```python
"""
CÓDIGO DE FUNCIONES AUXILIARES PARA CORRECCIÓN GEOMÉTRICA

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 rasterio
import os
import re



# ═══════════════ PARÁMETROS DE CALIBRACIÓN ═══════════════
def extract_camera_params(meta_row, tipo_imagen):
    """
    Extrae los parámetros de calibración de la cámara a partir de los metadatos de una imagen multiespectral o RGB.

    Esta función genera tres elementos clave utilizados en la corrección geométrica de imágenes aéreas:
    1. La matriz intrínseca de la cámara (`camera_matrix`), que contiene la longitud focal y el centro óptico.
    2. El vector de coeficientes de distorsión radial y tangencial (`dist_coeffs`), que permite corregir la deformación causada por la lente.
    3. La matriz de homografía (`homography_matrix`), que se utiliza para corregir inclinaciones (roll, pitch, yaw) aplicando una transformación en perspectiva.

    La extracción depende del tipo de imagen:
    - Para imágenes multiespectrales (`TIF`), se obtienen los coeficientes desde el campo `Perspective Distortion`.
    - Para imágenes RGB (`JPG`), se extraen desde el campo `Dewarp Data`, que requiere un preprocesamiento adicional para aislar los valores relevantes.

    Args:
        meta_row (pd.Series): Fila de un DataFrame con los metadatos de la imagen correspondiente.
        tipo_imagen (str): Tipo de imagen a procesar. Acepta 'TIF' para multiespectrales y 'JPG' para RGB.

    Returns:
        tuple:
            - camera_matrix (np.ndarray): Matriz 3×3 con los parámetros intrínsecos de la cámara.
            - dist_coeffs (np.ndarray): Vector con cinco coeficientes de distorsión (k1, k2, p1, p2, k3).
```

```
            - homography_matrix (np.ndarray): Matriz 3×3 de transformación homográfica.

    Raises:
        ValueError: Si el tipo de imagen especificado no es reconocido.
        None values: Si los campos necesarios están mal formateados o incompletos en los metadatos.
    """
    # Extraer valores comunes
    fx = fy = float(meta_row['Calibrated Focal Length'])
    cx = float(meta_row['Calibrated Optical Center X'])
    cy = float(meta_row['Calibrated Optical Center Y'])

    if tipo_imagen == "TIF":
        # Coeficientes de distorsión para TIF (Perspective Distortion)
        k1, k2, k3, p1, p2 = map(float, meta_row['Perspective Distortion'].split(','))

    elif tipo_imagen == "JPG":
        # Coeficientes para JPG (Dewarp Data)
        dewarp_data_str = str(meta_row['Dewarp Data']).strip().split(';')[-1]
        dewarp_data_values = list(map(float, re.split(r'[\s,]+', dewarp_data_str)))

        if len(dewarp_data_values) < 9:  # Validación de estructura
            print(f"Error: `Dewarp Data` incompleto para {meta_row['File Name']}")
            return None, None, None

        k1, k2, p1, p2, k3 = dewarp_data_values[4:9]

    else:
        raise ValueError(f"Formato de imagen no reconocido: {tipo_imagen}")

    # Construcción de la matriz intrínseca de la cámara
    camera_matrix = np.array([
        [fx, 0, cx],
        [0, fy, cy],
        [0,  0,  1]
    ])

    dist_coeffs = np.array([k1, k2, p1, p2, k3])

    # Extraer y validar la matriz de homografía
    try:
        homography_values = list(map(float, meta_row['Dewarp H Matrix'].split(',')))
        homography_matrix = np.array(homography_values).reshape(3, 3)
    except (ValueError, IndexError):
        print(f"Error al extraer `Dewarp H Matrix` para {meta_row['File Name']}")
        return None, None, None

    return camera_matrix, dist_coeffs, homography_matrix




# ═══════════════ CORRECCIÓN DE DISTORSIÓN Y RECORTE ═══════════════
def correct_lens_distortion(image, camera_matrix, dist_coeffs):
    """
```

```
    Aplica corrección de distorsión de lente a una imagen aérea utilizando los parámetros intrínsecos de la cámara.

