

# Fundamentos de Análisis de Imágenes

Gonzalo Lope Carrasco - d21c004

Javier Pérez Vargas - d21c017

Mario Ruiz Vaquett - d21c023

December 31, 2023

# Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Homografía para Detección de Objetos</b>	<b>3</b>
2.1. Detección de puntos clave . . . . .	3
2.2. Emparejamiento de los keypoints . . . . .	4
2.3. RANSAC y Matriz de Homografía . . . . .	5
2.4. Warping . . . . .	5
2.5. Resultado con ORB . . . . .	6
<b>3. Creacion de Panorama con Cinco Imágenes</b>	<b>8</b>
3.1. Detección de puntos clave . . . . .	8
3.2. Emparejamiento de los keypoints . . . . .	9
3.3. RANSAC y Matriz de Homografía . . . . .	9
3.4. Warping . . . . .	9
3.5. Resultado con ORB . . . . .	11
<b>4. Ampliación a Siete Imágenes evitando Distorsiones</b>	<b>12</b>
<b>5. Conclusión</b>	<b>14</b>

## 1. Introducción

El objetivo de esta práctica es la aplicación de conceptos clave del procesamiento de imágenes usando la biblioteca OpenCV. Dividido en tres partes, el proyecto aborda la homografía para detección de objetos, la creación de panoramas a partir de cinco imágenes y la expansión a siete imágenes evitando distorsiones en los bordes.

## 2. Homografía para Detección de Objetos

En la primera parte del proyecto, el objetivo consiste en determinar la homografía que describe la transformación geométrica entre un cartel publicitario y la imagen de una marquesina con ese cartel desde una perspectiva concreta. Esta homografía permitirá aplicar la misma transformación a otro cartel con el propósito de integrarlo de manera realista en la escena representada por la foto de la marquesina.

Las imágenes con las que vamos a trabajar son las siguientes:



Imagen 1



Imagen 2



Imagen 3

### 2.1. Detección de puntos clave

En este apartado, hemos aplicado el detector SIFT a nuestras imágenes para detectar los puntos clave, tanto del cartel como de la marquesina. Inicialmente hemos decidido usar SIFT por su capacidad y complejidad, sabiendo que el resultado iba a ser mejor que con otros detectores. No obstante, al final veremos cómo funciona el proceso utilizando ORB.

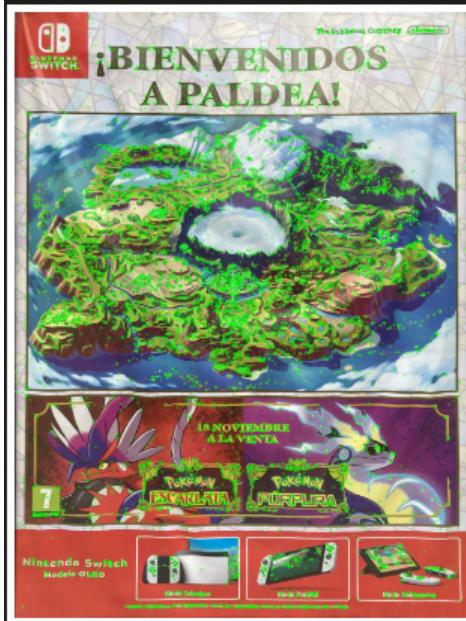


Figura 1: Keypoints del Cartel



Figura 2: Keypoints de la Marquesina

Figura 3: Keypoints

Podemos observar que muchos de los keypoints detectados en la marquesina no serán útiles, pues no encontrarán un match correcto en el cartel. No obstante, con los que hemos detectado serán suficientes para el proyecto. Utilizamos el descriptor SIFT para cada keypoint detectado y pasamos al matching.

