Visión por Computador

grado en ingeniería informática

# PRÁCTICA 2: DETECCIÓN DE PUNTOS RELEVANTES Y COSTRUCCIÓN DE PANORAMAS

**Realizado por:**
Alba Casillas Rodríguez

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2021-2022

# Índice

# 1. Ejercicio 1

**Extracción de regiones relevantes en un espacio de escalas. Este punto se centra en detectar KeyPoints sobre cada una de las imágenes de Yosemite.rar (usar versiones en rango de gris) y dibujarlos sobre las imágenes haciendo uso de la función drawKeyPoints(). Para ello, se ha de construir un Espacio de Escalas (pirámide) de Lowe con cuatro octavas en total y tres escalas dentro de cada octava. Suponer que la imagen original está afectada por un alisamiento debido a la captura de $\sigma = 0.8$ y queremos introducir una primera octava de índice-0 de manera que la imagen semilla en el espacio de escalas tenga un $\sigma$ =1.6.**

## 1.1. Apartado A

**¿Qué operaciones sobre la imagen original de $\sigma = 0.8$ nos permite fijar una imagen semilla de $\sigma = 1.6$ ?**

En el algoritmo SIFT se parte de una imagen semilla con $\sigma = 1.6$, ya que se asume que este valor proporciona una mayor garantía de obtener el máximo número de puntos.

Para conseguir esto, se tiene en cuenta de que **no** se aplica una convolución con un kernel gaussiano a la imagen original, puesto que de hacerlo, no podríamos obtener mucha información de la primera imagen; por eso se necesita una *imagen anterior* a la original (con la que poder comparar "por debajo").

Esta imagen anterior se consigue **duplicando** el tamaño de la imagen original. Tras esto, se realiza una **aplicación iterativa de los sigmas mediante convolución**. Por último, se **interpola** para reducir el tamaño a la mitad y obtener la *imagen interpolada*.

## 1.2. Apartado B

**Implementar una función que calcule las escalas de cualquier octava de la forma más eficiente posible. Es decir, reusable para cualquier escala.**

Para obtener el cálculo de las escalas de forma eficiente, debemos usar kernels pequeños, donde se utilizará los mismos $\sigma$ para todas las escalas de las octavas. Dichos $\sigma$ se obtendrán mediante la expresión:

$$\sigma_s = \sigma_0 \times \sqrt{2^{\frac{2s}{ns}} - 2^{\frac{2(s-1)}{ns}}}, s = 1, ..., ns + 2$$

donde $ns$ es el número de escalas y $\sigma_0 = \frac{\sigma\_ini}{\delta_0}$ , siendo $\sigma\_ini = 0{,}8$ y $\delta_0 = 0{,}5$.

Aunque $ns = 3$, se calculan dos escalas extra para hacer posible cálculos posteriores.

Una vez calculados los $\sigma$, los utilizamos para convolucionar la imagen en cada una de las escalas. La convolución ha sido implementada con el código realizado en la práctica anterior, por lo que no se dará detalles sobre ella.

De esta manera, esta función se puede generalizar para todas las octavas, ya que solo se necesita pasar como argumento la imagen a convolucionar, $\sigma_0$ y el número de escalas por octava.

Esta generalidad del cálculo de escalas se debe a las **octavas**, donde el concepto de octava es que $\sigma$ se haga el doble, permitiendo reducir a la mitad la siguiente imagen sin prejuicio ninguno y volver a construir la imagen siguiente con exactamente la misma técnica que la anterior. Esto se cumple ya que partimos de un $\sigma$ con el que al alisar $ns=3$ veces se obtiene el doble de $\sigma$, de forma que al reducir la imagen, volverá a quedar el $\sigma$ del que se partió.

Este *cálculo incremental* es el que evita que a medida que aumentamos de octava, los kernels sean demasiado grandes.

## 1.3. Apartado C

**Usar la función implementada en (b) para calcular las escalas de todas las octavas (0-3). Mostrar las imágenes calculadas agrupadas por octava (mostrar solo las escalas 1,2 y 3 en cada octava).**

***Imagen: Yosemite1.jpg***

(a) Octava 0 - Escala 1 (b) Octava 0 - Escala 2 (c) Octava 0 - Escala 3

Figura 1: Escalas 1,2 y 3 de la octava 0.

(a) Octava 1 - Escala 1 (b) Octava 1 - Escala 2 (c) Octava 1 - Escala 3

Figura 2: Escalas 1,2 y 3 de la octava 1.

(a) Octava 2 - Escala 1 (b) Octava 2 - Escala 2 (c) Octava 2 - Escala 3

Figura 3: Escalas 1,2 y 3 de la octava 2.

(a) Octava 3 - Escala 1 (b) Octava 3 - Escala 2 (c) Octava 3 - Escala 3

Figura 4: Escalas 1,2 y 3 de la octava 3.

Se muestran las escalas 1, 2 y 3 para las cuatro octavas que se deben calcular. Como es de esperar, el suavizado a medida que aplica iterativamente las convoluciones se hace cada vez mayor.

***Imagen: Yosemite2.jpg***

(a) Octava 0 - Escala 1 (b) Octava 0 - Escala 2 (c) Octava 0 - Escala 3

Figura 5: Escalas 1,2 y 3 de la octava 0.

(a) Octava 1 - Escala 1 (b) Octava 1 - Escala 2 (c) Octava 1 - Escala 3

Figura 6: Escalas 1,2 y 3 de la octava 1.

(a) Octava 2 - Escala 1 (b) Octava 2 - Escala 2 (c) Octava 2 - Escala 3

Figura 7: Escalas 1,2 y 3 de la octava 2.

(a) Octava 3 - Escala 1 (b) Octava 3 - Escala 2 (c) Octava 3 - Escala 3

Figura 8: Escalas 1,2 y 3 de la octava 3.

## 1.4. Apartado D

**Calcular el espacio de escalas Laplaciano a partir de la pirámide e identificar los 100 extremos locales con mayor respuesta, presentes en la misma. Extraer en un vector (x,y,$\sigma$) para cada uno de ellos en los ejes de la imagen original.**

En este apartado se construye el Espacio de Escalas Laplaciano mediante la construcción de las ***Diferencias de Gaussianas (DoG)*** entre una escala *j* y la escala *j+1* de cada octava *i*. Este cálculo se realiza ya que de esta manera la Laplaciana ya estará normalizada, y solo se deberá buscar los máximos entre cada conjunto de tres escalas consecutivas.

De esta manera, si tenemos 6 escalas por cada octava (ya que se necesita contar las tres escalas extra), debemos obtener 5 Diferencias de Gaussianas por octava:

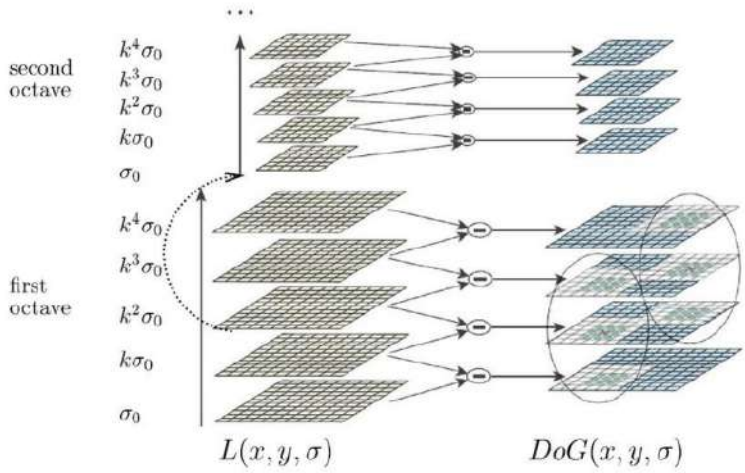
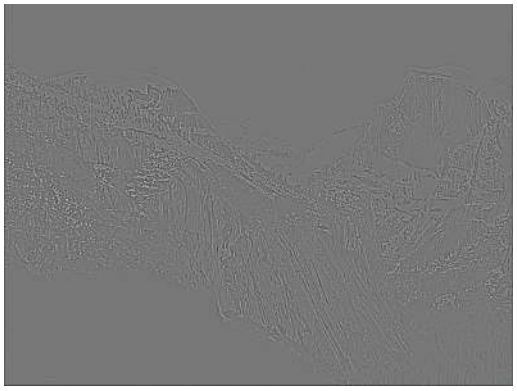
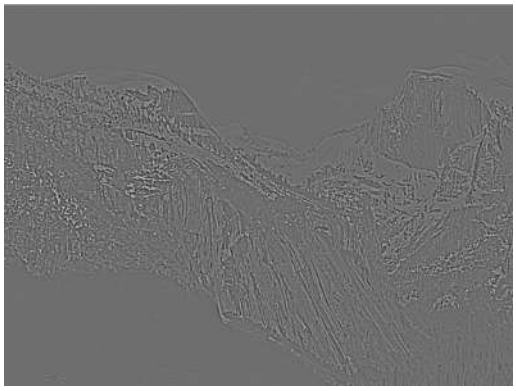



Figura 9: Diferencias de Gaussianas por octavas en SIFT.

