```
"""
MÓDULO PRINCIPAL DE CORRECCIÓN GEOMÉTRICA Y COREGISTRO MULTIESPECTRAL

Entradas:
- `data/Input/[zona]/`: Carpeta con las imágenes originales (.TIF y .JPG) de cada zona.
- Metadatos (`metadata_tif.csv`, `metadata_jpg.csv`) si ya existen.

Salidas:
- `data/Output/[zona]/correccion_geometrica/`: Imágenes corregidas geométricamente.
- `data/Output/[zona]/coregistro/`: Imágenes coregistradas multibanda y RGB transferidas.
- Archivos `progreso.txt` y `progreso_coregistro.txt` que registran el avance del procesamiento por imagen.

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
  ```
- Estructura modular compatible con `config.py` y scripts auxiliares en `scripts/`.

Uso:
    python main_correccion_geometrica.py
"""

# ═══════════════════════════════ IMPORTACIÓN DE LIBRERÍAS Y CONFIGURACIÓN ═══════════════════════════════
import os
import numpy as np
import pandas as pd
import rasterio
import cv2
import shutil
from pathlib import Path
from config import INPUT_DIR, OUTPUT_DIR, SUBCARPETA_CORRECCION, SUBCARPETA_COEGISTRO, listar_zonas_disponibles
from scripts.metadata_extractor import generate_metadata_csv
from scripts.utils import correct_lens_distortion, crop_center, extract_camera_params, save_corrected_image, save_processed_image
from scripts.coregistro import ejecutar_coregistro
from scripts.generar_rgb import generar_rgb_transferido_batch



# ═══════════════════════════════ FUNCIÓN: CORRECCIÓN GEOMÉTRICA DE UNA IMAGEN ═══════════════════════════════
def corregir_imagen(image_path, metadata, output_folder, tipo_imagen="TIF", progress_file="progreso.txt"):
```

```
    """
    Aplica la corrección geométrica a una imagen aérea multiespectral (TIF) o RGB (JPG) a partir de sus metadatos de calibración.

    Esta función realiza tres pasos fundamentales para cada imagen:
    1. Corrección de distorsión de lente con los parámetros de la cámara.
    2. Corrección de perspectiva o paralaje mediante homografía, si hay inclinación significativa (roll, pitch, yaw).
    3. Recorte centrado para normalizar el tamaño final a un ROI definido (1500×1150 px).

    El proceso se aplica tanto a imágenes multiespectrales (`.TIF`) como RGB (`.JPG`), utilizando metadatos previamente
    extraídos. La imagen corregida se guarda en una carpeta de salida y su nombre se registra en un archivo
    de progreso para evitar reprocesamiento en ejecuciones futuras.

    Args:
        image_path (str): Ruta completa del archivo de imagen a corregir (TIF o JPG).
        metadata (pd.DataFrame): DataFrame con los metadatos extraídos vía ExifTool (uno por tipo de imagen).
        output_folder (str): Carpeta donde se guardará la imagen corregida geométricamente.
        tipo_imagen (str): Tipo de imagen a procesar. Puede ser 'TIF' (multiespectral) o 'JPG' (RGB).
        progress_file (str): Ruta del archivo de texto donde se registran los nombres de imágenes ya procesadas.

    Returns:
        None. Guarda directamente la imagen corregida en el disco.

    Notas:
        - Si la imagen ya ha sido procesada previamente (aparece en `progress_file`), se omite automáticamente.
        - La corrección geométrica por homografía solo se aplica si alguno de los valores de roll, pitch o yaw supera ±0.1.
        - Las imágenes TIF se guardan con su perfil rasterio original actualizado (multibanda o monobanda).
        - Las imágenes JPG se guardan en formato comprimido (`uint8`) utilizando OpenCV.

