Práctica obligatoria 1

Detección de objetos

El siguiente enunciado corresponde a la primera práctica puntuable de la asignatura.

1 Normas

Las prácticas se realizarán en grupos de 2 alumnos y se presentará usando el aula virtual.

La fecha límite de presentación es el día del examen, pero si se presenta antes del 31 de Marzo a las 23:00 la nota de esta práctica se incrementará en un 10%.

Para presentarla se deberá entregar un único fichero ZIP que contendrá el código fuente y un fichero PDF con la descripción del sistema desarrollado. Dicho fichero PDF incluirá una explicación con el algoritmo desarrollado, los métodos de OpenCv utilizados, copias de las pantallas correspondientes a la ejecución del programa y estadísticas correspondientes al resultado de la ejecución del programa sobre la muestra de test.

La puntuación de esta práctica corresponde al 35% de la asignatura. En particular:

- El ejercicio 1 corresponde al 25%
- El ejercicio 2 al 5%
- El ejercicio 3 al 5%

La práctica deberá ejecutarse sobre Python 3.6 y OpenCV 3.4. y consistirá en 3 ficheros python:

- deteccion_orb.py
- detección_haar.py
- deteccion_video.py.

Para ejecutar la práctica deberá escribirse en la consola de comandos “python” seguido del nombre del fichero sin ningún otro parámetro adicional. Se supondrá que los directorios test, train y haar están en el mismo directorio que los ficheros python. Al ejecutar estos ficheros python se mostrará por pantalla el resultado sobre cada una de las imágenes o vídeos de test.

El código desarrollado en las prácticas debe de ser original. La copia (total o parcial) de prácticas será sancionada, al menos, con el SUSPENSO global de la asignatura en la convocatoria correspondiente. En estos casos, además, no regirá la liberación de partes de la asignatura (habrá que volver a presentarse al examen) y podrá significar, en la siguiente convocatoria y a discreción del profesor, el tener que **resolver nuevas pruebas y la defensa de las mismas de forma oral**. **Las sanciones derivadas de la copia, afectarán tanto al alumno que copia como al alumno copiado.**

Para evitar que **cuando se usa código de terceros** sea considerado una copia, se debe **citar siempre la procedencia** de cada parte de código no desarrollada por el propio alumno (con comentarios en el propio código y con mención expresa en la memoria de las prácticas). El plagio o copia de terceros (p.ej. una página web) ya sea en el código a desarrollar en las prácticas y/o de parte de la memoria de las prácticas, sin la cita correspondiente, acarrearán las mismas sanciones que en la copia prácticas de otros alumnos.

2 Detección de coches mediante puntos de interés

Se desea construir un sistema que permita la detección de la posición de un coche dentro de una imagen. Las imágenes de coches se encontrarán en forma frontal. Junto con este enunciado se suministra un directorio (training) con 48 imágenes de coches en los que se ha recortado la parte más importante que está libre de reflejos para que se use como muestra de aprendizaje (ver figura 1a). Además se suministra otro directorio (testing) con imágenes de 33 coches para que se usen como muestra de prueba (ver figura 1b). Sobre este segundo grupo de imágenes se realizarán las estadísticas de funcionamiento del sistema.

(a) (b)

Figura 1.- Ejemplo de las imágenes de aprendizaje (a) y de prueba (b).

2.1 Desarrollo de la práctica

Para construir el sistema de detección de coches se deberá utilizar un enfoque basado en la detección de puntos de interés (Harris, FAST, SIFT, etc.), seguido de una descripción de los mismos (BRIEF, SIFT, etc.) y votación a la Hough utilizando los descriptores tal y como se vió en clase. En este caso se recomienda seguir los siguientes pasos:

1. En primer lugar hará falta programar un bucle que cargue las imágenes de training.
    1. La carga debería realizarse en niveles de gris. Esto se consigue poniendo a 0 el segundo parámetro del comando cv2.imread.
    2. Mientras que se construye el sistema, por disponer de un ejemplo simplificado fácil de depurar, se recomienda que en el bucle solo se carguen 2 de las imágenes de training. Luego, cuando todo el sistema funcione, se cargarán las 48 imágenes que corresponden al conjunto completo de training

2. Utilizar la clase cv2.ORB_create (ver referencia [1] al final del enunciado) para obtener los puntos de interés (keypoints) y los descriptores de cada imagen.

    1. Al constructor de ORB se le pueden pasar varios parámetros. Los más importantes son: el número de keypoints a detectar, el número de pasos de la pirámide de imágenes a diferentes resoluciones y el factor de escala entre los diferentes escalones de la pirámide.

    2. Para desarrollar el sistema se pueden poner sólo 3 keypoints y sólo 1 nivel para la pirámide. De esta forma, se podrá depurar el sistema fácilmente.

    3. Cuando ya se tenga la certeza de que todo el sistema funciona, se puede aumentar el número de keypoints a un valor más realista (por ejemplo 100 keypoints, una pirámide de 4 niveles y un factor de escala de 1.3 entre cada paso).

    4. La función cv2.drawKeypoints permite mostrar los keypoints detectados sobre la propia imagen. El flag cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS permite mostrar algunos detalles de los keypoints de ORB.

