

Trabajo 4: Panorama

Visión por Computador

Alejandro Paricio García, 761783
Fernando Peña Bes, 756012

Universidad de Zaragoza
2 de mayo de 2021

Índice

1. Introducción	1
2. Extracción de características	1
2.1. Comparación de métodos de extracción de características	1
2.2. Búsqueda de emparejamientos	5
2.3. Corrección de la distorsión de la cámara	5
3. Panoramas	7
3.1. Estimación robusta de la homografía	7
3.2. Creación del panorama	9
3.3. Fusión de las imágenes	9
3.4. Extracción de bordes de costura	12
3.5. Captura de panoramas	13
4. Galería de panoramas	14

1. Introducción

En este trabajo se ha desarrollado un programa capaz de unir imágenes para crear panoramas. Para ello extrae puntos de interés de las imágenes junto a sus descriptores utilizando diversos métodos, los empareja y alinea las imágenes calculando la homografía que las relacionan de forma robusta con RANSAC.

Adicionalmente se ha añadido la posibilidad de calibrar la cámara que ha tomado las imágenes para corregir distorsiones ópticas, diferentes opciones de fusión de imágenes y la captura y generación de panoramas en directo.

2. Extracción de características

El primer paso para alinear un conjunto de imágenes es extraer las características o *features* de cada una, de forma que el computador pueda reconocer los objetos de interés de la escena que son relevantes con respecto a su entorno, por ejemplo bordes o texturas. Estos puntos deberían ser invariantes a perturbaciones y transformaciones que pueden sufrir las imágenes capturadas, como cambios en la iluminación, escala, rotación o perspectiva.

Para identificar a los puntos de interés, se calculan sus descriptores en base a la información de los píxeles vecinos, con el objetivo de poder encontrar los puntos comunes entre imágenes de una misma escena.

2.1. Comparación de métodos de extracción de características

Siguiendo el guión de la práctica se han probado los siguientes extractores de características: Harris, SIFT, SURF, ORB y AKAZE.

Se han usado dos escenas de prueba: la primera es un cuadro, al tomar las fotos la cámara se ha intentado mover dentro del mismo plano para realizar un barrido; la segunda es una escena 3D de exterior, las imágenes se tomaron rotando la cámara sin que el centro óptico se trasladase.

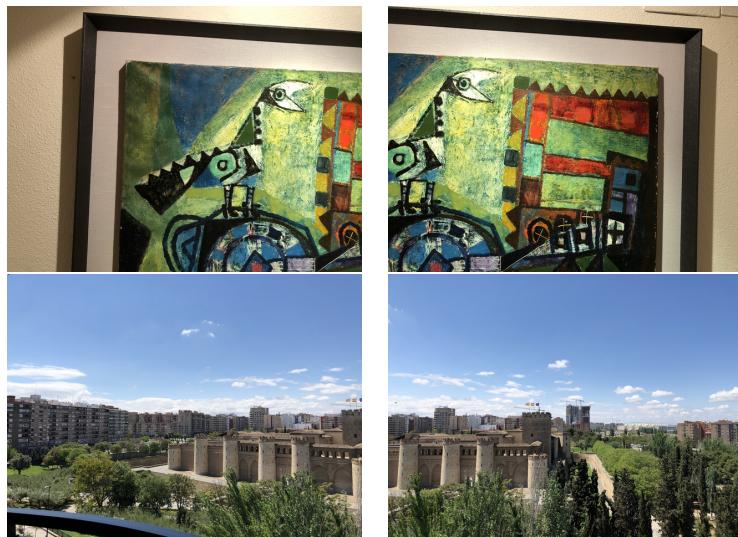


Figura 1: Imágenes de prueba para la extracción de características

2.1.1. Harris

Hay un gran número de algoritmos de detección de características que se basan en la búsqueda de esquinas. El detector de Harris es uno de ellos. Se basa en la idea de localizar patrones que son muy diferentes de los píxeles de su entorno. Para implementarlo se ha utilizado la función *cornerHarris()* de OpenCV. Esta función toma como entrada una imagen y devuelve otra imagen del mismo tamaño pero cada píxel tienen un valor más alto cuanto más interesante se considera. Como keypoints se eligen los píxeles cuyo valor supera un cierto umbral (p.e., 0.01 por el valor del píxel máximo). Una vez hecho esto se han calculado los descriptores de cada punto usando SIFT.

