E3- Registro

Instrucciones

En respuesta a las preguntas que se plantean a continuación, se debe entregar en la plataforma de enseñanza virtual (por separado, **no** en un archivo comprimido)

- un archivo con extensión .ipynb creado con notebook Jupyter de Anaconda que contenga, tanto las respuestas teóricas, como el código programado;
- el archivo anterior en .pdf (File->Print preview->guardar como pdf);
- una declaración de autoría firmada por el alumno/a según la plantilla proporcionada;
- las imágenes/vídeos usados.

La resolución de todos los ejercicios (tanto teóricos como prácticos) debe ser **razonada** y el código desarrollado debe ser **explicado en detalle, justificando todos los parámetros** escogidos y **referenciando las fuentes.**

Objetivo

El objetivo de este entregable es realizar algunos ejercicios de registro o alineación de imágenes para formar una imagen panorámica.

Ejercicio [1.5 puntos]

En este ejercicio vamos a calcular una correspondencia entre dos imágenes tomadas de una misma escena. Para ello:

1. Toma dos imágenes de un mismo paisaje desde distintas perspectivas, entre las cuales exista parte común (una a continuación de la otra, como se muestra en las imágenes de ejemplo “paisaje1.jpg” y “paisaje2.jpg”).
2. El siguiente fragmento de código (modificado a partir de https://docs.opencv.org/3.4/d5/dde/tutorial_feature_description.html) utiliza SIFT (cv.SIFT_create()) para calcular puntos de interés en el par de imágenes “paisaje1.jpg” y “paisaje2.jpg”. Ejecuta el código para calcular los puntos de interés en las imágenes que has tomado y responde a las siguientes cuestiones:

```
img1 = cv2.imread("paisaje1.jpg", cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread("paisaje2.jpg", cv2.IMREAD_GRAYSCALE)
if img1 is None or img2 is None:
        print('No se pudieron abrir las imágenes')
```

```
        exit(0)

    sift = cv2.SIFT_create()
    keypoints1, descriptors1 = sift.detectAndCompute(img1, None)
    keypoints2, descriptors2 = sift.detectAndCompute(img2, None)
```

a) ¿Cúantos puntos característicos ha seleccionado SIFT al aplicarlo sobre cada una de las imágenes que has tomado?
b) ¿Qué información, almacenada en las variables *descriptors1* y *descriptors2*, devuelve el método SIFT una vez ejecutado?
c) Modifica el código para que los descriptores sean calculados usando ORB y compara el resultado con el obtenido por SIFT.
d) Dibuja utilizando la función *cv.drawKeypoints* los puntos obtenidos por cada uno de los dos métodos.

3. La función cv.DescriptorMatcher_create de OpenCV permite instanciar un método de correspondencia para emparejar puntos de interés.

    a) Aplica el siguiente fragmento de código para calcular dicho emparejamiento entre los puntos característicos obtenidos por SIFT en el apartado anterior. Observa que la función "match" encuentra la mejor correspondencia en "descriptors2" para cada uno de los descriptores en "descriptors1".

    ```
    matcher = cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_BRUTEFORCE)
    matches = matcher.match(descriptors1, descriptors2)
    ```

    b) ¿Qué distancia se ha utilizado por defecto? Razona si se trata o no de una distancia adecuada.
    c) Testea otras posibles distancias de entre las que oferta OpenCV para matching, y utiliza la función *cv.drawMatches* para dibujar los emparejamientos obtenidos y comparar los resultados.
    d) Modifica el código anterior para realizar un emparejamiento de los puntos característicos obtenidos en el apartado anterior mediante ORB, y comenta qué distancia has decidido usar en este caso y por qué.

