Procesado de imagen y visión por computador

Programación de aplicaciones. Eventos

1. Manejo de eventos con OpenCV

OpenCV proporciona un conjunto de herramientas que permite al usuario interactuar en el transcurso de la ejecución del programa y tomar cierto control sobre el flujo de ejecución a partir de ciertas acciones (eventos). Estos eventos pueden desencadenarse, por ejemplo, cuando el usuario pulsa una tecla determinada del teclado o cuando hace *click* en un botón del ratón. Asociado a cada evento se define una función de respuesta llamada *callback.*

1.1. Eventos del teclado

Con OpenCV es posible capturar los eventos del teclado con distintas funciones propias. En concreto, la función *cv2.waitKey(delay)→ retval* detiene la ejecución del programa durante el tiempo especificado en *delay* (milisegundos) o hasta que el usuario pulsa una tecla. Si el valor especificado de *delay* es cero, la ejecución del programa se detendrá indefinidamente hasta que el usuario pulse una tecla. Hay que considerar que debido a que en el ordenador habrá más procesos abiertos, en realidad lo que garantiza el valor de *delay* es el tiempo mínimo que se detendrá la ejecución. La función devuelve el código de la tecla presionada por el usuario o bien -1 si no se ha llegado a presionar ninguna tecla durante el tiempo de espera. El valor del código de la tecla pulsada por el usuario nos permitirá poder utilizarlo dentro del código para tomar decisiones y acciones distintas en función de la tecla pulsada. Es importante destacar que esta función sólo funciona si hay una ventana activa de OpenCV y necesita ser llamada periódicamente para detectar si hay un evento.

A la hora de comparar el código devuelto, es necesario tener en cuenta que la función devuelve un valor *int* (32 bits, aunque puede depender de la plataforma). Para poder comparar u obtener las teclas pulsadas tenemos las funciones de Python: *ord()* que devuelve el entero correspondiente a un carácter y *chr()* que devuelve el carácter asociado a un entero (8 bits):

```
>>> ord('q')
113
>>> chr(113)
'q'
```

El uso más común del valor devuelto por *cv2.waitKey* es incluirlo dentro de estructuras *if* o *switch* con el objetivo de realizar una acción u otra, dependiendo de la tecla pulsada, tal como se verá a continuación.

Hay que tener en cuenta que las teclas especiales del teclado modificarán el código de salida de la función. Los 16 bits más significativos representan la combinación de teclas modificadoras pulsadas, tales como *Ctrl*, *Alt*, *NumLock*, *May*, etc. En cualquier caso, los 8 menos significativos contienen el carácter ASCII de la tecla pulsada. Por ello muchas veces veremos en los códigos de Python algo como:

```
if cv2.waitKey(20) & 0xFF == ord('q'):
        break
```

En este ejemplo comprobamos cómo solo se comparan los 8 bits menos significativos para comprobar si la tecla pulsada es 'q'.

1.2. Eventos del ratón

El uso más común de los eventos del ratón en aplicaciones de procesado de imagen es poder marcar coordenadas dentro de una imagen de forma interactiva. Para ello, necesitamos la capacidad de capturar la posición del ratón cuando el usuario pulsó alguno de sus botones, y en ocasiones también saber cuál fue el botón pulsado (botón izquierdo, botón derecho, rueda central, …). Para poder realizar ésto, necesitamos relacionar la ventana de la imagen donde interactuar con la función que atenderá los eventos que se generen. Es necesario tener en cuenta que cada ventana puede tener su propia función, es decir, si el usuario pulsa con el ratón en una ventana, se ejecutará una función concreta, y si lo hace sobre otra, se podrá ejecutar otra función.

Para relacionar una ventana con la función que se ejecutará cuando el usuario actúe con el ratón sobre dicha ventana, emplearemos la siguiente función:

- `cv2.setMouseCallback(ventana, funcion, datos)`
  - "ventana": Nombre de la ventana de OpenCV sobre la que interactuar.
  - "funcion": Nombre de la función (*callback*) que atiende el evento.
  - "datos": parámetros opcionales pasados a la *callback*.