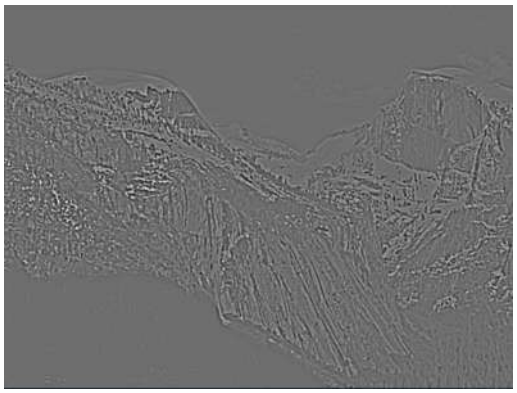
***Imagen: Yosemite1.jpg***



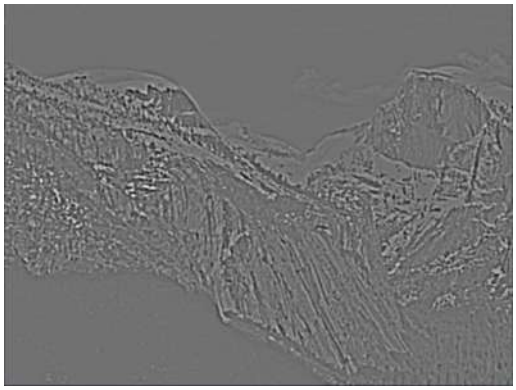
(a) Octava 0 - DoG entre escalas 1 y 2



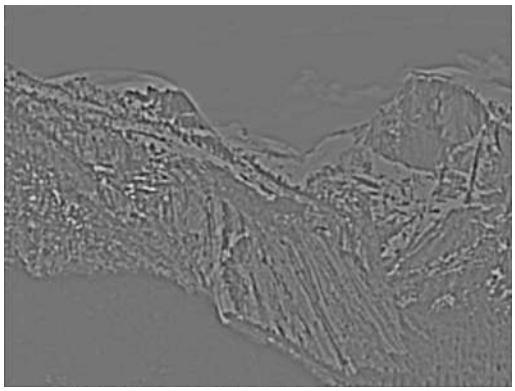
(b) Octava 0 - DoG entre escalas 2 y 3



(c) Octava 0 - DoG entre escalas 3 y 4

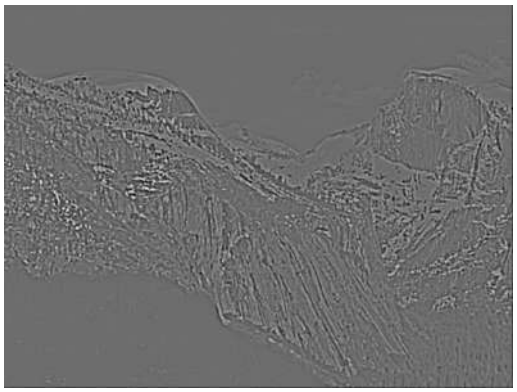


(d) Octava 0 - DoG entre escalas 4 y 5

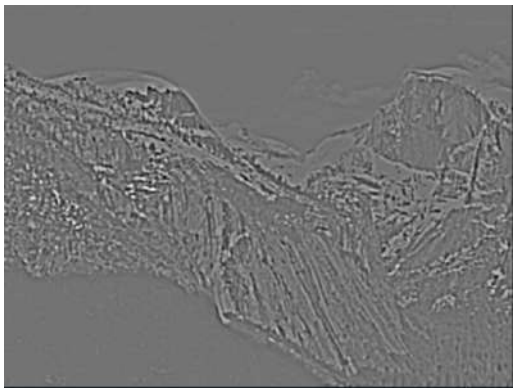


(e) Octava 0 - DoG entre escalas 5 y 6

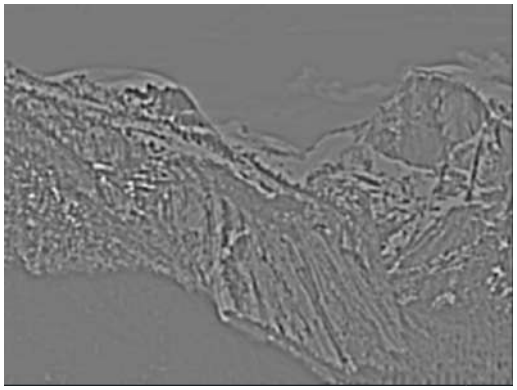
Figura 10: YOSEMITE 1 - Diferencias de Gaussianas (DoG) en Octava 0.



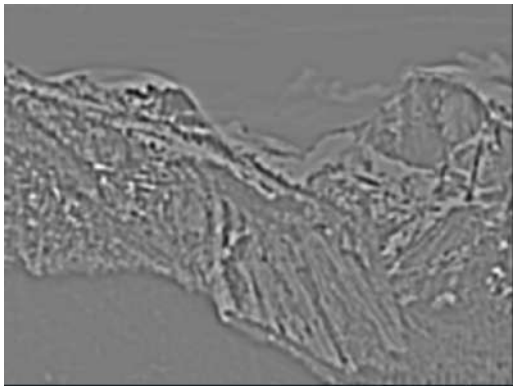
(a) Octava 1 - DoG entre escalas 1 y 2



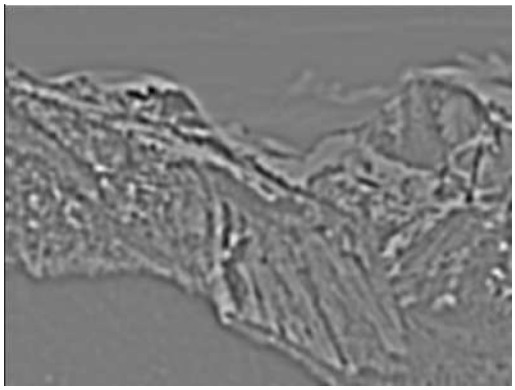
(b) Octava 1 - DoG entre escalas 2 y 3



(c) Octava 1 - DoG entre escalas 3 y 4

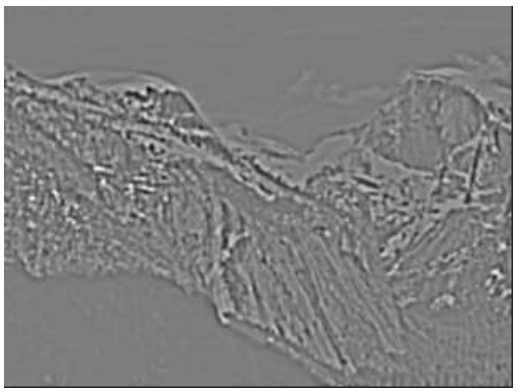


(d) Octava 1 - DoG entre escalas 4 y 5

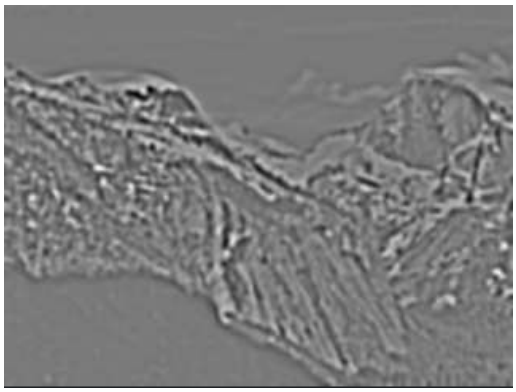


(e) Octava 1 - DoG entre escalas 5 y 6

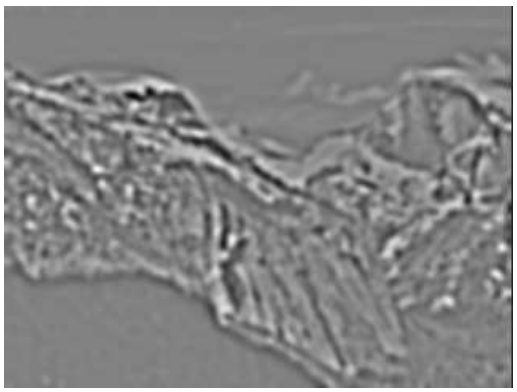
Figura 11: YOSEMITE 1 - Diferencias de Gaussianas (DoG) en Octava 1.



