```
"""
CÓDIGO DE CORRECCIÓN RADIOMÉTRICA

Requisitos:
- Python 3.8+
- Instalar las dependencias con:
  pip install -r requirements.txt
- Instalar `ExifTool` manualmente:
  **Windows**: Descargar desde [https://exiftool.org/](https://exiftool.org/)
  **Mac/Linux**:
  ```
  sudo apt install libimage-exiftool-perl # Ubuntu/Debian
  brew install exiftool # MacOS
  ```
"""



#Librerias
import os
import numpy as np
import pandas as pd
import rasterio
from rasterio.plot import reshape_as_image


# ——————————————— FUNCIONES DE CORRECCIÓN ———————————————
def subtract_black_level(band, black_level):
    """
    Aplica la corrección del nivel negro a una banda espectral.

    El nivel negro (black level) representa el valor mínimo registrado por el sensor
    en ausencia de luz, y su sustracción permite eliminar el sesgo inherente del sensor
    para obtener una señal más precisa. Esta corrección es fundamental en el preprocesamiento
    radiométrico, ya que asegura que los valores de los píxeles reflejen únicamente la señal real
    capturada por la cámara.

    La función se asegura de que los valores negativos resultantes después de la sustracción
    se ajusten a cero, evitando errores en las etapas posteriores de calibración y normalización.

    Parámetros:
    - band (np.ndarray): Banda espectral en formato array de 2D (una sola banda).
    - black_level (float): Valor del nivel negro extraído desde los metadatos de la imagen.

    Retorno:
    - np.ndarray: Banda corregida, con valores ajustados a cero donde corresponda.

    Ejemplo:
    corrected = subtract_black_level(banda_nir, 64.5)
```

```
    Notas:
    - Esta función no modifica los metadatos ni el perfil de la imagen original.
    - Es el primer paso del flujo de corrección radiométrica.
    """
    band = band - black_level
    band = np.where(band < 0, 0, band)  # Asegurar que no haya valores negativos
    print(f"Valores después de restar nivel negro: min={band.min()}, max={band.max()}")
    return band


def apply_sensor_calibration(band, calibration_factor):
    """
    Aplica la calibración del sensor a una banda espectral.

    Este paso multiplica cada valor de la banda por el factor de calibración del sensor,
    el cual compensa las ganancias electrónicas aplicadas por la cámara durante la captura
    de la imagen. El valor es extraído directamente de los metadatos y varía por banda
    y condiciones de vuelo.

    Esta corrección es necesaria para escalar los valores registrados por el sensor
    y llevarlos a unidades proporcionales de energía luminosa real recibida por el detector.

    Parámetros:
    - band (np.ndarray): Banda espectral en formato array 2D.
    - calibration_factor (float): Factor de calibración del sensor proveniente de los metadatos
      (usualmente bajo el campo "Sensor Gain Adjustment").

    Retorno:
    - np.ndarray: Banda calibrada.

    Ejemplo:
    calibrated = apply_sensor_calibration(banda_nir, 1.021)

    Notas:
    - Esta función debe ejecutarse después de aplicar la corrección del nivel negro.
    - La precisión de esta corrección depende directamente de la calidad de los metadatos.
    """
    band = band * calibration_factor
    print(f"Valores después de la calibración del sensor: min={band.min()}, max={band.max()}")
    return band


def convert_to_reflectance(band, irradiance):
    """
    Convierte los valores digitales de una banda espectral en reflectancia aparente.

    Esta función divide cada valor de la banda por la irradiancia registrada por el sensor
```

```
    en el momento de la captura. La reflectancia aparente representa la fracción de radiación
    incidente reflejada por la superficie en cada banda espectral, normalizada respecto
    a las condiciones de iluminación.

    Este paso permite comparar imágenes tomadas en distintos momentos o condiciones,
    ya que elimina la variabilidad causada por cambios en la luz solar, nubosidad o geometría solar.

    Parámetros:
    - band (np.ndarray): Banda espectral en formato array 2D.
    - irradiance (float): Valor de irradiancia extraído de los metadatos para la banda correspondiente.
                          Debe ser mayor que cero.

    Retorno:
    - np.ndarray: Banda convertida a reflectancia.

    Excepciones:
    - ValueError: Si el valor de irradiancia es cero o negativo.

    Ejemplo:
    reflectance = convert_to_reflectance(banda_nir, 1.215)

    Notas:
    - Esta función debe ejecutarse después de aplicar la calibración del sensor.
    - No aplica ningún tipo de corrección atmosférica, solo radiométrica.
    """
    if irradiance ⩽ 0:
        raise ValueError("El valor de irradiancia es inválido o cero.")
    band = band / irradiance
    print(f"Valores después de la división por irradiancia: min={band.min()}, max={band.max()}")
    return band



def apply_vignetting_correction(band, vignette_coeffs, center_x, center_y):
    """
    Aplica la corrección de viñeteo a una banda espectral utilizando un modelo polinomial
    basado en los coeficientes de calibración y el centro óptico de la imagen.

