Ejercicios Entrega 4

SIFT

Añade al ejercicio CLASIFICADOR un método basado en el número de coincidencias de keypoints SIFT. Utilízalo para reconocer objetos con bastante textura (p. ej. carátulas de CD, portadas de libros, cuadros de pintores, etc.).

Para este ejercicio, en primer lugar he implementado el método SIFT con las funciones `precompute()` y `compare()`.

Para ello, he modificado el fichero `sift.py` del directorio `code/SIFT` de modo que para la función `precompute()` obtuviera todos los keypoints del modelo que le paso como parámetro, que como se explicó anteriormente, esta función se ejecuta para todos los modelos de el directorio pasada como argumento.

Posteriormente, he implementado la función `compare()`, la cual dada el frame y los keypoints de un modelo, devuelve cuantos de ellos son comunes, de modo que cuando se ejecute el compare para cada uno de los modelos, podamos obtener el que mayor coincidencia tiene con el frame actual.

He de decir también que he creado el directorio images en la ruta `Entregas/Entrega3/src/images` para que al ejecutar se guarden los modelos nuevos allí.

De modo que la ejecución ahora se haría desde el directorio `Entrega3/src/` de la siguiente manera:

`python main.py --models=./images --method=sift --save`

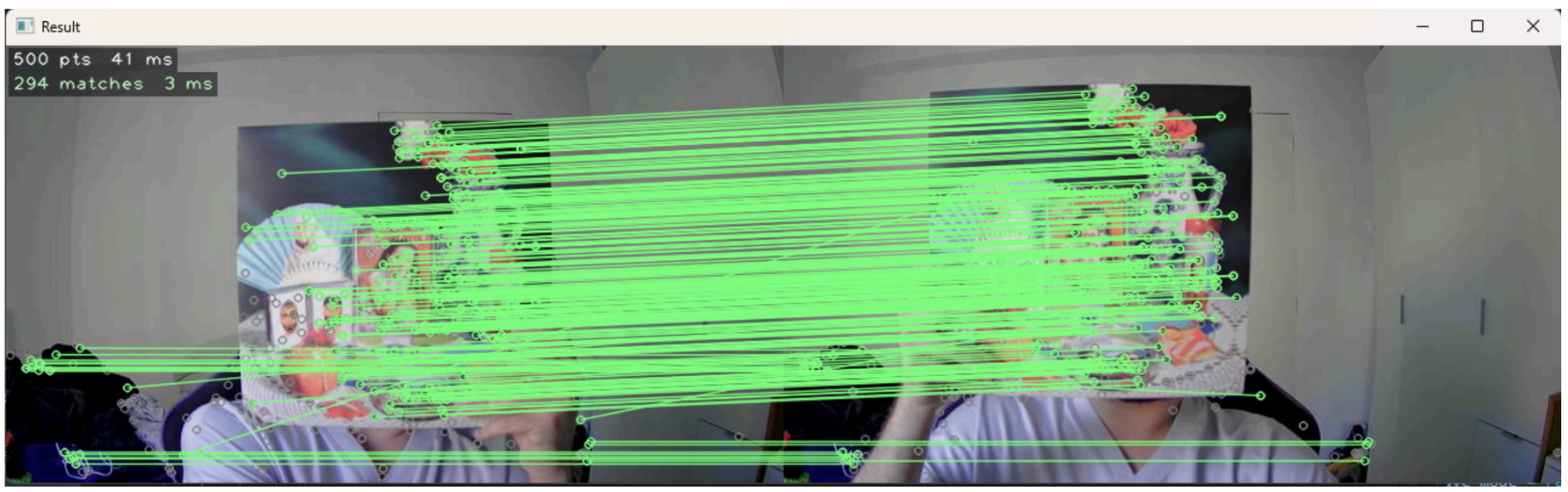
Hasta aquí todo nos es familiar puesto que es muy similar a la entrega anterior. Sin embargo, ahora en el main hay que tratar de representar esas conincidencias de manera visual.

