نیکی مجیدی فرد

تمرین سوم

98522382

سوال 1

الف) طبق فرمول زیر عمل می کنیم:

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

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1-K \\ 1-K \\ 1-K \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

توجه داریم که مقیاس rgb 255 و cymk 100 است.طبق فرمول بالا ابتدا k را به دست آورده و سپس هر یک از مقادیر را محاسبه می کنیم. از دو تابع rgb_to_cmyk و cmyk_to_rgb برای تبدیل ها استفاده کردیم:

```
def rgb_to_cmyk(r, g, b, RGB_SCALE = 255, CMYK_SCALE = 100):
    maxvalue = max(r, g , b)
    maxscale = float(maxvalue/255)

    c = round((float((maxvalue- r)/RGB_SCALE)/(maxscale))* CMYK_SCALE)
    m = round((float((maxvalue- g)/RGB_SCALE)/(maxscale))*CMYK_SCALE)
    y = round((float((maxvalue- b)/RGB_SCALE)/(maxscale))*CMYK_SCALE)
    k = round((1 - maxscale)*100)


    return c, m, y, k
print(rgb_to_cmyk(50,70,130))
```

```
(62, 46, 0, 49)
```

```
def cmyk_to_rgb(c, m, y, k, CMYK_SCALE = 100, RGB_SCALE = 255):
    r = round(RGB_SCALE * (1 - c/CMYK_SCALE) * (1 - k/CMYK_SCALE))
    g = round(RGB_SCALE * (1 - m/CMYK_SCALE) * (1 - k/CMYK_SCALE))
    b = round(RGB_SCALE * (1 - y/CMYK_SCALE) * (1 - k/CMYK_SCALE))


    return r, g, b

print(cmyk_to_rgb(62,46,0,49))
```

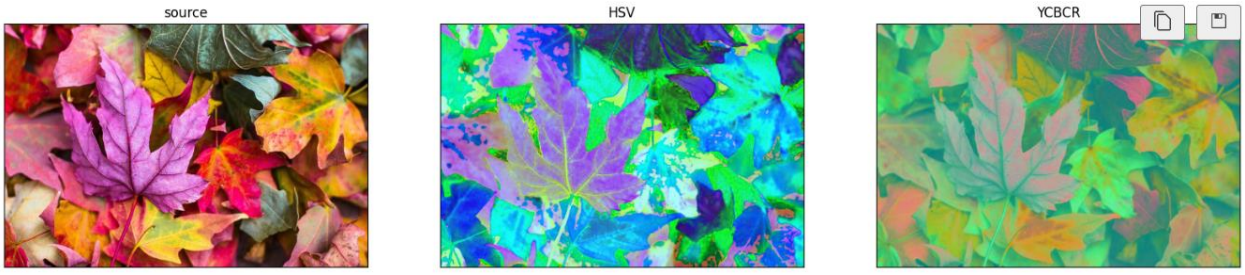
```
(49, 70, 130)
```

ب)برای بردن تصویر 1.jpg به hsv و YCbCr از توابع زیر استفاده کردیم:

```
image = cv2.imread('images/Q1/1.jpg')
result = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

