Mouse as a Paint-Brush

Goal

- Learn to handle mouse events in OpenCV
- You will learn these functions : **cv.setMouseCallback()**

Simple Demo

Here, we create a simple application which draws a circle on an image wherever we double-click on it.

First we create a mouse callback function which is executed when a mouse event take place. Mouse event can be anything related to mouse like left-button down, left-button up, left-button double-click etc. It gives us the coordinates (x,y) for every mouse event. With this event and location, we can do whatever we like. To list all available events available, run the following code in Python terminal:

```python
import cv2 as cv
events = [i for i in dir(cv) if 'EVENT' in i]
print( events )
```

Creating mouse callback function has a specific format which is same everywhere. It differs only in what the function does. So our mouse callback function does one thing, it draws a circle where we double-click. So see the code below. Code is self-explanatory from comments :

```python
import numpy as np
import cv2 as cv

# mouse callback function
def draw_circle(event,x,y,flags,param):
    if event == cv.EVENT_LBUTTONDBLCLK:
        cv.circle(img,(x,y),100,(255,0,0),-1)

# Create a black image, a window and bind the function to window
img = np.zeros((512,512,3), np.uint8)
cv.namedWindow('image')
cv.setMouseCallback('image',draw_circle)

while(1):
    cv.imshow('image',img)
    if cv.waitKey(20) & 0xFF == 27:
        break
cv.destroyAllWindows()
```

More Advanced Demo

Now we go for a much better application. In this, we draw either rectangles or circles (depending on the mode we select) by dragging the mouse like we do in Paint application. So our mouse callback function has two parts, one to draw rectangle and other to draw the circles. This specific example will be really helpful in creating and understanding some interactive applications like object tracking, image segmentation etc.

```python
import numpy as np
import cv2 as cv

drawing = False # true if mouse is pressed
mode = True # if True, draw rectangle. Press 'm' to toggle to curve
ix,iy = -1,-1

# mouse callback function
def draw_circle(event,x,y,flags,param):
    global ix,iy,drawing,mode

    if event == cv.EVENT_LBUTTONDOWN:
        drawing = True
        ix,iy = x,y

    elif event == cv.EVENT_MOUSEMOVE:
        if drawing == True:
            if mode == True:
                cv.rectangle(img,(ix,iy),(x,y),(0,255,0),-1)
            else:
                cv.circle(img,(x,y),5,(0,0,255),-1)

    elif event == cv.EVENT_LBUTTONUP:
        drawing = False
        if mode == True:
            cv.rectangle(img,(ix,iy),(x,y),(0,255,0),-1)
        else:
            cv.circle(img,(x,y),5,(0,0,255),-1)
```

Next we have to bind this mouse callback function to OpenCV window. In the main loop, we should set a keyboard binding for key 'm' to toggle between rectangle and circle.

```
img = np.zeros((512,512,3), np.uint8)
cv.namedWindow('image')
cv.setMouseCallback('image',draw_circle)

while(1):
    cv.imshow('image',img)
    k = cv.waitKey(1) & 0xFF
    if k == ord('m'):
        mode = not mode
    elif k == 27:
        break

cv.destroyAllWindows()
```

Additional Resources

Exercises

1. In our last example, we drew filled rectangle. You modify the code to draw an unfilled rectangle.

```
img = np.zeros((512,512,3), np.uint8)
cv.namedWindow('image')
cv.setMouseCallback('image',draw_circle)

while(1):
    cv.imshow('image',img)
    k = cv.waitKey(1) & 0xFF
    if k == ord('m'):
        mode = not mode
    elif k == 27:
        break
```