

Práctica 4 y 5

Visión por computador

Features y panoramas

César Moro Latorre - 815078

Alejandro Lalaguna Maza - 819860

1. Introducción

En este trabajo, se van a comparar diferentes métodos de extracción de características y emparejamiento de OpenCV para la creación de panoramas a partir de imágenes cuyo objetivo es evaluar su rendimiento en términos de precisión y eficiencia. Los métodos seleccionados para este estudio han sido ORB, SIFT y AKAZE.

- **ORB:** Es un método rápido y eficiente para la detección y descripción de puntos de interés en imágenes. Utiliza un algoritmo basado en orientación para identificar keypoints y calcular descriptores.
- **SIFT:** Proporciona una detección robusta de keypoints invariantes a cambios de escala y rotación. Utiliza técnicas basadas en diferencias de Gaussianas para encontrar puntos clave y calcular descriptores detallados.
- **AKAZE:** Es un método que funciona bien en una amplia variedad de condiciones de iluminación y escala. Similar a SIFT, emplea características no lineales para detectar keypoints y describir características locales.

2. Implementación de los métodos de extracción

El proceso seguido para la implementación de los siguientes métodos es el siguiente:

- **Extracción de características (*extract_features*):** Lo primero que hay que hacer es identificar puntos de interés en una imagen utilizando los diferentes métodos de extracción mencionados anteriormente. Se convierte la imagen a escala de grises para simplificar el procesamiento y posteriormente se aplica el método seleccionado para detectar puntos clave (*keypoints*) y calcular sus descriptores (*descriptors*) asociados. Si se proporciona un método no válido, por defecto utiliza el método ORB.
- **Emparejamiento por fuerza bruta (*match_features_brute_force*):** En este proceso, se comparan los descriptores obtenidos de dos conjuntos de características de diferentes imágenes. Utiliza el algoritmo de coincidencia de fuerza bruta proporcionado por OpenCV (*BMatcher*) con distancia Hamming, diseñado para encontrar coincidencias directas entre los descriptores. Este algoritmo ordena y devuelve las coincidencias según la distancia entre los descriptores, proporcionando así una lista ordenada de emparejamientos.
- **Emparejamiento por ratio test (*match_features_ratio_test*):** Similar al emparejamiento por fuerza bruta. Además, este método añade un paso adicional de filtrado para mejorar la calidad de los emparejamientos. Después de encontrar las coincidencias iniciales, calcula el ratio entre la distancia del mejor y el segundo mejor emparejamiento. Las coincidencias que no cumplen

con un criterio de ratio predefinido son descartadas, lo que resulta en un conjunto más selectivo de emparejamientos significativos y de alta calidad.

3. Comparativa de los métodos

Para comparar los 3 métodos, hemos realizado diferentes pruebas, variando el número de **features** en cada una de ellas, evaluando los tiempos y el número medio de *good matches* en cada caso.

Método	nFeatures	Tiempo promedio (s)	Media good matches
ORB	100	0.195	14.9
ORB	1000	0.463	156.8
ORB	10000	7.369	983.1
ORB	20000	9.932	1169.4
SIFT	100	1.030	4.8
SIFT	1000	1.341	20.9
SIFT	10000	1.938	39
SIFT	20000	1.968	39
AKAZE	100	1.146	296.7
AKAZE	1000	1.186	296.7
AKAZE	10000	1.158	296.7
AKAZE	20000	1.120	296.7

Como se puede observar en la tabla, conforme aumenta el número de features, ORB muestra un aumento significativo en la cantidad de keypoints detectados y descriptores generados. Sin embargo, este aumento está asociado con una mayor propensión a detectar keypoints en áreas que pueden no ser distintivas, lo que resulta en más coincidencias erróneas. Por otro lado, SIFT y AKAZE tienden a una respuesta menos lineal en la cantidad de keypoints detectados con respecto al número de features, lo que sugiere una capacidad más adaptativa en la determinación de keypoints basada en la imagen de entrada.