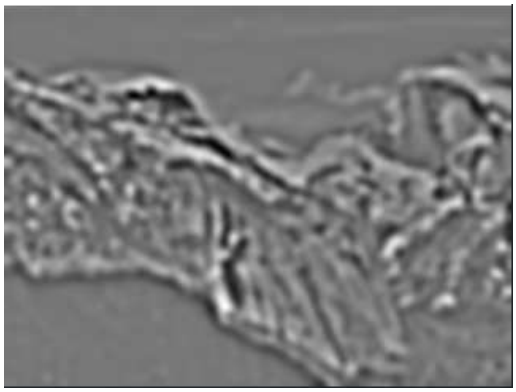
(a) Octava 2 - DoG entre escalas 1 y 2



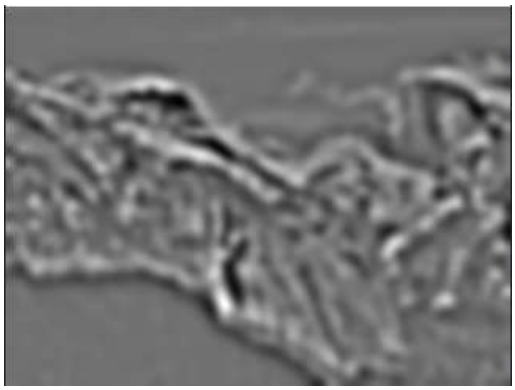
(b) Octava 2 - DoG entre escalas 2 y 3



(c) Octava 2 - DoG entre escalas 3 y 4

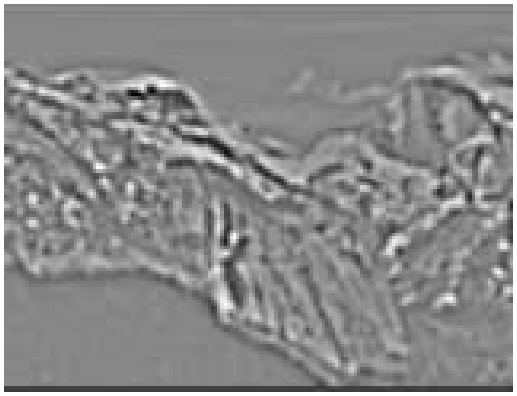


(d) Octava 2 - DoG entre escalas 4 y 5

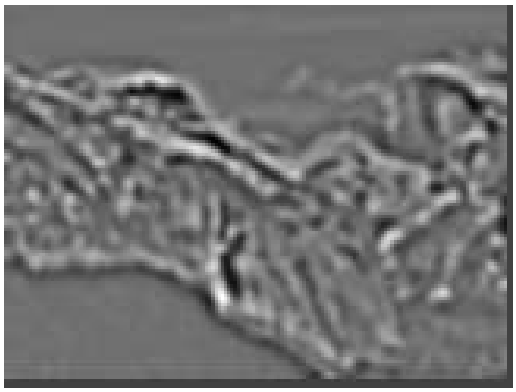


(e) Octava 2 - DoG entre escalas 5 y 6

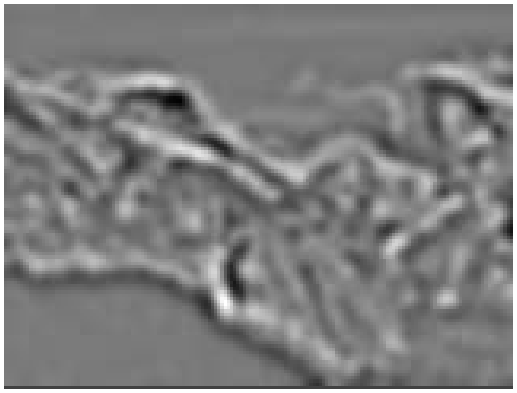
Figura 12: YOSEMITE 1 - Diferencias de Gaussianas (DoG) en Octava 2.



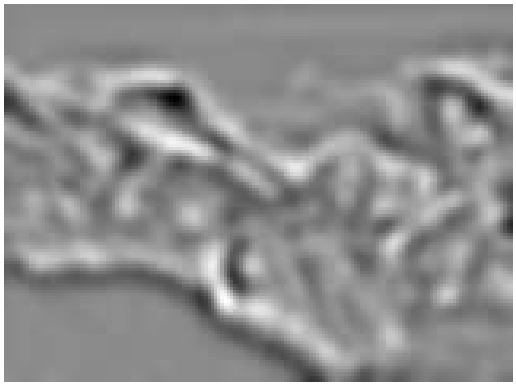
(a) Octava 3 - DoG entre escalas 1 y 2



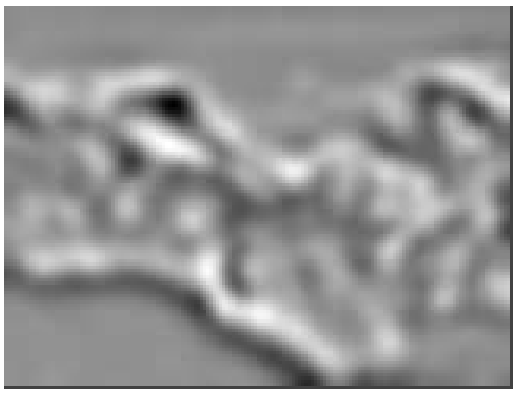
(b) Octava 3 - DoG entre escalas 2 y 3



(c) Octava 3 - DoG entre escalas 3 y 4



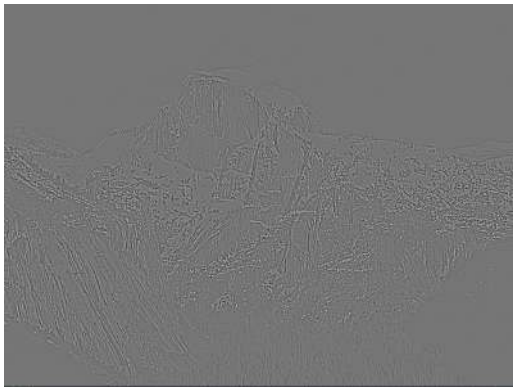
(d) Octava 3 - DoG entre escalas 4 y 5



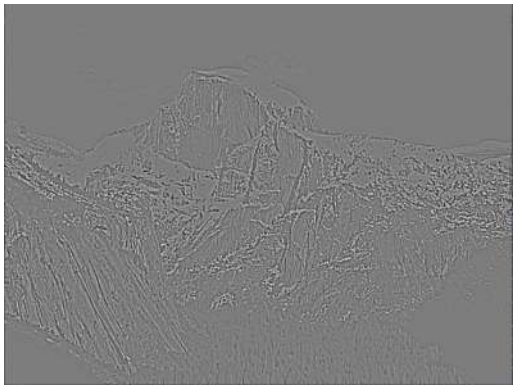
(e) Octava 3 - DoG entre escalas 5 y 6

Figura 13: YOSEMITE 1 - Diferencias de Gaussianas (DoG) en Octava 3.

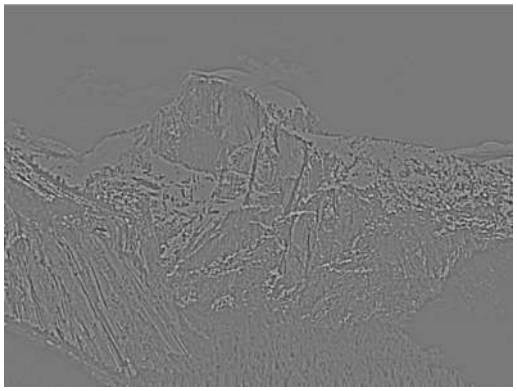
***Imagen: Yosemite2.jpg***



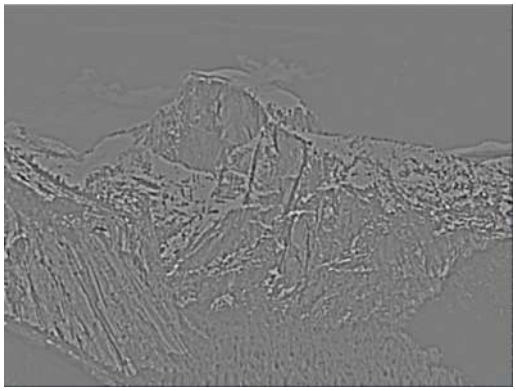
(a) Octava 0 - DoG entre escalas 1 y 2



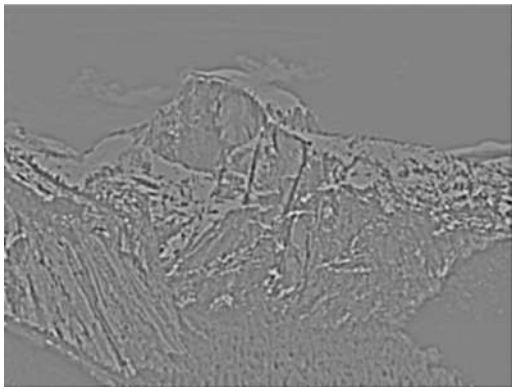
(b) Octava 0 - DoG entre escalas 2 y 3



(c) Octava 0 - DoG entre escalas 3 y 4

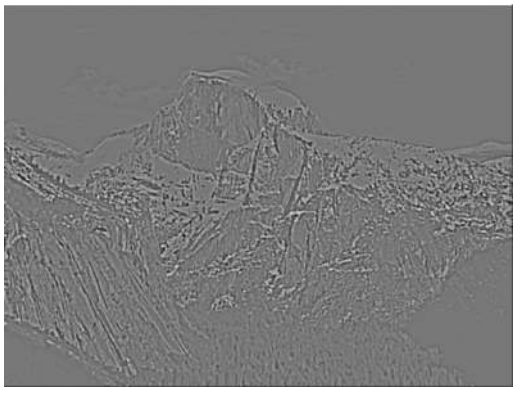


(d) Octava 0 - DoG entre escalas 4 y 5

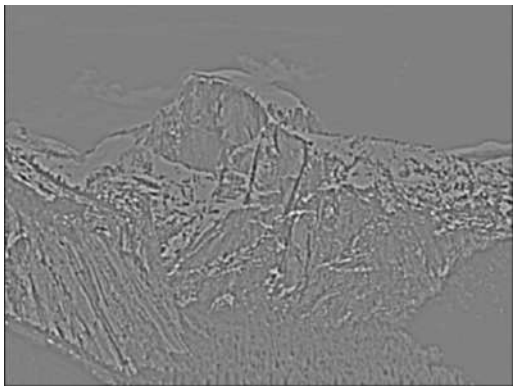


(e) Octava 0 - DoG entre escalas 5 y 6

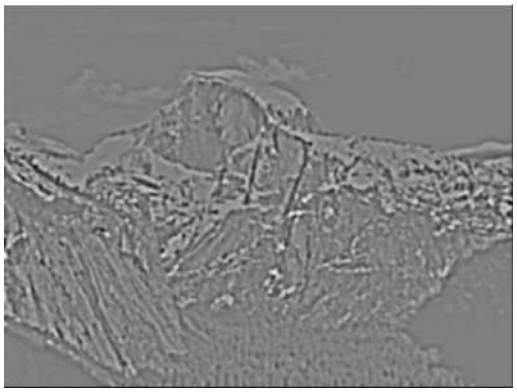
Figura 14: YOSEMITE 2 - Diferencias de Gaussianas (DoG) en Octava 0.



