

# Práctica 2 - Visión por Computador

## Detección de puntos relevantes y construcción de panoramas

David Gil Bautista A1

45925324M

### Ejercicio 1

Detección de puntos SIFT y SURF. Aplicar la detección de puntos SIFT y SURF sobre las imágenes, representar dichos puntos sobre las imágenes haciendo uso de la función drawKeyPoints. Presentar los resultados con las imágenes Yosemite.rar.

**(a) Variar los valores de umbral de la función de detección de puntos hasta obtener un conjunto numeroso ( $\geq 1000$ ) de puntos SIFT y SURF que sea representativo de la imagen. Justificar la elección de los parámetros en relación a la representatividad de los puntos obtenidos.**

Para realizar la detección con SIFT he probado con los siguientes parámetros para el umbral de contraste y de fronteras.

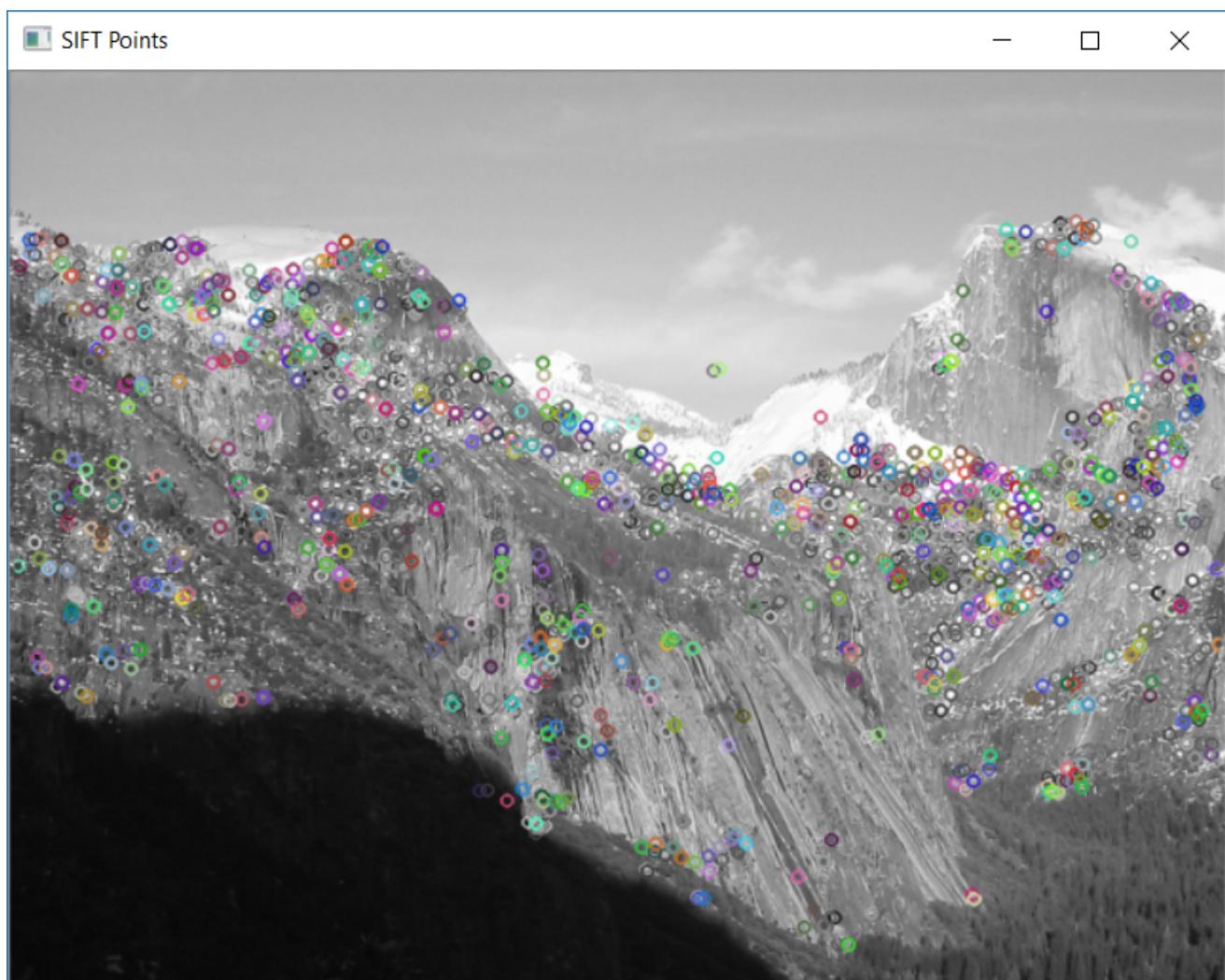
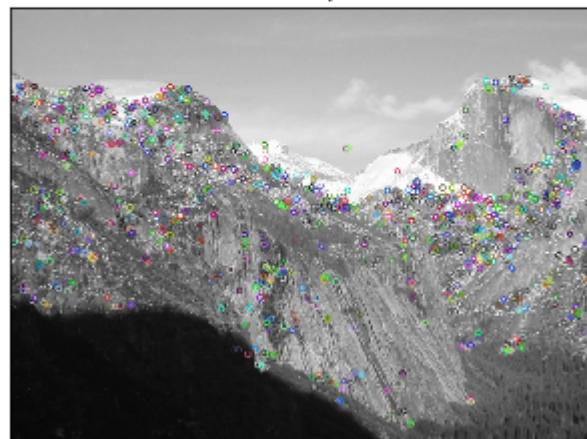
```
ct = [0.2, 0.15, 0.1, 0.08, 0.06, 0.04, 0.02]
```

```
et = [5, 10, 15, 20, 25, 30]
```

