EMPAREJAMIENTO DE CARACTERÍSTICAS SIMILARES USANDO EL MÉTODO ORB

Antes de comenzar si quieres saber cómo funciona el método ORB y entender mejor las primeras líneas del código te recomiendo que visites el siguiente repositorio https://github.com/Enock13/DETECTION-OF-KEYPOINTS-AND-DESCRIPTORS-WITH-ORB.git.

En ese repositorio encontraras la explicación paso a paso de la función del método ORB, así como la identificación de los puntos clave y descriptores.

Una vez entendiendo lo anterior podemos pasar a la parte del emparejamiento de las características similares, para este código usaremos las librerías de NumPy y OpenCV, una vez importadas estas librerías inicializaremos nuestra captura de video con la función .videoCapture() y también leeremos nuestra imagen de referencia, previamente guardada en la carpeta del código, con la función .imread().

```
import numpy as np
import cv2

Cap= cv2.VideoCapture(0)
Img = cv2.imread("BOOK_IMAGE.jpg")
```

Ahora crearemos nuestro ciclo while que contendrá la parte más importante del código para después leer nuestros frames obtenidos de la captura de video a través de la función .read().

```
while True:
    Rec, Frame = Cap.read()
```

Una vez leídos los frames pasaremos a transformar a escala de grises, tanto los frames como la imagen de referencia, esto mediante la función .cvtColor() a la cual le daremos como parámetros el nombre de la variable a convertir, así como el modo de conversión el cual sería de BGR (escala de colores que usa OpenCV por defecto) a GRAY.

```
Gray_Frames=cv2.cvtColor(Frame, cv2.COLOR_BGR2GRAY)
Gray_Img=cv2.cvtColor(Img, cv2.COLOR_BGR2GRAY)
```

Antes de iniciar el método ORB, le daremos nuestra cantidad de puntos deseada y una vez teniendo esto podemos inicializar el método con la función .ORB_create() y como parámetro la variable donde se almacenan la cantidad de puntos.

```
#NUMBERS OF POINTS TO DETECT
Num_pt=500
#START THE FUNCTION .ORB_create(NUMBERS OF POINTS)
Orb = cv2.ORB_create(Num_pt)
```

Ahora pasaremos a almacenar nuestros puntos clave y descriptores con la función .detectAndCompute() y como parámetro la variable designada a la escala de grises que convertimos anteriormente, esto se hará tanto para los frames como para la imagen de referencia.

```
#DETECT THE POINTS WITH FUNCTION .detectAndComputer(Img, None )
Keypoints1, Descriptor1= Orb.detectAndCompute(Gray_Frames,None)
Keypoints2, Descriptor2= Orb.detectAndCompute(Gray_Img,None)
```

Si deseamos conocer los valores de los descriptores solo tendríamos que imprimirlos y así obtendríamos los valores.

```
print(Descriptor1)
print(Descriptor2)
```

Para mostrar los puntos claves tenemos que dibujar primero esos puntos, para eso usaremos la función .drawKeypoints() y como parámetros los siguientes elementos:

- Imagen en la cual se mostrarán los puntos.
- Puntos clave detectados.
- Imagen de salida.
- Color de los puntos.
- Bandera que configura la característica de dibujo.

```
#DRAW THE POINT DETECTED
Frames_Display= cv2.drawKeypoints(Frame,Keypoints1,outImage=np.array([]),
color=(255,0,0),flags=cv2.DRAW_MATCHES_FLAGS_DEFAULT)
Img_Display= cv2.drawKeypoints(Img,Keypoints2,outImage=np.array([]), color=(
255,0,0),flags=cv2.DRAW_MATCHES_FLAGS_DEFAULT)
```

Para poder emparejar las características es necesario crear el objeto con la función .BFMatcher() esta función necesita de dos parámetros, que son:

- normType, en este especificamos la medición de distancia a usar, como nosotros usamos el método ORB, usaremos la medición .NORM_HAMMING.
- croosCheck, este parámetro booleano evalúa, si es True, que los puntos coincidan en ambas imágenes.

```
#MATCH THE POINT
#FIRST CREATE A OBJECT COMPARE OF DESCRIPTORS
Match=cv2.BFMatcher(cv2.NORM_HAMMING,crossCheck=True)
```