## 2.2. Emparejamiento de los keypoints

Una vez hecho esto, emparejamos los descriptores entre las dos imágenes mediante el uso de la función **BFMatcher**. Esta función realiza un matching mediante fuerza bruta, es decir, toma cada descriptor del primer set y lo empareja con todos los descriptores del segundo set, utilizando como criterio de mejor emparejamiento la distancia. En nuestro caso, utilizamos la norma **L2**, es decir, la distancia euclídea. El resultado de la aplicación de esta función sería:

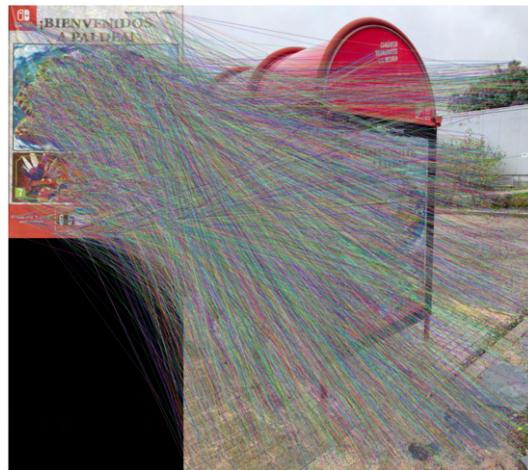


Figura 4: Matching

Como vemos, aparecen una abundante cantidad de matches. Por tanto, vamos a tener que limpiar y quedarnos solo con los mejores.

## 2.3. RANSAC y Matriz de Homografía

En este punto tenemos una gran cantidad de correspondencias, correctas e incorrectas. Para eliminar las peores correspondencias, utilizamos el algoritmo RANSAC, que nos permitirá estimar la matriz de homografía que contenga el mayor número de inliers, es decir, de correspondencias correctas.

Para ello, usaremos la función **findHomography** de OpenCV, que nos dará la matriz de homografía y los inliers. Con esto, podremos dibujar los matches correctos.

La matriz de homografía es la siguiente:

$$H = \begin{bmatrix} 0,9144 & 0,0980 & 1359,88 \\ 0,2752 & 1,0832 & 1095,08 \\ 0,0001 & 0 & 1,0 \end{bmatrix}$$

Y los inliers:

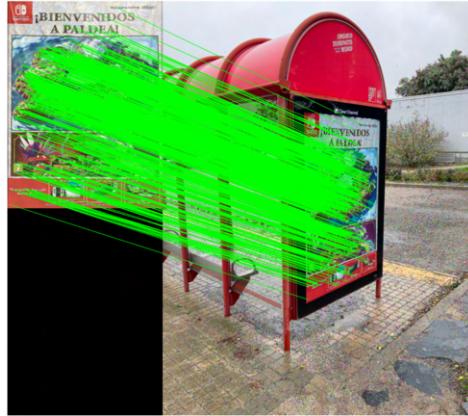


Figura 5: Matches correctos

Como observamos, tenemos una menor cantidad de matches en comparación con la imagen anterior, pero esta vez todos son correctos.

## 2.4. Warping

Ahora que tenemos la matriz de homografía, la aplicamos a la Imagen 1 (el nuevo cartel) para que encaje en la marquesina adecuadamente. Para ello, utilizamos la función **warpPerspective** con la matriz de homografía H que hará que la imagen resulte de la siguiente manera:



Figura 6: Imagen Ayuntamiento con Perspectiva

Teniendo la imagen con la forma correcta, debemos superponerla en la imagen 3 (la marquesina). Para ello, creamos una máscara para que solo se superpongan aquellos píxeles que no sean negros. Finalmente, el resultado es el siguiente:



Figura 7: Imagen Final

## 2.5. Resultado con ORB

Una vez visto el resultado final, vamos a ver como quedaría el resultado en el caso de que utilizáramos ORB. en nuestro caso haremos que calcule 10000 features.