¿Cómo definimos la función (*callback*)? Siempre después de haber creado la ventana, por ejemplo, con *imshow*, debemos definir la función que atenderá el evento del ratón del siguiente modo:

- `def funcion(evento, x, y, flags, datos):`
  - "evento": Tipo de evento: click, dbl click, mouse over,...
  - "x, y": Cordenadas del puntero del ratón
  - "flags": Teclas modificadoras (Ctrl, Alt, May, ...)
  - "datos": Opcional.

En general, será necesario chequear en primer lugar el tipo de evento (por ejemplo, detectar si se ha pulsado el botón derecho o el izquierdo del ratón) para realizar la acción oportuna o incluso no hacer nada. La lista completa de eventos disponibles puede consultarse en varias web, por ejemplo: https://www.opencv-srf.com/2011/11/mouse-events.html.

El script *apartado1.py* implementa un programa que, después de abrir una imagen y mostrarla en la pantalla, asocia a la ventana "*Imagen*" la callback *EventoRaton*, la cual está implementada en el mismo script. Finalmente, crea un bucle infinito que sólo finaliza cuando el usuario pulsa la tecla *Esc*. Dentro del bucle el programa atiende tanto los eventos del teclado (cada vez que el usuario pulsa una tecla, se imprime por pantalla el código de la tecla pulsada), como los del ratón (cada vez que el usuario pulsa con el ratón en la ventana, imprime por pantalla el tipo de evento y las coordenadas donde se pulsó). En el caso de hacer doble click con el botón izquierdo del ratón, se dibuja un círculo en la imagen.

El archivo demuestra cómo pasar variables a la callback de atención al evento. En este caso,

pasamos a la callback el color deseado para pintar los puntos al hacer doble click. Si fuese necesario pasar más de una variable a esta callback, la forma más simple sería crear una lista o tupla donde empaquetar todas las variables. Las variables que hemos usado son variables globales accesibles desde todo el programa, aunque podríamos haber usado el parámetro *datos* si sólo queremos que sean accesibles por la función de callback.

Una vez que haya entendido el código del script *apartado1.py*, intente modificar el mismo para conseguir los siguientes objetivos:

1. El usuario podrá cambiar el color de representación de los puntos, de tal modo que si el usuario pulsa la tecla 'r' se representarán en rojo, si pulsa la tecla 'g' se representarán en verde y si pulsa la tecla 'b' se representarán en azul. Para ello, se usará la función *ord()* que devuelve el entero correspondiente a un carácter.
2. A continuación, se modificará el mismo script para que, utilizando los eventos *cv2.LBUTTONDOWN* y *cv2.LBUTTONUP*, el usuario pueda dibujar un rectángulo dentro de la imagen marcando con el ratón las dos esquinas opuestas que lo definen. Hay que tener en cuenta que el proceso debe hacerse en dos pasos. El primer paso será cuando se pulse el botón sobre la imagen (evento *LBUTTONDOWN*), lo que constituirá una de las esquinas del rectángulo. Por tanto, será necesario capturar y guardar las coordenadas del ratón en ese momento. El segundo paso sería cuando se suelta el botón (después de haberlo arrastrado). Estas coordenadas constituirán la otra esquina del rectángulo. Para pintar el rectángulo sobre la imagen, se deberá utilizar la función cv2.rectangle() de OpenCV.
3. Como mejoras opcionales, si el usuario pulsa la tecla '+', el rectángulo deberá incrementarse el grosor de la línea y si pulsa la tecla '-' se reducirá. Compruebe cómo al reducir el grosor por debajo de 0, cambia el modo de representación del rectángulo pasando de representar el contorno a representar el área interna del rectángulo. Analice este efecto de acuerdo al modo de representación del rectángulo con la función cv2.rectangle.

