تمرین سوم بینایی

بنفشه قلی نژاد 98522328

سوال اول)

الف) فرمول تبدیل rgb به cymk

$$K = 1 - max(R', G', B')$$

$$C = \frac{(1 - R' - K)}{1 - K}$$

$$M = \frac{(1 - G' - K)}{1 - K}$$

$$Y = \frac{(1 - B' - K)}{1 - K}$$

و برای حالت معکوس آن، فرمول را برعکس می کنیم یعنی : r = (1-c/100) * 255 * (1-k/100)

نتیجه :
```
def rgbtocmyk(r, g, b, RGB_SCALE, CMYK_SCALE):

    kp = 1 -(max(r, g , b)/(RGB_SCALE))
    c =((1-kp) - (r/RGB_SCALE))/ (1-kp)
    m =((1-kp) - (g/RGB_SCALE))/ (1-kp)
    y = ((1-kp) - (b/RGB_SCALE))/ (1-kp)
    return int(c * CMYK_SCALE), int(m * CMYK_SCALE), int(y *
CMYK_SCALE),int(kp*CMYK_SCALE)
def cmyktorgb(c, m, y, k, CMYK_SCALE, RGB_SCALE ):
    scale = 1 - k/CMYK_SCALE
    r = int(RGB_SCALE * (1 - c/CMYK_SCALE) * scale)
    g = int(RGB_SCALE * (1 - m/CMYK_SCALE) * scale)
    b = int(RGB_SCALE * (1 - y/CMYK_SCALE) * scale)
    return r, g, b
```
result: **cymk : 61 46 0 49**

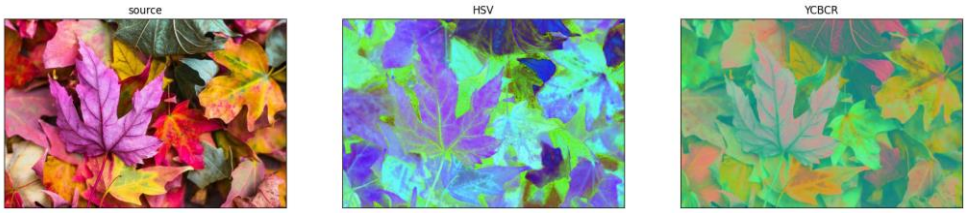
rgb : (50, 70, 130)

ب) با استفاده از توابع `cv.cvtcolor()`

Ycbcr:

```
 result = cv2.cvtColor(image, cv2.COLOR_BGR2YCR_CB)
```
hsv:

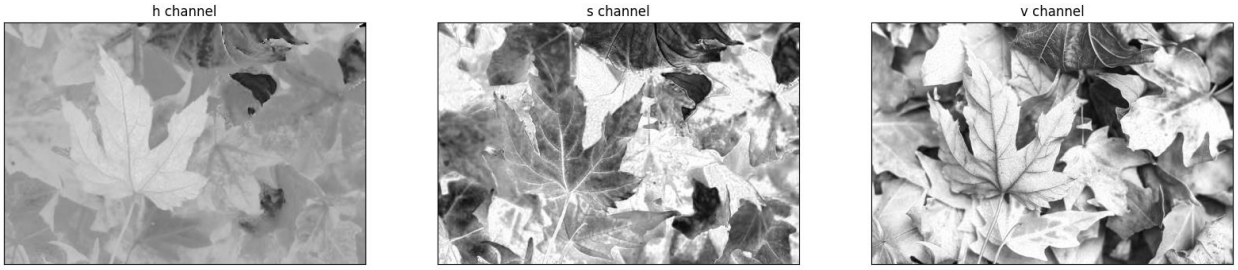
```
result = cv2.cvtColor(image , cv2.COLOR_RGB2HSV)
```
Result:



ج) تصویری را که به کانال hsvبرده ایم، هر کانال را جدا کرده و نمایش میدهیم.