En este caso, el tiempo de ejecución es menor, en el caso de SIFT fue de 4.4 segundos mientras que con ORB es de 0.8 segundos, esto se debe a que ORB nos devuelve una menor cantidad de puntos:

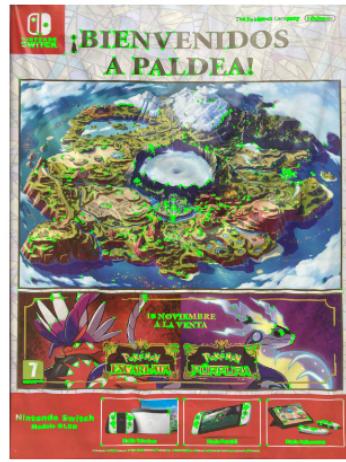


Figura 8: Keypoints del Cartel con ORB



Figura 9: Keypoints de la Marquesina con ORB

Figura 10: Keypoints

Como vemos en las imágenes, tenemos muchos menos keypoints que cuando utilizamos SIFT.

Como tenemos menos puntos y el tiempo de ejecución es menor cuando se utiliza ORB, ahora cuando vamos a realizar los matches vemos que la diferencia es todavía mayor. En el caso de SIFT, en realizar los matches se tardaban 40 segundos mientras que con ORB se tardan 2 segundos. Para el uso de ORB, hemos tenido que modificar la distancia que utilizamos, pasamos de utilizar la distancia L2 a la Hamming. Este es el resultado:



Figura 11: Matches con ORB

Una vez hecho esto, creamos la matriz de Homografía utilizando RANSAC al igual que con SIFT y nos quedamos con los matches correctos:

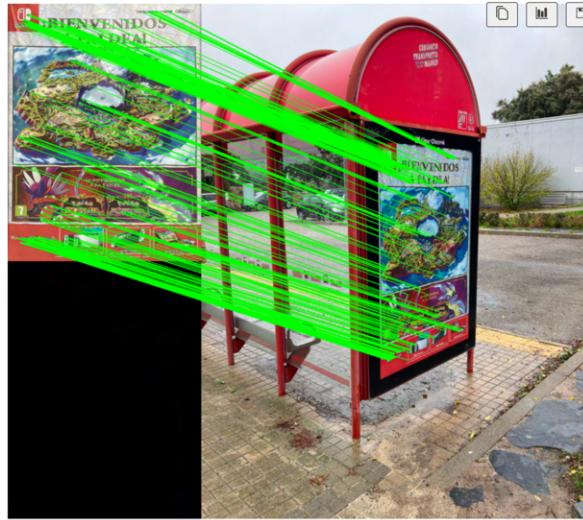


Figura 12: Matches Correctos con ORB

Por último, observamos el resultado de la homografía:



Figura 13: Resultado Final Con ORB

Como observamos, los resultados usando SIFT y usando ORB son muy similares.

### 3. Creacion de Panorama con Cinco Imágenes

En la segunda parte, exploraremos la unión de cinco imágenes horizontalmente solapadas para crear un panorama único. Abordaremos la recuperación de parámetros de transformación entre cada par de imágenes, la rectificación de las imágenes y la creación de un mosaico resultante. Dejaremos una imagen sin deformar y proyectaremos las otras sobre ella para lograr una alineación adecuada.

Las imágenes son las siguientes:



Figura 14: Imágenes para hacer la panorámica

#### 3.1. Detección de puntos clave

De la misma forma que en el ejercicio anterior, hemos decidido usar SIFT para detectar los puntos clave de las 5 imágenes.



Figura 15: Keypoints de las imágenes

### 3.2. Emparejamiento de los keypoints

Una vez descritos los keypoints, podemos pasar a calcular los matches. En las siguientes imágenes vemos dibujados los emparejamientos encontrados.