Mediante dos bucles for compruebo para cada par de umbrales el número de puntos obtenido y al llegar a un par que ofrezca más de 1000 puntos paramos de iterar. Comenzamos iterando por el umbral del contraste porque nos interesa que sea el más alto. Priorizando el umbral del contraste vamos a encontrar las zonas críticas de la imagen, no sólo los puntos más importantes en las fronteras.

Ejercicio 1 - Apartado a  
SIFT  
Contrast: 0.08  
Edge: 10

SIFT 1049 points



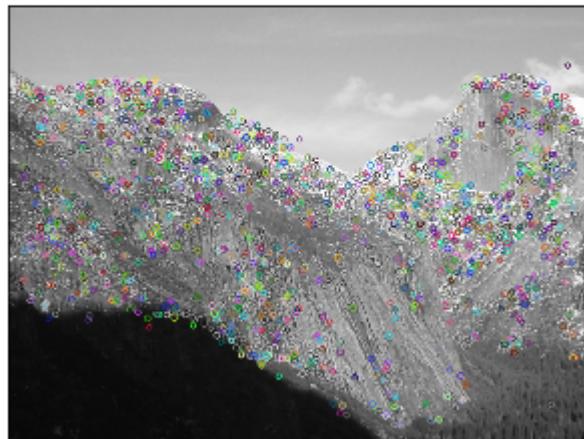
Para el caso del detector SURF he usado los siguientes parámetros para el umbral Hessiano.

