ساخت یک brush :

برای ساخت یک brush شبیه سایر برنامه های برنامه های ویرایش عکس مثل فتوشاپ می توانیم از همان رویدادهای کلیک ماوس و callback در کتابخانه open cv استفاده نماییم.برای انجام این موضوع ما به یک یا چند متغیر گلوبال در تابع callback نیاز داریم.

متغیر گلوبال در پایتون:

در پایتون، متغیر گلوبال (Global Variable) متغیری است که در خارج از هر تابع یا بلوک کد تعریف می‌شود و در سراسر برنامه قابل دسترسی است. به عبارت دیگر، هر تابعی می‌تواند به این متغیر دسترسی داشته باشد و مقدار آن را بخواند یا تغییر دهد. این متغیر در برابر متغیر لوکال قرار میگیرد که متغیری مربوط به یک تابع است.

جهت ساخت یک براش نسبتا ساده در کتابخانه open cv همیشه می توانیم از قطعه کد زیر استفاده نماییم:

```
# Creating a global variable
draw = False # True if mouse is pressed

# callback function
def brush(event, x, y, falgs, params): # you have to memorize this
    global draw
    if event == cv2.EVENT_LBUTTONDOWN:
        draw = True
    elif event == cv2.EVENT_MOUSEMOVE:
        if draw == True:
            cv2.circle(img,(x,y),5,(255,0,0), -1)
    elif event == cv2.EVENT_LBUTTONUP:
        draw = False

img = np.zeros((512,512,3), np.uint8)
cv2.namedWindow('image')
cv2.setMouseCallback('image', brush)

while True:
    cv2.imshow('image', img)
    if cv2.waitKey(1) & 0xFF == 27:
        break
cv2.destroyAllWindows()
plt.imshow(img[...,::-1])
```

این برنامه یک برنامه ساده نقاشی با موس در پایتون است. ابتدا یک متغیر سراسری به نام "draw" تعریف می‌شود که وضعیت دکمه چپ موس را نشان می‌دهد (فشرده شده یا نشده). سپس، یک تابع به نام "brush" تعریف می‌شود که به عنوان پاسخ به رویدادهای موس عمل می‌کند. این تابع، نوع رویداد موس، مختصات موس و سایر اطلاعات را دریافت می‌کند. اگر دکمه چپ موس فشرده شود، متغیر "draw" به "True" تغییر می‌کند. اگر موس حرکت کند و "draw" برابر "True" باشد، یک دایره آبی در مختصات موس رسم می‌شود. اگر دکمه چپ موس رها شود، "draw"به "False" تغییر می‌کند.

در ادامه، یک تصویر سیاه ایجاد می‌شود و یک پنجره با نام "image" باز می‌شود. تابع "brush" به عنوان پاسخ به رویدادهای موس در این پنجره تنظیم می‌شود. سپس، یک حلقه بی‌نهایت اجرا می‌شود که تصویر را نمایش می‌دهد و منتظر فشرده شدن کلید Esc می‌ماند. اگر کلید Esc فشرده شود، حلقه متوقف می‌شود و پنجره‌ها بسته می‌شوند. در نهایت، تصویر با استفاده از کتابخانه matplotlib نمایش داده می‌شود.

پس از اجرای قطعه کد بالا متوجه خواهیم شد که در صورتی که موس را با سرعت در صفحه حرکت دهیم خط رسم شده به شکل مقطع نمایش داده می شود نه یک خط مستقیم بنابراین باید این مشکل را بر طرف کنیم. اما قبل از رفع این مشکل می خواهیم آپشنی به کدمان اضافه کنیم تا کاربر بتواند رنگ قلم خود را تغییر دهد. برای انجام این کار می توانیم کد خود را به شکل زیر تغییر دهیم:

```
import matplotlib.pyplot as plt

import numpy as np
import cv2

drawing = False # true if mouse is pressed
color = (255,0,0)
# mouse callback function
def brush(event,x,y,flags,param):
    global drawing, color
    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing == True:
            cv2.circle(img,(x,y),5,color,-1)
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False

img = np.zeros((512,512,3), np.uint8)
cv2.namedWindow('image')
cv2.setMouseCallback('image',brush)
```

```
while True:
    cv2.imshow('image',img)
    k = cv2.waitKey(1) & 0xFF
    if k == ord('b'):
        color = (255,0,0)
    elif k==ord('g'):
        color = (0,255,0)
    elif k==ord('r'):
        color = (0,0,255)
    elif k == 27:
        break
cv2.destroyAllWindows()

plt.imshow(img[...,::-1])
```