Este operador es invariante a translación, rotación e iluminación, pero no a escala.

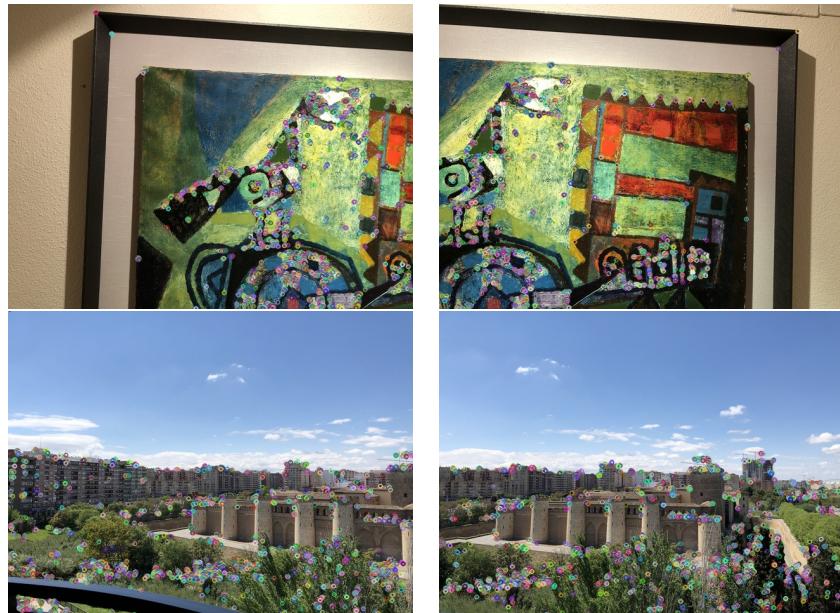


Figura 2: Extracción de características con Harris

2.1.2. SIFT

SIFT (Scale-invariant feature transform) además de ser invariante a translación, rotación e iluminación lo es también a escala. Se ha utilizado la clase *SIFT* de OpenCV.

2.1.3. SURF

SURF (Speeded up robust features), tiene características similares a SIFT. Es también invariante a escala, pero approxima el cálculo de derivadas Gaussianas de segundo orden media filtros de caja. Es más eficiente que SIFT pero menos robusto. Este algoritmo está patentado y para usarlo en OpenCV (clase *SURF*) ha hecho falta importar la librería *xfeatures/nonfree*.