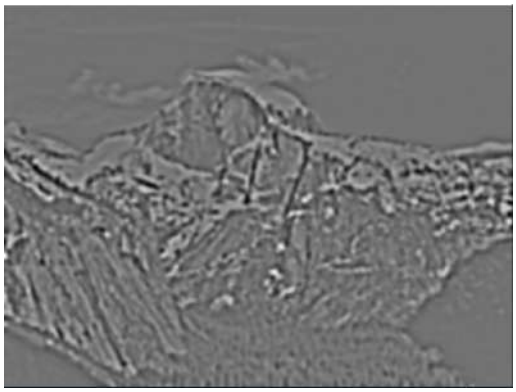
(a) Octava 1 - DoG entre escalas 1 y 2



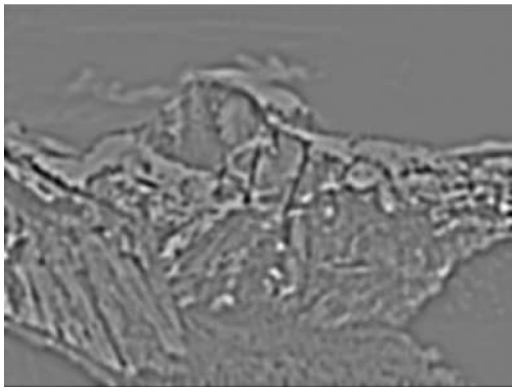
(b) Octava 1 - DoG entre escalas 2 y 3



(c) Octava 1 - DoG entre escalas 3 y 4

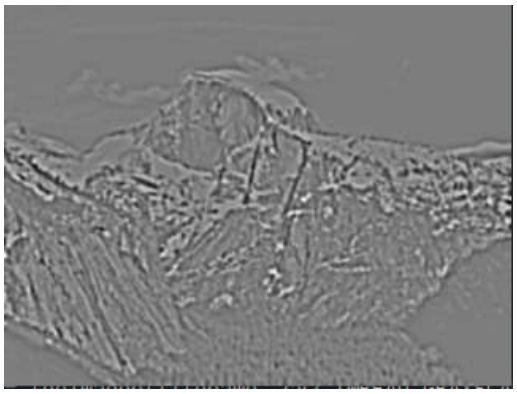


(d) Octava 1 - DoG entre escalas 4 y 5

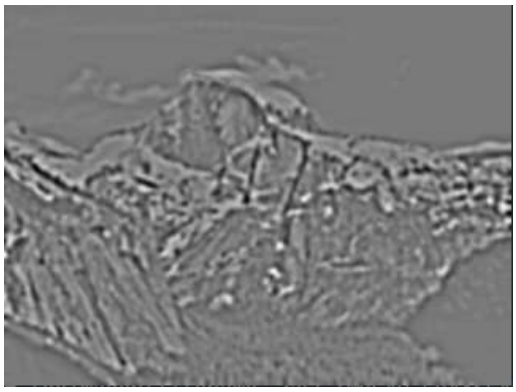


(e) Octava 1 - DoG entre escalas 5 y 6

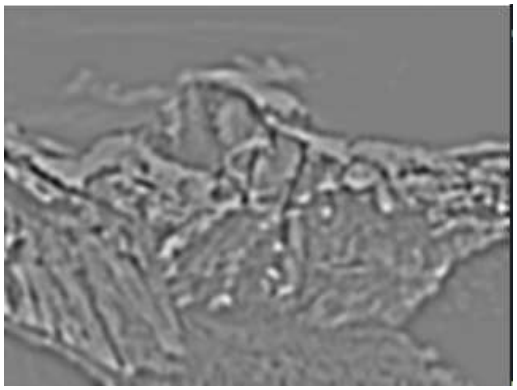
Figura 15: YOSEMITE 2 - Diferencias de Gaussianas (DoG) en Octava 1.



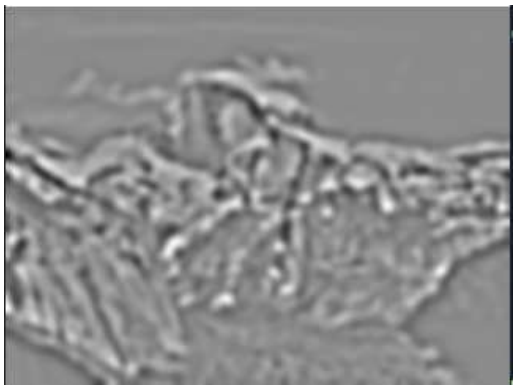
(a) Octava 2 - DoG entre escalas 1 y 2



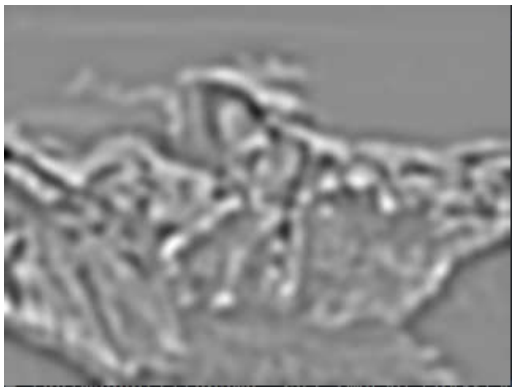
(b) Octava 2 - DoG entre escalas 2 y 3



(c) Octava 2 - DoG entre escalas 3 y 4

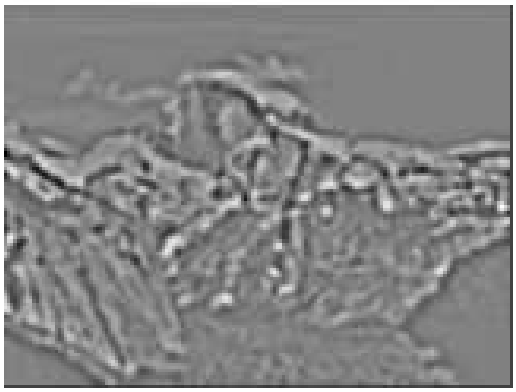


(d) Octava 2 - DoG entre escalas 4 y 5

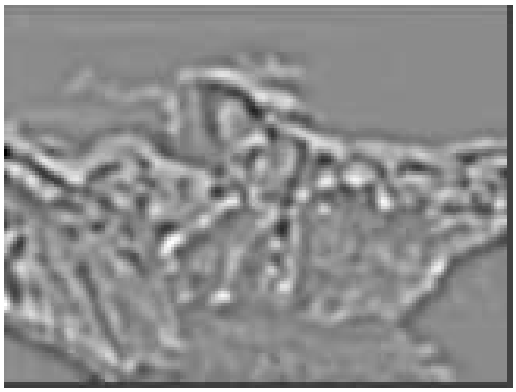


(e) Octava 2 - DoG entre escalas 5 y 6

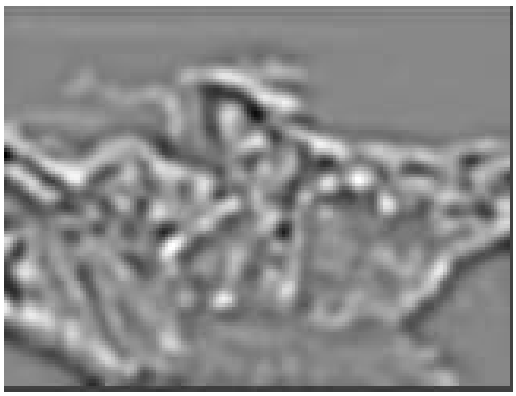
Figura 16: YOSEMITE 2 - Diferencias de Gaussianas (DoG) en Octava 2.



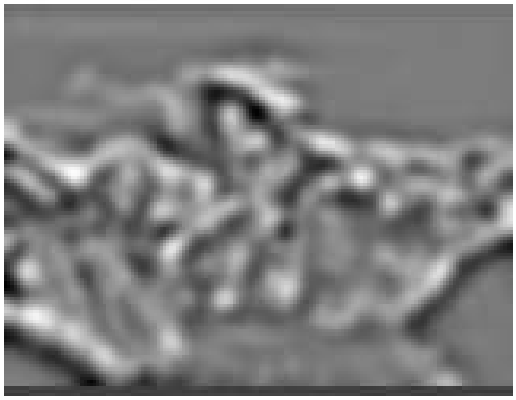
(a) Octava 3 - DoG entre escalas 1 y 2



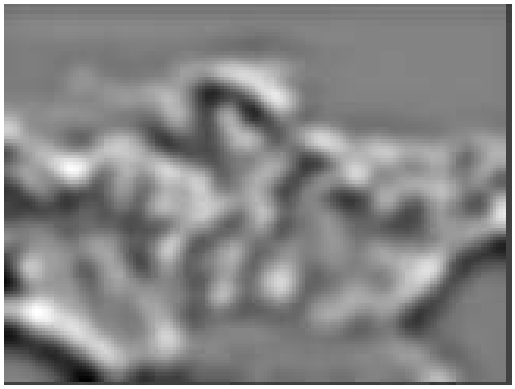
(b) Octava 3 - DoG entre escalas 2 y 3



(c) Octava 3 - DoG entre escalas 3 y 4



(d) Octava 3 - DoG entre escalas 4 y 5



(e) Octava 3 - DoG entre escalas 5 y 6

Figura 17: YOSEMITE 2 - Diferencias de Gaussianas (DoG) en Octava 3.

Una vez calculadas todas las Diferencias de Gaussianas, cada tres escalas y por cada píxel central, se analizan sus **vecinos** de la escala de arriba, de su escala y de la escala de abajo, de manera que nos quedamos con *(x, y, $\sigma$)* de los ***extremos***

***locales*** encontrados.

El objetivo de utilizar un Espacio de Escalas Laplaciano para detectar patrones en las imágenes es encontrar el mayor número de respuestas grandes; de forma que consideraremos como extremo local a los *máximos* y *mínimos* (en valor absoluto) locales.