    El viñeteo es un fenómeno óptico que causa una disminución progresiva en la intensidad
    de los píxeles hacia las esquinas de la imagen, debido a la geometría de la lente.
    Esta función corrige ese efecto multiplicando cada píxel por un factor de compensación
    calculado a partir de un polinomio de 6 grados sobre la distancia radial desde el centro óptico.

    Parámetros:
    - band (np.ndarray): Banda espectral en formato array 2D.
    - vignette_coeffs (list of float): Lista de coeficientes del polinomio de viñeteo [a1, a2, ... , a6],
```

```
145       extraídos del campo "Vignetting Data" de los metadatos.
146     - center_x (float): Coordenada X del centro óptico de la lente.
147     - center_y (float): Coordenada Y del centro óptico de la lente.
148 
149     Retorno:
150     - np.ndarray: Banda corregida por viñeteo.
151 
152     Ejemplo:
153     corregida = apply_vignetting_correction(banda_red, [0.0012, -0.003, 0.0021, ...],
    640, 512)
154 
155     Notas:
156     - Se usa `np.clip()` para restringir el factor de corrección entre 0.5 y 2.0,
    evitando
157       amplificaciones o atenuaciones excesivas.
158     - Esta función debe ejecutarse antes de la normalización final.
159     - El modelo de corrección se basa en la distancia radial `r` al centro óptico y
    puede ajustarse
160       a las especificaciones del fabricante del sensor.
161     """
162     y, x = np.indices(band.shape)
163     r_squared = (x - center_x)**2 + (y - center_y)**2
164     r = np.sqrt(r_squared)
165 
166     # Calcular el polinomio del viñeteo
167     vignette_correction = (
168         vignette_coeffs[5] * r**6 +
169         vignette_coeffs[4] * r**5 +
170         vignette_coeffs[3] * r**4 +
171         vignette_coeffs[2] * r**3 +
172         vignette_coeffs[1] * r**2 +
173         vignette_coeffs[0] * r +
174         1.0
175     )
176 
177     # Evitar valores extremos en la corrección
178     vignette_correction = np.clip(vignette_correction, 0.5, 2.0)  # Valores razonables
    para corrección
179     band = band * vignette_correction
180     print(f"Valores después de la corrección de viñeteo: min={band.min()}, max=
    {band.max()}")
181     return band
182 
183 
184 
185 def normalize_band(band):
186     """
187     Normaliza los valores de una banda espectral al rango [0, 1].
188 
189     Este paso estandariza la escala de valores de la imagen para facilitar su análisis y
    visualización,
190     especialmente en procesos posteriores como clasificación, segmentación o
    entrenamiento de modelos
191     de aprendizaje automático.
192 
```

```
    La función calcula el valor mínimo y máximo de la banda y aplica una transformación lineal
    para escalar todos los valores dentro del rango [0, 1]. En casos donde todos los valores de
    la banda sean iguales (por ejemplo, imagen uniforme), se evita la división por cero mediante `np.clip()`.

    Parámetros:
    - band (np.ndarray): Banda espectral en formato array 2D con valores en punto flotante.

    Retorno:
    - np.ndarray: Banda normalizada en el rango [0.0, 1.0].

    Ejemplo:
    banda_normalizada = normalize_band(banda_corr_nir)

    Notas:
    - Esta función debe ejecutarse como paso final del preprocesamiento radiométrico.
    - La salida es adecuada para visualización o uso en algoritmos que requieren escalado uniforme.
    - No modifica los metadatos asociados a la imagen.
    """
    min_val = np.min(band)
    max_val = np.max(band)
    if max_val > min_val:
        band = (band - min_val) / (max_val - min_val)
    else:
        band = np.clip(band, 0, 1)  # Evitar división por cero si min == max
    print(f"Valores después de la normalización: min={band.min()}, max={band.max()}")
    return band


# ——————————————— PROCESAMIENTO DE UNA IMAGEN ———————————————
def process_image_radiometrica(image_path, metadata, output_folder):
    """
    Aplica la corrección radiométrica paso a paso a una imagen multiespectral en formato .TIF,
    utilizando los metadatos asociados.

    El procesamiento incluye cinco pasos consecutivos:
    1. Sustracción del nivel negro
    2. Calibración del sensor (ganancia electrónica)
    3. Conversión a reflectancia aparente mediante irradiancia
    4. Corrección óptica por viñeteo (basada en distancia radial al centro óptico)
    5. Normalización de los valores al rango [0, 1]

    Adicionalmente, la función conserva y reasigna los metadatos tanto globales como por banda.

    Parámetros:
    - image_path (str): Ruta completa del archivo de imagen a procesar (.TIF).
    - metadata (pd.DataFrame): Tabla de metadatos extraídos con campos como:
        'File Name', 'Black Level', 'Sensor Gain Adjustment', 'Irradiance',
        'Vignetting Data', 'Vignetting Center'.
```

```
    - output_folder (str): Ruta donde se guardará la imagen corregida.

    Proceso:
    - Verifica que existan metadatos correspondientes para la imagen.