Luego se creará una estructura de datos indexe todos los descriptores de puntos detectados sobre la muestra de aprendizaje. Estos puntos se entenderán etiquetados como pertenecientes a coche:

1. En el caso de ORB, para construir el índice se puede utilizar la clase cv2.FlannBasedMatcher. Como tendremos muchos descriptores almacenados, para acelerar la búsqueda se recomienda utilizar LSH (ver el uso de cv2.FlannBasedMatcher en la referencia [2] al final del enunciado). Además, como los descriptores son códigos binarios se debe usar la distancia de Hamming. Otra opción es usar el cv2.BFMatcher con distancia de Hamming. En este caso BF significa “Brute Force” (Fuerza Bruta) y permite buscar el descriptor más cercano a un conjunto de descriptores calculando todas las distancias y quedándose con el de menor distancia.

2. El método detectAndCompute de la clase ORB, devuelve los keypoints y los descriptores en forma de matriz de bytes, con tantas filas como keypoints y 32 columnas (32 x 8 bits = 256 bits). Para construir un índice de búsqueda con un objeto de clase FlannBasedMatcher hay que acumular en los descriptores correspondientes a todas las imágenes mediante el método add (la entrada es una lista de matrices con descriptores).

3. Además, se deberá guardar la información de cada keypoint, así como un vector que partiendo del keypoint indique en que punto concreto de la imagen se podría convenir que está el coche: **el vector de votación.**

Al proceso de detección de puntos de interés, extracción de descriptores, extracción de vectores de votación y construcción del índice de búsqueda por descriptor (usando el cv2.FlannBasedMatcher) **le llamaremos entrenamiento de un detector de objetos** basado en votación a la Hough con puntos de interés.

Una vez realizado el entrenamiento, lo que se querrá es utilizarlo para procesar una imagen y encontrar el centro del coche. De nuevo se buscarán los puntos de interés en la imagen de entrada y se calcularán sus descriptores. Para cada punto de interés, P, detectado en la imagen de entrada se realizará una búsqueda de los puntos de interés de entrenamiento más cercanos en descriptor (y por tanto de apariencia semejante) a P. Los descriptores de los puntos de interés del entrenamiento los tenemos guardados en un objeto de la clase cv2.FlannBasedMatcher y podremos usar su método knnSearch para

buscar los k más cercanos en descriptor a P. El punto de interés P, emitirá k votos a la Hough utilizando los **k vectores de votación** (no sólo el del más cercano). Una vez procesados todos los votos de los puntos de interés en la imagen de entrada, las coordenadas de la imagen con el máximo número de votos indicará la localización más probable del coche.

A continuación se enumeran los pasos a realizar para la detección:

1. Para realizar la búsqueda de un descriptor, encontrado en la imagen de test, sobre el objeto cv2.FlannBasedMatcher creado con las imágenes de training, se debe utilizar el método knnSearch.

2. El parámetro k indica que se devolverán los k descriptores más parecidos. Durante el desarrollo se puede utilizar un k bajo (por ejemplo 2). Luego, cuando el sistema esté en funcionado se deberá utilizar un parámetro más realista (5 o 6).

3. Como knnSearch devuelve los números de índice de los descriptores más parecidos, es fácil localizar los keypoints asociados. Con ellos se puede obtener el vector que concreta dónde está el coche. Con las votaciones correspondientes a los diferentes keypoints hallados se realiza una votación a la Hough que resultará en la posición del coche.

    1. Para que la acumulación funcione hay que tener en cuenta la escala (el “size” del keypoint) de P y la escala de P' (punto de entrenamiento). Si el vector de votación de P' se construyó con un keypoint de size=2 y, por contra, el punto de interés P, se ha detectado con escala 3 (size=3) entonces el vector de votación se debe de multiplicar por 3/2 antes de emitir el voto.

    2. Para que la acumulación sea evidente suele reducirse la resolución de la votación. Por ejemplo, si la imagen es de 240x480 píxeles, dividiendo por 10 las dimensiones del acumulador a 24x48 celdas. En la práctica esto significa que tendremos una incertidumbre de 5 píxeles (10/2) en la posición del coche en cada coordenada.

3 Detección de coches usando cv2.CascadeClassifier

Se desea construir un sistema basado en el detector de P.Viola y M. Jones (cv2.CascadeClassifier en OpenCV) que permita la detección de la posición de un coche dentro de una imagen. Las imágenes de coches se encontrarán en forma frontal, y se utilizarán las imágenes de coches del directorio test (ver figura 1b).

Para la realización de esta práctica se suministra un fichero (coches.xml) que corresponde con un clasificador de coche/no-coche ya entrenado. Para utilizar este clasificador en la detección del frontal del coche se debe de:

- Crear un objeto de tipo cv2.CascadeClassifier.

- Utilizar el método cv2.detectMultiScale para obtener los rectángulos donde el sistema detecta coches.

4 Detección del coche en secuencias de vídeo

Desarrollar un sistema que utilice los detectores desarrollados en la sección 1 y 2 sobre las secuencias de vídeo suministradas (el alumno puede añadir otras secuencias de prueba). Para la lectura de vídeo se recomienda utilizar el objeto de clase cv2.VideoCapture de OpenCV.

5 Referencias

1. Detector de Puntos de Interés ORB y Descriptor BRIEF con orientación:

   https://docs.opencv.org/3.1.0/d1/d89/tutorial_py_orb.html

2. Emparejamiento de descriptores en OpenCV sobre Python:

   https://docs.opencv.org/3.3.0/dc/dc3/tutorial_py_matcher.html