Cabe destacar que el $\sigma$ a almacenar en este caso es $\sigma$ *absoluto*, el cual se denota por la expresión:

$$\sigma_k = \sigma_0 * 2^{\frac{k}{n_s}}$$

ya que aunque se calculen los incrementos por cuestiones de eficiencia, este $\sigma$ es el que indica la escala real en la que ha sido detectado el extremo.

Por tanto en esta práctica, por cada octava calculamos los extremos entre las Diferencias de Gaussianas calculadas 1,2 y 3, las Diferencias de Gaussianas 2,3 y 4 y las Diferencias de Gaussianas 3,4 y 5.

Para ello, por cada píxel de la escala central debemos comparar con sus vecinos adyacentes de la escala perteneciente, la escala de arriba y la escala de abajo. Esta operación es tan simple como calcular para cada píxel una región de dimensiones 3x3x3, y comprobar si es máximo o mínimo de dicho cubo.

Como ya se ha dicho anteriormente, por cada extremo encontrado, se guardaran sus coordenadas y el $\sigma$ absoluto correspondiente. En la práctica, a parte de estos valores, almaceno también el valor de cada píxel, así como la octava en la que ha sido encontrado con el único motivo de facilitar operaciones posteriores.

Finalmente, ordeno los extremos encontrados de mayor a menor respuesta para poder elegir los 100 con mayor respuesta.

## 1.5. Apartado E

**Mostrar la imagen con los extremos locales extraídos usando para ello un círculo de radio 6x sobre la escala de detección $\sigma$.**

Una vez calculados los extremos, debemos visualizar el resultado. Para ello, se hace uso de *KeyPoint* y *drawKeypoints* de **OpenCv**.

Se utiliza el método *KeyPoint*[3] para guardar los extremos encontrados en formato Keypoint, donde se puede indicar diversos parámetros como la orientación, escala y posición 2D.

***(cv2.KeyPoint(x, y, size))***

donde:

- **x** es la coordenada x del Keypoint.
- **y** es la coordenada y del Keypoint.
- **size** es el diámetro del Keypoint. En nuestro caso se indica en el enunciado que debemos usar un radio de 6x, por lo que no solo debemos multiplicar el $\sigma_k$ por 6, sino también multiplicar de nuevo por 2 para obtener el diámetro.

Además, para una correcta visualización del resultado, se debe tener en cuenta:

- Se debe escalar las coordenadas a las de la imagen original.
- OpenCv considera las coordenadas (x,y) lo que nosotros consideramos comunmente como (y,x), por lo que lo parámetros (x,y) deberan estar invertidos en la llamada a la función.

Con el método *DwarKeypoints*[4] se dibujan los Keypoints encontrados en la imagen.

***(cv2.KeyPoint(image, keypoints, outImage, flags))***

donde:

- **image** es la imagen de entrada, en nuestro caso: la imagen original en escala de grises.
- **keypoints** son los keypoints detectados.
- **outImage** es la imagen de salida, su contenido varía dependiendo del flag que se defina.

- **flags**: ***cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS***. Este parámetro configura las características con las que se dibujan los Keypoints. En nuestro caso, el flag usado asegura que el tamaño del cícurlo corresponde al de la "mancha"detectada.

Una vez explicado esto, procedemos a mostrar y analizar los resultados:

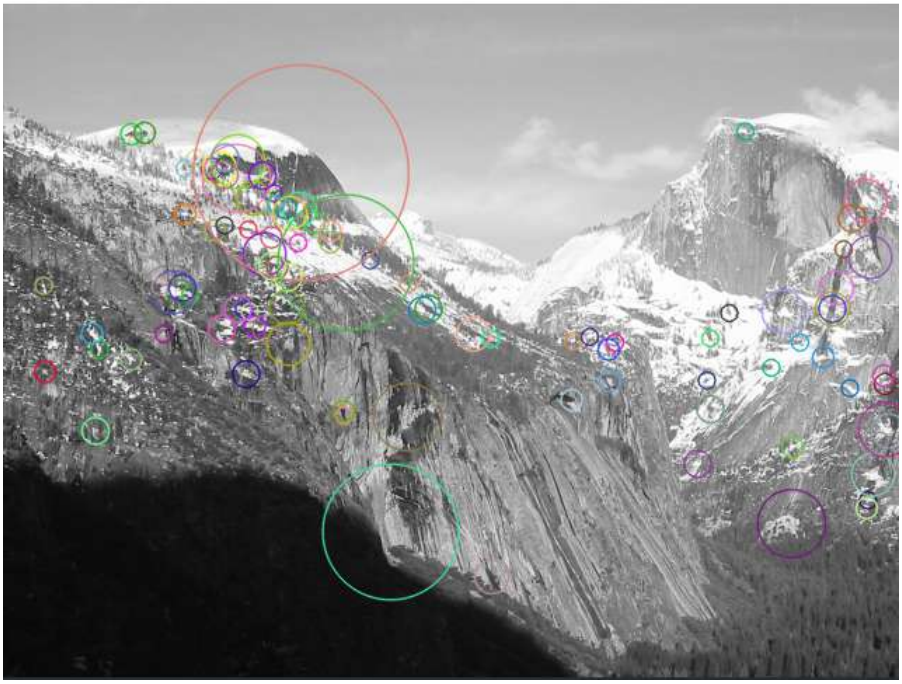


Figura 18: Keypoints detectados en YOSEMITE 1

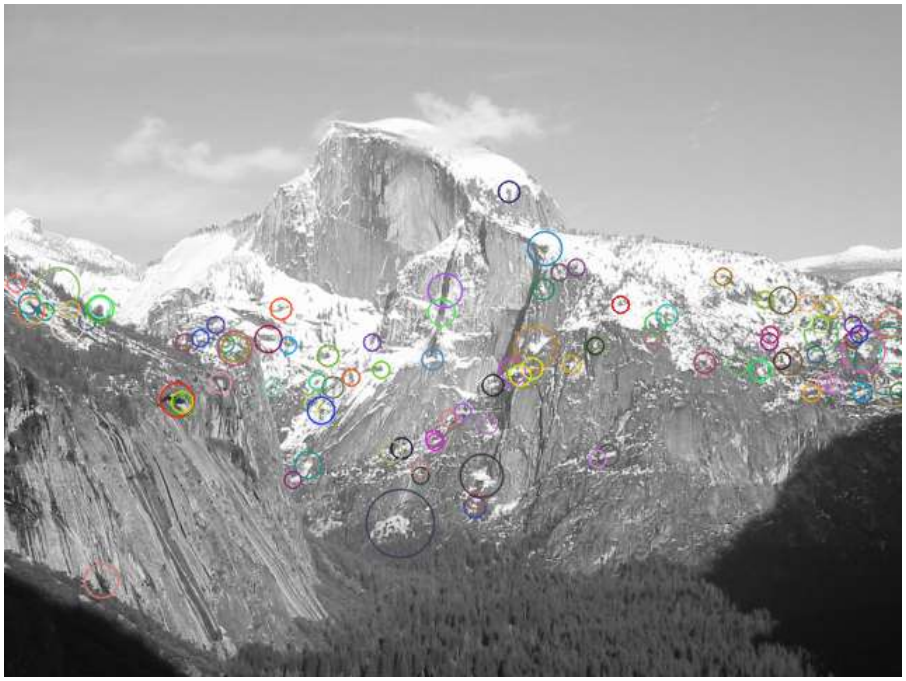


Figura 19: Keypoints detectados en YOSEMITE 2

Podemos considerar cierta calidad en los puntos obtenidos ya que, como se puede observar, están concentrados por la ladera de la montaña, mayoritariamente en aquellos lugares en los que la montaña se mezcla con la nieve. De igual forma, no encontramos ningún punto de interés ni en el cielo, ni en las sombras negras, ni en la arbolada. Esto se debe a que en esas zonas existe una variación pequeña de los niveles de grises de la imagen, por lo que, como es de esperar, SIFT no lo detecta como una región de interés.

Los círculos correspondientes a los Keypoint rodean zonas de la montaña claras (por la presencia de nieve o de la luz) que tienen a su alrededor zonas más oscuras; además de zonas oscuras (de la ladera o de alguna sombra producida por la iluminación) que está rodeada de zonas más claras; es decir, detecta aquellas regiones donde existe un salto notable en la diferencia de intensidad.

Otro detalle a destacar y que se aprecia claramente en la imagen *Yosemite 1*, es un Keypoint detectado en la última octava (el círculo más grande); del cual podemos observar que en la zona en la que este ha sido detectado, a su vez han sido detectados una cantidad grande de extremos en diferentes escalas; por lo que se podría considerar que esa zona tiene un alto interés en la imagen.

## 1.6. BONUS

**Pintar los mayores extremos de acuerdo a la siguiente distribución por octavas: octava 1 (50 % del total), octava 2 (25 % del total), octava 3 (15 % del total) y octava 4 (10 % del total).**

Para este apartado, simplemente se debe tener también en cuenta en qué octava se ha encontrado el máximo. Por ello, se guardan los extremos locales encontrados por cada octava en listas diferentes, de las cuales solo mostraremos los porcentajes indicados por el enunciado del problema.

Figura 20: BONUS - Keypoints detectados en YOSEMITE 1

De nuevo, los Keypoints son detectados en aquellas zonas de la montaña que despiertan mayor interés, sin encontrar extremos en zonas planas como el cielo, la sombra negra o la arbolada, aunque se hayan detectado puntos en octavas altas cuyo radio abarque un poco esas zonas.

Además, como cabe de esperar, en esta imagen encontramos puntos de interés que ya se encontraban en la imagen del apartado anterior (porque son extremos locales de la imagen y de la propia escala), además de encontrar nuevos puntos añadidos al aumentar el porcentaje dentro de cada escala.