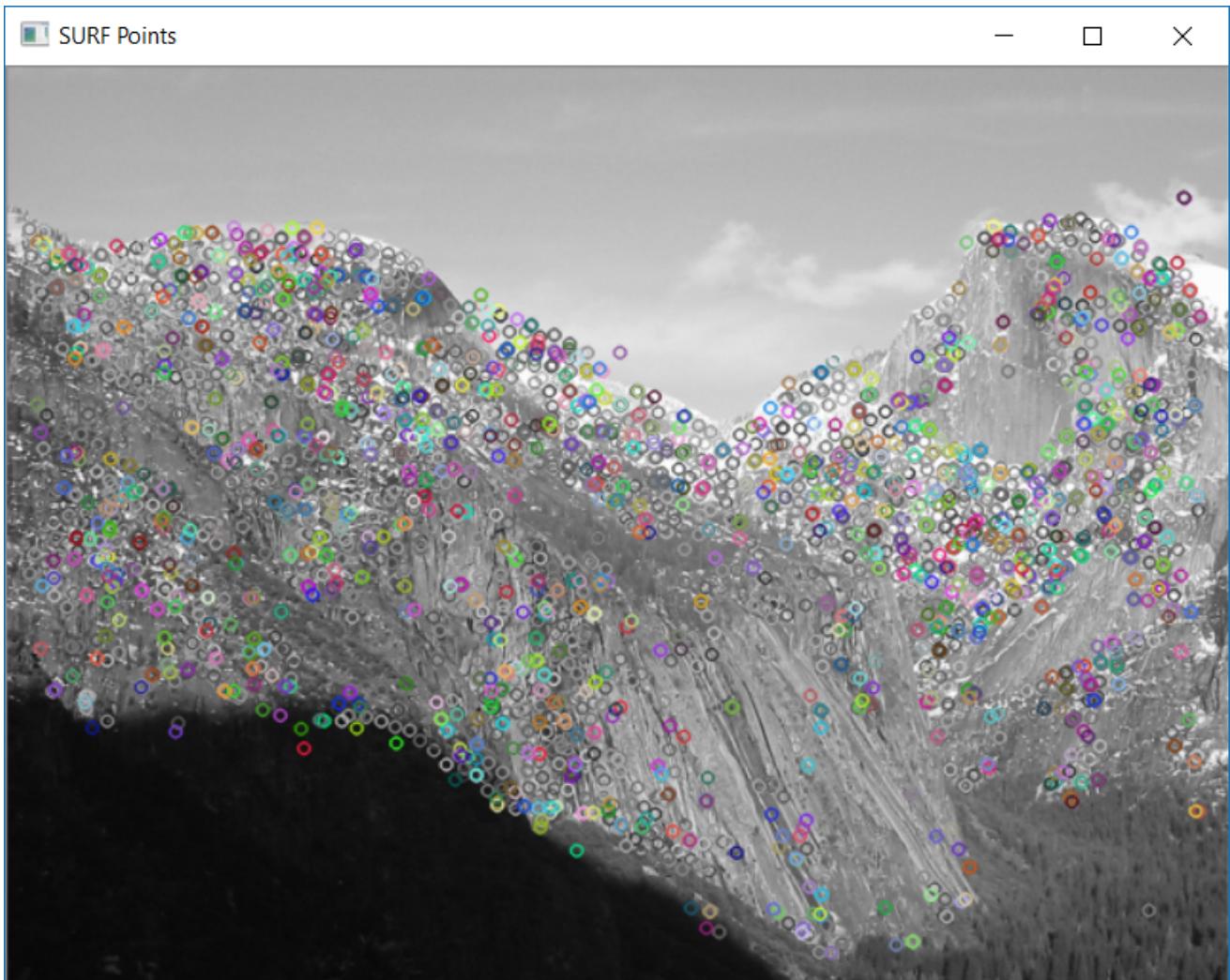
```
hess = [1000, 900, 800, 700, 600, 500, 400]
```

Al igual que en el caso anterior, itero con los distintos umbrales y obtengo un conjunto de puntos mayor que 1000 con un umbral de 600.

Ejercicio 1 - Apartado a  
SURF  
Hessian: 600

SURF 1034 points





En este caso podemos ver que al contrario que con el detector SIFT, los puntos aparecen más dispersos por toda la imagen.

**(b) Identificar cuántos puntos se han detectado dentro de cada octava. En el caso de SIFT, identificar también los puntos detectados en cada capa. Mostrar el resultado dibujando sobre la imagen original un círculo centrado en cada punto y de radio proporcional al valor de sigma usado para su detección (ver circle()) y pintar cada octava en un color.**

Usando el detector SIFT y con unos umbrales de **0.06** para el contraste y **5** para las fronteras obtenemos un conjunto de **1607** puntos.

Haciendo uso de KeyPoint.octave podemos obtener la información sobre la capa y la octava a la que pertenece cada KeyPoint. Usando Numpy hacemos el recuento y obtenemos el siguiente número de octavas y capas, y sus respectivos puntos.