    Ejemplo de uso:
        corregir_imagen("DJI_0011_ob6_e_t3.TIF", metadatos_tif, "Output/Exconvento/correccion_geometrica", tipo_imagen="TIF")

    Requiere:
        - Funciones auxiliares: `extract_camera_params`, `correct_lens_distortion`, `crop_center`,
          `save_corrected_image`, `save_processed_image`.
        - Metadatos válidos y alineados con el nombre exacto del archivo (`File Name` en el CSV).
    """

    image_name = os.path.basename(image_path)
    meta_row = metadata[metadata['File Name'].str.strip().str.lower() == image_name.lower()]
```

```
    if meta_row.empty:
        print(f"Metadatos no encontrados para {image_name}. Omitiendo.")
        return
    meta_row = meta_row.iloc[0]

    # Verificar si la imagen ya ha sido procesada
    if os.path.exists(progress_file):
        with open(progress_file, "r") as f:
            processed_images = set(f.read().splitlines())
        if image_name in processed_images:
            print(f"{image_name} ya fue procesada. Omitiendo.")
            return

    # Cargar la imagen
    if tipo_imagen == "TIF":
        with rasterio.open(image_path) as src:
            img = np.stack([src.read(i + 1) for i in range(src.count)], axis=-1)
            profile = src.profile
    else:
        img = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)

    # Extraer parámetros de calibración
    camera_matrix, dist_coeffs, homography_matrix = extract_camera_params(meta_row,
tipo_imagen)
    if camera_matrix is None:
        return

    # Paso 1: Corrección de distorsión
    undistorted_img, roi = correct_lens_distortion(img, camera_matrix, dist_coeffs)

    # Paso 2: Corrección geométrica
    h, w = undistorted_img.shape[:2]
        # Extraer Roll, Pitch, Yaw
    roll, pitch, yaw = float(meta_row['Roll']), float(meta_row['Pitch']),
float(meta_row['Yaw'])
        # Aplicar homografía en todas las imágenes, pero en ortogonales solo si hay
inclinación significativa
    if abs(roll) > 0.1 or abs(pitch) > 0.1 or abs(yaw) > 0.1:
        undistorted_img = cv2.warpPerspective(undistorted_img, homography_matrix, (w,
h), flags=cv2.INTER_LINEAR)
        print(f"Corrección geométrica aplicada en {image_name} debido a Roll={roll},
Pitch={pitch}, Yaw={yaw}")
    else:
        print(f"No se requiere corrección en la imagen ortogonal {image_name}, sin
inclinación significativa.")

    # Paso 3: Recorte al tamaño deseado
    corrected_img = crop_center(undistorted_img)

    # Guardar imagen corregida
    output_path = os.path.join(output_folder, image_name)
    save_corrected_image(corrected_img, output_path, tipo_imagen, profile if tipo_imagen
== "TIF" else None)

    # Guardar imagen en progreso.txt
```

```
135     save_processed_image(progress_file, image_name)
136 
137     print(f"Imagen corregida guardada en: {output_path}")
138 
139 
140 
141 # ═══════════════════════════════ FUNCIÓN: PROCESAMIENTO COMPLETO DE UNA ZONA DE ESTUDIO ═══════════════════════════════
142 def procesar_zona(zona):
143     """
144     Ejecuta el preprocesamiento completo para una zona de estudio, incluyendo:
145     1. Corrección geométrica de imágenes TIF y JPG.
146     2. Coregistro espectral multibanda respecto a la banda NIR.
147     3. Generación de imágenes RGB visualmente naturalizadas por transferencia de color.
148 
149     Este flujo es aplicado a cada carpeta individual dentro del directorio de entrada (`INPUT_DIR`),
150     correspondiente a una zona o sitio de estudio. El procesamiento asegura la generación de productos
151     corregidos geométricamente, alineados espacialmente y listos para análisis espectral o segmentación.
152 
153     Pasos del procesamiento:
154     - Verifica y/o genera automáticamente los archivos de metadatos (`metadata_tif.csv`, `metadata_jpg.csv`)
155       usando `ExifTool`, en caso de que no existan.
156     - Aplica corrección geométrica a todas las imágenes .TIF y .JPG usando `corregir_imagen()`.
157     - Ejecuta el coregistro multiespectral con `ejecutar_coregistro()`, usando la banda NIR como referencia.
158     - Genera una imagen RGB naturalizada a partir de las bandas 3-2-1 del TIF, transferida al estilo del JPG.
159 
160     Args:
161         zona (str): Nombre de la carpeta de la zona de estudio, ubicada dentro de `data/Input`.
162 
163     Returns:
164         None. Todos los productos son guardados en disco en subcarpetas dentro de `data/Output/[zona]/`.
165 
166     Notas:
167         - Las imágenes corregidas geométricamente se almacenan en `Output/[zona]/correccion_geometrica/`.
168         - Las imágenes coregistradas (TIF multibanda y RGB transferido) se almacenan en `Output/[zona]/coregistro/`.
169         - El archivo de progreso `progreso.txt` evita reprocesamiento redundante por imagen.
170         - Si ExifTool no está instalado, el procesamiento se detiene con un mensaje de error informativo.
171 
172     Requiere:
173         - Librerías: `os`, `pandas`, `shutil`, `cv2`, `rasterio`
174         - Funciones auxiliares: `corregir_imagen`, `ejecutar_coregistro`, `generar_rgb_transferido_batch`
```