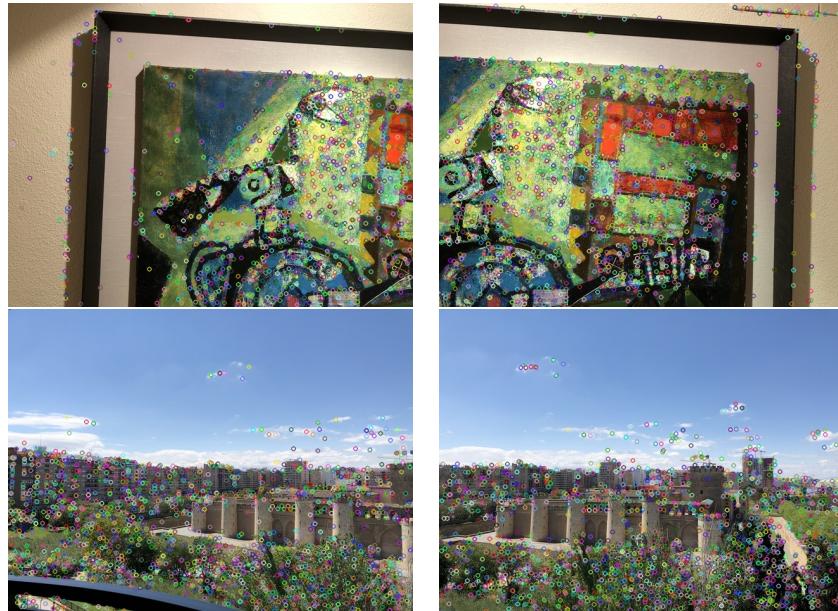


Figura 3: Extracción de características con SIFT

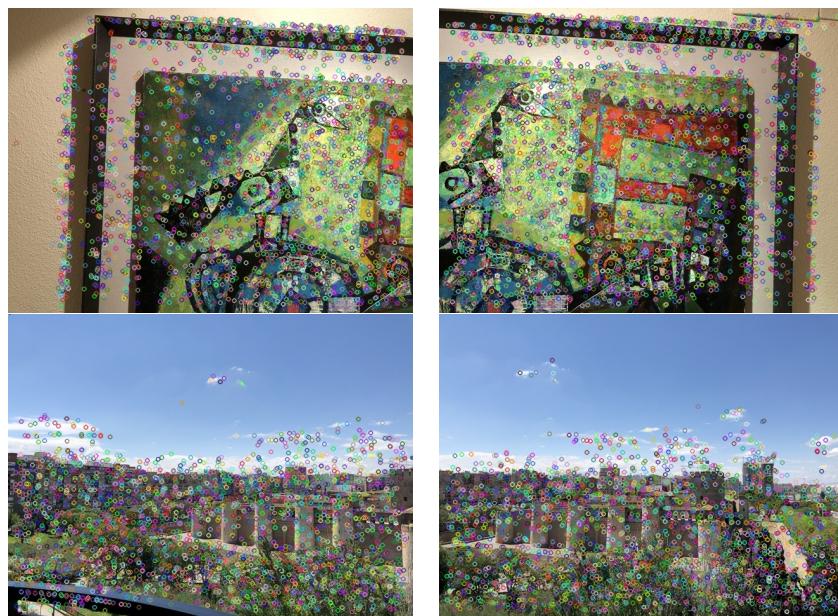


Figura 4: Extracción de características con SURF

2.1.4. ORB

ORB (Oriented FAST and rotated BIEF) es una alternativa más eficiente a SIFT y SURF. Es una fusión del detector de keypoints FAST y el generador de descriptores BRIEF con varias modificaciones para mejorar su rendimiento. Es también invariante a escala. Se ha usado la clase *ORB* de OpenCV.

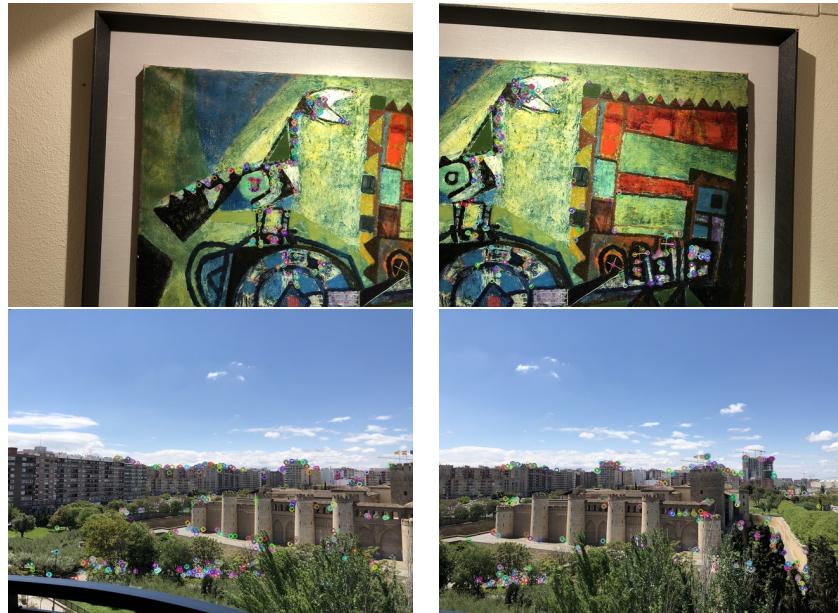


Figura 5: Extracción de características con ORB

2.1.5. AKAZE

AKAZE es un método más reciente que los anteriores y que intenta mejorar el rendimiento, tanto a nivel de precisión en el reconocimiento como en velocidad, de SIFT y SURF al operar en un espacio de escala no lineal. Intento preservar los contornos naturales de los objetos al moverse por el espacio de escala. Además, su implementación es libre, por lo que es bastante popular.