    Esta función corrige las deformaciones ópticas causadas por la lente (distorsión radial y tangencial), que son
    comunes en sensores de cámaras aéreas montadas en drones. La corrección se realiza utilizando los parámetros de
    calibración de la cámara obtenidos previamente desde los metadatos (`camera_matrix` y `dist_coeffs`).

    Internamente, se calcula una nueva matriz de cámara óptima para maximizar el campo de visión útil y se aplica
    la transformación utilizando las funciones `getOptimalNewCameraMatrix` y `undistort` de OpenCV.

    Args:
        image (np.ndarray): Imagen de entrada (puede ser multicanal o monocanal), sin corregir.
        camera_matrix (np.ndarray): Matriz intrínseca 3×3 de la cámara.
        dist_coeffs (np.ndarray): Vector con cinco coeficientes de distorsión (k1, k2, p1, p2, k3).

    Returns:
        tuple:
            - undistorted_image (np.ndarray): Imagen corregida, con la distorsión óptica eliminada.
            - roi (tuple): Región de interés (x, y, w, h) útil de la imagen corregida, recomendada para recorte posterior.

    Notas:
        Esta corrección es un paso previo necesario antes de aplicar transformaciones geométricas como homografías
        o coregistro multiespectral, ya que garantiza que las imágenes estén libres de efectos ópticos sistemáticos.
    """
    h, w = image.shape[:2]
    new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(camera_matrix, dist_coeffs, (w, h), 1, (w, h))
    undistorted_image = cv2.undistort(image, camera_matrix, dist_coeffs, None, new_camera_matrix)
    return undistorted_image, roi



def crop_center(image, target_width=1500, target_height=1150):
    """
    Recorta una región central de la imagen con dimensiones fijas, centrada respecto al eje óptico.

    Esta función permite normalizar el tamaño de las imágenes corregidas geométricamente
    extrayendo un ROI (región de interés) de dimensiones específicas centrado en la imagen. Es
    especialmente útil cuando se desea mantener un área común de análisis libre de distorsiones
    ópticas y bordes potencialmente inválidos tras la corrección de lente.
```

```
135
136     Si la imagen es más pequeña que el tamaño especificado, no se recorta y se devuelve
    la imagen original.
137
138     Args:
139         image (np.ndarray): Imagen de entrada (corregida geométricamente), de 2 o 3
    canales.
140         target_width (int): Ancho del recorte centrado en píxeles. Por defecto 1500 px.
141         target_height (int): Alto del recorte centrado en píxeles. Por defecto 1150 px.
142
143     Returns:
144         np.ndarray: Imagen recortada centrada. Si la imagen es más pequeña que el ROI,
    se retorna sin modificar.
145
146     Notes:
147         - Este tamaño de ROI se definió con base en el área mínima común que resulta
    tras aplicar la corrección
148           de distorsión en la base de datos completa.
149         - Se recomienda aplicar este recorte después de la corrección de lente y antes
    del coregistro o segmentación.
150     """
151     h, w = image.shape[:2]
152     if h < target_height or w < target_width:
153         print(f"Imagen demasiado pequeña para recortar a {target_width}x{target_height},
    se mantiene original.")
154         return image
155
156     center_x = w // 2
157     center_y = h // 2
158
159     x_start = center_x - (target_width // 2)
160     x_end = center_x + (target_width // 2)
161     y_start = center_y - (target_height // 2)
162     y_end = center_y + (target_height // 2)
163
164     return image[y_start:y_end, x_start:x_end]
165
166
167
168
169 # ═══════════════════ GUARDADO DE IMÁGENES Y PROGRESO ═══════════════════
170 def save_corrected_image(image, output_path, tipo_imagen, profile=None):
171     """
172     Guarda una imagen corregida en el formato y tipo de dato adecuado según su tipo
    ('TIF' o 'JPG').
173
174     Esta función gestiona el guardado de imágenes corregidas geométricamente asegurando:
175     - En imágenes TIF, que se mantenga la estructura multibanda y el formato `uint16`,
    comúnmente utilizado para
176       imágenes multiespectrales de alta precisión.