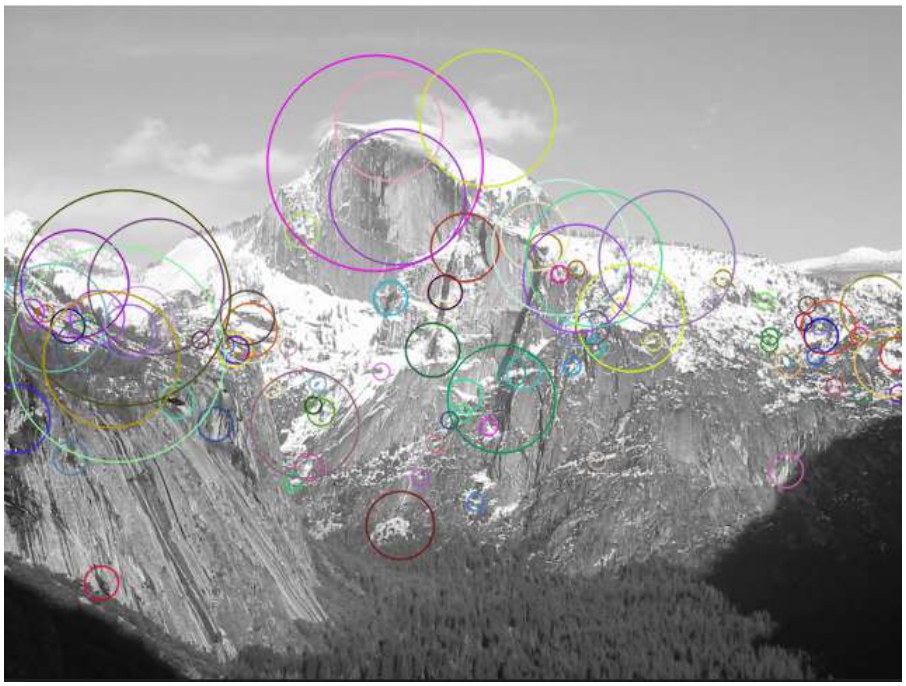


Figura 21: BONUS - Keypoints detectados en YOSEMITE 2

Al igual que la imagen anterior, se visualizan los Keypoints que dan la información más relevante de la imagen. Da la sensación de que se han detectado sobre todo a la izquierda-alta de la imagen, pero esto se debe a que gran parte de la derecha de la imagen está formada por sombras y por árboles que no varían mucho en su intensidad.

# 2. Ejercicio 2

**Con cada dos de las imágenes de Yosemite con solapamiento detectar y extraer los descriptores SIFT de OpenCV, usando para ello la función cv2.detectAndCompute(). Establecer los puntos en correspondencias existentes entre las dos imágenes usando el objeto BFMatcher de OpenCV y los criterios de correspondencias "BruteForce+crossCheck y "Lowe-Average-2NN". Mostrar ambas imágenes en un mismo canvas y pintar líneas de diferentes colores entre las coordenadas de los puntos en correspondencias. Mostrar en cada caso un máximo de 100 elegidas aleatoriamente.**

Para la realización del ejercicio se va a utilizar el descriptor **SIFT** para extraer información en las imágenes que nos permita establecer puntos de correspondencias entre ellas.

De esta forma, el descriptor SIFT calcula los *keypoints* y los *descriptores* en un espacio de escalas para asegurarse de que imágenes con escalas diferentes seguirán produciendo los mismos valores. El hecho de que el descriptor sea invariable frente a ciertas transformaciones permite reconocer la misma información de imágenes diferentes que representen el mismo objeto.

Para el cálculo de dichos keypoints y descriptores, primero se creará un objeto SIFT para extaer las características mediante *SIFT_create* y tras esto se utilizará la función *detectAndCompute* de **OpenCV** para cada una de las imágenes de Yosemite. Este método, a parte de la imagen, recibe una máscara utilizada a la hora de realizar cálculos, aunque en nuestro caso no haremos uso de ella y estableceremos su valor como 'None'.

## 2.1. Brute Force + CrossCheck

A continuación, se establecen los puntos de correspondencias entre las dos imágenes. La primera técnica para encontrar parejas de correspondencias usada es ***Brute Force*** (Fuerza Bruta) con ***CrossCheck***.

Para ello, primero se crea una instancia de la clase **BFMatcher**[5] mediante el método *BFMatcher_create* de **OpenCV**, con el propósito de hacer coincidir los descriptores de los Keypoints buscando el descriptor más similar en las diferentes imágenes. En este método, especificamos el parámetro *normType* con el valor *cv2.NORM_L2*; de esta manera indicamos que el criterio elegido para establecer correspondencias será la *Distancia Euclídea*.

Para realizar dicha correspondencia, se realizará una llamada al método *match*,

indicando los dos descriptores de las imágenes.

La técnica de Fuerza Bruta, compara cada punto $p_{1i}$ de la primera imagen con cada punto $p_{2j}$ de la segunda imagen y establece una correspondencia con aquel punto con el que haya obtenido la *distancia mínima (Euclídea)* entre ambos descriptores.

Para disminuir el número de emparejamientos incorrectos, se activa el parámetro *CrossCheck*, de manera que se invertirá las comparaciones (se compara cada punto $p_{2j}$ con cada $p_{1i}$). De esta forma, existirá un *match* si $p_{1i}$ y $p_{2j}$ se eligen mutuamente como aquellos que tienen la distancia mínima.

## 2.2. Lowe-Avergae-2NN

Esta técnica es propuesta por David Lowe. Se propone, de nuevo, comparar cada punto de $p_{1i}$ de la primera imagen con cada uno de los puntos $p_{2j}$ de la segunda imagen. Sin embargo, para evitar un segundo rastreo (no se tiene que activar el parámetro CrossCheck), obtenemos aquel punto con el que se obtiene la distancia mínima *y el siguiente*.

Si *el radio de la distancia entre el mínimo y el siguiente* es cercano (se parecen), se considera que existe una ambigüedad y no se considera. Si la distancia es grande, se asume que ninguno de los demás puntos se parecen y aceptamos la correspondencia.

Por tanto, el procedimiento trata de crear una instancia de **BFMatcher**, realizar la correspondencia entre ambos descriptores con el criterio de Lowe mediante el método *KnnMatch*, donde establecemos el parámetro *K=2* para indicar que se quiere los dos puntos más cercanos y comprobar cuáles de ellos se aceptan como correspondencias buenas. El umbral establecido para que haya una cantidad suficiente de puntos de calidad está entre 0.7-0.8; por lo que existirá un match si:

$$distance(p_{1i}, p_{2,mejor}) < 0{,}8 distance(p_{1i}, p_{2,segundomejor})$$

Una vez calculadas las correspondencias, se elgirán aleatoriamente 100 de ellas. Cabe destacar que se ha decidido hacer uso de una semilla para no obtener una muestra diferente cada vez que se ejecute el ejercicio.

**Valorar la calidad de los resultados obtenidos a partir de un par de ejemplos aleatorios de 100 correspondencias. Hacerlo en términos de las correspondencias válidas observadas por inspección ocular y las tendencias de las líneas dibujadas. ¿concluiría que ambos métodos son equivalentes u observa uno mejor que otro?**

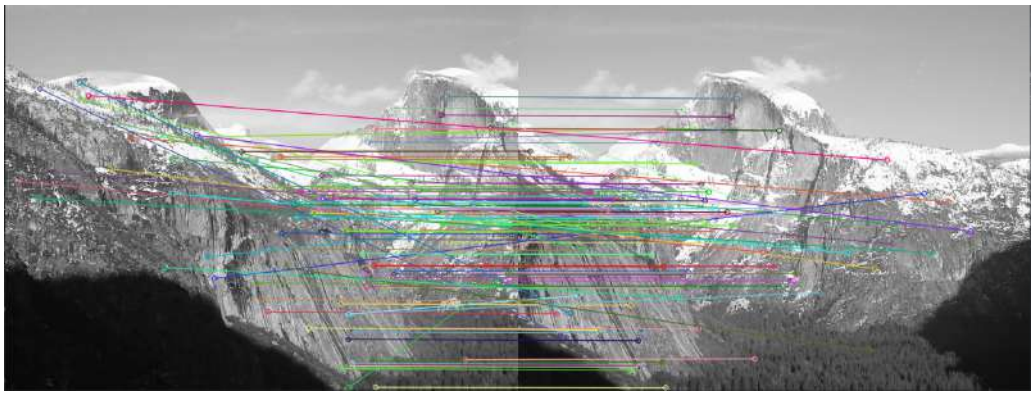


Figura 22: Puntos de Correspondencias - Brute Force + CrossCheck

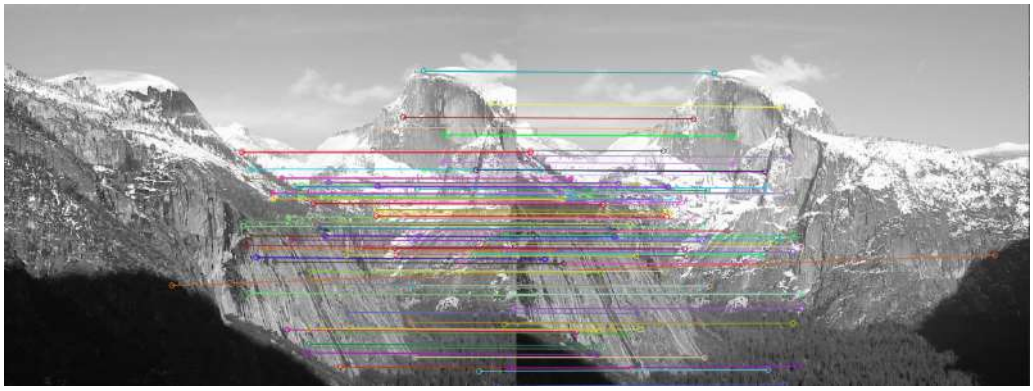


Figura 23: Puntos de Correspondencias - Lowe Avergae 2NN

No hay demasiada cabida a dudas de que la técnica de **Lowe Avergae 2NN** es la que proporciona mejores resultados, encontrando ciertas observaciones que lo confirman. Primero, debemos pensar en que nos encontramos con dos fotos del tipo panorámica en horizontal, de modo que es lo más lógico es que no se crucen las lineas. Esto no ocurre en el resultado de *Brute Force*, donde encontramos un gran número de correspondencias erroneas que se curzan.

