```python
"""
CÓDIGO DE COREGISTRO DE BANDAS MULTIESPECTRALES

Requisitos:
- Python 3.8+
- Instalar dependencias con:
    pip install -r requirements.txt
"""


# ═══════════════════════════════ IMPORTACIÓN DE LIBRERÍAS Y CONFIGURACIÓN ═══════════════════════════════
import cv2
import numpy as np
import pandas as pd
import os
import rasterio
import re
from collections import defaultdict



# ═══════════════════ FUNCIONES COMPLEMENTARIAS═══════════════════
def obtener_centros_opticos(ruta_imagen, metadata_tif, metadata_jpg):
    """
    Extrae las coordenadas relativas del centro óptico (X, Y) desde los metadatos de una imagen TIF o JPG.

    Esta función identifica automáticamente si la imagen es multiespectral (TIF) o RGB (JPG), selecciona
    el DataFrame de metadatos correspondiente, y busca una coincidencia exacta por nombre de archivo.
    Una vez encontrada, extrae las coordenadas del centro óptico desde las columnas:
    - 'Relative Optical Center X'
    - 'Relative Optical Center Y'

    Estas coordenadas son utilizadas para estimar el desplazamiento geométrico entre sensores durante el coregistro
    multiespectral.

    Args:
        ruta_imagen (str): Ruta absoluta del archivo de imagen (.TIF o .JPG).
        metadata_tif (pd.DataFrame): Metadatos de imágenes TIF, leídos desde 'metadata_tif.csv'.
        metadata_jpg (pd.DataFrame): Metadatos de imágenes JPG, leídos desde 'metadata_jpg.csv'.

    Returns:
        tuple:
            - relative_x (float): Coordenada relativa X del centro óptico.
            - relative_y (float): Coordenada relativa Y del centro óptico.

    Notas:
```

```
        - Si no se encuentra el archivo en los metadatos o hay error en la lectura, se retorna (0.0, 0.0) como fallback.
        - La comparación del nombre de archivo es insensible a mayúsculas y espacios.
        - El campo de metadatos 'File Name' debe coincidir exactamente con el nombre del archivo, incluyendo extensión.
    """
    try:
        nombre_imagen = os.path.basename(ruta_imagen).strip().lower()

        # Determinar si es TIF o JPG
        if nombre_imagen.endswith(".tif"):
            metadata = metadata_tif
        elif nombre_imagen.endswith(".jpg") or nombre_imagen.endswith(".jpeg"):
            metadata = metadata_jpg
        else:
            print(f"No se reconoce el formato de {nombre_imagen}. Omitiendo ... ")
            return 0.0, 0.0

        # Buscar la fila en el CSV que coincide exactamente con el nombre del archivo
        fila_metadatos = metadata[metadata['File Name'].str.strip().str.lower() == nombre_imagen]

        if fila_metadatos.empty:
            print(f"No se encontraron metadatos para {nombre_imagen}. Usando X=0, Y=0 como fallback.")
            return 0.0, 0.0

        # Extraer valores
        relative_x = float(fila_metadatos['Relative Optical Center X'].iloc[0])
        relative_y = float(fila_metadatos['Relative Optical Center Y'].iloc[0])

        print(f"Metadatos encontrados para {nombre_imagen} → X: {relative_x}, Y: {relative_y}")
        return relative_x, relative_y

    except Exception as e:
        print(f"Error al extraer metadatos de {ruta_imagen}: {str(e)}")
        return 0.0, 0.0



def parse_nombre_archivo(nombre_archivo):
    """
    Parsea el nombre de un archivo multiespectral o RGB y extrae su información estructural clave,
    generando un identificador base para agrupar todas las bandas asociadas a una misma imagen.

    Esta función se basa en un patrón de nomenclatura específico utilizado en archivos generados por drones DJI,
    donde cada banda multiespectral tiene un número de banda codificado en la séptima posición del nombre.