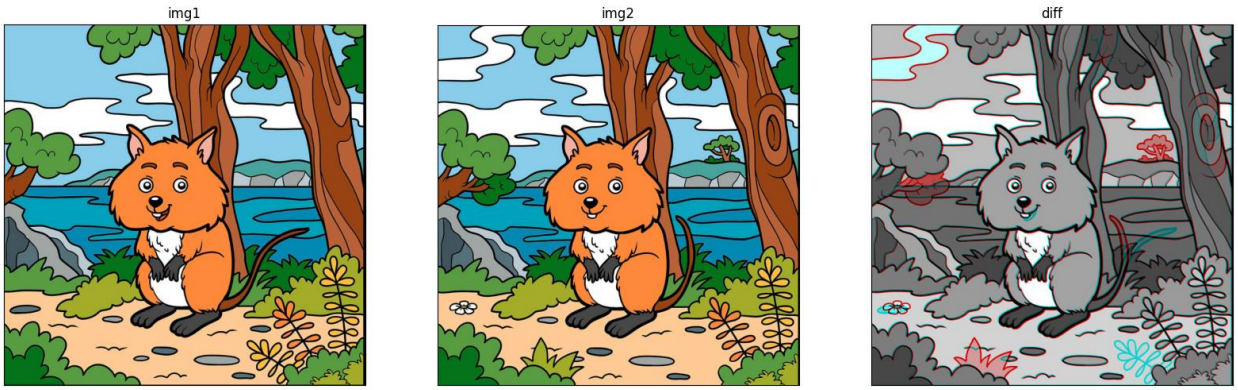
```
hsv = convert_to_hsv(image)
h, s, v = hsv[:,:,0] , hsv[:,:,1] , hsv[:,:,2]
channels = [[h, 'h channel' , 'img'] , [s ,  's channel' , 'img'] , [v , 'v
channel' , 'img']]
plotter(channels,1 , 3, True, 20, 10, '2A')
```

result:



د)روش کلاسی : برای یافتن تفاوت دو عکس سیاه و سفید ، آن ها را به کانال رنگی می بریم. یعنی عکس اول در یک کانال و عکس دیگر در دو کانال دیگر جای بگیرد. در این حالت ، در قسمت های شبیه رنگ پیکسل عکس سه کاناله سیاه وسفید و در قسمت های متفاوت به رنگ همان کانالی است که عکس درون آن قرار گرفته است. (مثلا قرمز و فیروزه ای (ترکیب سبز و آبی))

```
h,v = image1.shape
    blank_image = np.zeros((h-1,v,3), np.uint8)
    print(h , v)
    blank_image[:, : , 0]  = image1[1: , :]
    blank_image[:, : , 1]  = image2[: , 1:]
    blank_image[:, : , 2]  = image2[: , 1:]
```



ه) به خاطر کاربرد های گستره ی آن ها و فضای رنگی rgbنمی تواند همیشه پاسخگوی ما باشد. برای مثال برای تغییر روشنایی ، با کانال rgb نمی توان به شکل خوبی تغییر داد اما در فضای hsvبه راحتی قابل تغییر است و برای تغییرات دیگر عکس (مثل خلوص رنگ یا شدت رنگ)

یا برای دستگاه های پیشرفته تر و طیف رنگ های خارج از مرئی و ..

سوال 2)

در قسمت اول عکس ها را کنار هم خوانده و آن را resizeمی کنیم.

برای نمایش آن ها کنار هم از plt.subpltاستفاده می کنیم .