```
175         - Script de metadatos: `generate_metadata_csv()` desde
    `scripts.metadata_extractor.py`
176     """
177
178     print(f"\n Procesando zona: {zona}")
179     ruta_entrada = os.path.join(INPUT_DIR, zona)
180     ruta_corr = os.path.join(OUTPUT_DIR, zona, SUBCARPETA_CORRECCION)
181     ruta_coreg = os.path.join(OUTPUT_DIR, zona, SUBCARPETA_COEGISTRO)
182
183     os.makedirs(ruta_corr, exist_ok=True)
184     os.makedirs(ruta_coreg, exist_ok=True)
185
186     # Cargar o Extraer metadatos
187     metadata_tif_csv = os.path.join(ruta_entrada, 'metadata_tif.csv')
188     metadata_jpg_csv = os.path.join(ruta_entrada, 'metadata_jpg.csv')
189
190     if not os.path.exists(metadata_tif_csv) or not os.path.exists(metadata_jpg_csv):
191         print(f"\nMetadatos no encontrados en {zona}. Generando automáticamente con
    ExifTool ... ")
192
193         # Verificar si ExifTool está disponible en el sistema
194         if shutil.which("exiftool") is None:
195             print("Error: ExifTool no está instalado o no está en el PATH del sistema.")
196             print("Instala ExifTool desde https://exiftool.org/ y asegúrate de que esté
    accesible por consola.")
197             return
198
199         # Generar metadatos
200         exito = generate_metadata_csv(ruta_entrada)
201         if not exito:
202             print(f"No se pudieron generar los metadatos para {zona}. Omitiendo esta
    zona.")
203             return
204
205     # Cargar los metadatos una vez estén asegurados
206     metadata_tif = pd.read_csv(metadata_tif_csv)
207     metadata_jpg = pd.read_csv(metadata_jpg_csv)
208
209     # Definir ruta del archivo de progreso geométrico por zona
210     progress_file = os.path.join(ruta_corr, "progreso.txt")
211
212     # Corrección geométrica de imágenes TIF y JPG
213     for fname in os.listdir(ruta_entrada):
214         fpath = os.path.join(ruta_entrada, fname)
215         if fname.lower().endswith(".tif"):
216             corregir_imagen(fpath, metadata_tif, ruta_corr, tipo_imagen="TIF",
    progress_file=progress_file)
217         elif fname.lower().endswith((".jpg", ".jpeg")):
218             corregir_imagen(fpath, metadata_jpg, ruta_corr, tipo_imagen="JPG",
    progress_file=progress_file)
219
220
221     # Coregistro espectral respecto a NIR
222     ejecutar_coregistro(ruta_corr, ruta_corr, ruta_coreg, metadata_tif_csv,
    metadata_jpg_csv)
```

```python

    # Generar imagen RGB desde TIF coregistrado y JPG corregido
    generar_rgb_transferido_batch(directorio_tif=ruta_coreg, directorio_jpg=ruta_corr, directorio_salida=ruta_coreg, verbose=True)


# ═══════════════════════════════ BLOQUE PRINCIPAL DE EJECUCIÓN ═══════════════════════════════
if __name__ == "__main__":
    zonas = listar_zonas_disponibles()
    if not zonas:
        print("No se encontraron zonas en la carpeta de entrada.")
    for zona in zonas:
        procesar_zona(zona)


```