    La estructura esperada es:
        DJI_XXXX_S_CC_Z_tN.ext
```

```
    Donde:
        - XXXX: identificador de imagen (numérico)
        - S: número de banda (1-5) que se eliminará para formar el nombre base
        - CC: código de orientación (`ob` o `ot`)
        - Z: identificador de zona
        - tN: tiempo (`t1`, `t2`, etc.)
        - ext: extensión (.TIF, .JPG, etc.)

    Args:
        nombre_archivo (str): Nombre del archivo a analizar (ej. "DJI_0011_ob3_j_t1.TIF").

    Returns:
        tuple:
            - nombre_base_grupo (str): Nombre base para agrupar todas las bandas de una imagen (con 'S' omitido).
            - banda (int): Número de banda extraído desde la posición 7 del nombre.
            - nombre_completo (str): Nombre original del archivo (sin modificaciones).

    Returns (None, None, None) si el nombre no cumple con el patrón esperado.
    """
    pattern = r"^DJI_(\d{3})\d_(ob|ot)(\d+)_([a-zA-Z])_t(\d)\..+$"
    match = re.match(pattern, nombre_archivo, re.IGNORECASE)

    if not match:
        return None, None, None

    # Componentes del nombre
    xxx = match.group(1)       # Primeros 3 dígitos de la secuencia
    banda = int(match.group(0)[7])  # El dígito en posición S (índice 7)
    cc = match.group(2)        # ob/ot
    l = match.group(3)         # Número de carpeta
    k = match.group(4)         # Zona de estudio
    tp = match.group(5)        # Tiempo

    # Construir nombre base del grupo (todo excepto S)
    nombre_base = f"DJI_{xxx}_{cc}{l}_{k}_t{tp}"

    return nombre_base, banda, nombre_archivo



def cargar_grupos_imagenes(directorio_tif, directorio_rgb, progress_file):
    """
    Agrupa imágenes TIF multiespectrales y sus correspondientes imágenes RGB (JPG) por nombre base común.

    Esta función itera sobre los directorios de entrada y agrupa las imágenes en un diccionario estructurado
    por nombre base (`nombre_base`), el cual es derivado de `parse_nombre_archivo()`. Este agrupamiento es
    esencial para permitir el coregistro entre bandas multiespectrales (1–5) y la imagen RGB.

    Para cada grupo:
```

```
    - Las bandas multiespectrales se almacenan con su número de banda como clave.
    - La imagen RGB (banda 0) se asocia como imagen de referencia si está disponible.

    Args:
        directorio_tif (str): Ruta al directorio que contiene imágenes TIF geométricamente corregidas.
        directorio_rgb (str): Ruta al directorio que contiene imágenes JPG corregidas.
        progress_file (str): Ruta del archivo de progreso (no se usa directamente aquí, pero se mantiene por consistencia del flujo).

    Returns:
        dict: Diccionario con la siguiente estructura por grupo:

        {
            'nombre_base': {
                'bandas': {
                    1: {'imagen': np.ndarray, 'ruta': str},
                    2: { ... },
                     ...
                    5: { ... }
                },
                'rgb': {'imagen': np.ndarray, 'ruta': str} or None
            },
            ...
        }

    Notes:
        - Sólo se incluyen las imágenes cuyo nombre cumpla con el patrón esperado por `parse_nombre_archivo()`.
        - Las bandas deben estar numeradas del 1 al 5. Las JPG se consideran banda 0.
        - Si no hay JPG correspondiente, el grupo se crea de todas formas con `'rgb': None`.
        - Este agrupamiento es esencial para aplicar el coregistro multibanda en etapas posteriores.