سپس تابع stitchرا که با مد PANAROMAاست را فرا میخوانیم.

stitcher = cv2.Stitcher_create(0)

st, stitch = stitcher.stitch(images)

در صورتی که ،stصفر باشد، به معنی انجام نشدن عملیات است و در صورت موفقیت آمیز بودن آن را رسم می کنیم .در این حالت، این تابع با کمک نقاط کلیدی ، نقاط مشترک را پیدا کرده و تصاویر را متصل می کند

```
if st == 0:
   plt.imshow(stitch)
   cv2.imwrite('images/panaroma.png' , stitch)
else:
    print("stitching failed".format(st))
```

سوال 3)

الگوریتم به صورت کلی به این صورت عمل می کند که نقاط کلیدی صورت (سمت چپ، راست بینی و چانه) و نقاط کلیدی ماسک را مشخص کنیم.

سپس با تبدیل نقاط متناظر ماسک را روی صورت بیاوردیم.

مرحله اول) نقاط کلیدی صورت با استفاده از کتابخانه dlib:توابع detectorو predictorرا فراخوانی می کنیم. تابع اول وجود چهره را بررسی می کندو تابع دوم نقاط کلیدی صورت را به صورت 68 عدد مختصات برمیگرداند. طبق شکل داک نقاط 3و 15و 30و 9 می توانند نقاط مناسبی باشند. پس آن ها را داخل ارایه میریزیم.

```
detector = dlib.get_frontal_face_detector()
    myface = detector(face)
    predictor = dlib.shape_predictor(p)
    coords = np.zeros((68, 2))
    landmarks = predictor(gray, myface[0])
    for n in range(0,68):
        x = landmarks.part(n).x
        y = landmarks.part(n).y
        coords[n] = [x, y]
```

مرحله ی دوم) انتخاب نقاط کلیدی ماسک: این مرحله رابه صورت چشمی انجام میدهیم و چهار نقطه ی گوشه ای (بالا، پایین ، چپ ، راست) را انتخاب کرده و در آرایه قرار میدهیم.

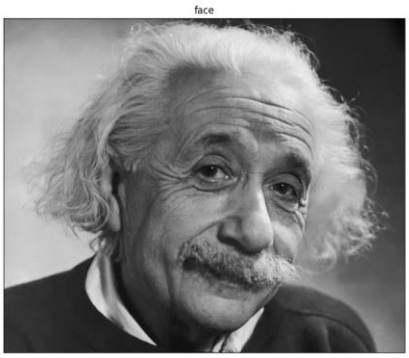
```
dst =
np.array([[coords[2][0],coords[2][1]],[coords[14][0],coords[14][1]],[coords[8][0],coo
rds[8][1]],[coords[28][0],coords[28][1]]],np.float32)
    src = np.array([[110,400] , [1100,400] , [600,810] ,[600,190]],np.float32)
```

مرحله ی سوم) در این مرحله تبدیل حالت اول ماسک به حالتی که روی صورت بنشیند اتفاق می افتد. از توابع getperspectivetransform استفاده می کنیم تا ماتریس تبدیل نقاط منبع و مقصد را به دست بیاوریم. سپس آن را به تابع warperspectivetransform می دهیم تا ماسک را با مدلی که میخواهیم تبدیل کند.

```
   matrix = cv2.getPerspectiveTransform(src, dst)
    print(matrix)
    result = cv2.warpPerspective(maskk, matrix, (einstein.shape[1] ,
einstein.shape[0])) face2= cv2.addWeighted(face,1, result,1.0,0)
```

مرحله ی چهارم) ماسک تغییر یافته را روی صورت می اندازیم

```
face2= cv2.addWeighted(face,1, result,1.0,0)
```



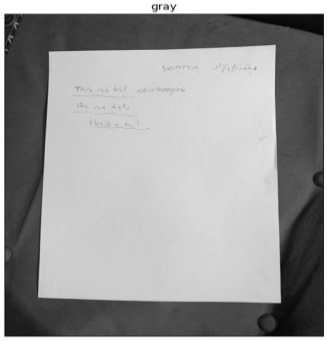
.