177     - En imágenes JPG, que se convierta adecuadamente a `uint8` para visualización o
    tareas donde no se requiere
178       profundidad espectral extendida.
179
```

```
    Para imágenes TIF, también actualiza el perfil de metadatos (`profile`) con las dimensiones actuales y el tipo
    de datos antes de la escritura. Es compatible tanto con imágenes de una sola banda como con multibanda.

    Args:
        image (np.ndarray): Imagen corregida (2D o 3D), lista para guardarse.
        output_path (str): Ruta completa donde se almacenará la imagen.
        tipo_imagen (str): Tipo de imagen. Debe ser 'TIF' (multiespectral) o 'JPG' (RGB).
        profile (dict, optional): Perfil de raster (metadatos de rasterio) necesario para guardar archivos .TIF.
                                  Solo es requerido cuando `tipo_imagen == 'TIF'`.

    Notes:
        - Las imágenes TIF se guardan usando Rasterio, permitiendo el manejo de múltiples bandas.
        - Las imágenes JPG se guardan usando OpenCV (`cv2.imwrite`) en formato comprimido.
        - Si los tipos de datos no son consistentes (`uint16` para TIF, `uint8` para JPG), se ajustan automáticamente
          mediante recorte al rango permitido y conversión de tipo.
    """
    if tipo_imagen == "TIF":
        # Asegurar que la imagen sea de tipo uint16
        if image.dtype != np.uint16:
            image = np.clip(image, 0, 65535).astype(np.uint16)
        # Obtener dimensiones actualizadas
        h, w = image.shape[:2]
        # Actualizar el perfil con el nuevo tamaño
        profile.update(height=h, width=w, dtype=rasterio.uint16)

        # Si la imagen tiene solo 2 dimensiones (una sola banda), actualizar perfil correctamente
        if len(image.shape) == 2:
            profile.update(count=1)  # Imagen de una sola banda
            with rasterio.open(output_path, 'w', **profile) as dst:
                dst.write(image, 1)  # Guardar como banda única
        else:
            profile.update(dtype=rasterio.uint16, count=image.shape[2]) # Imagen multibanda
            with rasterio.open(output_path, 'w', **profile) as dst:
                for i in range(image.shape[2]):  # Escribir cada banda
                    dst.write(image[:, :, i], i + 1)

    else:
        # Para JPG, guardar como uint8
        if image.dtype != np.uint8:
            image = np.clip(image, 0, 255).astype(np.uint8)

        cv2.imwrite(output_path, image)



def save_processed_image(progress_file, image_name):
```

```
    """
    Registra el nombre de una imagen procesada en un archivo de progreso para evitar reprocesamientos innecesarios.

    Esta función añade de forma incremental (`append`) el nombre de cada imagen que ha sido procesada con éxito
    durante la ejecución del flujo de preprocesamiento. Esto permite implementar una lógica de control de progreso
    basada en persistencia, útil para:
    - Reanudar ejecuciones interrumpidas sin duplicar trabajo.
    - Omitir imágenes ya corregidas en iteraciones futuras.

    Si el archivo de progreso no existe, la función lo crea automáticamente antes de escribir.

    Args:
        progress_file (str): Ruta al archivo de texto donde se almacenan los nombres de imágenes procesadas.
        image_name (str): Nombre del archivo de imagen (con extensión) que se va a registrar.

    Notes:
        - Cada nombre se escribe en una nueva línea.
        - Este archivo suele llamarse `progreso.txt` o `progreso_coregistro.txt` según la etapa del flujo.
        - El archivo puede analizarse posteriormente para reejecución selectiva.
    """
    try:
        # Asegurar que el archivo existe antes de escribir en él
        if not os.path.exists(progress_file):
            open(progress_file, "w").close()  # Crea el archivo vacío si no existe

        # Escribir el nombre de la imagen procesada
        with open(progress_file, "a") as f:
            f.write(image_name + "\n")

    except Exception as e:
        print(f"Error al escribir en progress_file: {e}")
```