    """
    grupos = defaultdict(lambda: {'rgb': None, 'bandas': {}})

    print("Cargando imágenes ... ")

    # Procesar TIF (multiespectral)
    for archivo in os.listdir(directorio_tif):
        nombre_base, banda, _ = parse_nombre_archivo(archivo)
        if not nombre_base or banda not in [1,2,3,4,5]:
            continue

        # Guardar imagen y metadatos
        ruta = os.path.join(directorio_tif, archivo)
        with rasterio.open(ruta) as src:
            img = src.read(1)
        grupos[nombre_base]['bandas'][banda] = {'imagen': img, 'ruta': ruta}
        print(f"TIF agregado - {nombre_base} | Banda: {banda} | Ruta: {ruta}")

    # Procesar JPG (RGB - Banda 0)
    for archivo in os.listdir(directorio_rgb):
        nombre_base, banda, _ = parse_nombre_archivo(archivo)
```

```
        if not nombre_base or banda != 0:  # Asumiendo RGB=banda 0
            continue

        ruta = os.path.join(directorio_rgb, archivo)
        img = cv2.imread(ruta, cv2.IMREAD_COLOR)
        if nombre_base in grupos:
            grupos[nombre_base]['rgb'] = {'imagen': img, 'ruta': ruta}
            print(f"RGB agregado - {nombre_base} | Ruta: {ruta}")

    print("Verificación final de grupos:")
    for nombre_base, datos in grupos.items():
        bandas_presentes = list(datos['bandas'].keys())
        print(f" {nombre_base}: RGB {'OK' if datos['rgb'] else 'F'} | Bandas:
{bandas_presentes}")

    return grupos



def preparar_para_sift(image):
    """
    Prepara una imagen para detección de características mediante SIFT, asegurando el
formato uint8 requerido.

    Esta función convierte temporalmente imágenes de 16 bits (`uint16`), típicas en
imágenes multiespectrales TIF,
    a imágenes de 8 bits (`uint8`) necesarias para el correcto funcionamiento del
algoritmo SIFT de OpenCV,
    sin modificar la imagen original ni su versión de trabajo de mayor precisión.

    La conversión se realiza mediante normalización lineal al rango [0, 255] utilizando
`cv2.normalize()`.

    Args:
        image (np.ndarray): Imagen original en formato `uint16` o `uint8`.

    Returns:
        np.ndarray: Imagen convertida a `uint8`, adecuada para la detección de keypoints
con SIFT.

    Notas:
        - Esta conversión es sólo para el cálculo de keypoints y descriptores.
        - La imagen resultante no debe usarse para análisis espectral o visualización de
precisión.
        - En imágenes que ya están en `uint8`, se retorna una copia segura convertida
con `.astype(np.uint8)`.

    Example:
        >> sift_ready = preparar_para_sift(imagen_nir)
        >> keypoints, descriptors = cv2.SIFT_create().detectAndCompute(sift_ready, None)
    """
    if image.dtype == np.uint16:
        image_uint8 = cv2.normalize(image, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8)
        return image_uint8
```

```
    return image.astype(np.uint8)


def ncc(fixed, moving):
    """
    Calcula la Correlación Cruzada Normalizada (NCC) entre dos imágenes para evaluar la calidad del coregistro.

    Esta función mide el grado de similitud entre dos imágenes alineadas, típicamente la imagen de referencia
    (`fixed`) y la imagen coregistrada (`moving`). La NCC es una métrica simétrica y libre de unidades que toma
    valores en el rango [-1, 1], donde:

        -  1  indica correspondencia perfecta,
        -  0  indica ausencia de correlación lineal,
        - -1  indica correlación inversa perfecta.

    La fórmula implementada es:

        NCC = Σ[(f - μ_f) * (m - μ_m)] / sqrt[Σ(f - μ_f)^2 * Σ(m - μ_m)^2]

    Args:
        fixed (np.ndarray): Imagen de referencia (por ejemplo, banda NIR).
        moving (np.ndarray): Imagen que ha sido alineada respecto a la referencia.

    Returns:
        float: Valor de NCC entre las dos imágenes.

    Notes:
        - Esta métrica se utiliza comúnmente para cuantificar la calidad de alineación tras aplicar homografías
          o transformaciones afines.
        - Ambas imágenes deben tener el mismo tamaño y tipo de dato numérico (ej. `uint8` o `float32`).
        - Una NCC cercana a 1 es deseable tras un coregistro exitoso.