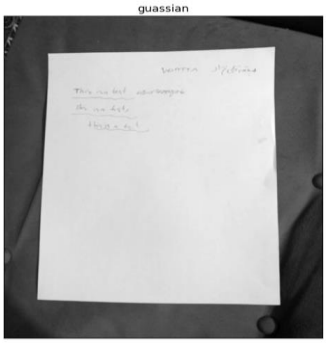
سوال 4)

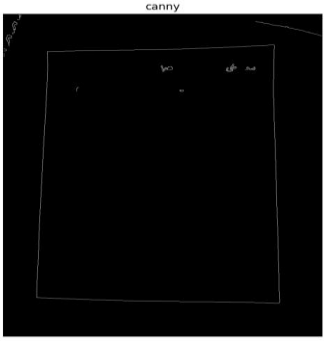
الف) 1- gray-scale، 2- canny – 3 guassian filter

دو ناحیه ی استانه گزاری کنی را برابر 170 و 30 قرار دادم تا لبه های نسبتاکناری مشخص تر باشند.

```
1- gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
2-  img_blur = cv2.GaussianBlur(im, (7,7), 0)
3- edges = cv2.Canny(im,170,30)
```







ب) از کانتور برای یافتن نقاط گوشه استفاده می کنیم. مشابه تمرین های پیشین نقاطی که 4 نقطه دارند را جدا میکنیم و علاوه بر آن ، این چهار نقطه باید بیشترین سطح مساحت ممکن را تشکیل دهند. (بزرگ ترین مستطیل) . می توان مرز مستطیل را با تغییر پارامتر کانتور نیز نشان داد.

```
    ret, im = cv2.threshold(im, 127, 255, 0)
    contours, hierarchy = cv2.findContours(im, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    biggest = None
    max_area = 0
    approx_contour = None
    list = []
    for i in contours:
        area = cv2.contourArea(i)
        perim = 0.1* cv2.arcLength(i, True)
        approx = cv2.approxPolyDP(i,  perim, True)
        if len(approx) == 4 and area > max_area:
                biggest = approx
                max_area = area
                approx_contour = biggest
```

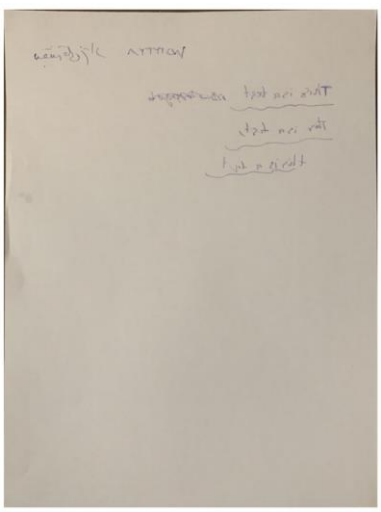
پ) در مرحله ی نهایی ، 4 گوشه ی قاب عکس گرفته را در نظر میگیریم و نسبت به یک مستطیل صاف آن را نگاشت میدهیم تا اطراف عکس از بین برود و از تابع perspectivetransformبرای به دست اوردن تابع تبدیل استفاده می کنیم.

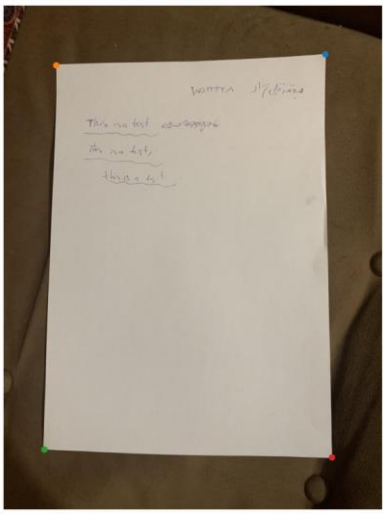
```
  w , h, *channels = im.shape

    new_vertices = np.array([vertices[1][0],vertices[2][0],vertices[3][0],
vertices[0][0]],np.float32)
    target = np.array([[0,0],[0,w],[h, w],[h,0]],np.float32)

    transform = cv2.getPerspectiveTransform(new_vertices, target)  # get the top
or bird eye view effect
    return cv2.warpPerspective(im, transform, (h, w))
```

