MEMORIA PRÁCTICA 5

Autores:
Aris Vazdekis Soria & Alejandra Ruiz de Adana Fleitas

Ejercicio 1

Apartado A: Detección de características con SIFT

Para este primer apartado, nos sumergimos en el uso de SIFT (Scale-Invariant Feature Transform) y desarrollamos una aplicación en Python que permite detectar características en una imagen y ajustar sus parámetros a través de una interfaz gráfica. A continuación, detallamos el proceso que seguimos y las dificultades que encontramos.

Para empezar, utilizamos la libreria Tkinter para construir una herramienta que fuera funcional y a la vez fácil de usar, con una interfaz modesta y moderna. Esto nos permitió cargar imágenes, aplicar SIFT y visualizar los resultados con los parámetros ajustables desde la interfaz. Estos parámetros incluyen Numero de caracteristicas, Umbral de contraste, Umbral de borde y Sigma, todos ellos esenciales para afinar la detección de características.

Aquí es donde realmente notamos el impacto de los ajustes de SIFT en la detección de puntos de interés. La interfaz nos permite ver visualmente cómo afectan los cambios de cada parámetro, algo que fue una experiencia muy valiosa y nos permitió entender de manera práctica cómo estos controles alteran la precisión de la detección. Durante este proceso, aprendimos que configurar correctamente el Umbral de contraste y el Umbral de borde es primordial para evitar la detección de demasiadas características o que las mismas no se detecten (valores recomendados 0.04 y 10 respectivamente). Además, para asegurar que los cambios fueran efectivos y observables, empleamos una escala en el parámetro sigma, lo que aumentó considerablemente la precisión de los puntos detectados.

En este apartado, logramos una comprensión más profunda de cómo funciona SIFT y de los parámetros que afectan la detección de puntos de interés. Nos sorprendió gratamente cómo este método es capaz de reconocer patrones en diferentes tipos de imágenes, independientemente de la escala o de cambios de contraste.

Apartado B, C, D: Transformaciones y búsqueda de áreas de interés

En estos apartados B, C y D, realmente mejoramos la aplicacion anterior, aplicando varias transformaciones sobre una imagen para luego detectar en versiones transformadas un área específica de interés. Implementamos distorsiones de barril (de nuestra practica numero 2), cambios de rotación, escalado y traslación, todas ajustables mediante sliders en la interfaz gráfica.

Para empezar, desarrollamos una función que aplica diferentes transformaciones a la imagen: rotación, escalado y traslación. También implementamos un efecto de distorsión de barril, que añade una variación interesante a la imagen original ya que en especial las

esquinas se estiran bastante (bajo valores negativos) y ponemos asi a prueba al detector de caracteristicas.

Para seleccionar un área de interés, usamos la función *selectROI* de OpenCV, que permite recortar una sección de la imagen sobre la cual queremos enfocar la detección. El reto aquí fue lograr que la aplicación localizara esta área en las versiones transformadas de la imagen de partida. Aquí es donde el trabajo previo con SIFT realmente brilló: una vez seleccionada el área de interés, el programa intenta localizarla en las imágenes transformadas.

Para lograr una detección robusta, utilizamos un matcher FLANN, que garantiza una coincidencia eficiente y precisa entre los puntos de interés de la imagen original y la transformada. Fue sorprendente ver lo bien que funcionaba esta técnica, y en gran medida, nos ayudó a superar los desafíos de alineación y escalado que presenta este tipo de búsqueda en imágenes transformadas.

Uno de los mayores retos aquí fue asegurarnos de que, cuando fuera imposible detectar las características debido a una fuerte distorsión de la imagen (por ejemplo, un escalado muy alto que haga desaparecer los bordes de la imagen o un efecto de barril que excluya las esquinas), el programa lanzara una advertencia en lugar de detectar puntos que bajo nuestras observaciones se veian aparentemente aleatorios. Para lograrlo, usamos varias funciones de verificación de contornos y comprobación de límites que garantizan que el área de interés se encuentre dentro de los bordes visibles y en una zona no distorsionada. De esta manera, la herramienta identifica si el área de interés está fuera de los límites de la imagen transformada antes de intentar detectar características.

```
def find_area_in_transformed_image(transformed_img, template_img, roi):
    # Verificar si el ROI está dentro de los límites de la imagen transformada
    roi_x, roi_y, roi_w, roi_h = roi
    if roi_x < 0 or roi_y < 0 or roi_x + roi_w > transformed_img.shape[1] or roi_y + roi_h > transformed_img.shape[0]:
        # Mostrar mensaje de error si el ROI está fuera de los límites
        messagebox.showerror("Error", "El área seleccionada está fuera de la imagen transformada. No se buscarán caract
        return None
```

Aquí observamos cómo se verifica que las dimensiones de nuestra área de interés se encuentren dentro de los límites de la imagen transformada, como vemos en el *if*. Este condicional se encarga de comprobar que el ROI no esté fuera del área visible de la imagen.

```
matches = flann.knnMatch(des_template, des_transformed, k=2)
good_matches = []
for m, n in matches:
    if m.distance < 0.7 * n.distance:  # Filtro de Lowe para coincidencias válidas
        good_matches.append(m)

# Verificar si se encontraron coincidencias suficientes
if len(good_matches) < 10:  # Ajusta este valor según sea necesario
    messagebox.showerror("Error", "No se encontraron coincidencias válidas en el área de interés transformada.")
    return None
```

En este otro apartado de la misma función, se verifica si las coincidencias son buenas calculando la distancia entre las coincidencias obtenidas. Usando el filtro de Lowe *(if m.distance < 0.7 * n.distance)*, el código filtra aquellas coincidencias que cumplan con el criterio de ser significativamente similares. Luego, el código evalúa si el número de coincidencias válidas (*good_matches)* es suficiente que bajo nuestro criterio el valor debe ser 10. Es de esta manera como logramos abordar los principales errores de nuestro codigo.