تغییرات این کد نسبت به کد قبلی به شرح زیر است:

1. **اضافه شدن متغیر `color`:**
   - یک متغیر سراسری جدید به نام `color` اضافه شده است که رنگ دایره‌ای که با موس رسم می‌شود را تعیین می‌کند.
   - مقدار اولیه `color` برابر `(0, 255, 0)` (سبز) است.
2. **تغییر در تابع `brush`:**
   - تابع `brush` اکنون از متغیر `color` برای تعیین رنگ دایره استفاده می‌کند.
   - `global color` به تابع اضافه شده است تا بتواند به متغیر سراسری color دسترسی داشته باشد.
   - `cv2.circle(img,(x,y),5,color,-1)` به جای رنگ ثابت آبی از متغیر color استفاده میکند.
3. **اضافه شدن قابلیت تغییر رنگ با کلیدهای صفحه کلید:**
   - در حلقه اصلی، کد اکنون منتظر فشرده شدن کلیدهای صفحه کلید است.
   - اگر کلید 'b' فشرده شود، `color` به `(255, 0, 0)` (آبی) تغییر می‌کند.
   - اگر کلید 'g' فشرده شود، `color` به `(0, 255, 0)` (سبز) تغییر می‌کند.
   - اگر کلید 'r' فشرده شود، `color` به `(0, 0, 255)` (قرمز) تغییر می‌کند.
   - اگر کلید ESC فشرده شود برنامه خاتمه پیدا میکند.

اکنون که توانستیم رنگ قلم را تغییر دهیم وقت آن رسیده است تا قلم را تصحیح کرده و یک خط صاف رسم کنیم. براین این کار می توانیم از متد line در کتابخانه open cv استفاده نماییم. بنابراین می توانیم کد خود را به شکل زیر تغییر دهیم:

```
img = np.zeros((512,512,3), np.uint8)
drawing = False
ix = 0
iy = 0
# Adding Function Attached To Mouse Callback
def draw(event,x,y,flags,params):
    global ix,iy,drawing
    # Left Mouse Button Down Pressed
    if event==cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix = x
        iy = y
    if event==cv2.EVENT_MOUSEMOVE:
        if drawing==True:
            #For Drawing Line
            cv2.line(img,pt1=(ix,iy),pt2=(x,y),color=(255,255,255),thickness=3)
            ix = x
            iy = y
    if event==cv2.EVENT_LBUTTONUP:
        drawing = False



# Making Window For The Image
cv2.namedWindow("Window")

# Adding Mouse CallBack Event
cv2.setMouseCallback("Window",draw)

# Starting The Loop So Image Can Be Shown
while True:

    cv2.imshow("Window",img)

    if cv2.waitKey(1) & 0xFF ==27:
        break

cv2.destroyAllWindows()
plt.imshow(img[...,::-1])
```

تغییرات این کد نسبت به کد اول (که دایره رسم می‌کرد) به شرح زیر است:

1. **تغییر در عملکرد نقاشی:**

- به جای رسم دایره، اکنون خط رسم می‌کند.
- برای رسم خط، مختصات نقطه شروع (ix, iy) و نقطه پایان (x, y) را ذخیره می‌کند.