2. Barras de desplazamiento

Las barras de desplazamiento constituyen una manera cómoda de modificar el valor de una variable o parámetro en un programa de forma interactiva. Una barra de desplazamiento está asociada a una variable y, a medida que se desliza el cursor a lo largo de la barra, el valor de esta variable varía entre un valor mínimo y un valor máximo.

Una barra de desplazamiento en OpenCV se crea con la siguiente instrucción:

- `cv2.createTrackbar(nombre, ventana, valor, maximo, callback)`
  - "nombre": nombre de la barra de desplazamiento.
  - "ventana": ventana OpenCV sobre la que se acopla la barra.
  - "valor": valor inicial de la barra.
  - "maximo": valor máximo de la barra. El mínimo valor es siempre 0.
  - "callback": función que es llamada cada vez que la barra cambia de posición.

La instrucción anterior acopla la barra de desplazamiento horizontal sobre la ventana indicada. El valor asociado a esta barra puede variar entre 0 y el valor indicado en "*maximo"*. Podemos acceder

al valor de la barra de dos formas distintas:

1. Usando *cv2.getTrackbarPos()* que nos devuelve el valor entero correspondiente a la barra.
2. Si en "*callback*" especificamos el nombre de una función con un argumento, cada vez que se cambie el valor de la barra, se lanzará un evento que ejecutará esta función (igual que, por ejemplo, con los eventos del ratón vistos antes). El argumento que hayamos definido en esa función tomará el valor de la barra.

La opción 1 permite una programación más sencilla del código, ya que para obtener el valor de la barra, basta con leer su valor, pero a cambio debemos acceder continuamente a dicho valor, lo cual puede ralentizar la ejecución del programa. La opción 2 es más eficiente, ya que sólo se ejecuta cuando realmente se produce un cambio en el valor de la barra.

El archivo *apartado2.py* implementa un ejemplo muy simple de uso de una barra de desplazamiento siguiendo la opción 1. El código cambia el color del fondo de la ventana en función del valor de las componentes RBG. Analice el código para entender el funcionamiento de las barras de desplazamiento.

Cree ahora un nuevo programa que haga la misma operación que el *apartado2.py,* pero usando la segunda opción de acceso a los valores de las barras de desplazamiento. Tenga en cuenta que en este caso sólo deberá modificarse la componente de color que hayamos cambiado en la barra de desplazamiento y la nueva imagen se mostrará tras la modificación. Nota: considere que para modificar únicamente las componentes *r,g,b* y *s* cuando se modifica el valor de las barras, la lectura de éstas debe realizarse fuera del bucle *while*.

3. Programación de un editor gráfico (Apartado opcional)

En este apartado se programará una aplicación 'Paint', que mediante los movimientos del ratón, nos permitirá pintar a mano alzada sobre una ventana OpenCV. Para dibujar deberemos mantener presionado el botón izquierdo del ratón y movernos sobre el área que deseamos pintar. Para hacer ésto simplemente dibujaremos un círculo en la posición definida por el movimiento del ratón (evento cv2.EVENT_MOUSEMOVE). Para saber cuando tenemos activado el dibujo del trazo manejaremos una variable booleana *start* que se activará cuando se pulse el botón izquierdo del ratón (evento *LBUTTONDOWN*) y se mantendrá activa mientras el usuario arrastre hasta que suelte el botón (evento *LBUTTONUP*).

El programa permitirá al usuario poder cambiar el color del trazo y su grosor en tiempo de ejecución. Para ello, se definirán cuatro barras de desplazamiento (una para cada canal de color RGB (rango de valores [0-255]) y otra para el grosor del pincel (rango de valores [0-15]). A partir del archivo *apartado3.py dado*, complete el código necesario especificado por los comentarios incluyendo el código necesario debajo de la etiqueta TODO. Adicionalmente, puede implementar la opción de deshacer el último trazo del ratón. Para ello, deberá guardar todas las posiciones del trazo a medida que se van dibujando y en el momento de activarse esta opción deberá representar un círculo negro por cada una de esas posiciones.

Guarde el archivo *apartado3.py* con las modificaciones realizadas.