Una vez teniendo el objeto creado, vamos a usar la variable donde almacenamos el objeto con sus parámetros para usarlos en los descriptores de la imagen de referencia y la captura de video, esto mediante la función .match() y como parámetros los descriptores antes mencionados, ya que estos tienen las coordenadas de cada punto característico que serán usadas para lograr el emparejamiento.

```
Matches=Match.match(Descriptor1,Descriptor2)
```

Ahora, ese emparejamiento que hemos creado lo vamos a ordenar, esto se hará con la función sorted() de Python, lo que hace esta función es ordenar los emparejamientos de acuerdo a él orden que le daremos como parámetro. Primero le daremos la variable que queremos ordenar, en este caso será Matches, ya que esta variable es la que contine el emparejamiento de los descriptores, veremos que el parámetro key tiene una igualdad, esta igualdad es dada por "lambda x: x.distance", lo que hace esta igualdad es tomar la lista dada por la variable Matches y ordenarla en función de la distancia, a menor distancia tendrá mejor

emparejamiento. Por último, el parámetro booleano reverse que será False, si lo ponemos en True ordenara la lista de mayor a menor distancia.

```
#NOW WE ORDER THE LIST OF MATCHES WITH THE FUNCTION sorted()
Matches=sorted(Matches, key=lambda x: x.distance, reverse=False)
```

Teniendo ya el orden vamos a filtrar esos emparejamientos que tenemos hasta ahora, esto será mediate el valor entero que se obtendrá del 50% del total de emparejamientos.

Después este valor entero lo almacenaremos de nuevo en la variable Matches, dándole como valor la lista de esta filtración que acabamos de realizar. Finalmente, la variable Matches tendrá la mitad de los emparejamientos realizados esto con el fin de obtener los mejores emparejamientos y eliminar los emparejamientos con distancias no recomendadas.

```
#FILTER MATCHES
Good_Matches=int(len(Matches)*0.5)
Matches=Matches[:Good_Matches]
```

La siguiente parte es opcional, pero si quisiéramos conocer los valores de los emparejamientos, podríamos crear un ciclo for en el cual obtengamos tres valores los cuales son:

- query_idx: es el índice que tiene el descriptor en el conjunto de características de la imagen de consulta.
- train_idx: es el índice que tiene el descriptor en el conjunto de características de la imagen de entrenamiento.
- distance: es la distancia entre los descriptores emparejados.

```
for Mat in Matches:
    query_idx = Mat.queryIdx
    train_idx = Mat.trainIdx
    distance = Mat.distance
    print(f"queryIdx: {query_idx}, trainIdx: {train_idx}, distance: {distance}")
```

Ya teniendo todo esto hecho y siguiendo con el código, tendríamos que dibujar estos emparejamientos encontrados, esto mediante la función .drawMatches() al cual le daremos como parámetros lo siguiente:

- Variable de los frames.
- Variable de los puntos clave de los frames.
- Variable de la imagen de referencia.
- Variable de los puntos clave de la imagen de referencia.
- Variable que contiene los emparejamientos.
- La imagen de salida (en este caso ninguna).

```
#DRAW THE MATCHES
Img_Matches = cv2.drawMatches(Frame,Keypoints1,Img,Keypoints2,Matches,None)
```

Con los emparejamientos dibujados, podemos mostrarlos con la función .imshow() y como parámetros el nombre que tendrá la ventana y la variable de lo que mostraremos.

```
#SHOW THE MATCHES IN SCREEN
cv2.imshow("MATCHES",Img_Matches)
#SHOW THE POINTS IN SCREEN
cv2.imshow("IMAGE",Img_Display)
```