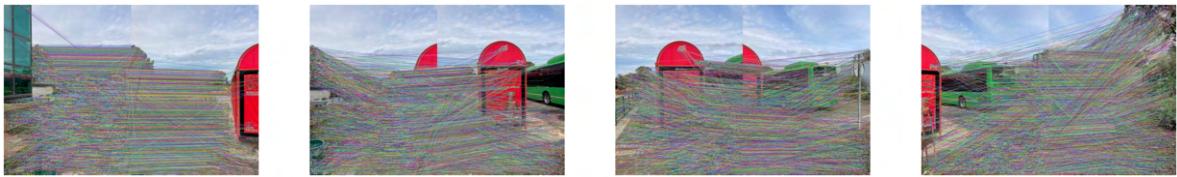


Figura 16: Matches detectados en los pares de imágenes

### 3.3. RANSAC y Matriz de Homografía

Como hay muchos emparejamientos incorrectos, aplicamos de nuevo el algoritmo RANSAC para quedarnos solo con los inliers. Los dibujamos en las siguientes imágenes:



Figura 17: Matches correctos utilizando RANSAC

### 3.4. Warping

Para crear la panorámica, hemos decidido crear dos funciones. La primera de ellas es **WarpCompleteImage**:

```
def WarpCompleteImage(img1, img2, img2_keypoints, H):
    # Calculamos las dimensiones de las dos imágenes: Altura y ancho
    h1,w1 = img1.shape[0], img1.shape[1]
    h2,w2 = img2.shape[0], img2.shape[1]

    # Calculamos los puntos de las esquinas de ambas imágenes
    pts1 = np.float32([[0,0],[0,h1],[w1,h1],[w1,0]]).reshape(-1,1,2)
    pts2 = np.float32([[0,0],[0,h2],[w2,h2],[w2,0]]).reshape(-1,1,2)

    # Transformamos los puntos de la imagen 2 para obtener las nuevas coordenadas
    pts2_transformed = cv2.perspectiveTransform(pts2, H)

    # Unimos todos los puntos para luego hallar el maximo y el minimo
    pts = np.concatenate((pts1, pts2_transformed), axis=0)
    # Calculamos el minimo y el maximo de las coordenadas de x e y
    [xmin,ymin] = np.int32( pts.min(axis= 0).ravel() - 0.5)
```

```

[xmax,ymax] = np.int32( pts.max(axis=0).ravel() +0.5)

# Obtenemos el tx y el ty para la matriz de traslación
tx = -xmin
ty = -ymin

# Creamos la matriz de traslación para que se vea toda la imagen, usando
# el mínimo de las coordenadas de x e y transformadas
Ht = np.array([[1,0,tx],[0,1,ty],[0,0,1]])

# Aplicamos la transformación multiplicando la matriz de traslación por la matriz de homografía
result = cv2.warpPerspective(img2, Ht.dot(H), (xmax-xmin, ymax-ymin))

# Sustituimos los píxeles de la primera imagen en la segunda transformada
mask = np.all(img1 == 0, axis=2)
img1 = np.where(img1 == 0, img1+1, img1)
img1[mask] = [0,0,0]
result[ty:h1+ty, tx:w1+tx] = np.where(img1 != 0, img1, result[ty:h1+ty, tx:w1+tx])

# Transformamos los keypoints de la imagen 1
test_keypoints_arr = np.array([i.pt for i in img2_keypoints])
keypoints_transformed = cv2.perspectiveTransform(test_keypoints_arr.reshape(-1,1,2), Ht.dot(H))
# Creamos los nuevos keypoints trasladados
keypoints_transformed = [cv2.KeyPoint(x, y, 1) for x,y in keypoints_transformed.reshape(-1,2)]

return result, keypoints_transformed, (tx, ty)

```

La función **WarpCompleteImage** toma dos imágenes, img1 y img2, junto con los keypoints de esta última (img2.keypoints) y una matriz de transformación H que representa la homografía entre las imágenes. El objetivo es proyectar img2 sobre img1. Primero, calcula las dimensiones y los puntos de esquina de ambas imágenes. Luego, transforma los puntos de las esquinas de la segunda imagen usando la homografía y determina las dimensiones del área combinada de las imágenes, ajustando una matriz de traslación para asegurar que se visualicen completamente ambas imágenes. Después, aplica la transformación perspectiva a la segunda imagen multiplicando por la traslación y sustituye sólo los píxeles negros de la primera imagen en la región solapada. Finalmente, ajusta los keypoints de la segunda imagen transformada y devuelve la imagen combinada resultante, los nuevos keypoints trasladados y las coordenadas de traslación (tx, ty) usadas para alinear las imágenes.

