Procesado de imagen y visión por computador

Programación de aplicaciones con Python y OpenCV (II)

1. Trabajar con vídeos

La librería OpenCV incluye funciones para trabajar con vídeos de distintos formatos, como por ejemplo los formatos MPEG4, MJPG o XVID. Dichas funciones permiten, entre otras cosas, extraer fotogramas del vídeo y trabajar con ellos como si fueran imágenes independientes. Para poder trabajar con un vídeo se debe emplear la clase *VideoCapture* y sus correspondientes métodos o funciones miembro:

- **`cv2.VideoCapture(filename)` → `Objeto para capturar.`** Abre el vídeo especificado por el nombre del archivo *filename* y devuelve un objeto que permite acceder a él.
- **`cv2.VideoCapture.read()`→ `retval, image.`** Devuelve dos argumentos: *retval* e *image*. En la variable *image* se extrae el fotograma actual del vídeo. Una posterior llamada a la misma función extraería la siguiente imagen y así sucesivamente hasta el final del vídeo. Devuelve como La variable *retval* indica si la operación de lectura ha sido correcta (True). Por tanto, a partir del valor de esta variable es fácil chequear el fin del vídeo.

En algún caso, el objeto puede no ser capaz de inicializar la captura. Para comprobarlo se puede utilizar el método **cv2.VideoCapture.isOpened()**, que devuelve True si la inicialización se ha realizado correctamente. El archivo *apartado1.py* incluye el código que permite abrir y mostrar un vídeo por pantalla. La variable *time* calcula el tiempo de visualización de la imagen en pantalla a partir de la lectura del número de frames por segundo (fps) del vídeo (si, por ejemplo, la velocidad del vídeo es de 25 imágenes por segundo, la variable *time* es igual a 1000 ms/25fps=40 ms).

En cualquier caso, se puede acceder a algunas características del video utilizando el método **cv2.VideoCapture.get(propId)**, donde *propId* es un número identificador con valor entre 0 y 18. Cada número denota una propiedad del vídeo[1]. Por ejemplo los identificadores 3 y 4 devuelven, respectivamente, el ancho y alto de los frames y el 5 la velocidad del vídeo.

Ejecutar el programa, abriendo un vídeo de los proporcionados en la carpeta Videos, y comprobar que dicho vídeo se muestra por pantalla a la velocidad correcta. Posteriormente, modificar el código anterior para que el vídeo muestre, superpuesto a la secuencia original, el nombre completo del alumno. Para poder imprimir texto sobre una imagen, se puede emplear la función *cv2.PutText.* Para definir el color del texto, puede emplearse una tupla (B,G,R)[2]. Para probarlo ponga el texto en verde.

Guardar el código fuente del programa anterior incluyendo la modificación para imprimir sobre la imagen el nombre del alumno.

1 https://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html#videocapture-get

2 Como ayuda: http://docs.opencv.org/master/dc/da5/tutorial_py_drawing_functions.html#gsc.tab=0

Procesado de imagen y visión por computador

Programación de aplicaciones con Python y OpenCV (II)

1a.- Guardar un vídeo:

Para guardar un vídeo con OpenCV se empleará la clase *VideoWriter*. Dicha clase incluye, entre otros, los siguientes métodos:

- **`cv2.VideoWriter([filename, fourcc, fps, frameSize[, isColor]])`** : Inicializa la creación del vídeo especificado. Si el archivo ya existe, se sobreescribirá. Admite los siguientes parámetros de entrada:
  - *filename*: nombre del archivo de vídeo.
  - *fourcc*: código de 4 caracteres que indica el codec usado para la compresión. Por ejemplo: CV_FOURCC(`'P','I','M','1'`) es un codec MPEG-1.
  - fps: velocidad de frames por segundo.
  - FrameSize: tamaño de los frames del vídeo.
  - IsColor: si no es cero, el codificador esperará frames de color. En caso contrario, trabajará con frames en escala ded gris (este flag solo es soportado en Windows).
- **`cv2.VideoWriter.write(img)`:** Añade una nueva imagen al vídeo.

Al definir el archivo tenemos que tener en cuenta varios detalles. En primer lugar debemos especificar el codec y el formato del vídeo. Esto se especifica con el código *FOURCC*, que es un conjunto de cuatro caracteres que identifican el codec y formato a utilizar. En la web www.fourcc.org pueden consultarse todos los códigos disponibles. Para construir el código a partir de los 4 caracteres, podemos utilizar:

- `fourcc = cv2.VideoWriter_fourcc(*'MJPG')`

En el ejemplo se especifica el formato MJPEG. Bastará con incluir, como segundo argumento del constructor la variable `fourcc` creada, por ejemplo:

- `out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))`

En la llamada al constructor `fourcc` debemos pasar correctamente las dimensiones de los fotogramas del vídeo. El tamaño de los fotogramas en el método `cv2.VideoWriter.write` debe ser el mismo que el que pasemos como argumento al crear el objeto *VideoWriter*. Si el tamaño no coincide, el vídeo no se creará. Así mismo, todas las imágenes que añadamos deben ser en color (formato BGR).