    """
    # Normalización
    fixed_mean = fixed.mean()
    moving_mean = moving.mean()
    fixed_std = fixed.std()
    moving_std = moving.std()

    # Correlación cruzada normalizada
    numerator = np.sum((fixed - fixed_mean) * (moving - moving_mean))
    denominator = np.sqrt(np.sum((fixed - fixed_mean) ** 2) * np.sum((moving - moving_mean) ** 2))
    return numerator / denominator




# ═══════════════════ FUNCIÓN COREGISTRO ═══════════════════
def coregistrar_banda(fixed_rgb, moving_espectral, ruta_fixed, ruta_moving, metadata_tif, metadata_jpg, ncc_log_path=None):
```

```
    """
    Coregistra una banda espectral (moving) con respecto a una imagen de referencia (fixed), típicamente la banda NIR.

    El proceso de coregistro se realiza en dos etapas:

    1. **Corrección geométrica inicial por desplazamiento de centros ópticos**
       Se calcula un desplazamiento (ΔX, ΔY) entre los centros ópticos de las imágenes, extraídos de los metadatos.
       Esta traslación se aplica mediante una transformación afín a la imagen `moving_espectral` en su formato original `uint16`.

    2. **Refinamiento mediante homografía estimada con SIFT**
       La imagen alineada se convierte temporalmente a `uint8` para aplicar SIFT (Scale-Invariant Feature Transform).
       Se detectan puntos clave y descriptores tanto en la imagen de referencia como en la alineada, y se calcula una
       matriz de homografía para refinar el alineamiento si se encuentran suficientes coincidencias válidas.
       Finalmente, la homografía se aplica directamente a la imagen `uint16` alineada.

    Este procedimiento garantiza una alineación precisa sin alterar la resolución radiométrica de los datos.

    Args:
        fixed_rgb (np.ndarray): Imagen de referencia corregida geométricamente (usualmente la banda NIR o una RGB).
        moving_espectral (np.ndarray): Banda espectral a coregistrar, en formato `uint16`.
        ruta_fixed (str): Ruta al archivo de la imagen de referencia.
        ruta_moving (str): Ruta al archivo de la imagen a alinear.
        metadata_tif (pd.DataFrame): Metadatos correspondientes a las imágenes TIF.
        metadata_jpg (pd.DataFrame): Metadatos correspondientes a las imágenes JPG.

    Returns:
        np.ndarray: Imagen `moving_espectral` coregistrada respecto a la referencia, en formato `uint16`.

    Notas:
        - Si no se detectan suficientes puntos clave para estimar la homografía, se utiliza únicamente la traslación afín.
        - La calidad del coregistro puede evaluarse mediante el valor de NCC (Normalized Cross-Correlation), que se imprime.
        - Este proceso se realiza banda por banda (1 a 4), tomando como referencia la banda 5 (NIR) u otra predefinida.
    """

    print(f"Iniciando coregistro para {ruta_moving}")
    print(f"Tamaño original moving_espectral (uint16): {moving_espectral.shape}")

    # Reiniciar variables de transformación antes de cada iteración
    kp1, kp2, des1, des2 = None, None, None, None
    M_affine = None
    M_homography = None

```

```
    # Alinea una banda espectral con la referencia RGB
    # Obtener centros ópticos desde metadatos
    center_x_fixed, center_y_fixed = obtener_centros_opticos(ruta_fixed, metadata_tif, metadata_jpg)
    center_x_moving, center_y_moving = obtener_centros_opticos(ruta_moving, metadata_tif, metadata_jpg)

    # Calcular desplazamiento inicial
    delta_x = center_x_fixed - center_x_moving
    delta_y = center_y_fixed - center_y_moving


    # **PASO 1: Aplicar traslación afín en imagen ORIGINAL `uint16`**
    M_affine = np.float32([[1, 0, delta_x], [0, 1, delta_y]])
    aligned_uint16 = cv2.warpAffine(moving_espectral, M_affine, (fixed_rgb.shape[1], fixed_rgb.shape[0]),flags=cv2.INTER_NEAREST)
    print(f"Traslación afín aplicada: ΔX={delta_x}, ΔY={delta_y}")

    # COREGISTRO CON SIFT
    # **PASO 2: Convertir imagen alineada y fija a uint8 TEMPORALMENTE para SIFT**
    moving_sift = preparar_para_sift(aligned_uint16)

    if fixed_rgb.ndim == 3:
        fixed_gray = cv2.cvtColor(fixed_rgb, cv2.COLOR_BGR2GRAY)
    else:
        fixed_gray = fixed_rgb

    fixed_sift = preparar_para_sift(fixed_gray)
    print(f"Aplicando SIFT en {ruta_moving}")

    # **PASO 3: Detectar puntos clave con SIFT**
    sift = cv2.SIFT_create()
    kp1, des1 = sift.detectAndCompute(moving_sift, None)
    kp2, des2 = sift.detectAndCompute(fixed_sift, None)
    print(f"Puntos clave detectados: {len(kp1)} en moving, {len(kp2)} en fixed")

    if des1 is None or des2 is None or len(kp1) < 10 or len(kp2) < 10:
        print(f"No se detectaron suficientes puntos clave en {ruta_moving}. Se usará solo la traslación afín.")