Cabe destacar que, para realizar la media de *good matches* se han tenido en cuenta los emparejamientos de todas las imágenes con todas, de entre las cuales algunas entre sí no tienen ningún good match, por ello la media del método SIFT es más baja, ya que en este caso no detecta ninguno, indicando una mayor robustez en la detección de coincidencias significativas entre imágenes. Sin embargo, el método ORB detecta “falsos” good matches en estas imágenes.

En términos de tiempo empleado, ORB tiende a volverse más costoso computacionalmente a medida que se aumenta *nFeatures*, con un tiempo de procesamiento considerablemente mayor en comparación con SIFT y AKAZE. Estos últimos mantienen tiempos de procesamiento más estables y menos sensibles al aumento en *nFeatures*, lo que indica una eficiencia relativa en términos de rendimiento.

Por todo esto, aunque ORB y AKAZE pueden ser útiles en ciertos escenarios debido a su eficiencia computacional y capacidad para manejar grandes cantidades de características, **SIFT** destaca por su robustez y precisión en la detección y descripción de características. Gracias a su capacidad para encontrar coincidencias más precisas y significativas entre imágenes nos ha hecho decantarnos por utilizar este método para la reconstrucción del panorama.

4. Cálculo de la homografía

El cálculo de la homografía se calculó utilizando la función *findHomography* que ofrece *openCV* para asegurar que los problemas que pudiese haber no fuesen causados por la implementación propia.

Una vez implementado el resto del código, se implementó una versión propia llamada **calcularHomografiaRANSAC**.

Esta función calcula la homografía entre dos conjuntos de puntos utilizando *RANSAC* para manejar los outliers. Primero, se verifica que ambos conjuntos de puntos tengan la misma cantidad de elementos y que haya al menos cuatro puntos disponibles para calcular la homografía. Luego, se realiza un bucle sobre un número máximo de iteraciones especificadas.

En cada iteración, se seleccionan aleatoriamente cuatro puntos de cada conjunto (*pts1* y *pts2*). Se calcula la homografía inicial utilizando estos puntos mediante la función que se ha implementado llamada *calcularHomografia*, sin utilizar la proporcionada por OpenCV. Esta función realiza el cálculo de la homografía a partir de los puntos de origen y destino, aplicando las fórmulas explicadas en clase y con la ayuda de [esta explicación](#).

Posteriormente, se aplican estos puntos transformados a través de la homografía, y se calculan las distancias entre estos puntos transformados y los puntos reales.

Se cuentan los puntos que están dentro de un umbral de distancia especificado. Si el número de *inliers* (puntos dentro del umbral) supera al número de *inliers* encontrados anteriormente, se actualiza la mejor homografía (*mejor_H*) y se guarda el conjunto de *inliers* correspondientes.

Finalmente, se refina la homografía utilizando todos los *inliers* encontrados y se devuelve la mejor homografía encontrada junto con el conjunto de *inliers*.

Tras implementar la función del cálculo de la homografía se han comparado los resultados obtenidos con la implementación de *OpenCV* y se ha visto que la implementación de *OpenCV* genera mejores resultados. En promedio *OpenCV* detecta un mayor número de *inliers* que nuestra función (28% más de *inliers* de media que nuestra implementación).

5. Creación del panorama

El primer paso para la creación del panorama consiste en definir un orden en el que se van a unir nuestras imágenes. Este paso es muy importante, ya que no se obtiene un resultado correcto si se intenta unir la imagen 1 y 2 siguiendo el orden 2 a la izquierda, 1 a la derecha.

Para definir el orden de las imágenes se decidió utilizar una estructura de datos, en este caso un diccionario. En el diccionario se almacenan el número de imágenes que hay a la izquierda de cada imagen. Para añadir al diccionario se recorren todas las imágenes comparando unas con otras y si la imagen A está a la derecha de la imagen B, se suma el contador de la imagen B.

Al final del bucle se tiene un diccionario completo con las imágenes y cuantas tienen a la izquierda. Esto permite decidir que la imagen con más elementos a la izquierda será la primera por la derecha, e ir definiendo de esta forma el orden de las imágenes.