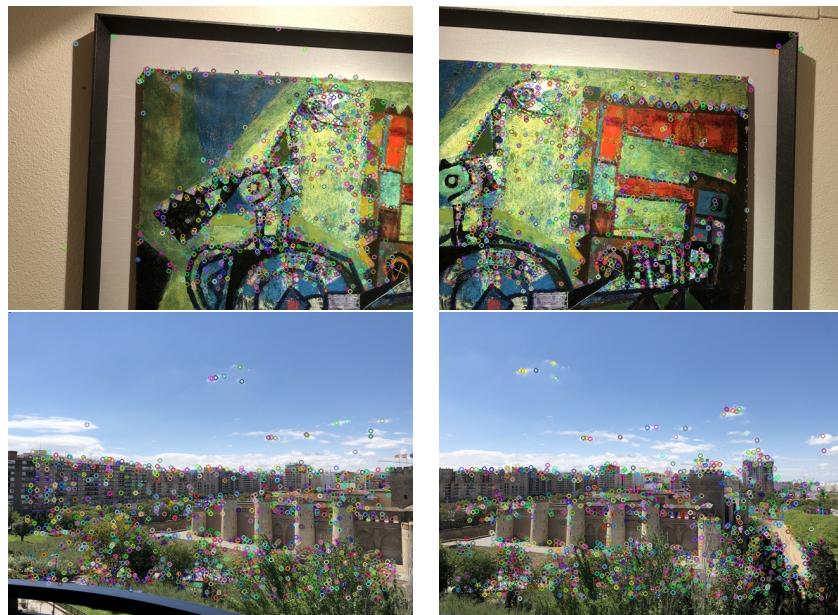


Figura 6: Extracción de características con AKAZE

2.2. Búsqueda de emparejamientos

Partiendo de los keypoints detectados en un par de imágenes, es necesario llevar a cabo el emparejamiento de los descriptores de los mismos para poder encontrar correspondencias entre los puntos de ambas imágenes. La primera aproximación probada fue un emparejamiento por fuerza bruta, similar para todos los puntos de interés excepto en un punto clave, que es el modo de calcular la distancia entre descriptores. En el caso de ORB y AKAZE, descriptores binarios, ha de emplearse la distancia Hamming para ello, mientras que para SIFT, SURF y Harris, se emplea la distancia L2. No obstante, la primera opción fue sustituida por el uso de los métodos que emplean FLANN de OpenCV, con el objetivo de realizar una búsqueda aproximada del vecino más próximo que sea más rápida. OpenCV ofrece una implementación directa que no ha supuesto mayores complicaciones en su uso.

A continuación, y tras cualquiera de las dos opciones anteriores se aplica una fase de refinamiento de los emparejamientos obtenidos, descartando así aquellos de menor calidad. Para ello se ha implementado el test del ratio al segundo vecino. En el caso general se ha empleado un valor de 0.8, comúnmente utilizado, pero para SURF y SIFT, especialmente para el caso de las imágenes empleadas para la creación del panorama, se redujo un poco este valor (hasta 0.7), descartándose así un mayor número de emparejamientos y obteniéndose un mejor panorama.

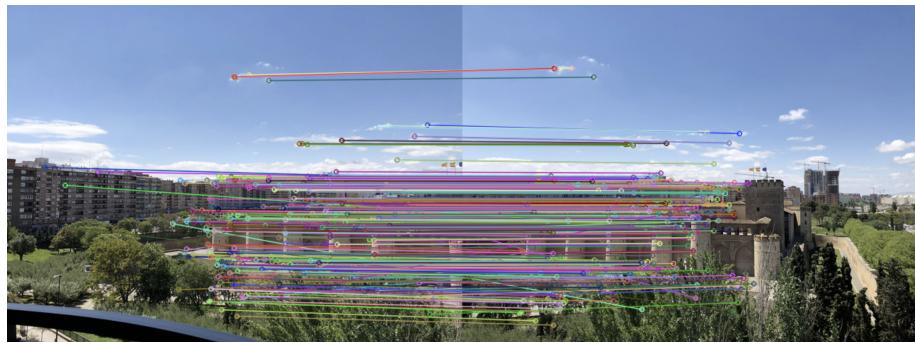


Figura 7: Emparejamientos encontrados con AKAZE y 0.8 como ratio al segundo vecino

2.3. Corrección de la distorsión de la cámara

Las imágenes capturadas por una cámara tienen una distorsión que puede afectar a la hora de obtener emparejamientos correctos, especialmente si después se intenta calcular un modelo, por ejemplo una homografía, que requiere de una gran precisión en la proyección de los puntos de una imagen sobre la otra. Por ello, se ha tratado de corregir esa distorsión, restaurando una apariencia de las imágenes más similar a la forma que tienen los objetos en el mundo real. Para llevar a cabo este proceso se suelen utilizar múltiples imágenes de un tablero de ajedrez. En este caso se han cogido las imágenes del tablero de ajedrez proporcionadas por la instalación de OpenCV (`samples/data/left{01..14}.jpg`). El proceso completo es el siguiente:

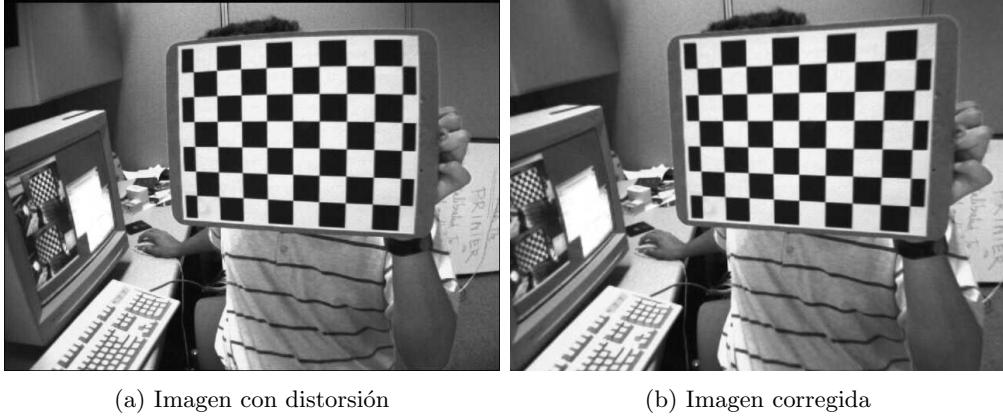
- Definir las coordenadas del tablero de ajedrez en el mundo real. Como las casillas tienen todas el mismo tamaño, sus coordenadas se han definido como $(0, 0, 0)$, $(0, 1, 0)$, $(1, 0, 0)$... Como el tablero es plano, el eje Z puede dejarse siempre a cero, mientras que el X y el Y se han ajustado al tamaño del tablero.
- Acto seguido, han de encontrarse las coordenadas 2D de las intersecciones de 4 casillas en el tablero de la imagen, para lo que se ha recurrido a la función de

OpenCV `findChessboardCorners()`. La anterior devuelve directamente las coordenadas buscadas, si las encuentra.

- Si se han podido localizar las coordenadas 2D, se pasa a una etapa de refinamiento subpíxel para buscar una mayor precisión. La función `cornerSubPix()` lleva a cabo este refinamiento, aumentando la precisión en la localización de las mismas, que va a ser fundamental en la posterior estimación de los parámetros de la cámara. Los resultados obtenidos tras este punto se corresponden a localizaciones como las marcadas en la siguiente imagen:



- En este punto se han definido las localizaciones de los puntos de intersección entre 4 casillas del tablero en el mundo real, siendo estas las mismas para cada punto a lo largo de todas las imágenes. También se han obtenido las coordenadas 2D de los puntos en todas las imágenes del tablero. La función `calibrateCamera()` nos permite obtener la matriz intrínseca de la cámara (considerados sólo distancia focal y centro óptico en OpenCV), los vectores de translación y rotación, y los coeficientes de distorsión de la lente.
- Una vez obtenidos los parámetros e la cámara, el único paso restante es corregir la imagen para poder proceder con los emparejamientos. La función `undistort()` emplea la matriz intrínseca de la cámara y los parámetros de distorsión de la lente para corregir la imagen, obteniéndose resultados como los siguientes, en los que claramente se aprecia cómo las líneas que deberían ser rectas en la imagen izquierda, pero no lo son, pasan a sí serlo en la derecha.



El proceso anterior de calibración de la cámara realizado como paso previo a la extracción de puntos de interés puede suponer la diferencia entre encontrar un conjunto de emparejamientos que permitan obtener una homografía tras un algoritmo de estimación robusta como RANSAC y no hacerlo. Esto se debe a que la distorsión de la imagen pueden provocar tanto una estimación de homografía incorrecta, como que los puntos proyectados mediante el modelo calculado no estén situados a una distancia inferior al ϵ necesario para declararlo como inlier, y no siendo este utilizado en otros procesos posteriores como la creación de panoramas.

Para la consecución de este apartado se han utilizado como guía los siguientes sitios web: <https://learnopencv.com/camera-calibration-using-opencv/> y https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html.

3. Panoramas

En esta sección se comenta como, partiendo de los emparejamientos conseguidos en la sección anterior, se ha creado un programa de generación de panoramas.