Modificar el código del apartado anterior para que se guarde una copia del vídeo original (ver código de salvar un vídeo de la página web de la documentación de OpenCV)[3], pero con la modificación indicada en dicho apartado, es decir, superponiendo el nombre del alumno. Opcionalmente, el alumno puede también superponer un logo (por ejemplo: el de la Universidad de Alcalá u OpenCV) en cada imagen del vídeo. Como formato, emplearemos el formato MJPEG. Si el vídeo no se genera, habrá que revisar todos los detalles mencionados anteriormente, ya que si no se ajustan todos los parámetros a su valor correcto, el codec no funciona correctamente. Guardar el vídeo con el nombre *videoNombreAlumno.avi*.

3 Ayuda: http://docs.opencv.org/3.4.3/dd/d43/tutorial_py_video_display.html

2. Trabajar con cámaras de vídeo

OpenCV permite trabajar con los fotogramas en tiempo real capturados por una cámara de vídeo y no ofrece ninguna diferencia significativa con respecto a trabajar con vídeos, excepto que algunas de las propiedades del vídeo no se pueden leer, por ejemplo, la velocidad del vídeo no es accesible y dará un error si se intenta leer. Para abrir una cámara, emplearemos, igual que se hizo con los vídeos en el apartado anterior, la clase *VideoCapture*. La única diferencia estriba en que, a la hora de abrir el vídeo, en lugar de especificar el nombre del archivo que contiene el vídeo, habrá que especificar el dispositivo (cámara de vídeo) a emplear, utilizando la función:

- `cap=cv2.VideoCapture(device)`: abre el dispositivo especificado.

donde *device* es el identificador de la cámara. Por ejemplo, cuando trabajamos con cámaras *USB*, y sólo tenemos una única cámara, el dispositivo sería el 0. Si tuviéramos dos cámaras, una de ellas sería la 0, y la siguiente la 1, y así sucesivamente.

Utilice como base el código de la página web de la documentación de OpenCV[3] para abrir, capturar y mostrar por pantalla la escena visualizada por la propia cámara del alumno. Deberá modificar el código para que guarde la última imagen visualizada al salir de la aplicación al presionar la tecla 'q'. Opcionalmente, intente visualizar simultáneamente la imagen captada por la cámara y las imágenes de los tres canales de color R,G,B.

3. Detector de movimiento (Opcional)

Una de las muchas aplicaciones del procesado de imagen es la detección de movimiento a partir de cámaras estáticas. Por ejemplo, para contar las personas que entran a un recinto o los vehículos que pasan por un punto de peaje. En ambos casos debemos detectar los objetos que se mueven en la escena. La aproximación más simple para la detección de movimiento está basada en calcular la diferencia entre fotogramas consecutivos de un vídeo, de tal forma que los píxeles sin cambios se mostrarán como negros (igual intensidad en frames consecutivos) y los píxeles con movimiento se apreciarán como blancos o grises (diferente intensidad en frames consecutivos).
Para realizar este apartado, se utilizará la propia captura de la cámara en el ordenador. Como salida se deberán visualizar dos ventanas: una que vaya mostrando la captura de la cámara en color y otra que vaya visualizando la detección de movimiento.
El archivo *apartado3.py* incluye la estructura base del código y los comentarios para editar/completar el mismo. Los pasos a seguir se describen a continuación :

1. Convertir imágenes a escala de gris, ya que resulta menos complejo calcular la diferencia en escala de gris que con imágenes en color. Para ello emplearemos la función *cv2.cvtColor,* tanto para la imagen actual *img* en el instante dado como para la imagen anterior, img*_ref,* que será la referencia.
2. A la hora de restar una imagen con otra, debemos tener en cuenta que podemos obtener valores negativos. Para evitar ésto, lo que haremos será restar los valores de cada píxel mediante el método c*v2.absdiff* de OpenCV con el que obtendremos la imagen diferencia *dif.*
3. Con la finalidad de descartar ruido en la detección de movimiento, solo vamos a considerar

como movimiento aquellos píxeles para los que se haya detectado un valor significativo por encima de un umbral *th* en la imagen diferencia *dif*. Por ello, se trabajará con una imagen *dif_threshold* que se inicializará a 0 y se fijará a blanco para aquellos píxeles con valor de movimiento por encima del umbral. Intente fijar de manera experimental un valor umbral adecuado.

4. Visualizar simultáneamente dos ventanas en cada instante: una con la imagen original y otra con la imagen diferencia umbralizada de la detección de movimiento.

Guarde el script modificado como *apartado3Mod.py* y varias capturas de pantalla en las que se vea el correcto funcionamiento del sistema.

Como actividad adicional, si desea, puede generar un vídeo en el que se muestre en cada fotograma la concatenación de la imagen de la cámara y la imagen de detección de movimiento. Para ello, deberá utilizar la función `np.dstack(image1,image2)`[4] del paquete *numpy*, que tomará como entradas las dos imágenes a concatenar.

4 https://numpy.org/doc/stable/reference/generated/numpy.dstack.html