### Ejercicio 1 - Apartado b

SIFT

Key points 1607

Octave -1, key points: 1220

Octave 0, key points: 286

Octave 1, key points: 68

Octave 2, key points: 26

Octave 3, key points: 3

Octave 4, key points: 3

Octave 5, key points: 1

Layer 1, key points: 798

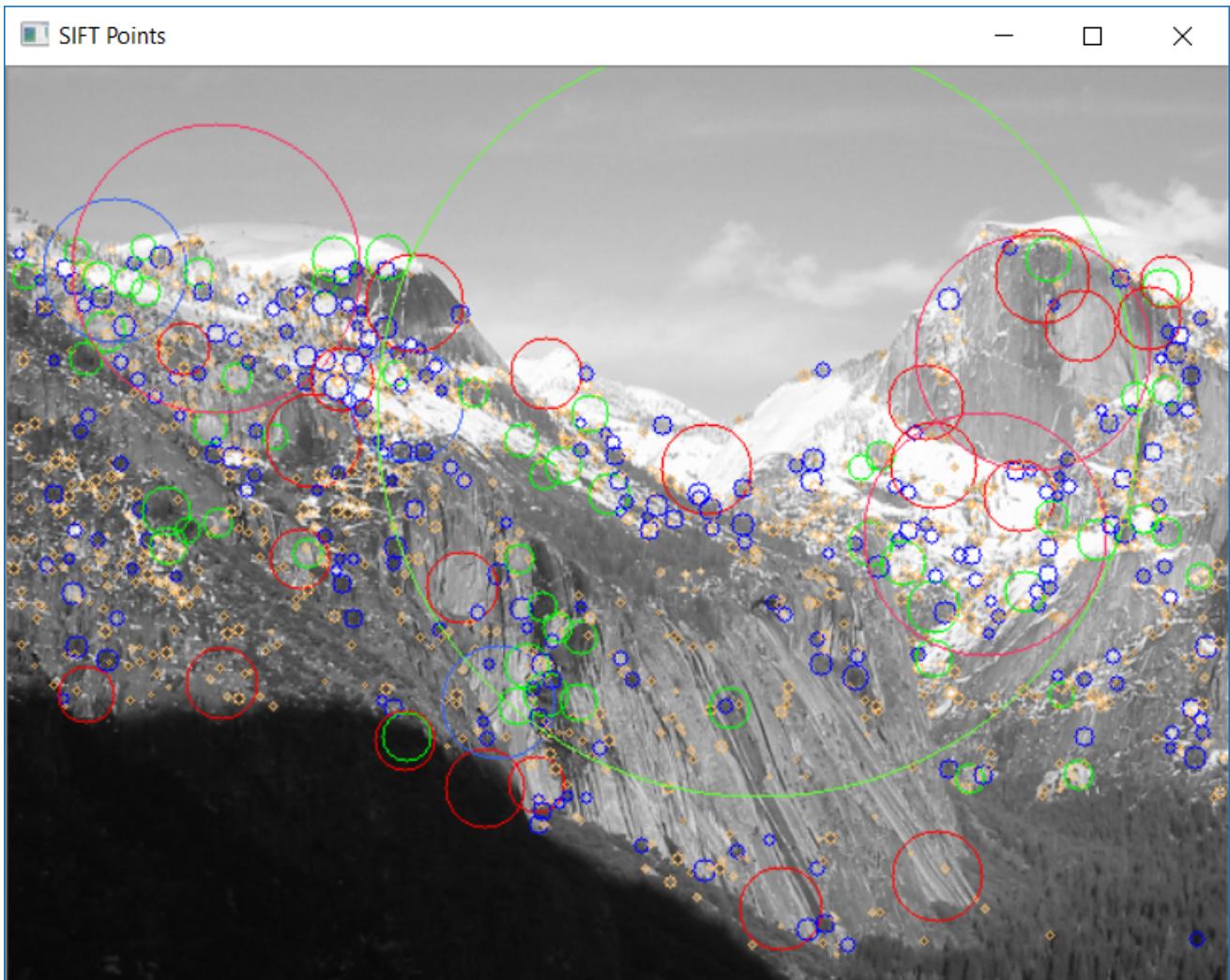
Layer 2, key points: 490

Layer 3, key points: 319

Usando SIFT vemos que las octavas no comienzan en 0, si no que hay una en -1. Dicha octava pertenece a una ampliación de la imagen que realiza el algoritmo, es por eso que la mayoría de los puntos pertenecen a esta octava.