3.1. Estimación robusta de la homografía

El primer paso para realizar la fusión de al menos dos imágenes en una que supone el panorama, es estimar la transformación entre ellas. Expresado con mayor detalle, hay que intentar obtener la homografía que representa la transformación de los puntos de interés comunes entre ambas imágenes. Aplicar dicha transformación a los puntos de la primera imagen que aparezcan en la segunda debería dar como resultado las posiciones de los mismos en la segunda imagen. Para ello, debe realizarse una extracción de puntos de interés en ambas imágenes, obtener emparejamientos entre ellos y, finalmente, aplicar un algoritmo de estimación robusta, en este caso RANSAC adaptativo. El algoritmo implementado sigue parcialmente el siguiente esquema de pseudocódigo, siendo necesarios algunos cambios:

- Considerar un modelo que requiere k puntos y un conjunto de puntos P con $|P| > k$.
- For $i = 1 \dots t$ intentos:
 - Elegir un subconjunto aleatorio S_i de P , con k puntos y calcular el modelo M_i .

- Buscar el conjunto de consenso S_i^* : puntos de P que son compatibles con el modelo M_i (utilizando una cierta tolerancia ε).
- Si $|S_i^*| >$ threshold de consenso, utilizar S_i^* para recalcular un modelo de consenso M_i^* utilizando mínimos cuadrados.
- Fin For
- Devolver el mejor modelo M_i^* encontrado. Si en ningún intento se ha logrado sobrepasar el threshold de consenso, devolver fallo.

En cada iteración se eligen cuatro emparejamientos aleatorios, se obtiene con ellos y la función *findHomography()* de OpenCV (sin usar el flag de RANSAC, indicándole que emplee todos los puntos. Se usa por simplicidad y así evitar la programación de la resolución del sistema de ecuaciones) una homografía. Acto seguido se evalúa cuántos, de entre todos los emparejamientos, la proyección del punto de la primera imagen mediante la homografía queda a menos de ε píxeles de su emparejamiento en la segunda, para así declararlo con inlier en esta iteración. Si el número de inliers es mayor que el del mejor modelo obtenido hasta el momento, se conserva el nuevo modelo. El número de intentos se calcula de acuerdo a la siguiente fórmula, en la que p es el ratio de inliers y P es la probabilidad de éxito:

$$t = \frac{\log(1 - P)}{\log(1 - p^4)}$$

Al comienzo del algoritmo se establece p con un valor pequeño, 0.5 por ejemplo, lo que da un número de intentos totales bastante alto. Al final de cada iteración, t se recalcula, actualizando previamente p , que se corresponde con el ratio de inliers del modelo calculado. Si el número de intentos calculado con ese ratio es menor que el anterior t , se reduce el número de intentos, reduciendo la necesidad de seguir iterando si se obtiene un buen modelo rápidamente, de ahí el nombre de RANSAC adaptativo. Filtrados los inliers con éste método podemos obtener resultados sin emparejamientos espurios. Gracias a ello podemos, únicamente con todos los inliers del mejor modelo calculado, recalcular la homografía, permitiendo obtener el solapamiento de imágenes que será imprescindible para los panoramas. La siguiente imagen muestra el dibujo de la proyección de la primera imagen sobre el espacio de la segunda.



Figura 9: A la derecha se muestra el cuadrilátero correspondiente a transformar la primera imagen para que encaje con la segunda.

3.2. Creación del panorama

Una vez que se ha encontrado la homografía que relaciona dos imágenes hace falta aplicarla y unirlas de forma correcta para que queden alineadas.

Para calcular las dimensiones de la imagen resultante, se aplica la homografía a las posiciones de las esquinas de la imagen que se va a trasformar (con la función *perspectiveTransform()*) y juntándolas con las esquinas de la segunda imagen se calculan las dimensiones del rectángulo que contendrá ambas imágenes.

A continuación, se aplica la homografía a la primera imagen y las dos imágenes se desplazan mediante una matriz de translación para evitar que haya píxeles con posiciones negativas. Estas transformaciones se realizan con *warpPerspective()*.

En la Figura 10 se muestra el resultado de transformar dos imágenes para que queden alineadas. Una vez hecho está todo preparado para fusionarlas en una sola.



