(photometry)):
```

```
            color = 'red' if bad_mask[i] else 'blue'
            ax1.annotate(f'{band_prefix}_{i+1}', (time_offsets[i], photometry[i]),
                         textcoords="offset points", xytext=(0,5), ha='center',
                         fontsize=7, alpha=0.7, color=color)

        ax1.axhline(median_flux, color='purple', linestyle='-', alpha=0.5,
                    linewidth=2, label='Median')
        ax1.axhline(median_flux - outlier_threshold * mad / 0.6745,
                    color='orange', linestyle='--', alpha=0.7, linewidth=2,
                    label=f'-{outlier_threshold}Ï\203 threshold')
        ax1.set_ylabel('Flux (DN/s)', fontsize=11)
        ax1.set_title('Light Curve and Trail Detection', fontsize=12, fontweight='bold')
        ax1.legend(loc='best')
        ax1.grid(True, alpha=0.3)

        # Middle panel: Modified z-scores
        ax2.plot(time_offsets, modified_zscores, 'o', alpha=0.6, color='gray')
        ax2.axhline(0, color='purple', linestyle='-', linewidth=2, label='Median (z=0)')
        ax2.axhline(-outlier_threshold, color='orange', linestyle='--',
                    linewidth=2, label=f'-{outlier_threshold}Ï\203 threshold')
        ax2.fill_between([time_offsets.min(), time_offsets.max()],
                         -outlier_threshold, -5, alpha=0.2, color='red',
                         label='Trail region')

        # Highlight rejected points
        if np.any(bad_mask):
            ax2.plot(time_offsets[bad_mask], modified_zscores[bad_mask],
                     'rx', markersize=10, linewidth=2)

        ax2.set_ylabel('Modified Z-Score', fontsize=11)
        ax2.set_title('Modified Z-Scores vs Time', fontsize=12)
        ax2.legend(loc='best')
        ax2.grid(True, alpha=0.3)

        # Bottom panel: Distribution with box plot
        ax3.boxplot([photometry[good_mask]], positions=[1], widths=0.6,
                    labels=['Valid Data'], patch_artist=True,
                    boxprops=dict(facecolor='lightblue', alpha=0.7))
        if np.any(bad_mask):
            ax3.plot(np.ones(np.sum(bad_mask)), photometry[bad_mask], 'rx',
                     markersize=12, linewidth=2, label='Rejected (trails)')
        ax3.axhline(median_flux, color='purple', linestyle='--',
                    alpha=0.5, linewidth=2, label='Median')
        ax3.set_ylabel('Flux (DN/s)', fontsize=11)
        ax3.set_title('Flux Distribution with Outliers', fontsize=12)
        ax3.legend(loc='best')
        ax3.grid(True, alpha=0.3, axis='y')
        ax3.set_xlim(0.5, 1.5)

        pylab.tight_layout()
        pylab.show()

    # Flag outliers (ONLY LOW VALUES)
    for i in range(len(photometry)):
        if modified_zscores[i] < -outlier_threshold:
            trailed_indices.append(i)
            print(f"    TRAIL DETECTED in {image_files[i]}:")
            print(f"      Measured flux: {photometry[i]:.1f} DN/s")
            print(f"      Median flux: {median_flux:.1f} DN/s")
            print(f"      Modified Z-Score: {modified_zscores[i]:.2f}")
            print(f"      Time: {time_offsets[i]:.1f} s")
        else:
            valid_indices.append(i)
```

```
    valid_indices = np.array(valid_indices)
    trailed_indices = np.array(trailed_indices)

    print("\n" + "=" * 60)
    print(f"SUMMARY: Found {len(trailed_indices)} trails, "
          f"{len(valid_indices)} good images")
    if len(trailed_indices) > 0:
        print(f"Excluded images: {[image_files[i] for i in trailed_indices]}")
    print("=" * 60)

    return valid_indices


if __name__ == "__main__":
    print("Star Photometry Utilities Module")
    print("================================")
    print("\nAvailable functions:")
    print("  - find_star_center_processed()")
    print("  - detect_and_exclude_star_trails()")
    print("\nImport this module in your notebook with:")
    print("  from star_photometry_utils import find_star_center_processed, detect_and_exclude_
star_trails")
```