4. OpenCV proporciona una serie de métodos para determinar correspondencias. Un tutorial sobre el uso de estos métodos puede encontrarse en: https://docs.opencv.org/4.9.0/dc/dc3/tutorial_py_matcher.html

   Incluye la librería FLANN (Fast Library for Approximate Nearest Neighbour), que es una librería para realizar búsquedas rápidas aproximadas de vecinos más cercanos en espacios de alta dimensión (más información: https://www.cs.ubc.ca/research/flann/ y https://github.com/flann-lib/flann)

   a) Modifica el código del apartado anterior para utilizar estos métodos optimizados, tanto en el caso de descriptores obtenidos con ORB como en el de los descriptores obtenidos por SIFT.
   b) Utiliza la función *cv.drawMatchesKnn* para visualizar los emparejamientos obtenidos por ambos métodos y comenta los resultados comparándolos con los obtenidos en el apartado anterior.

5. El siguiente fragmento de código permite seleccionar únicamente aquellos emparejamientos en los que el NDDR test supere un determinado umbral, y los utiliza para calcular una homografía que permita crear una imagen panorámica a partir de las imágenes de las cuales estos emparejamientos fueron calculados. Selecciona la correspondencia (o matching) que consideres oportuno de entre las obtenidas en los apartados anteriores, y utilízala para ejecutar el siguiente código en el cual debes:
   a) Seleccionar un umbral (**threshold**) adecuado para que el resultado permita realizar una composición panorámica correcta entre las dos imágenes.
   b) Modificar el parámetro **method** de la función cv.findHomography para que utilice tanto el algoritmo RANSAC como el método de mínimos cuadrados.
   c) Comentar los umbrales utilizados en cada uno de los casos así como las diferencias obtenidas en el resultado.

```
good_matches = []
  for m1,m2 in matches:
     if m1.distance < threshold*m2.distance:
        good_matches.append(m1)


   if len(good_matches) > 4:
      src_pts = np.float32([ keypoints1[good_match.queryIdx].pt for good_match
in good_matches ]).reshape(-1,1,2)
      dst_pts = np.float32([ keypoints2[good_match.trainIdx].pt for good_match
in good_matches ]).reshape(-1,1,2)
```

```
        M, mask = cv2.findHomography(src_pts, dst_pts, method, 5.0)
        result = warpImages(img2, img1, M)
        cv2.imshow('Stitched output', result)
        cv.waitKey()
        if cv.waitKey(0) & 0xff == 27: #ESC para salir
            cv.destroyAllWindows()
    else:
        print("No hay suficientes correspondencias entre las dos imágenes")
        print("Se encontraron sólo %d. Se necesitan al menos %d." %
(len(good_matches), min_match_count))
```

Nota: El código anterior requiere cargar la siguiente función para su funcionamiento:

```
# Warp img2 to img1 using the homography matrix H
def warpImages(img1, img2, H):
    rows1, cols1 = img1.shape[:2]
    rows2, cols2 = img2.shape[:2]
    list_of_points_1 = np.float32([[0,0], [0,rows1], [cols1,rows1],
[cols1,0]]).reshape(-1,1,2)
    temp_points = np.float32([[0,0], [0,rows2], [cols2,rows2], [cols2,0]]).reshape(-
1,1,2)
    list_of_points_2 = cv2.perspectiveTransform(temp_points, H)
    list_of_points = np.concatenate((list_of_points_1, list_of_points_2), axis=0)
    [x_min, y_min] = np.int32(list_of_points.min(axis=0).ravel() - 0.5)
    [x_max, y_max] = np.int32(list_of_points.max(axis=0).ravel() + 0.5)
    translation_dist = [-x_min, -y_min]
    H_translation = np.array([[1, 0, translation_dist[0]], [0, 1, translation_dist[1]],
[0,0,1]])
    output_img = cv2.warpPerspective(img2, H_translation.dot(H), (x_max-x_min,
y_max-y_min))
    output_img[translation_dist[1]:rows1+translation_dist[1],
translation_dist[0]:cols1+translation_dist[0]] = img1
    return output_img
```