Después, creamos la función **CreatePanoramic**.

```

def CreatePanoramic(images, keypoints_imgs, matches_pairs, idx):
    # Se selecciona la imagen en la posición 'idx' como base para la panorámica
    result = images[idx]
    # Fusión hacia la izquierda de la imagen base
    for i in range(idx-1, -1, -1):
        # Se seleccionan el conjunto de matches correspondientes
        matches_it = matches_pairs[i]

        # Se obtienen los keypoints correspondientes de las imágenes
        src_pts = np.float32([keypoints_imgs[i][m.queryIdx].pt for m in matches_it])
        dst_pts = np.float32([keypoints_imgs[i+1][m.trainIdx].pt for m in matches_it])

        # Se calcula la homografía entre los keypoints utilizando RANSAC
        M, inliers = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 50)

        # Se realiza la transformación y fusión de las imágenes
        result, new_kps, t = WarpCompleteImage(result, images[i], keypoints_imgs[i], M)

        # Se ajustan los keypoints de la imagen base para mantener la coherencia en la panorámica
        keypoints_imgs[idx] = [cv2.KeyPoint(kp.pt[0]+t[0], kp.pt[1]+t[1], 1) for kp in
                               keypoints_imgs[idx]]
        # Se actualizan los keypoints de la imagen transformada
        keypoints_imgs[i] = new_kps

    # Fusión hacia la derecha de la imagen base
    for i in range(idx+1, len(imgs)):
        # Se seleccionan el conjunto de matches correspondientes

```

```

matches_it = matches_pairs[i-1]

# Se obtienen los keypoints correspondientes de las imágenes
src_pts = np.float32([ keypoints_imgs[i-1][m.queryIdx].pt for m in matches_it ])
dst_pts = np.float32([ keypoints_imgs[i][m.trainIdx].pt for m in matches_it ])

# Se calcula la homografía entre los keypoints (en sentido contrario) utilizando RANSAC
M, inliers = cv2.findHomography(dst_pts, src_pts, cv2.RANSAC, 50)

# Se realiza la transformación y fusión de las imágenes
result, new_kps, t = WarpCompleteImage(result, images[i], keypoints_imgs[i], M)

# Se actualizan los keypoints de la imagen transformada
keypoints_imgs[i] = new_kps

return result

```

La función **CreatePanoramic** crea una panorámica tomando una serie de imágenes y sus coincidencias de keypoints como entrada. Utilizando la imagen central (determinada por idx), calcula homografías entre imágenes adyacentes empleando el algoritmo RANSAC para encontrar transformaciones precisas. Luego, realiza fusiones iterativas (primero con las imágenes de la izquierda, luego con las de la derecha) aplicando transformaciones basadas en estas homografías para alinear las imágenes, actualizando los keypoints en cada iteración. Al final, genera una imagen panorámica completa y coherente que incorpora todas las imágenes alineadas.

Haciendo uso de estas funciones, obtenemos el resultado final de la panorámica realizada.



Figura 18: Panorámica completa

Claramente observamos que la imagen se acaba deformando en los extremos de la panorámica. Esto se debe a que estamos proyectando todas las imágenes sobre el plano de la imagen central. Esto es útil si queremos proyectar una imagen sobre otra, pero en el caso de aplicar la transformación a varias imágenes, se produce esta deformación.

La solución es utilizar una proyección cilíndrica o esférica, tarea de la que nos encargaremos en el siguiente apartado.

### 3.5. Resultado con ORB

Como en el ejercicio anterior, vamos a comparar los resultados como saldrían en el caso de que se utilizara ORB.