Para el caso de las capas podemos ver que por defecto tenemos 3, aunque hay un parámetro que nos permite cambiar el número de capas que tenemos.

Para representar los puntos gráficamente hacemos uso de las variables ***pt*** que nos proporciona el centro del punto y ***size*** que nos da el tamaño. Depende de la octava a la que pertenezca lo pintamos de un color distinto y obtenemos lo siguiente.



En el caso de SURF no tenemos capas, y haciendo el mismo cálculo que para SIFT obtenemos el siguiente número de octavas y sus puntos.

**Ejercicio 1 - Apartado b**

SURF

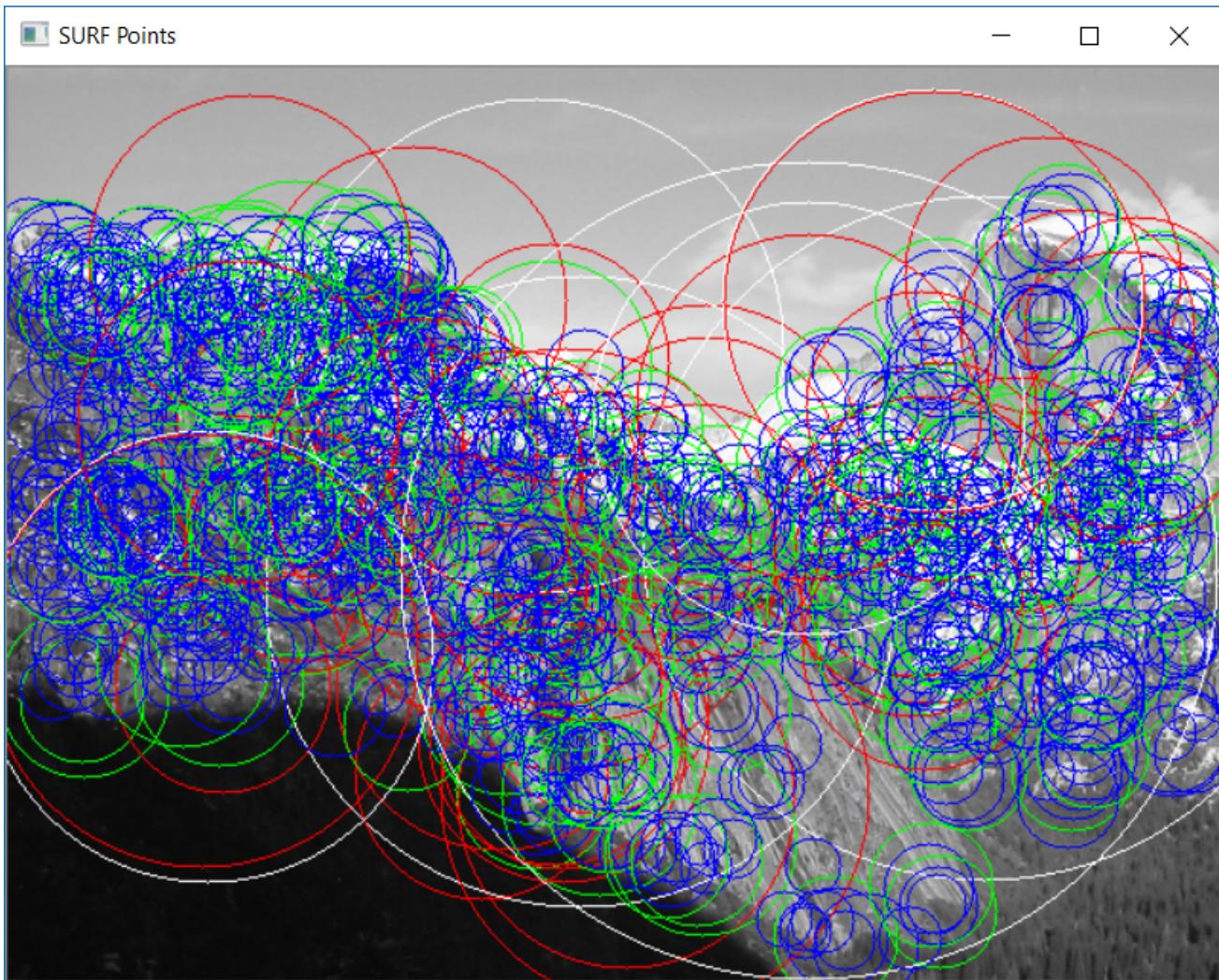
Key points 1040

Octave 0, key points: 813

Octave 1, key points: 177

Octave 2, key points: 43

Octave 3, key points: 7



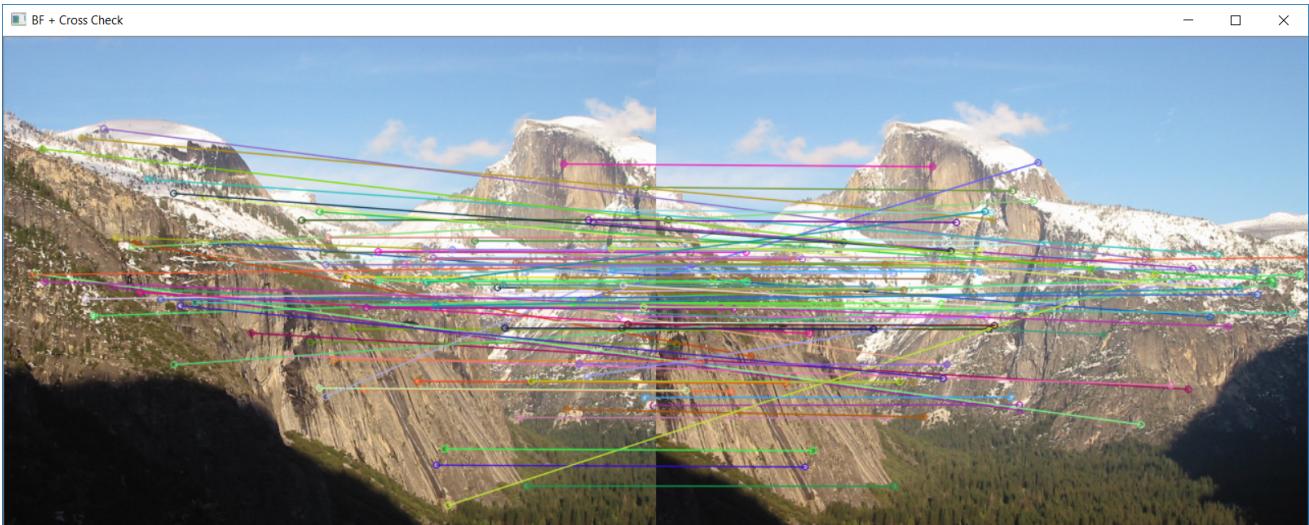
(c) Mostrar cómo con el vector de keyPoint extraídos se pueden calcular los descriptores SIFT y SURF asociados a cada punto usando OpenCV.