Otra observación a tener en cuenta es el hecho de encontrar una correspondencia en una parte de la segunda imagen que no se ve en la primera (*outliers*,

siendo completamente erróneo. Esto se debe a que cada punto se evalúa ***fuera del contexto*** de la imagen, por lo que sus descriptores pueden parecerse aunque no se correspondan.

También se puede encontrar un outlier establecido por la técnica de Lowe, fruto de haber extraído el descriptor del contexto al establecer la correspondencia; sin embargo, el uso de esta técnica marca una mejoría notable al encontrar correspondencias de puntos.

# 3. Ejercicio 3

**Construicción de un panorama rectangular.Usar las imágenes del fichero mosaico.rar. Escribir una función que dadas 3 imágenes con solapamiento 2-a-2: a) Extraiga los KeyPoints SIFT y establezca un conjunto de puntos en correspondencias entre cada dos imágenes solapadas. b) Estime la homografía entre las imágenes a partir de dichas correspondencias y C) muestre un Mosaico bien registrado a partir de dichas imágenes. Estimar las homografías usando la función cv2.findHomography() con RANSAC.**

Se quiere construir un mosaico con 3 imágenes. Para ello, primero debemos plantearnos dónde se va a montar la composición de imágenes. Esto nos lleva a crear un ***canvas*** negro (una matriz de ceros).

El *tamaño del canvas* depende bastante de las imágenes y el tipo de mosaico a construir. En nuestro caso se quiere conseguir un mosaico que muestre una panorámica con un movimiento lineal unidimensional de izquierda a derecha, por lo que se asume que no habrá una gran variación en la altura; de esta manera, utilizar una altura ligeramente mayor de la altura de la imagen sería suficiente. En cuanto al ancho, se puede intuir que como máximo tendrá la suma de los anchos de todas las imágenes, aunque teniendo en cuenta de que las imágenes no se solapan solamente por la zona de los bordes, sino que prácticamente se solapa la mitad de cada imagen, debería ser suficiente con sumar los anchos de la mitad de la lista de imágenes.

La homografía inicial debe de ser una ***homografía de traslación al canvas***, para tener la imagen central en el centro del canvas. Para el cálculo de esta homografía no es necesario ninguna función, simplemente una operación de traslación:

**H0 = [[1,0, ancho_canvas//2 - ancho_imagen_central//2]**
**[0,1,alto_canvas//2 - alto_imagen_central],[0,0,1]]**

A continuación, se compone la parte izquierda del mosaico (es decir, una homografía entre la imagen izquierda y la central) y la parte derecha del mosaico (entre la imagen derecha y la central). Para ello se extraen los Keypoint y se establece un conjunto de puntos de correspondencias entre cada dos imágenes solapadas. Realizaremos estos pasos usando las funciones vistas en el ejercicio anterior (y la técnica de David Lowe), con la diferencia de que, una vez obtenidos los matches, debemos calcular cuáles han sido los keypoints emparejados para poder calcular la homografía posteriormente.

Para obtenerlos, simplemente nos basamos en la información que guardan los matches, los cuales son una lista de objetos **DMatch** formados por los parámetros:

*queryIdx*, refiriéndose a los Keypoints de la primera imagen, y *trainIdx*, refiréndose a los Kyepoints de la segunda imagen. Estos son los puntos que necesitamos.

El siguiente paso es establecer una ***homografía*** óptima entre ambas imágenes solpadas. Se utilizará la función *findHomography*[6] de **openCV**:

***(cv2.findHomography(srcPoints, dstPoints, method, ransacReprojThreshold))***

donde:

- **srcPoints** son los Keypoints obtenidos por la imagen fuente.
- **dstPoints** son los Keypoints obtenidos por la imagen destino.
- **method** es el método usado para calcular la matriz de la homografía, en nuestro caso será: RANSAC.
- **ransacReprojThreshold** es el error máximo de reproyección para tratar una pareja de puntos como un inlier. OpenCV aconseja que el valor se encuentre entre 0 y 10; aunque usaré el valor por defecto: 3. Tras haber probado con distintos valores de este parámetro y no haber notado diferencia, al tratarse de una imagen con mucho detalle considero que es preferible tener un valor de error máximo lo más pequeño posible.

Una vez calculadas las homografías entre cada dos imágenes, las multiplicamos con la homografía de traslación calculada para que las transformaciones se apilen correctamente en el resultado.Además, debemos aplicar una transformación de perspectiva a una imagen para añadirla al resultado. Para ello, haremos uso de la función *warpPerspective*[7] de **openCV**:

***(cv2.warpPerspective(src, M, dsize, dst, borderMode))***

donde:

- **src** es la imagen de entrada.
- **dst** es la imagen de salida donde se guardará el resultado. En nuestro caso será en el canvas.
- **M** es la matriz 3x3 de transformación, es decir, la homografía resultante de la composición de las imágenes.
- **dsize** es el tamaño de la imagen de salida. Como nuestra salida es el propio canvas, el tamaño será las dimensiones establecidas para dicho canvas.
- **borderMode** hace referencia al método de extrapolación de los píxeles, el cual establecemos como *BORDER_TRANSPARENT*.

A continuación, mostramos los resultados obtenidos:

Figura 24: Mosaico de 3 imágenes de la Alhambra.

Figura 25: Mosaico de 3 imágenes de la ciudad de Granada.

Como se puede observar, dadas tres imágenes consecutivas, se construye una composición bien hecha en la cual apenas se nota el lugar donde se cortan las imágenes. Además, como las fotografías parecen haber sido tomadas sin un cambio del ángulo (o giro) de la cámara notable, apenas se producen deformaciones en las imágenes de los laterales.

# 4. BONUS

## 4.1. BONUS B1

**Implementar el refinamiento de localización de puntos extremos en el espacio de escalas Laplaciano. Aplicarlo a los puntos encontrados en el punto.1 y mostrar que la implementación realmente mejora la estimación inicial usando imágenes de los entornos de los puntos.**

Una vez que hemos encontrado los extremos locales, se necesita calcular una ***corrección*** que permita realizar un ajuste detallado a los datos cercanos para la úbicación, escala, y otras características del keypoint que permita obtener su localización verdadera. Además esta información permite rechazar los puntos que tienen poco contraste (y que, por tanto, son sensibles al ruido) o que están mal localizados a lo largo de un borde.

Para la implementación del ejercicio, me baso en el artículo *Anatomy of the SIFT method* [1] donde, para cada extremo discreto $(s_e, m_e, n_e)$ obtenido en una escala $o_e$:

- Se calcula ${w^o}_{s,m,n}(\alpha)$, es decir, la ***interpolación cuadrática*** en un punto $(s, m, n)$ en una octava $o$, donde:

$$ {w^o}_{s,m,n}(\alpha) = {w^o}_{s,m,n} + \alpha^T {g^o}_{s,m,n} + \tfrac{1}{2}\alpha^T {H^o}_{s,m,n}\alpha $$

  Para ello, debemos de saber que $H$ representa la **Matriz Hessiana** y $g$ el **gradiente 3D, Jacobiana** de *(s,m,n)* en la octava $o$, calculados mediante las expresiones:

$$ {H^o}_{s,m,n} = \begin{bmatrix} \mathrm{h}_{11} & \mathrm{h}_{12} & \mathrm{h}_{13} \\ \mathrm{h}_{12} & \mathrm{h}_{22} & \mathrm{h}_{23} \\ \mathrm{h}_{13} & \mathrm{h}_{23} & \mathrm{h}_{33} \end{bmatrix}, {g^o}_{s,m,n} = \begin{bmatrix} (\mathrm{w}^o_{s+1,m,n} - {w^o}_{s-1,m,n})/2 \\ (\mathrm{w}^o_{s,m+1,n} - {w^o}_{s,m-1,n})/2 \\ (\mathrm{w}^o_{s,m,n+1} - {w^o}_{s,m,n-1})/2 \end{bmatrix} $$

  donde:

$$ \begin{aligned} h_{11} &= {w^o}_{s+1,m,n} + {w^o}_{s-1,m,n} - 2{w^o}_{s,m,n} \\ h_{22} &= {w^o}_{s,m+1,n} + {w^o}_{s,m-1,n} - 2{w^o}_{s,m,n} \\ h_{33} &= {w^o}_{s+1,m,n} + {w^o}_{s-1,m,n} - 2{w^o}_{s,m,n} \\ h_{12} &= ({w^o}_{s+1,m+1,n} - {w^o}_{s+1,m-1,n} - {w^o}_{s-1,m+1,n} + {w^o}_{s-1,m-1,n})/4 \\ h_{13} &= ({w^o}_{s+1,m,n+1} - {w^o}_{s+1,m,n-1} - {w^o}_{s-1,m,n+1} + {w^o}_{s-1,m,n-1})/4 \\ h_{22} &= ({w^o}_{s,m+1,n+1} - {w^o}_{s,m+1,n-1} - {w^o}_{s,m-1,n+1} + {w^o}_{s,m-1,n-1})/4 \end{aligned} $$

y que $\alpha$ denota $(\alpha_1, \alpha_2, \alpha_3)^T$ como el **desplazamiento** desde el centro del extremo 3D interpolado y que se puede calcular como:

$$\alpha^* = -(H^o{}_{s,m,n})^{-1} g^o{}_{s,m,n}$$

La Diferencia de Gaussianas tiene una fuerte respuesta a lo largo de los bordes, incluso si la ubicación a lo largo del borde está mal determinada; por lo que la hace ser inestable a pequeñas cantidades de ruido. Este cálculo con la matriz Hessiana permite eliminar estos puntos inestables, de manera que acabaremos con menos puntos pero de buena calidad.