1- نقاط گوشه 2- برش عکس



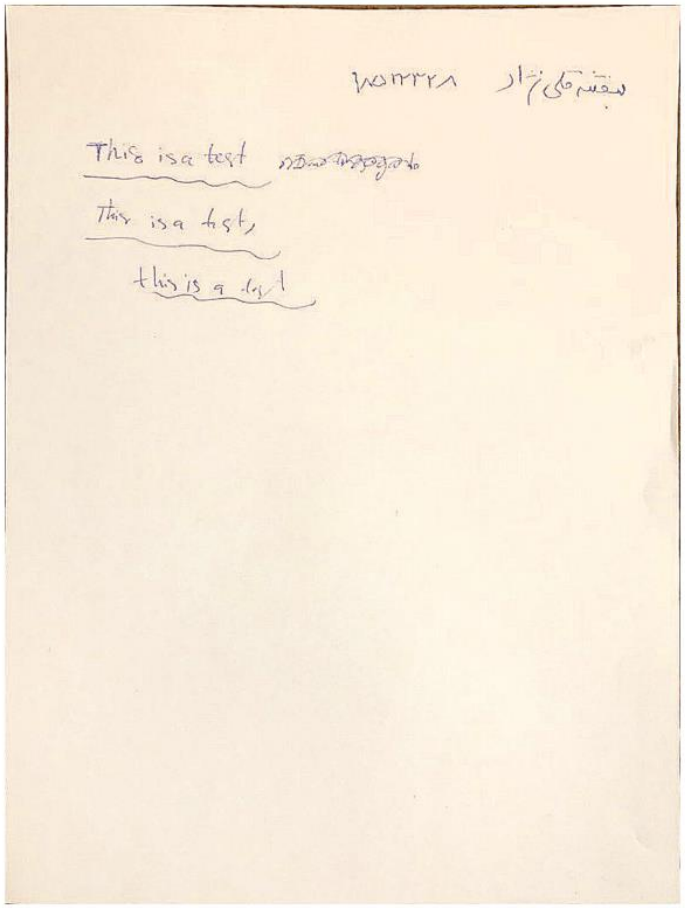


ه) برای بهبود کیفیت تصویر ، می توان کنتراست تصویر و نور آن را بالا برد. برای بهبود کنتراست می توانیم از کرنل های بهبود تصویر و لاپلاسین استفاده کرد.

برای افزایش نور تصویر نیز می توان تصویر را به کانال hsvببریم و مقادیر را یک مقدار ثابتی زیاد کرده و برگردانیم.

```
kernel_sharpening = np.array([[-1,-1,-1],
                              [-1, 9,-1],
                             [-1,-1,-1]])
   sharpened = cv2.filter2D(im, -1, kernel_sharpening)
   value = 60
   hsv = cv2.cvtColor(sharpened, cv2.COLOR_BGR2HSV)
   h, s, v = cv2.split(hsv)
   lim = 255 - value
   v[v > lim] = 255
   v[v <= lim] += value
   final_hsv = cv2.merge((h, s, v))
   img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
```

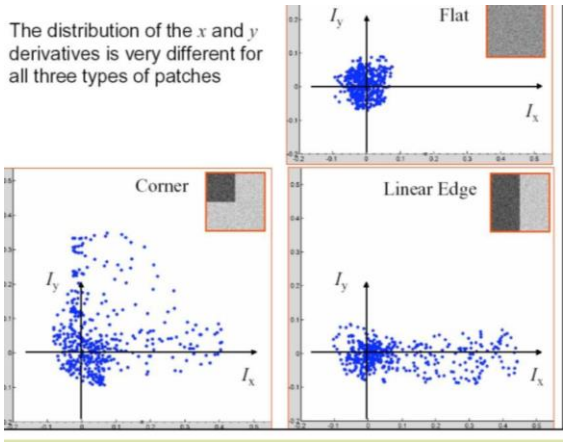
نتیجه :