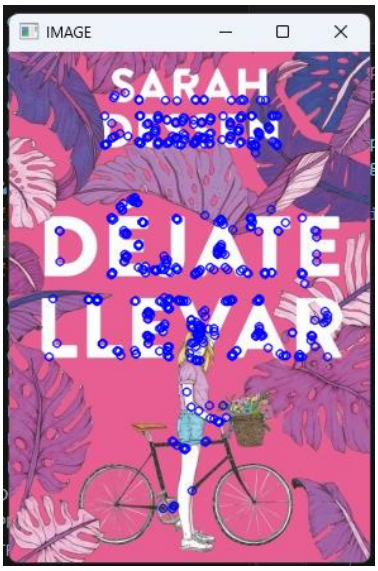
Ya para terminar nuestro código, crearemos el comando de salida con la función .waitKey() y la tecla “ESC” de nuestro teclado que corresponde al número 27 en el sistema ASCII, también borraremos nuestra captura de video con la función .release() y nuestras ventanas con .destroyAllWindwos() para finalizar el programa.

```
    t = cv2.waitKey(1)
    if t == 27:
        break


Cap.release()
cv2.destroyAllWindows()
```

RESULTADOS

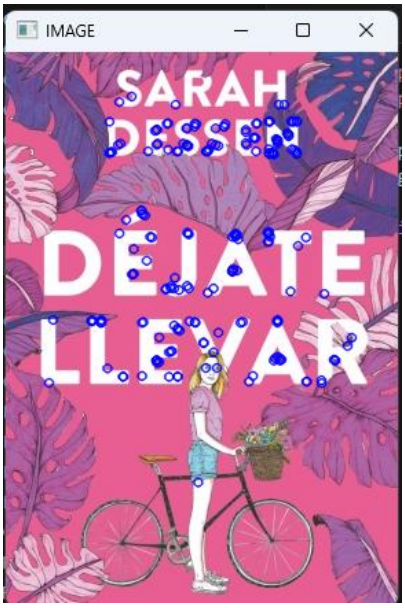
A continuación, se verán los resultados obtenidos en este código, primero con una cantidad de 500 puntos y después la misma prueba, pero con una cantidad de 200 puntos a detectar, al igual se mostrarán los valores del emparejamiento final.