### Ejercicio 2

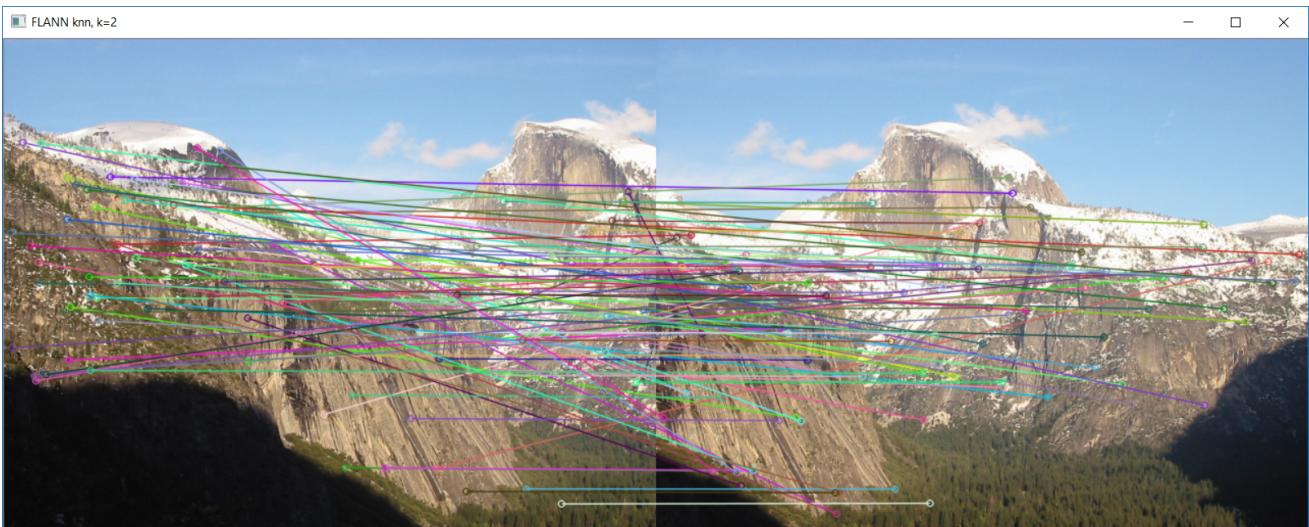
Usar el detector-descriptor SIFT de OpenCV sobre las imágenes de Yosemite.rar (cv2.xfeatures2d.SIFT create()). Extraer sus listas de keyPoints y descriptores asociados. Establecer las correspondencias existentes entre ellos usando el objeto BFMatcher de OpenCV y los criterios de correspondencias "BruteForce+crossCheck" y "Lowe-Average2NN". (NOTA: Si se usan los resultados propios del puntos anterior en lugar del cálculo de SIFT de OpenCV se añaden 0.5 puntos)

(a) 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 100 elegidas aleatoriamente.

Tras extraer los KeyPoint y descriptores de cada imagen usamos **BFMatcher con correlación** para calcular los puntos en correspondencia. Desordenamos la lista de vectores y mostramos los 100 primeros.



Al igual que con BruteForce+crossCheck hacemos el mismo cálculo con los keyPoint y descriptores pero esta vez usando **Flann con k=2**. Volvemos a desordenar la lista de vectores y mostramos los 100 primeros.



**(b) Valorar la calidad de los resultados obtenidos en términos de las correspondencias válidas observadas por inspección ocular y las tendencias de las líneas dibujadas.**

Para ambos algoritmos podemos apreciar un resultado parecido. Los vectores tienden a ser horizontales y paralelos aunque podemos encontrar algunos vectores diagonales que no representan bien el mismo punto en las dos imágenes.

**(c) Comparar ambas técnicas de correspondencias en términos de la calidad de sus correspondencias (suponer 100 aleatorias e inspección visual).**

El primer algoritmo, BFMatcher, es un algoritmo de fuerza bruta, lo que nos garantiza que va a encontrar la solución óptima aunque puede que tarde más. Flann, en cambio, es un algoritmo rápido que garantiza encontrar una buena solución, no la mejor, en un tiempo más reducido.

Al haber representado una muestra aleatoria de todo el conjunto de vectores no podemos garantizar a simple vista que BFMatcher sea mejor. En ambos encontramos falsos positivos.