سوال 5)

الف). ایده این است که نقاط مورد علاقه را در جایی که محله اطراف لبه ها را در بیش از یک جهت نشان می دهد، قرار داده شود. ایده اصلی الگوریتم یافتن تفاوت در شدت برای جابجایی (u,v) در همه جهات به کمک مربع مشتقات است.

w یک پنجره مستطیلی یا یک پنجره گاوسی است که به پیکسل ها در (x,y) وزن می دهد. معادله را می توان با استفاده از بسط Tayler تقریب زد .



$$E(u,v) \approx [u \quad v]\left(\sum_{x,y} w(x,y)\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}\right)\begin{bmatrix} u \\ v \end{bmatrix} \qquad R = \lambda_1\lambda_2 - k(\lambda_1+\lambda_2)^2$$

$$M \triangleq \sum_{x,y} w(x,y)\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \qquad R = det(M) - k\big(trace(M)\big)^2$$

مرحله ی 1 : یافتن مشتق افقی و عمودی به کمک سوبل:

```
Ix = cv2.filter2D(img1_gray,cv2.CV_64F ,  SOBEL_X)
    Iy = cv2.filter2D(img1_gray,cv2.CV_64F,  SOBEL_Y)
```

2 – محاسبه مربع مشتق ها :

```
 Ix2 = np.square(Ix)
    Iy2 = np.square(Iy)
    IxIy = Ix*Iy
```

3– اعمال اثر پنجره ی w(برای این کار از 3 فیلتر گاوسی استفاده شد)

```
I_dx2  = cv2.GaussianBlur(Ix2,(3,3),0)
    I_dy2 = cv2.GaussianBlur(Iy2,(3,3),0)
    I_dxdy = cv2.GaussianBlur(IxIy,(3,3),0)
```

4–محاسبه مقادیر R و k = 0.06 (با استفاده از فرمول بالا)

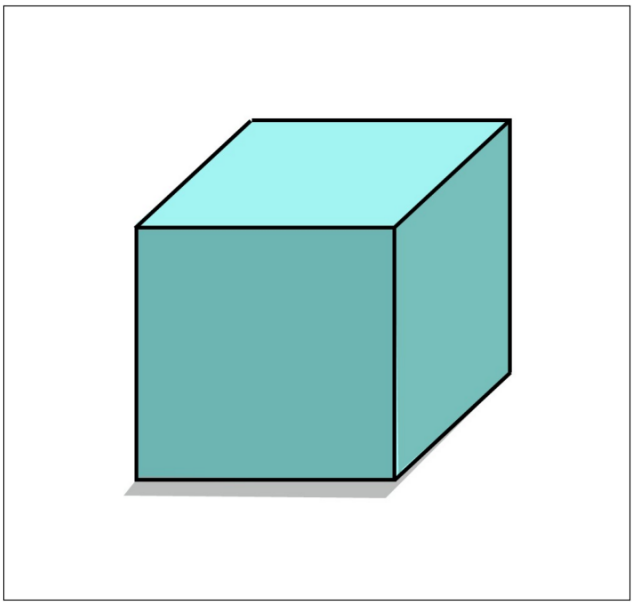
```
harris = I_dx2*I_dy2 - np.square(I_dxdy) - 0.06*np.square(I_dx2 + I_dy2)
```