        return aligned_uint16  # Devolver solo la alineación afín si no hay suficientes puntos clave

    # **PASO 4: Calcular homografía**
    FLANN_INDEX_KDTREE = 1
    index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
    search_params = dict(checks=50)
    flann = cv2.FlannBasedMatcher(index_params, search_params)
    matches = flann.knnMatch(des1, des2, k=2)

    # Filtrar buenos matches
    good_matches = [m for m, n in matches if m.distance < 0.7 * n.distance]
    print(f"Imagen de referencia fija para {ruta_moving}: {ruta_fixed}")

    if len(good_matches) > 4:
```

```
        src_pts = np.float32([kp1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
        dst_pts = np.float32([kp2[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)
        M_homography, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
        print(f"Matriz de homografía para {ruta_moving}:\n{M_homography}")

        if M_homography is not None:
            # **PASO 5: Aplicar la homografía sobre la imagen alineada `uint16`**
            aligned_final = cv2.warpPerspective(aligned_uint16, M_homography, (fixed_rgb.shape[1], fixed_rgb.shape[0]), flags=cv2.INTER_NEAREST)
            print(f"Homografía aplicada exitosamente en {ruta_moving}")

            # **PASO 6: Calcular métrica de calidad NCC**
            valor_ncc = np.corrcoef(fixed_sift.flatten(), preparar_para_sift(aligned_final).flatten())[0, 1]
            print(f"NCC (Correlación Cruzada Normalizada) para {ruta_moving}: {valor_ncc:.4f}")

            # Guardar en archivo
            if ncc_log_path:
                with open(ncc_log_path, 'a') as f:
                    f.write(f"{os.path.basename(ruta_moving)}: {valor_ncc:.4f}\n")
        else:
            print("No se pudo calcular la homografía. Usando solo traslación afín.")
            aligned_final = aligned_uint16
    else:
        print("No se encontraron suficientes coincidencias para la coregistración. Usando solo traslación afín.")
        aligned_final = aligned_uint16

    print(f"Tamaño aligned final antes de devolver (uint16): {aligned_final.shape}")

    return aligned_final, valor_ncc if M_homography is not None else None




# ═══════════════ PROCESAMIENTO PRINCIPAL ═══════════════
def procesar_grupos(grupos, directorio_salida, progress_file, metadata_tif, metadata_jpg):
    """
    Procesa grupos de imágenes multiespectrales agrupadas por nombre base y genera imágenes TIF multibanda coregistradas.

    Para cada grupo de imágenes que contiene las cinco bandas espectrales (Blue, Green, Red, RedEdge, NIR), esta función:

    1. Usa la banda NIR (banda 5) como imagen de referencia fija.
    2. Coregistra cada una de las bandas 1–4 respecto a la NIR usando:
        - Traslación inicial basada en centros ópticos desde metadatos.
        - Refinamiento geométrico mediante homografía estimada con SIFT.
    3. Ensambla las cinco bandas alineadas en una única imagen multibanda (TIF).
```

```
    4. Guarda el archivo resultante en el directorio de salida, insertando un '0' en la posición 7 del nombre
       para mantener la consistencia con el nombre base de las imágenes RGB.