El proceso es el mismo que en el ejercicio anterior, vamos a usar la distancia de Hamming para ORB y 10000 features, los resultados de los keypoints y de los matches serían los siguientes:



Figura 19: Keypoints de las Imágenes



Figura 20: Matching de las Imágenes



Figura 21: Matching Correctos de las Imágenes

Con todo esto, vemos que se realizan unos buenos matches utilizando ORB, por tanto, vamos a pasar a ver el resultado final de la panorámica.



Figura 22: Panorámica final

Como observamos, la panorámica resultante de aplicar ORB es un buen resultado.

#### 4. Ampliación a Siete Imágenes evitando Distorsiones

En la tercera y última parte, uniremos siete imágenes en un panorama más amplio. Nos enfocaremos en evitar distorsiones en los bordes, proponiendo el uso de una superficie alternativa, como una esfera o cilindro, en lugar de proyectar el mosaico en un plano.

Las imágenes que usaremos serán las siguientes:



Figura 23: Imágenes que usaremos para la creación del panorama

El procedimiento será igual que en el ejercicio anterior, con la diferencia de que procesaremos previamente las imágenes para evitar las distorsiones. Este preproceso consistirá en una proyección cilíndrica que aplicaremos a las 7 imágenes, usando la función **project\_image\_to\_cylinder**, que convertirá las coordenadas de las imágenes para adaptarlas a la forma que queremos:

```
def project_image_to_cylinder(img, f = 1700):
    x_cent = img.shape[1] // 2
    y_cent = img.shape[0] // 2
    temp = np.indices(img.shape[:2])
    x_coords = temp[1]
    y_coords = temp[0]
    theta = (x_coords - x_cent) / f
    h = (y_coords - y_cent) / f
    x_hat = np.sin(theta)
    y_hat = h
    z_hat = np.cos(theta)
    new_x = f * (x_hat / z_hat) + x_cent
    new_y = f * (y_hat / z_hat) + y_cent
    new_img = cv2.remap(img, new_x.astype(np.float32), new_y.astype(np.float32), cv2.INTER_CUBIC)
    return new_img
```

Empíricamente hemos decidido elegir 1700 como distancia focal, tras probar diferentes cantidades. El resultado que obtenemos aplicado a nuestras imágenes es el siguiente:



Figura 24: Imágenes que usaremos para la creación del panorama

Una vez tenemos las imágenes proyectadas sobre un cilindro, pasamos a realizar el mismo proceso del ejercicio 2: Detección de keypoints con SIFT, descripción de keypoints, emparejamiento de keypoints en imágenes consecutivas y, finalmente, búsqueda de los mejores matches con RANSAC, llegando a lo siguiente:

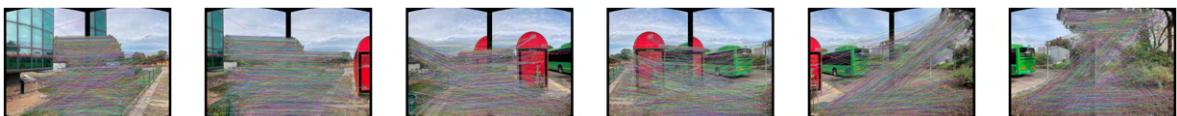


Figura 25: Matches correctos entre imágenes

Aplicando la función **CreatePanoramica** que vimos en el ejercicio 2, obtenemos el resultado final:



Figura 26: Panorámica final

## 5. Conclusión

Este proyecto exploró conceptos fundamentales del procesamiento de imágenes utilizando OpenCV. Desde la detección de keypoints mediante SIFT u ORB hasta la creación de panorámicas, se aplicaron algoritmos como RANSAC para filtrar correspondencias y obtener homografías precisas. La introducción de proyecciones cilíndricas mitigó las distorsiones en los bordes de las panorámicas, mejorando la coherencia visual. La combinación de estos métodos permitió la alineación efectiva de imágenes y la generación de panorámicas amplias y sin distorsiones, demostrando la eficacia de estas técnicas en el procesamiento de imágenes.