Figura 10: Aplicación de la homografía y las translaciones necesarias para alinear las dos imágenes

Para crear panoramas con más de dos imágenes, hemos decidido repetir este proceso hasta unir todas las imágenes, tratando los panoramas parciales como imágenes. Como se están realizando proyecciones planas, en vez de esféricas o cilíndricas, hay que tener cuidado al elegir la primera imagen del panorama ya que a esta no se le aplica ninguna transformación de perspectiva y las demás se intentan alinear con ella. Lo mejor, generalmente, es comenzar con la imagen que quedará en el centro del panorama, para evitar que las de los extremos se deformen demasiado.

3.3. Fusión de las imágenes

El paso final para construir el panorama es fusionar las imágenes alineadas. El problema que da lugar a diferentes estrategias de fusión aparece al tener elegir cómo mezclar los píxeles de las zonas en las que hay información de dos imágenes, en las que se encuentran solapadas.

La estrategia más sencilla es no realizar ninguna fusión, colocar una imagen encima de otra eligiendo siempre los píxeles de la misma imagen en la zona de solapamiento. El resultado se muestra en la Figura 11. Como es de esperar, los bordes en la zona de la unión son muy marcados sobre todo en el cielo.

Una pequeña mejora es utilizar un blending lineal. En las zonas en las que sólo hay información de una imagen, se copian esos píxeles directamente, y en las que hay información de las dos se hace una media ponderada de los valores de los píxeles de cada una (utilizando por ejemplo la función *addWeighted()*). En el ejemplo de la Figura 12 se pondrá 50 %-50 %. De esta forma los bordes quedan más disimulados, pero aparecen efectos de ghosting en zonas en las que ambas imágenes no están perfectamente alineadas.



Figura 11: Panorama sin blending



Figura 12: Panorama con blending lineal

OpenCV dispone de un módulo llamado *Stitching* con una pipeline muy completa para unir imágenes desde el principio hasta el final. Las primeras fases las hemos implementando durante la práctica, pero las últimas pueden ser de utilidad en esta parte. Al final de la pipeline hay varias opciones para fusionar imágenes de forma suave evitando efectos de ghosting. Hemos utilizado las clases *FeatherBlender* y *MultibandBlender*. El blender de tipo feather coloca una imagen encima de la otra pero realiza un suavizado en los punto de unión entre ambas, se muestra en la Figura 13. El blender multibanda aplica un algoritmo de fusión multibanda basado en el análisis de las frecuencias de las componentes de la imágenes. Los resultados con esta técnica se muestran la Figura 14.

El blending multibanda suele ser el que mejor resultados ofrece, pero no funciona bien cuando hay bandas negras alrededor los bordes de unión. Una solución podría ser conseguir que el fondo al aplicar las transformaciones con *warpPerspective()* no sea negro, si no que se rellene con información de la imagen transformada, por ejemplo con su reflejo. Esto funcionaba bien para la primeras imágenes, pero conforme el panorama parcial empezaba a tener bandas negras, su reflejo también tenía bandas negras, por lo que se seguía sin conseguir buenos resultados



Figura 13: Panorama con blending de tipo feather

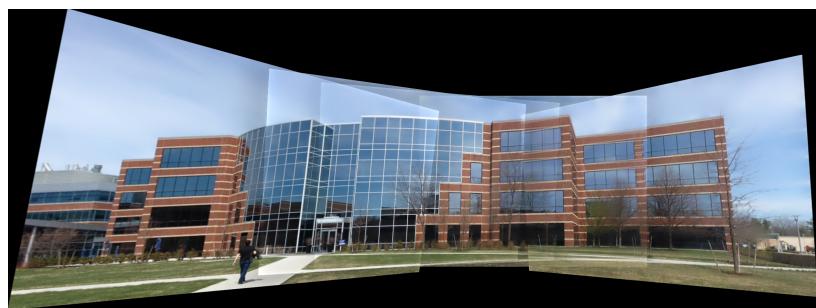


Figura 14: Panorama con blending multibanda

3.4. Extracción de bordes de costura

Una forma de mejorar las uniones es extraer los bordes de costura (*seams*). Los algoritmos que los encuentran buscan que la unión se realice por donde más variabilidad haya en la imagen para que se note menos. El módulo de stitching de OpenCV incluye implementaciones de estos algoritmos, como *VoronoiSeamFinder* y *DpSeamFinder*. En las Figuras 15 y 16 se muestra como al refinar las uniones de esta manera se consigue arreglar los problemas con el blender multibanda.