Para determinar si una imagen debe colocarse a la izquierda o a la derecha de otra en el panorama, se ha implementado la función *image_concatenation_direction*.

Esta función calcula las esquinas de la imagen de origen en relación con la imagen de destino después de aplicar la homografía. Si alguna coordenada x de las esquinas transformadas es negativa, la imagen de origen se superpone a la izquierda de la imagen de destino. De esta forma, se obtiene el orden correcto de unión de las imágenes en el panorama.

Para que la construcción final del panorama no salga muy distorsionada, hemos dividido las imágenes en tres grupos, izquierda, derecha y central. De esta forma se colocan las imágenes a la izquierda y a la derecha de la imagen central en su orden correspondiente, evitando la gran distorsión que sale en caso de ir uniéndose desde la imagen de más a la izquierda.

Hemos probado a construir diferentes panoramas, utilizando las imágenes de *buildingScene* proporcionadas, además de otros conjuntos de imágenes.

Establecer el orden de las imágenes es el proceso más costoso, ya que compara todas las imágenes entre sí. Para un panorama de 5 imágenes tarda 24 segundos. Unir cada imagen al panorama generado tarda de media 1,5 segundos, tardando el proceso completo 32 segundos*.

* Estos valores se pueden ver en la salida mostrada por el programa.

Los resultados obtenidos han sido los siguientes:

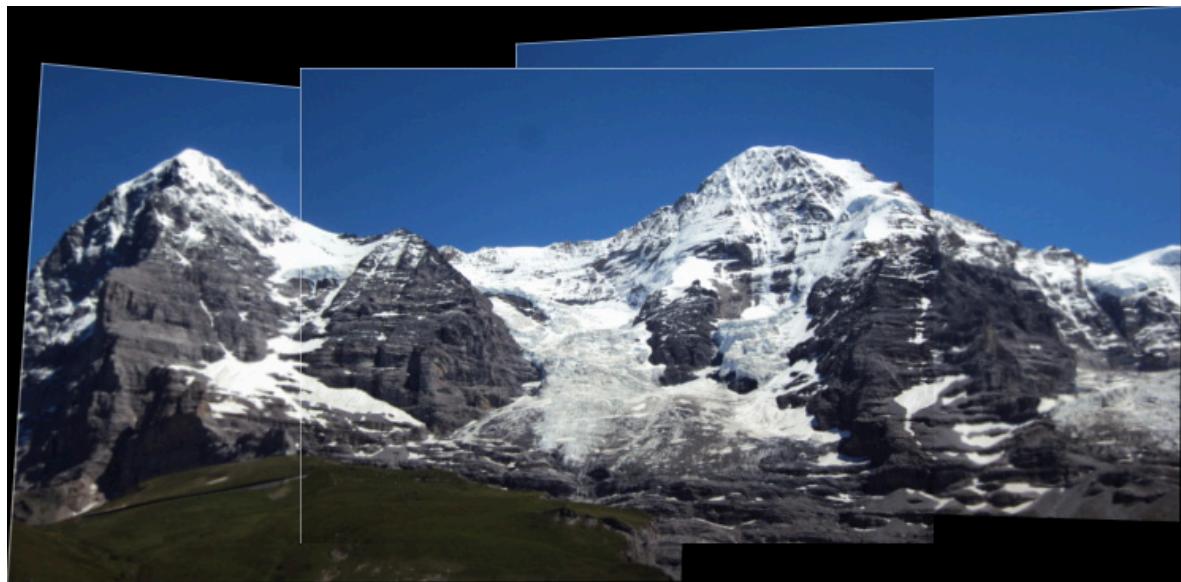


Imagen 1: Montañas nevadas del Pirineo Catalán.



Imagen 2: Building Scene proporcionada.