2. **اضافه شدن متغیرهای ix و iy:**
   - متغیرهای سراسری ix و iy برای ذخیره مختصات نقطه شروع خط اضافه شده‌اند.
3. **تغییر در تابع draw به جای brush):**
   - نام تابع callback موس به draw تغییر کرده است.
   - در رویداد cv2.EVENT_LBUTTONDOWN، علاوه بر تغییر drawing به True، مختصات نقطه شروع خط (ix, iy) نیز ذخیره می‌شوند.
   - در رویداد cv2.EVENT_MOUSEMOVE، اگر drawing برابر Trueباشد، یک خط از نقطه شروع (ix, iy)به نقطه فعلی موس (x, y) رسم می‌شود. سپس نقطه پایان خط را به نقطه شروع خط بعدی تبدیل میکند.
   - cv2.line(img,pt1=(ix,iy),pt2=(x,y),color=(255,255,255), thickness=3)برای رسم خط سفید استفاده میشود.
   - در رویداد cv2.EVENT_LBUTTONUP، drawing به Falseتغییر می‌کند.
4. **تغییر نام پنجره:**
   - نام پنجره از 'image' به 'Window' تغییر کرده است.

اکنون که با رویداد کال بک و ایجاد یک صفحه نقاشی آشنا شدیم می توانیم از این رویداد برای رسم یک مستطیل استفاده کنیم:

```
draw = False
start_point = (0,0)

def draw(event,x,y,flag,params):
    global draw, start_point
    if event == cv2.EVENT_LBUTTONDOWN:
        draw = True
        start_point = x,y
    elif event == cv2.EVENT_MOUSEMOVE:
        if draw == True:
            cv2.rectangle(img, start_point, (x,y),(255,0,0), -1)
    elif event == cv2.EVENT_LBUTTONUP:
        draw = False
img = np.zeros((512,512,3), np.uint8)
cv2.namedWindow('image')
cv2.setMouseCallback('image', draw)
```

```
while True:
    cv2.imshow('image', img)
    key = cv2.waitKey(1) & 0xFF
    if key == 27:
        break
cv2.destroyAllWindows()
plt.imshow(img[...,::-1])
```

این کد یک برنامه ساده برای رسم مستطیل با موس در پایتون است.

ابتدا، دو متغیر سراسری `draw` و`start_point`تعریف می‌شوند .وضعیت دکمه چپ موس را نشان می‌دهد (فشرده شده یا نشده) و `start_point`مختصات گوشه اولیه مستطیل را ذخیره می‌کند.

سپس، یک تابع به نام `draw`تعریف می‌شود که به عنوان پاسخ به رویدادهای موس عمل می‌کند. این تابع، نوع رویداد موس، مختصات موس و سایر اطلاعات را دریافت می‌کند.

اگر دکمه چپ موس فشرده شود(`cv2.EVENT_LBUTTONDOWN`)، متغیر `draw`به `True`تغییر می‌کند و مختصات موس در `start_point`ذخیره می‌شود.

اگر موس حرکت کند (`cv2.EVENT_MOUSEMOVE`) و `draw`برابر `True`باشد، یک مستطیل با گوشه اولیه `start_point`و گوشه مقابل مختصات فعلی موس (`x, y`) رسم می‌شود. مستطیل رسم شده آبی رنگ است و پر شده است.

اگر دکمه چپ موس رها شود(`cv2.EVENT_LBUTTONUP`)، متغیر `draw`به `False`تغییر می‌کند.

در ادامه، یک تصویر سیاه ایجاد می‌شود و یک پنجره با نام 'image' باز می‌شود. تابع `draw`به عنوان پاسخ به رویدادهای موس در این پنجره تنظیم می‌شود.

سپس، یک حلقه بی‌نهایت اجرا می‌شود که تصویر را نمایش می‌دهد و منتظر فشرده شدن کلید Esc می‌ماند. اگر کلید Esc فشرده شود، حلقه متوقف می‌شود و پنجره‌ها بسته می‌شوند. در نهایت، تصویر با استفاده از کتابخانه matplotlib نمایش داده می‌شود.