    5. Registra el archivo procesado en `progress_file`.

    Args:
        grupos (dict): Diccionario de grupos generado por `cargar_grupos_imagenes()`, con imágenes y rutas.
        directorio_salida (str): Carpeta donde se guardarán los TIF multibanda coregistrados.
        progress_file (str): Ruta del archivo de texto donde se registran los archivos procesados.
        metadata_tif (pd.DataFrame): Metadatos correspondientes a las imágenes TIF.
        metadata_jpg (pd.DataFrame): Metadatos correspondientes a las imágenes JPG.

    Notas:
        - Solo se procesan los grupos que contienen las cinco bandas espectrales.
        - La banda NIR no se coregistra (se agrega directamente al conjunto).
        - Se verifica que todas las bandas tengan el mismo tamaño antes de guardar.
        - El nombre del archivo de salida sigue el formato `DJI_XXX0_obX_z_tX.TIF` (con '0' insertado).

    Raises:
        Excepciones durante el procesamiento de un grupo se capturan y reportan, pero no detienen el flujo general.
    """

    # Configuración inicial
    os.makedirs(directorio_salida, exist_ok=True)
    bandas_orden = [1, 2, 3, 4, 5]  # Blue, Green, Red, RE, NIR

    # Inicializar archivo de resultados NCC
    ncc_log_path = os.path.join(directorio_salida, "ncc_resultados.txt")
    if os.path.exists(ncc_log_path):
        os.remove(ncc_log_path)
    with open(ncc_log_path, 'w') as f:
        f.write("Resultados de NCC por banda coregistrada\n# Formato: nombre_archivo : valor_NCC\n")

    for nombre_base, datos in grupos.items():
        try:
            # Validar que el grupo tiene las 5 bandas
            if len(datos['bandas']) != 5:
                print(f"Grupo incompleto {nombre_base}. Bandas disponibles: {list(datos['bandas'].keys())}")
                continue
            print(f"Procesando grupo: {nombre_base}")

            # Obtener imagen y ruta de la banda NIR
            if 5 not in datos['bandas']:
                print(f"Banda NIR ausente en {nombre_base}. Omitiendo...")
                continue

            # Obtener ruta y datos de la banda NIR
            nir_info = datos['bandas'][5]
```

```
            ruta_nir = nir_info['ruta']
            imagen_nir = nir_info['imagen']
            print(f"Usando banda NIR como referencia fija: {ruta_nir} | Tamaño: {imagen_nir.shape}")

            bandas_coreg = []
            ncc_vals = []
            # Verificación de bandas
            for banda in bandas_orden:
                if banda not in datos['bandas']:
                    print(f"Banda {banda} ausente. Omitiendo ... ")
                    continue

                imagen_banda = datos['bandas'][banda]['imagen']
                ruta_banda = datos['bandas'][banda]['ruta']

                if banda == 5:
                    # No se coregistra la NIR, se agrega directamente
                    bandas_coreg.append(imagen_banda)
                    print(f"Banda NIR añadida sin modificación.")
                    continue

                # Coregistrar banda con respecto a la NIR
                img_coreg, ncc_val  = coregistrar_banda(
                    fixed_rgb=imagen_nir,
                    moving_espectral=imagen_banda,
                    ruta_fixed=ruta_nir,
                    ruta_moving=ruta_banda,
                    metadata_tif=metadata_tif,
                    metadata_jpg=metadata_jpg
                )
                bandas_coreg.append(img_coreg)
                if ncc_val is not None:
                    ncc_vals.append(ncc_val)
                print(f"Banda {banda} coregistrada respecto a NIR.")

            if len(bandas_coreg) != 5:
                print(f"Error: {nombre_base} tiene menos de 5 bandas coregistradas. Omitiendo ... ")
                continue

            # **PASO EXTRA: Verificar que todas las bandas tengan el mismo tamaño**
            ref_shape = bandas_coreg[0].shape  # Usamos la primera banda como referencia
            for i, banda in enumerate(bandas_coreg):
                if banda.shape != ref_shape:
                    print(f"Inconsistencia en {nombre_base}: Banda {bandas_orden[i]} tiene tamaño {banda.shape}, se ajustará a {ref_shape}.")