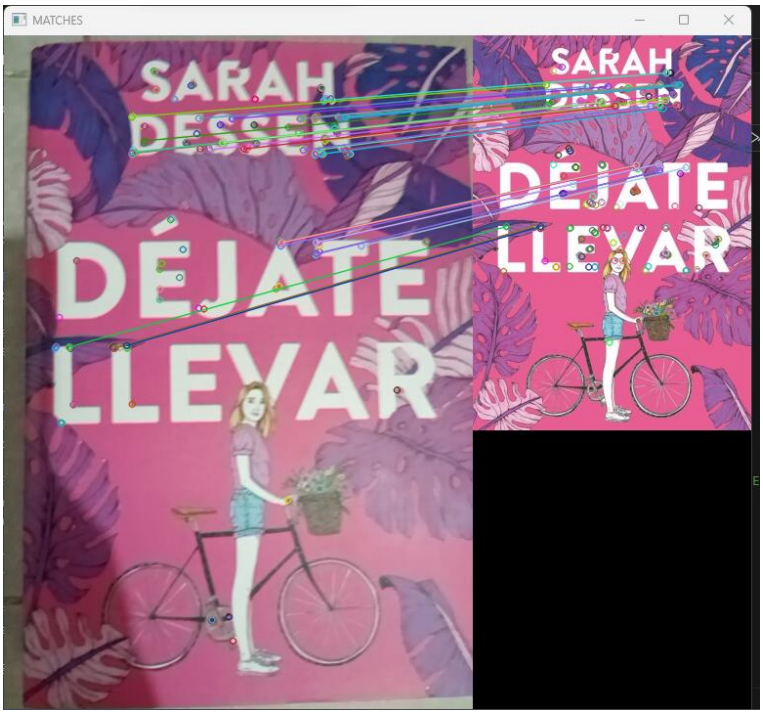


Del lado derecho podemos ver los puntos clave de la imagen y de lado izquierdo el emparejamiento, ahora veremos los resultados de los valores de este emparejamiento, veremos que en la parte de distancia están de menor a mayor esto indica que la función sorted si funciona correctamente.

```
queryIdx: 302, trainIdx: 135, distance: 18.0
queryIdx: 480, trainIdx: 366, distance: 18.0
queryIdx: 411, trainIdx: 232, distance: 19.0
queryIdx: 219, trainIdx: 62, distance: 20.0
queryIdx: 234, trainIdx: 12, distance: 20.0
queryIdx: 307, trainIdx: 113, distance: 20.0
queryIdx: 360, trainIdx: 161, distance: 20.0
queryIdx: 424, trainIdx: 284, distance: 20.0
queryIdx: 306, trainIdx: 56, distance: 21.0
queryIdx: 421, trainIdx: 294, distance: 21.0
queryIdx: 448, trainIdx: 353, distance: 21.0
queryIdx: 284, trainIdx: 48, distance: 22.0
queryIdx: 323, trainIdx: 65, distance: 22.0
queryIdx: 287, trainIdx: 94, distance: 23.0
queryIdx: 180, trainIdx: 261, distance: 24.0
queryIdx: 341, trainIdx: 255, distance: 24.0
queryIdx: 347, trainIdx: 215, distance: 24.0
queryIdx: 439, trainIdx: 345, distance: 24.0
queryIdx: 259, trainIdx: 102, distance: 25.0
queryIdx: 337, trainIdx: 140, distance: 25.0
queryIdx: 298, trainIdx: 26, distance: 26.0
queryIdx: 303, trainIdx: 136, distance: 26.0
queryIdx: 351, trainIdx: 218, distance: 26.0
queryIdx: 366, trainIdx: 222, distance: 26.0
queryIdx: 494, trainIdx: 376, distance: 26.0
```

Ahora la misma prueba, pero con 200 puntos y con esto tendríamos comprobado la eficiencia del código.






```
queryIdx: 191, trainIdx: 145, distance: 15.0
queryIdx: 179, trainIdx: 115, distance: 19.0
queryIdx: 140, trainIdx: 46, distance: 21.0
queryIdx: 162, trainIdx: 98, distance: 21.0
queryIdx: 163, trainIdx: 116, distance: 21.0
queryIdx: 120, trainIdx: 2, distance: 22.0
queryIdx: 138, trainIdx: 59, distance: 22.0
queryIdx: 164, trainIdx: 102, distance: 22.0
queryIdx: 112, trainIdx: 17, distance: 23.0
queryIdx: 172, trainIdx: 111, distance: 24.0
queryIdx: 141, trainIdx: 51, distance: 25.0
queryIdx: 84, trainIdx: 8, distance: 26.0
queryIdx: 93, trainIdx: 23, distance: 29.0
queryIdx: 142, trainIdx: 44, distance: 29.0
queryIdx: 196, trainIdx: 52, distance: 29.0
queryIdx: 122, trainIdx: 10, distance: 30.0
queryIdx: 115, trainIdx: 49, distance: 32.0
queryIdx: 111, trainIdx: 53, distance: 35.0
queryIdx: 118, trainIdx: 56, distance: 35.0
queryIdx: 180, trainIdx: 121, distance: 35.0
queryIdx: 153, trainIdx: 40, distance: 36.0
queryIdx: 155, trainIdx: 80, distance: 36.0
queryIdx: 86, trainIdx: 24, distance: 37.0
queryIdx: 174, trainIdx: 136, distance: 37.0
queryIdx: 88, trainIdx: 19, distance: 39.0
```

Si llegaste hasta esta parte te agradezco el tiempo que me regalaste y espero te haya ayudado con tus dudas o inquietudes de aprender más, esto solo es un pequeño fragmento de explicación si quieres seguir aprendiendo más puedes visitar las siguientes páginas que te ayudaran a seguir aprendiendo y mejorando en el tema de la visión artificial y Python.

1.- https://omes-va.com/

2.- https://www.pythonpool.com/

3.- https://hetpro-store.com/TUTORIALES/

4.- https://learnopencv.com/

Lo principal es la sabiduría; adquiere sabiduría, Y con todo lo que obtengas adquiere inteligencia. Proverbios 4:7