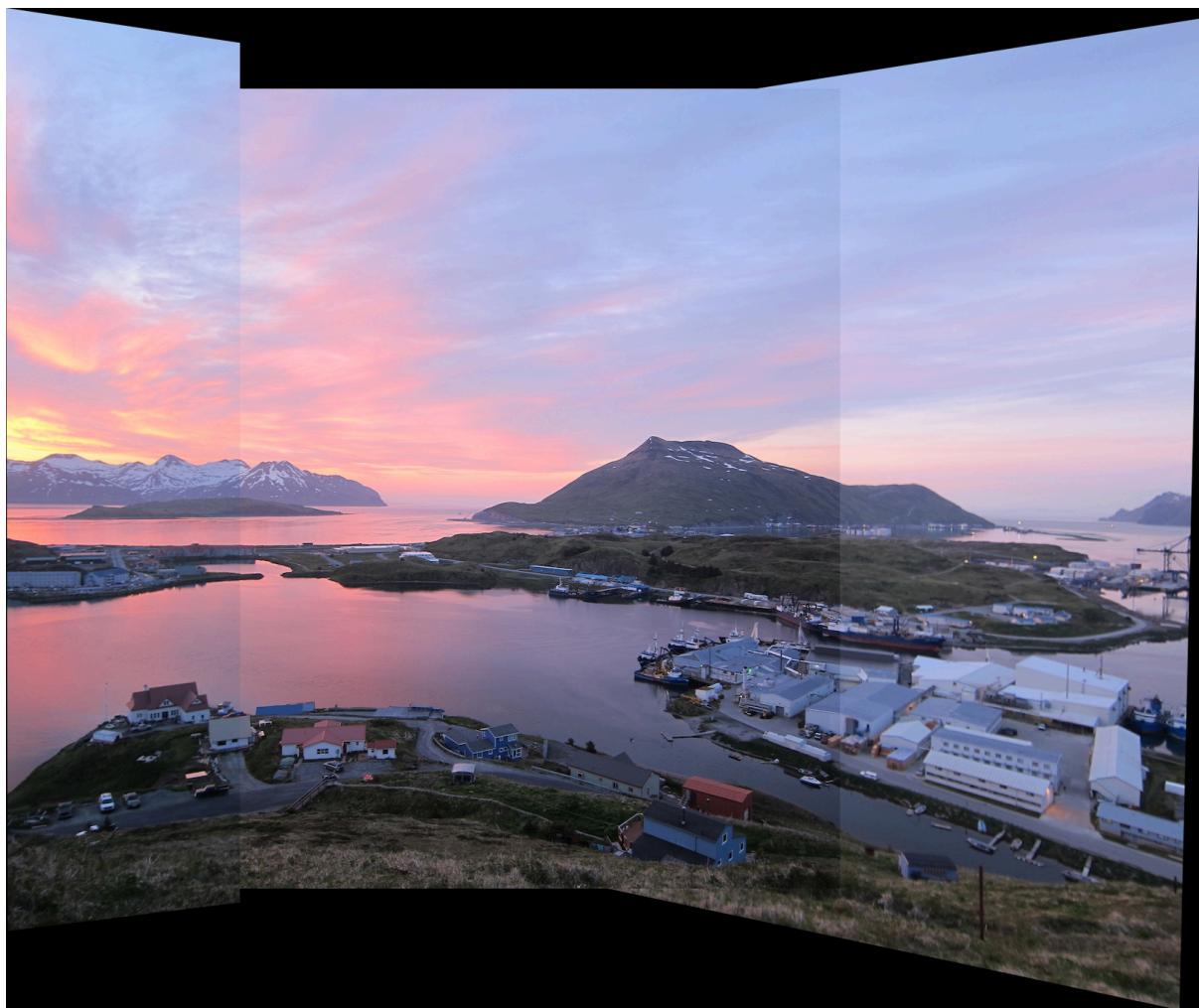


Imagen 3: Atardecer relajante en la costa catalana.

6. Bibliografía

- https://docs.opencv.org/4.x/dc/d0d/tutorial_py_features_harris.html
- https://docs.opencv.org/3.4/db/d70/tutorial_akaze_matching.html
- <https://kediarahul.medium.com/panorama-stitching-stitch-multiple-images-using-opencv-python-c-875a1d11236d>