5) حذف مقادیر غیر بیشینه :threshold = 03

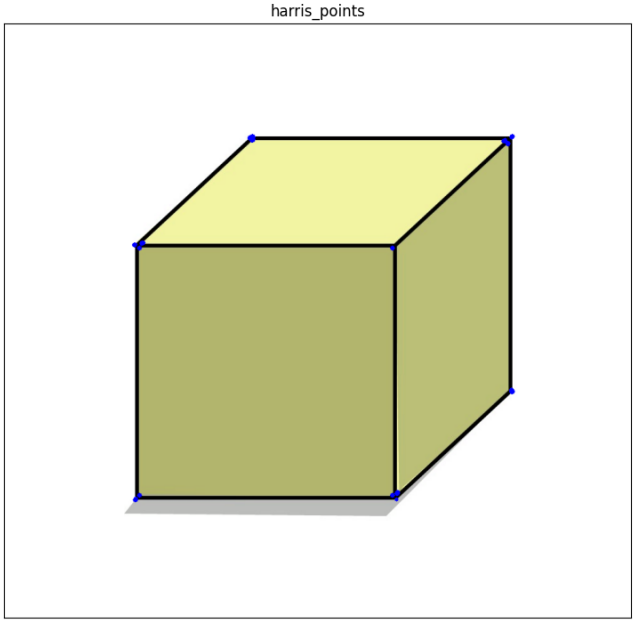
```
cv2.normalize(harris, harris, 0, 1, cv2.NORM_MINMAX)
```

```
points = np.where(harris >= 0.3)
```

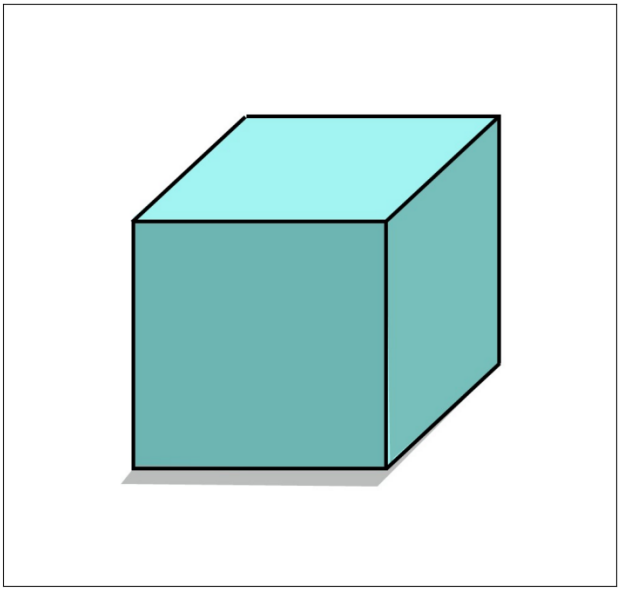
source



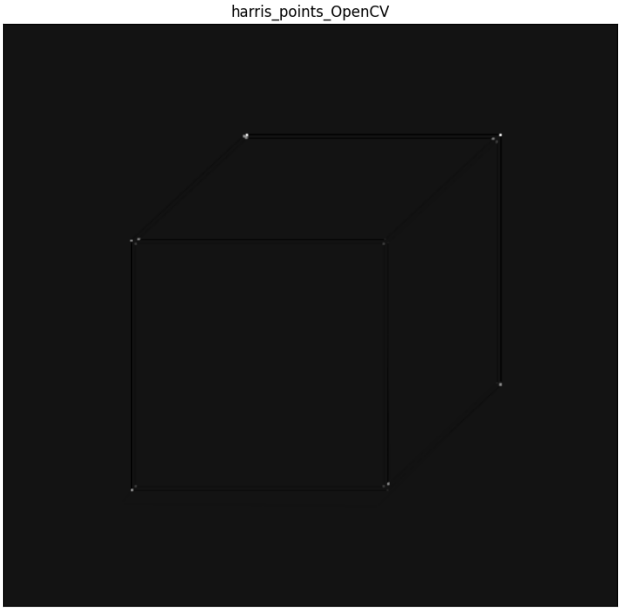
harris_points



source



harris_points_OpenCV



سوال 6)

