Ejercicio 1

Dibujado de rectángulo

Para dibujar rectángulos, debemos emplear la función cv2.rectangle, a la cual debemos pasarle por parámetros:

1. La imagen sobre la que se dibuja.
2. Dos coordenadas, las cuales marcan el vértice inicial y el final del rectángulo.
3. El color del rectángulo a dibujar.
4. El grosor del rectángulo a dibujar.

Para poder dibujarlo a través del ratón, basta con general una función que responda al evento "movimiento del ratón", tal y como nos indica la práctica (función tipo "callback"). En esta función, deberemos comprobar cuál de los eventos se ha producido.

Botón izquierdo pulsado: Se guardan las coordenadas en las que se encuentra el ratón. De forma adicional se imprime por terminal un mensaje que confirma esto.

Botón izquierdo soltado: Al soltar el botón, invocamos la función cv2.rectangle a la cual le pasamos por parámetro la coordenada previamente almacenada en memoria y la actual. Al igual que antes, se muestra un mensaje por terminal que confirma que hemos soltado el botón izquierdo.

De esta forma se genera un rectángulo que ira desde donde se pulsó el botón hasta la coordenada en que se haya levantado el botón izquierdo.

```
if evento == cv2.EVENT_MOUSEMOVE:
    pass

elif evento == cv2.EVENT_RBUTTONDBLCLK:
    print("Doble click botón izquierdo\n")
    cv2.circle(img, (x,y), 3, color, -1)
elif evento == cv2.EVENT_LBUTTONDOWN:
    coordenada_inicial = (x,y)
    print("Pulsar botón izquierdo\n")
elif evento == cv2.EVENT_LBUTTONUP:
    cv2.rectangle(img, coordenada_inicial, (x,y), (0,0,255), thickness)
    print("Soltar botón izquierdo\n")
else:
    print("Otro evento\n")

print("Coordenadas: x=", x, ", y=", y, "\n")
print(color)
```

El hecho de que cv2.rectangle tome el grosor gracias a una variable thickness se explica al final de la memoria. Permite cambiar el grosor de forma dinámica.

Dibujar puntos de colores

Para poder dibujar puntos de diferentes colores, se emplea la primera sentencia de la captura anterior

elif evento == cv2.EVENT_RBUTTONDBLCLK:

Esto nos permite realizar otra tarea diferente. En este caso cuando se pulse dos veces el botón derecho del ratón, se va a llamar a la función cv2.circle, la cual dibuja un círculo.

Como lo que se nos pedía era dibujar puntos, podemos obtener esto si dibujamos un círculo con un radio pequeño, de forma que salga un círculo relleno.

Para poder seleccionar el tipo de color del círculo, en el bucle “whlie” he creado 3 sentencias “if” que comprueban el valor de la tecla pulsada. Esto se muestra a continuación:

```
#Esperamos a que el usuario pulse una tecla.
c=cv2.waitKey(0) & 0xFF
if c == ord('g'):
    color =(0,255,0)
    print("Color cambiado a verde")
if c == ord('r'):
    color =(0,0,255)
    print("Color cambiado a rojo")
if c == ord('b'):
    color =(255,0,0)
    print("Color cambiado a azul")
```

Pese a no ser la mejor manera de hacerlo (al estar dentro del “while” estas sentencias se comprueban constantemente), he decidido dejarlo así por su simplicidad. La idea es que si c (variable que guarda la tecla pulsada) es igual a g,r o b, la variable global color cambie al color concreto.

Grosor del rectángulo

Para modificar el grosor del rectángulo, he seguido un sistema similar al descrito anteriormente para el cambio de colores.

```
if c == ord('+'):
    thickness +=1
    print("thickness aumentado a:",thickness)
if c == ord('-'):
    if thickness==-1:
        thickness = -1
        print("thickness decrementado a:",thickness)
    else:
        thickness -=1
        print("thickness decrementado a:",thickness)
```

La idea es la misma, con el pequeño detalle de que hay que establecer un “seguro” que evite que tickness (variable global) tome valores inferiores a “-1”. Esto es así para evitar que la función que dibuje el rectángulo nos devuelva un error, ya que “–1" es el mínimo que puede tomar.

Ejercicio 2

Para mantener la misma funcionalidad sin necesidad de comprobar constantemente si se ha modificado el valor de la barra, podemos apoyarnos en la propia definición de las “Trackbars”.

Para definir una “Trackbar” debemos declarar lo siguiente:

```
# Creamos las barras para cada color
cv2.createTrackbar('R','Imagen',50,255,red)
```

El significado de estos parámetros de entrada los explica la propia práctica, pero mirando en la documentación me fijé en que la función que le pasas como último parámetro se ejecuta siempre que se produce un cambio en la barra.

Esto quiere decir que, en el caso de esta captura, se va a llamar a la función “red()” cada vez que haya una variación en la barra. Donde esta variación es un cambio en los valores de la misma.

Con esto en mente, la solución a la que llegue fue generar una función para cada barra y hacer desde ella la actualización.

```
def nothing(x):
    global s
    s = cv2.getTrackbarPos(switch,'Imagen')

def red(x): # Actualizo "color" con la variable de la barra roja
    global r
    r = cv2.getTrackbarPos('R','Imagen')
def green(x):# Actualizo "color" con la variable de la barra verde
    global g
    g = cv2.getTrackbarPos('G','Imagen')
def blue(x): # Actualizo "color" con la variable de la barra azul
    global b
    b = cv2.getTrackbarPos('B','Imagen')
```

Estas funciones hacen exactamente lo mismo que había antes, solo que en lugar de ejecutarse en cada iteración del "while”, se ejecutan si y solo si hay cambios en la barra.