در نهایت ما می توانیم کدمان را به شکلی تغییر دهیم که در یک عکس یک مستطیل تو خالی برای annotate کردن اشیا رسم نماییم برای انجام این کار همیشه می توانیم از قطعه کد زیر استفاده نماییم:

```python
import cv2

# now let's initialize the list of reference point
start_point = (0,0)
points = []
drawing= False

def annotate_image(event, x, y, flags, param):
    # grab references to the global variables
    global start_point, points, drawing

    if event == cv2.EVENT_LBUTTONDOWN:
        start_point = (x, y)
        drawing = True
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing:
            temp_image = image.copy()
            cv2.rectangle(temp_image, start_point, (x, y), (120,255,0), 2)
            cv2.imshow("image", temp_image)
    # check to see if the left mouse button was released
    elif event == cv2.EVENT_LBUTTONUP:
        # record the ending (x, y) coordinates and indicate that
        # the cropping operation is finished
        points.append([start_point , (x, y)])

        # draw a rectangle around the region of interest
        cv2.rectangle(image, start_point, (x, y), (0, 255, 0), 3)
        cv2.imshow("image", image)
        drawing = False


# load the image, clone it, and setup the mouse callback function
image = cv2.imread('images/walking.jpg')
clone = image.copy()
temp_image = image.copy()
cv2.namedWindow("image")
cv2.setMouseCallback("image", annotate_image)

while True:
    # display the image and wait for a keypress
    if drawing == False:
        cv2.imshow("image", image)
    key = cv2.waitKey(5) & 0xFF
```

```
    # press 'r' to reset the window
    if key == ord("r"):
        image = clone.copy()
        points = []

    # if the 'c' key is pressed, break from the loop
    elif key == 27:
        break

# close all open windows
cv2.destroyAllWindows()

plt.imshow(image[...,::-1])
print(points)
```

این کد یک برنامه پایتون است که به کاربر اجازه می‌دهد با استفاده از موس، نواحی مستطیلی را روی یک تصویر مشخص کند و مختصات این نواحی را ذخیره کند.

در ابتدا، کتابخانه OpenCV (cv2) وارد می‌شود. سپس، سه متغیر سراسری تعریف می‌شوند: `start_point` برای ذخیره مختصات گوشه اولیه مستطیل، `points` برای ذخیره لیستی از مختصات مستطیل‌های مشخص شده و `drawing` برای نشان دادن وضعیت رسم مستطیل (در حال رسم یا خیر).

تابع `annotate_image` به عنوان تابع callback موس تنظیم می‌شود. این تابع رویدادهای موس را دریافت می‌کند و بر اساس آن‌ها عمل می‌کند.

- اگر دکمه چپ موس فشرده شود (`cv2.EVENT_LBUTTONDOWN`)، مختصات موس به `start_point` اختصاص داده می‌شود و `drawing` به `True` تغییر می‌کند.
- اگر موس حرکت کند (`cv2.EVENT_MOUSEMOVE`) و `drawing` برابر `True` باشد، یک کپی از تصویر اصلی ایجاد می‌شود (`temp_image`). سپس، یک مستطیل موقت با گوشه اولیه `start_point` و گوشه مقابل مختصات فعلی موس رسم می‌شود. این مستطیل موقت با رنگ سبز روشن نمایش داده می‌شود. تصویر موقت نمایش داده میشود.
- اگر دکمه چپ موس رها شود (`cv2.EVENT_LBUTTONUP`)، مختصات گوشه مقابل مستطیل ذخیره می‌شود و یک مستطیل دائمی با رنگ سبز تیره روی تصویر اصلی رسم می‌شود. مختصات دو گوشه مستطیل در لیست `points` ذخیره می‌شود و `drawing` به `False` تغییر می‌کند. تصویر اصلی نمایش داده میشود.

سپس، تصویر از فایل 'images/walking.jpg' بارگذاری می‌شود. یک کپی از تصویر اصلی (clone) و یک کپی موقتی (temp_image) ایجاد می‌شود. یک پنجره با نام 'image' ایجاد می‌شود و تابع annotate_image به عنوان تابع callback موس برای این پنجره تنظیم می‌شود.

یک حلقه بی‌نهایت اجرا می‌شود. در هر تکرار، تصویر نمایش داده می‌شود. اگر drawing برابر False باشد، تصویر اصلی نمایش داده میشود. منتظر فشرده شدن کلید می‌ماند.

- اگر کلید 'r' فشرده شود، تصویر به کپی اصلی (clone) بازنشانی می‌شود و لیست points پاک می‌شود.
- اگر کلید Esc (27) فشرده شود، حلقه متوقف می‌شود.

در نهایت، همه پنجره‌های OpenCV بسته می‌شوند و لیست points چاپ می‌شود. تصویر نهایی با استفاده از matplotlib نمایش داده میشود.