            ##  Crear y guardar imagen multibanda
            # Insertar '0' en la posición 7 para que el nombre coincida con la JPG
            nombre_base_corregido = nombre_base[:7] + '0' + nombre_base[7:]
            nombre_salida = f"{nombre_base_corregido}.TIF"

            # Guardar valor de NCC
            if len(ncc_vals) > 0:
```

```
                avg_ncc = sum(ncc_vals) / len(ncc_vals)
                with open(ncc_log_path, 'a') as f:
                    f.write(f"{nombre_salida}: {avg_ncc:.4f}\n")

            # Guardar TIF
            with rasterio.open(
                os.path.join(directorio_salida, nombre_salida),
                'w',
                driver='GTiff',
                height=bandas_coreg[0].shape[0],
                width=bandas_coreg[0].shape[1],
                count=5,
                dtype=np.uint16,
                photometric='MINISBLACK'
            ) as dst:
                for i, banda in enumerate(bandas_coreg, 1):
                    dst.write(banda, i)

            with open(progress_file, 'a') as f:
                f.write(f"{nombre_salida}\n")

            print(f"{nombre_salida} | Tamaño: {bandas_coreg[0].shape}")

        except Exception as e:
            print(f"Error en {nombre_base}: {str(e)}")
            continue




# ═══════════════ FUNCIÓN DE INTERFAZ ═══════════════
def ejecutar_coregistro(directorio_tif, directorio_rgb, directorio_salida,
metadata_tif_csv, metadata_jpg_csv):
    """
    Ejecuta el proceso completo de coregistro multiespectral por zona de estudio.

    Esta función actúa como interfaz principal del módulo de coregistro. A partir de las
imágenes geométricamente
    corregidas (TIF y JPG), realiza el agrupamiento por nombre base, selecciona la banda
NIR como referencia,
    y coregistra todas las bandas multiespectrales (1 a 4) con respecto a ella.
Posteriormente, ensambla y guarda
    una imagen multibanda alineada por grupo.

    Flujo general:
        1. Carga los metadatos desde los archivos CSV de TIF y JPG.
        2. Agrupa las imágenes corregidas en conjuntos coherentes por nombre base.
        3. Procesa cada grupo mediante `procesar_grupos`, que aplica:
            - Traslación basada en centros ópticos (metadatos).
            - Homografía refinada con SIFT (si es posible).
            - Ensamblaje y guardado de imágenes multibanda (TIF).
        4. Registra el progreso por zona en un archivo `progreso_coregistro.txt`.

    Args:
```

```
        directorio_tif (str): Ruta al directorio con imágenes TIF corregidas geométricamente.
        directorio_rgb (str): Ruta al directorio con imágenes JPG corregidas geométricamente.
        directorio_salida (str): Carpeta de salida para los resultados del coregistro (TIF multibanda).
        metadata_tif_csv (str): Ruta del archivo CSV con los metadatos de imágenes TIF.
        metadata_jpg_csv (str): Ruta del archivo CSV con los metadatos de imágenes JPG.

    Notas:
        - Este procedimiento debe ejecutarse después de la corrección geométrica individual.
        - El archivo de progreso evita duplicación en ejecuciones posteriores.
        - Es compatible con procesamiento por lotes en múltiples zonas de estudio.
    """
    progress_file = os.path.join(directorio_salida, "progreso_coregistro.txt")

     # 1. Cargar los metadatos
    metadata_tif = pd.read_csv(metadata_tif_csv)
    metadata_jpg = pd.read_csv(metadata_jpg_csv)

    # 2. Cargar y agrupar imágenes corregidas por nombre base
    grupos = cargar_grupos_imagenes(directorio_tif, directorio_rgb, progress_file)

    # 3. Procesar cada grupo → coregistrar bandas espectrales respecto a NIR
    procesar_grupos(grupos, directorio_salida, progress_file, metadata_tif, metadata_jpg)


```