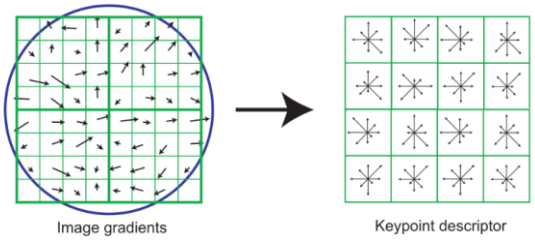
روش : SIFT

SIFT را می توان به دو بخش تقسیم کرد:

-1تشخیص نقطه کلید و-2 استخراج توصیفگر نقطه کلید.

نشخیص نقطه ی کلیدی با کمک تقریب لاپلاسین گاوسی انجام میپذیرد.این روش به این صورت است که با کمک یک stack در تمام همسایگی 3*3*3 می گرددتا به عنوان نقطه ی کلیدی شناسایی شود. هر نقطه نیز دارای یک جهت است. به این صورت که برای هر نقطه در هر همسایگی یک هیستوگرامی از جهت ها میسازد و نقطه پیک را به عنوان جهت در نظر میگیرد.

برای توصیف گر نیز برای هر همسایگی از نقطه کلیدی یک پنجره ی 16 * 16 طراحی میشود و به خانه های 4*4 تقسیم می شود. یک هیستوگرام جهت گیری برای هر خانه محاسبه می شود و هیستوگرام های ترکیبی به یک توصیفگر ویژگی 128 بعدی متصل می شوند.



روش : surf

این روش یک روش بهبودی برای روش بالا است که به جای تقریب و تفاوت گاوسی، از boxfilterها استفاده می کند. و مزیت آن بالا بردن سرعت محاسبه است و در مقیاس های مختلف.

برای توضیح جهت‌یابی، در هر دو جهت x و y در یک همسایگی s6 در اطراف نقطه کلید در یک مرحله نمونه‌برداری از s با s متناسب با مقیاس محاسبه می‌شوند. مجموع پاسخ ها در یک ناحیه اسکن کشویی برای تعیین جهت استفاده می شود.

برای استخراج ویژگی های نقطه کلید، یک همسایگی 20 *2 ث استخراج و به سلول های 4*4 تقسیم می شود. پاسخها از هر سلول استخراج می‌شوند و پاسخ‌های هر سلول به هم متصل می‌شوند تا یک توصیفگر ویژگی 64 بعدی را تشکیل دهند.

روش ORB:

این الگوریتم ترکیبی از دو الگوریتم Fastو BRIEF است.

FAST به عنوان آشکارساز نقطه کلید استفاده می شود. با انتخاب پیکسل‌هایی در شعاع اطراف یک نقطه کلیدی منتخب ، کار می‌کند و بررسی می‌کند که آیا n پیکسل پیوسته وجود دارد که همگی روشن‌تر یا تیره‌تر از پیکسل نامزد هستند. این سرعت فقط با مقایسه زیر مجموعه ای از این پیکسل ها قبل از آزمایش کل محدوده افزایش می یابد. FAST جهت گیری را محاسبه نمی کند، برای حل این موضوع، از مرکز وزنی شدت نقطه کلید و جهت این مرکز با اشاره به نقطه کلید به عنوان جهت استفاده می کنند.

BRIEF به عنوان توصیفگر نقطه کلیدی استفاده می شود. از آنجایی که BRIEF با چرخش ضعیف عمل می کند، جهت گیری محاسبه شده نقاط کلیدی برای هدایت جهت وصله نقطه کلید قبل از استخراج توصیفگر استفاده می شود. پس از انجام این کار، یک سری آزمایش باینری محاسبه می‌شود که الگوی پیکسل‌ها را در وصله مقایسه می‌کند. خروجی تست های باینری به هم پیوسته و به عنوان توصیفگر ویژگی استفاده می شود.