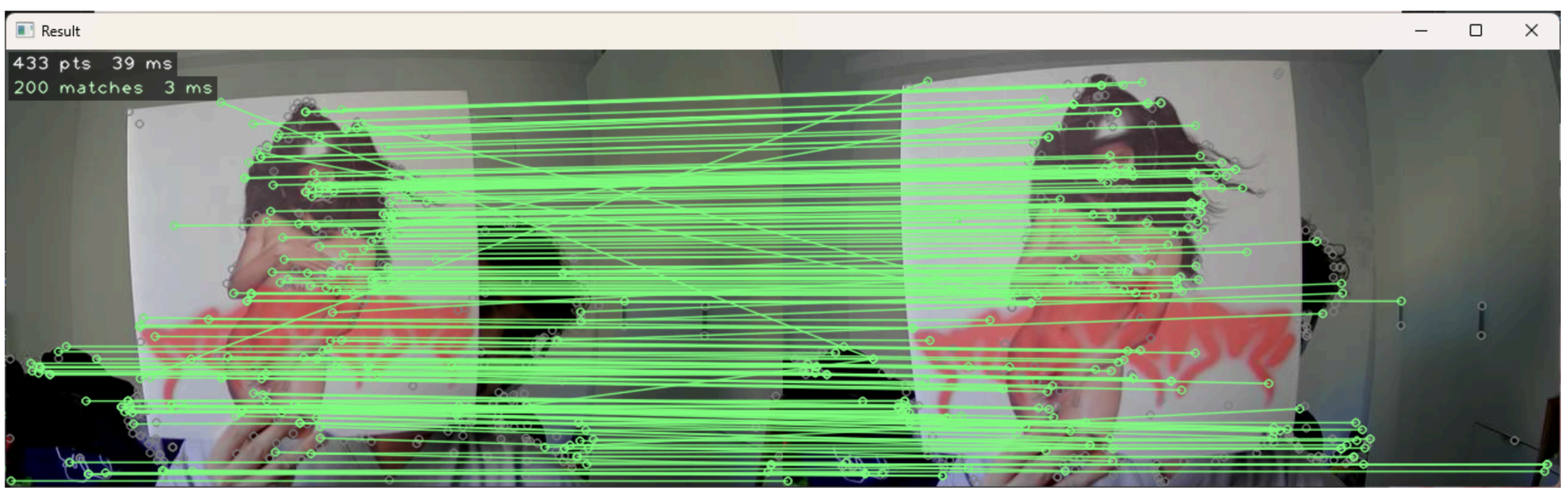
Para ello en el fichero `main.py` he modificado la función `main()` de modo que si el método actual es el sift, una vez comprobado cual es el modelo con mejor coincidencia, calcule de nuevo los keypoints del frame actual y los de dicho modelo para poder representarlos en el frame.

Esto se consigue de la siguiente manera:

In [ ]:
```python
def main():

    # ...
    # Hasta aquí el codigo no cambia

    # Iniciar captura de video
    for key, frame in autoStream():

        # ...
        # Hasta aquí el codigo no cambia

        if args.method != "sift":
            # Mostramos el frame de la misma manera que se hace en la entrega 3
        else:
            sift = cv2.SIFT_create(nfeatures=500)
            mejor_imagen=models[best_match[0]]['image']
            t0 = time.time()
            keypoints , descriptors = sift.detectAndCompute(result_frame, mask=None)
            t1 = time.time()
            putText(result_frame, f'{len(keypoints)} pts  {1000*(t1-t0):.0f} ms')

            k0,d0 = sift.detectAndCompute(mejor_imagen, mask=None)
            x0=mejor_imagen
            t2 = time.time()
            matches = cv2.BFMatcher().knnMatch(descriptors, d0, k=2)
            t3 = time.time()
            # Aplicar el "ratio test" para filtrar coincidencias
            good = []
            for m in matches:
                if len(m) >= 2:
                    best, second = m
                    if best.distance < 0.75 * second.distance:
                        good.append(best)

            if len(good) > 0:
                imgm = cv2.drawMatches(result_frame, keypoints, x0, k0, good,
                                       flags=0,
                                       matchColor=(128,255,128),
                                       singlePointColor=(128,128,128),
                                       outImg=None)
            else:
                print("No hay coincidencias buenas para dibujar.")
                imgm = result_frame.copy()
            putText(imgm ,f'{len(good)} matches  {1000*(t3-t2):.0f} ms',
                     orig=(5,36), color=(200,255,200))
            cv2.imshow('Result', imgm)
```

El resultado de todo ello es lo que se observa en las siguientes imágenes:





Donde, como se puede apreciar se calculan los keypoints para el frame actual y se asocian con los keypoints de cada una de las imágenes gracias a los descriptores de ambos frames, calculados gracias a la función `detectAndCompute()` y asociados mediante la función `knnMatch()` del matcher de OpenCV `cv2.BFMatcher()`