    - Extrae los parámetros de corrección desde el archivo CSV.
    - Lee la imagen como arreglo multibanda con `rasterio`.
    - Aplica secuencialmente las funciones de preprocesamiento a cada banda.
    - Guarda la imagen corregida en formato `float32` conservando los metadatos originales.

    Retorno:
    - No retorna ningún valor. Guarda la imagen corregida directamente en disco.

    Ejemplo de uso:
    process_image_step1("T1_/DJI_0010.TIF", metadata_df, "01_Corr_radiometrica/T1_")

    Notas:
    - Si la imagen no tiene metadatos asociados, se omite y muestra una advertencia.
    - Si el archivo no tiene sistema de coordenadas (CRS), se emite una alerta pero se continúa.
    - La salida conserva la estructura espectral original (número de bandas) y los tags.

    Excepciones:
    - En caso de error al escribir el archivo de salida, se imprime un mensaje con el detalle.
    """
    image_name = os.path.basename(image_path)
    meta_row = metadata.loc[metadata['File Name'] == image_name]

    if meta_row.empty:
        print(f"Metadatos no encontrados para {image_name}.")
        return

    print(f"Procesando imagen: {image_name}")
    print(f"Campo File Name en metadatos: {meta_row['File Name'].iloc[0]}")

    # Extraer el nivel negro de los metadatos
    black_level = float(meta_row['Black Level'].iloc[0])
    calibration_factor = float(meta_row['Sensor Gain Adjustment'].iloc[0])
    irradiance = float(meta_row['Irradiance'].iloc[0])
    vignette_coeffs = list(map(float, meta_row['Vignetting Data'].iloc[0].split(',')))
    center_x = float(meta_row['Vignetting Center'].iloc[0].split(',')[0])
    center_y = float(meta_row['Vignetting Center'].iloc[0].split(',')[1])

    print(f"Nivel negro: {black_level}, Factor de calibración: {calibration_factor}, Irradiancia: {irradiance}")
    print(f"Datos de viñeteo: {vignette_coeffs}, Centro X: {center_x}, Centro Y: {center_y}")

    # Leer la imagen
    with rasterio.open(image_path) as src:
        img = reshape_as_image(src.read())
        profile = src.profile
        crs = src.crs  # CRS original
        tags = src.tags()  # Metadatos globales
```

```
        band_tags = [src.tags(i + 1) for i in range(src.count)]  # Metadatos por banda

    # Verificar y asignar el CRS si falta
    if crs is None:
        print(f"Advertencia: CRS no definido en {image_name}.")
    else:
        profile.update(crs=crs)

    # Restar el nivel negro de cada banda
    corrected_bands = []
    for i in range(img.shape[2]):
        band = img[:, :, i]
        band = subtract_black_level(band, black_level)  # Paso 1: Nivel negro
        band = apply_sensor_calibration(band, calibration_factor)  # Paso 2: Calibración del sensor
        band = convert_to_reflectance(band, irradiance)  # Paso 3: Reflectancia
        band = apply_vignetting_correction(band, vignette_coeffs, center_x, center_y) # Paso 4: Corrección por viñeteo
        band = normalize_band(band)  # Normalización final
        corrected_bands.append(band)

    # Guardar la imagen corregida tras el paso 2
    corrected_bands = np.stack(corrected_bands, axis=-1).astype(np.float32)

    output_path = os.path.join(output_folder, image_name)
    profile.update(dtype='float32', count=corrected_bands.shape[2], nodata = None)

    try:
        with rasterio.open(output_path, 'w', **profile) as dst:
            for i in range(corrected_bands.shape[2]):
                dst.write(corrected_bands[:, :, i], i + 1)
                # Asignar metadatos específicos por banda
                dst.update_tags(i + 1, **band_tags[i])
            # Asignar etiquetas globales al archivo
            dst.update_tags(**tags)
        print(f"Imagen corregida guardada en: {output_path}")
    except Exception as e:
        print(f"Error al guardar la imagen {output_path}: {e}")


def main_radiometrica(input_folder, metadata_csv, output_folder):
    """
    Ejecuta la corrección radiométrica para todas las imágenes TIF en la carpeta definida.

    Utiliza los metadatos especificados en `metadata_csv` y guarda los resultados en `output_folder`.
    Las rutas son gestionadas desde el archivo `config.py`.

    Proceso:
    1. Carga los metadatos desde el CSV.
    2. Busca imágenes con extensión .TIF en la carpeta de entrada.
    3. Aplica `process_image_step1()` a cada archivo.
    """
    # Cargar los metadatos
```

```
    metadata = pd.read_csv(metadata_csv)

    # Crear la carpeta de salida si no existe
    os.makedirs(output_folder, exist_ok=True)

    # Procesar todas las imágenes TIFF
    tiff_files = [os.path.join(input_folder, f) for f in os.listdir(input_folder) if f.lower().endswith('.tif')]

    for image_path in tiff_files:
        process_image_radiometrica(image_path, metadata, output_folder)


if __name__ == "__main__":
    main_radiometrica()

```