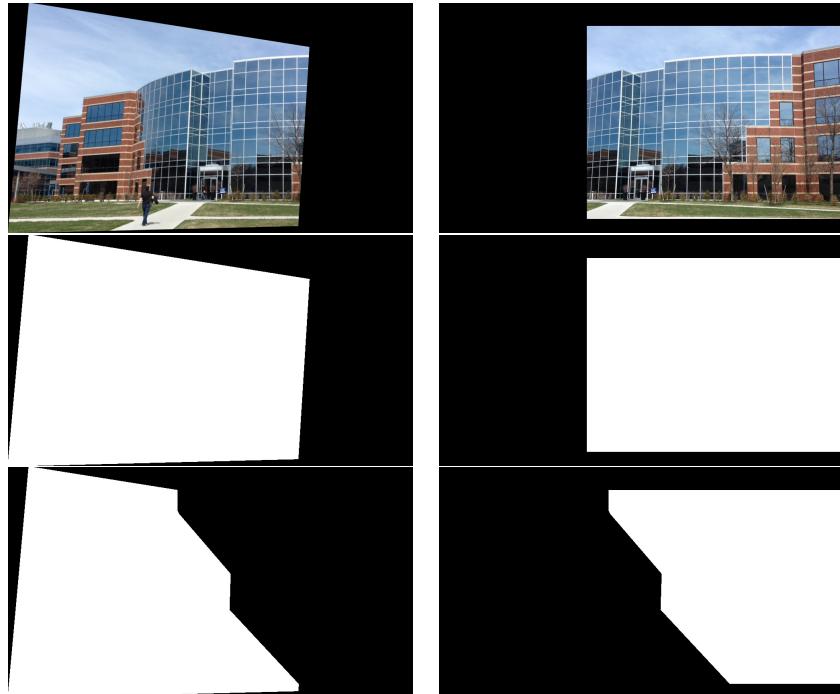


Figura 15: Búsqueda de bordes de costura utilizando el estimador basado en diagramas de Voronoi. En la primera fila se muestran las imágenes transformadas, en la segunda las máscaras originales y en la tercera las máscaras tras aplicar la búsqueda de *seams*



Figura 16: Mezcla de las dos imágenes con blending multibanda usando las máscaras con bordes de costura

3.5. Captura de panoramas

El programa permite realizar panoramas de tres maneras diferentes: leyendo imágenes de disco, en directo, tomando imágenes de una cámara conectada al ordenador cada vez que se pulse una tecla y también en directo pero tomando imágenes cada cierto intervalo configurable por el usuario.

A continuación se reproduce la ayuda que acompaña al programa:

```
Uso: ./panorama [--mode MODO] [opciones...]

Modos (por defecto images):
  images [imagenes...]
    Componer todas las imágenes que se pasen
  live_key
    Panorama en directo, capturando una imagen cada vez que se pulse una tecla
  live_auto [time]
    Panorama en directo, capturando una imagen cada time segundos.
    Por defecto 2 segundos
Opciones:
  --features {harris, orb, sift, surf, akaze}
    Tipo de features al hacer matching. Por defecto akaze
  --blend {no, linear, multiband, feather}
    Método para unir las imágenes. Por defecto no
  --seam {no, voronoi, dp_color}
    Tipo de estimación de uniones entre imágenes. Por defecto no
  --use_flann
    Usar FLANN para comparar los descriptores. Por defecto no se usa
  --nn_ratio
    Ratio al segundo vecino. Por defecto 0.8

Ejemplos:
./panorama --mode images BuildingScene/building{4,3,5,2,1}.JPG --blend multiband
./panorama --mode live_key --blend multiband --seam voronoi
./panorama --mode live_auto 5 --features sift
```

Para mostrar la captura de panoramas en directo se ha grabado un vídeo al que se puede acceder en el siguiente enlace:

https://drive.google.com/file/d/1twmZ6LGI Aryx0eSXDTwtd0pG_zjnRsBo/view?usp=sharing

4. Galería de panoramas

En esta última sección se muestran tres ejemplos de panoramas completos. El primero está formado por 5 imágenes y demás por 6. Para todos ellos se ha utilizado AKAZE para extraer las características, estimación robusta con RANSAC para obtener las homografías, blending multibanda aplicando la búsqueda de bordes de costura basada en diagramas de Voronoi.