### Ejercicio 3

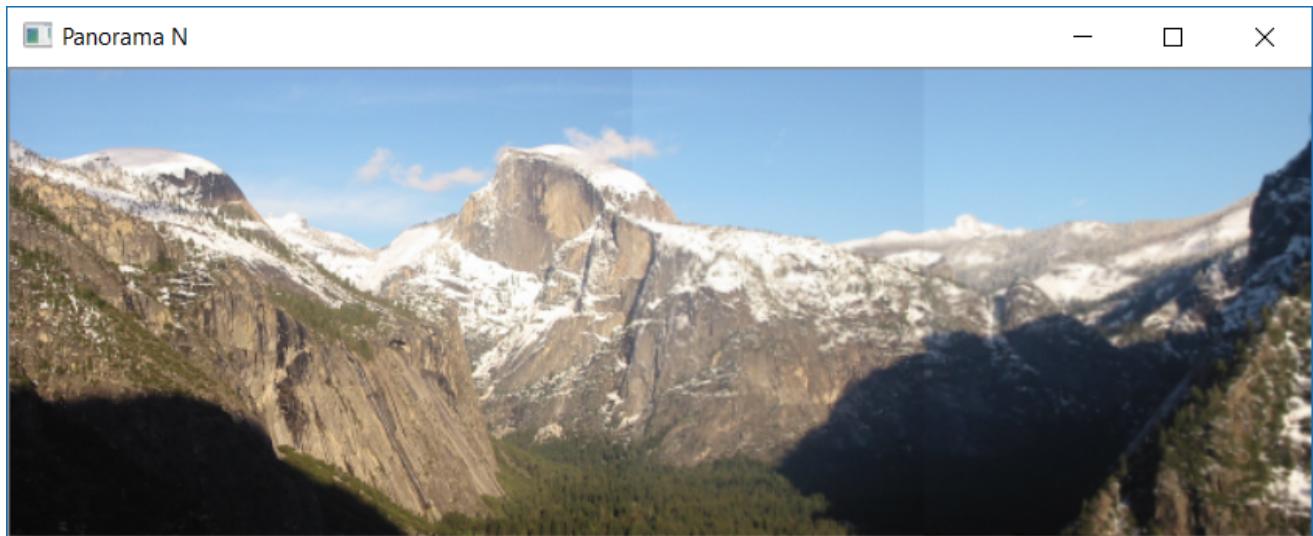
Escribir una función que genere un mosaico de calidad a partir de  $N = 3$  imágenes relacionadas por homografías, sus listas de keyPoints calculados de acuerdo al punto anterior y las correspondencias encontradas entre dichas listas. Estimar las homografías entre ellas usando la función cv2.findHomography(p1,p2, CV RANSAC,1). Para el mosaico será necesario.

**(a) Definir una imagen en la que pintaremos el mosaico.**

**(b) Definir la homografía que lleva cada una de las imágenes a la imagen del mosaico.**

**(c) Usar la función cv2.warpPerspective() para trasladar cada imagen al mosaico (Ayuda: Usar el flag BORDER TRANSPARENT de warpPerspective).**

Al igual que en el ejercicio anterior, hacemos uso de SIFT y BFMatcher para extraer los keyPoints de ambas imágenes y sus correspondencias entre ellos. Ordenamos los vectores por importancia, aquellos cuya distancia sea menor. Con la equivalencia de los puntos calculamos la homografía entre las imágenes usando el método de RANSAC. Usamos la homografía para juntar las imágenes con warpPerspective y mejoramos la imagen copiando la izquierda sobre el canvas creado y eliminamos el espacio en negro que se nos queda al desplazar la imagen de la derecha y hacerla coincidir con la de la izquierda.



#### Ejercicio 4

Lo mismo que en el punto anterior pero para  $N > 5$  (usar las imágenes para mosaico).



Para este ejercicio y el anterior he realizado un método iterativo que comienza a realizar el mosaico desde el centro de la imagen, ya que haciéndolo desde los extremos provocaba demasiada distorsión.

En caso de que tengamos dos imágenes realizamos el mosaico de estas dos, si son 3, realizaremos el de la primera imagen con la segunda, el de la segunda con la tercera y para acabar realizaremos el mosaico de los mosaicos que hemos creado previamente. Si tenemos más de 3 imágenes realizaremos el mosaico para los elementos de la derecha e izquierda partiendo desde  $N/2$ , siendo  $N$  el número de imágenes.