- Una vez calculada dicha interpolación, se calculan las coordenadas reales sumándole a cada una el sesgo de la interpolación:

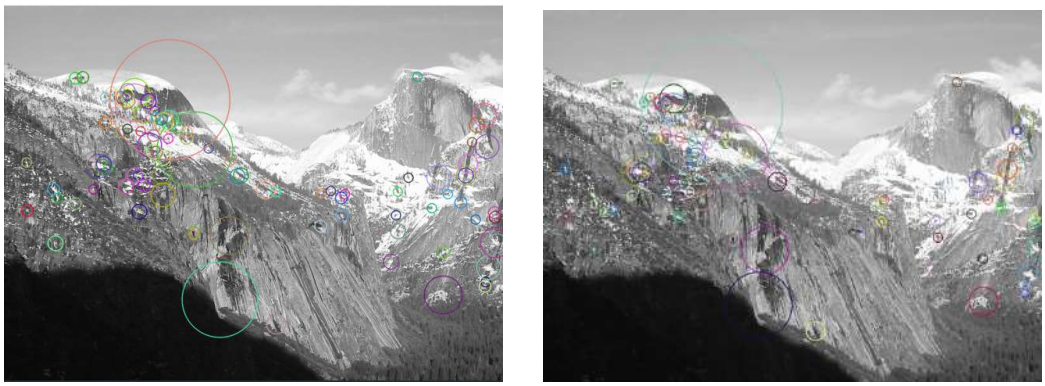
$$(\sigma, x, y) = (\tfrac{\delta_{oe}}{\delta_{min}} \sigma_{min} 2^{({\alpha_1}^* + s)/n_{spo}}, \delta_{oe}({\alpha_2}^* + m), \delta_{oe}({\alpha_3}^* + n))$$

- Además, actualizamos la posición interpolada sumándole a *(s,m,n)* el desplazamiento para obtener su valor discreto más cercano:

$$(s, m, n) = ([s + {\alpha_1}^*], [m + {\alpha_2}^*], [n + {\alpha_3}^*])$$

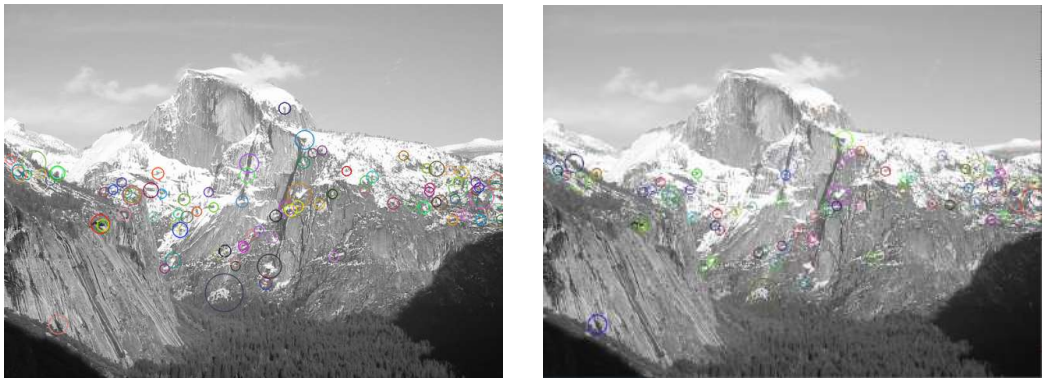
Este proceso se repite para cada punto hasta que el máximo de $({\alpha_1}^*, {\alpha_2}^*, {\alpha_3}^*)$ sea menor que 0.6 o durante cinco intentos fallidos. En realidad, el umbral teórico está establecido a 0.5, pero en la práctica se estima a 0.6 para evitar posibles inestabilidades numéricas. Si el máximo es menor a este umbral, dicho punto se acepta como extremo.

Una vez seguidos todos los pasos y utilizado las funciones del Ejercicio 1E para visualizar los Keypoints, mostramos los resultados:



(a) Puntos obtenidos originalmente. (b) Puntos refinados.

Figura 26: YOSEMITE 1 - Refinamiento de Keypoints.



(a) Puntos obtenidos originalmente. (b) Puntos refinados.

Figura 27: YOSEMITE 2 - Refinamiento de Keypoints.

Como se puede observar, podemos encontrar pequeñas diferencias al comparar los keypoints obtenidos en el ejercicio 1 con los refinados. La diferencia menos notoria, pero existente, es la localización de algunos de los puntos: al fijarnos con detenimiento podemos observar que algunos de ellos han sufrido un pequeño desplazamiento; lo cuál es lógico, no tendría sentido que la traslación de los puntos fuese muy grande, ya que la verdadera localización del extremo se encuentra cercana a este.Otra diferencia más notable es el cambio en la escala de algunos de ellos, de manera que parecen adaptarse con mayor precisión a la zona de interés detectada. También, se puede apreciar la desaparición de algunos de los puntos ya que, como se ha explicado anteriormente, este refinamiento permite la eliminación de puntos inestables con ruido.

## 4.2. BONUS B2 - Opción (A)

**Construir un panorama con proyección plana con todas las imágenes del fichero.**

Para este ejercicio, se propone realizar una extensión del Ejercicio 3 para generar un mosaico de N imágenes (en nuestro caso, serán 14 imágenes de las vistas a Granada). El procedimiento es prácticamente idéntico al del ejercicio anterior.

El tamaño del canvas es difícil de elegir en este caso, ya que se debe de tener en cuenta que las posibles deformaciones de las imágenes pueden aumentar el tamaño del canvas. Por eso, y como todas las imágenes tienen las mismas dimensiones, de alto se considerará algo más del doble de la altura de una de las imágenes; y de ancho se aplicará algo más de la suma de los anchos de todas las imágenes, ya que es preferible a que sobre canvas que obtener un mosaico cortado. El canvas resultante ha sido recortado mediante el uso de la función *boundingRect*[8] de **OpenCv**.

De nuevo, se separa la construcción la parte derecha del mosaico de la construcción de la parte izquierda, y se sigue la misma dinámica que en el mosaico para tres imágenes. Sin embargo, ahora se debe de tener en cuenta de que se debe de multiplicar la homografía calculada con todas las homografías hasta llegar a la foto central, además de realizar la multiplicación con la homografía de traslación. Para ello, se hace uso de los listas (una para cada parte del mosaico), las cuales irán alamacenando las multiplicaciones de las homografías iterativamente para multiplicar con la siguiente homografía calculada.

Los resultados obtenidos son:

Figura 28: Mosaico de 14 imágenes con el centro en la séptima imagen.

Aunque la construcción del mosaico sea correcta, se puede observar fácilmente que las proyecciones de las fotos del lado izquierdo no son paralelas a la cámara, por lo que los píxeles empiezan a abrirse y a generar ***distorsiones*** cada vez más notables. En este caso, se decidió tomar como centro la imagen central de la lista

que compone nuestras 14 imágenes, pero en vista a que el campo de visión es más amplio a la izquierda, es preferible elegir como imagen de referencia alguna que se encuentre más a la izquierda de la imagen central, de manera que la deformación sea la menor posible.

Figura 29: Mosaico de 14 imágenes con el centro en la sexta imagen.

Figura 30: Mosaico de 14 imágenes con el centro en la quinta imagen.

Se puede observar que al cambiar la imagen de referencia por una imagen situada en la parte izquierda de la lista de imágenes que compone el mosaico, la deformación se hace menor, hasta incluso llegar a equilibrarse (como ocurre al elegir como imagen de referencia la quinta imagen), obteniendo un acabado del canvas de mayor calidad.

## Referencias

[1] SIFT. *Anatomy of the SIFT method.*
https://www.ipol.im/pub/art/2014/82/?utm_source=doi

[2] Distinctive Keypoints. *Distinctive Image Features from Scale-Invariant Keypoints.*
https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[3] keypoint. *KeyPoint method specification in OpenCV doc.*
https://docs.opencv.org/3.4/d2/d29/classcv_1_1KeyPoint.html#a9d81b57ae182dcb3ceac86a6b0211e94

[4] drawKeypoints. *drawKeypoints method specification in OpenCV doc.*
https://docs.opencv.org/3.4/d4/d5d/group__features2d__draw.html#gab958f8900dd10f14316521c149a60433

[5] BFMatcher. *BFMatcher method specification in OpenCV doc.*
https://docs.opencv.org/4.5.3/d3/da1/classcv_1_1BFMatcher.html

[6] findHomography. *findHomography method specification.*
https://shimat.github.io/opencvsharp_docs/html/38333149-7bd7-5ddd-eadf-3401d0efab12.htm

[7] warpPerspective. *warpPerspective method specification in OpenCV doc.*
https://docs.opencv.org/4.x/da/d54/group__imgproc__transform.html#gaf73673a7e8e18ec6963e3774e6a94b87

[8] Bounding *OpenCV documentation - how to contour features*
https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html