```
image = cv2.imread('images/Q1/1.jpg')
result = cv2.cvtColor(image, cv2.COLOR_BGR2YCR_CB)
```

نتیجه به صورت زیر حاصل شد:

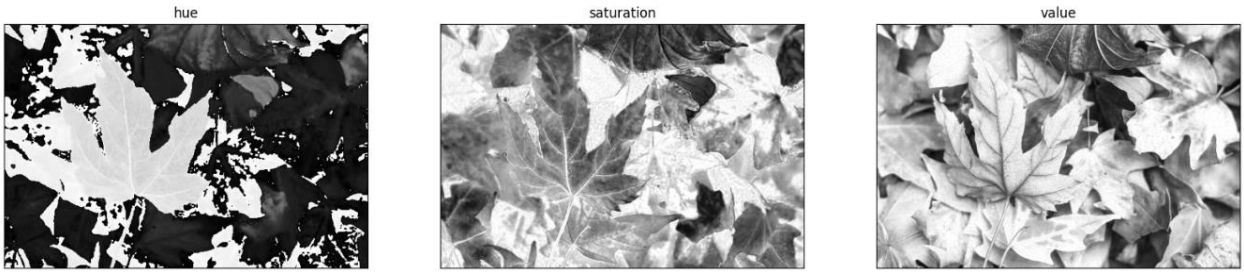


ج)

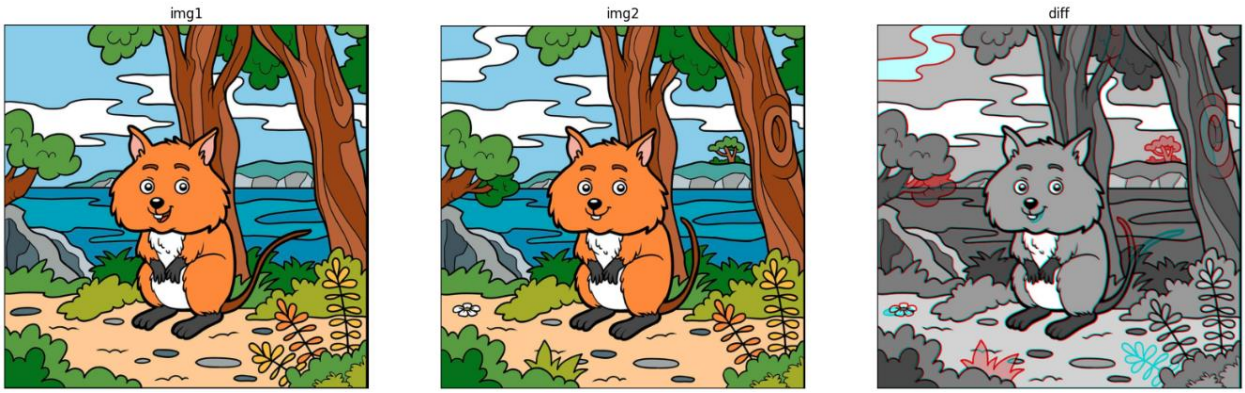
می خواهیم عکس 1.jpg را در کانال های مجزا h,s,v نمایش دهیم. طبق عکس زیر از تابع split برای جدا کردن کانال ها استفاده کردیم و سپس آنها را به صورت زیر نمایش دادیم.

```python
def sepearte_hsv(image):

    image = cv2.imread(image)
    result = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    h,s,v = cv2.split(result)
    return h,s,v
```

✓ 0.1s



د)عکس ها باید سیاه و سفید شوند و بعد آنها را به کانال رنگی ببریم. مثلا در rgb عکس اول را به جای کانال رنگی قرمز و عکس دوم را کانال سبز آبی بگذاریم. بدین ترتیب پیکسل هایی که فرق دارند، اگر در عکس اول باشند قرمز و اگر در دوم باشند سبزآبی نشان داده می شوند.درواقع عکس سیاه و سفید یک کاناله را به سه کانال بردیم که عکس اول کانال قرمز و عکس دوم کانال سبز و ابی است. نتیجه به صورت زیر است:



ه) داشتن درک بیشتر اینکه چگونه فضاهای رنگی بر تصاویر تاثیر می گذارد، می تواند به بهتر نشان دادن آنها کمک کند و همچنین کار را سریع تر می کند. به طور کلی از فضاهای رنگی برای نمایش به طوریکه بتوانند قابل تولید در دستگاه های مختلف مثل مانیتور، پرینتر و دوربین ها شوند، استفاده می شود. هر فضای رنگی از مجموعه معادلاتی متفاوتی برای مشخص کردن رنگ ها استفاده می کند و هرکدام استفاده خود را دارند، برای مثال فضای rgb برای نشان دادن رنگ ها روی دستگاه های الکترونیکی است ولی cmyk برای چاپ کردن رنگ ها روی کاغذ است. به علاوه فضاهای رنگی متفاوت برای نشان دادن رنگ ها در روشنایی های متفاوت یا برای ایجاد تاثیر خاصی مثل vibrant استفاده می شوند. به طور کلی کاربرد استفاده از چندین فضای رنگی این اجازه را به ما می دهد که کنترل دقیق تری روی درک و تولید رنگ ها داشته باشیم.

سوال 2

ابتدا یک آرایه از تصاویر می سازیم و تک تک تصاویر به کمک تابع resize کوچک می کنیم. سپس با کمک subplot عکسها را کنار هم می خوانیم تا یک دید کلی پیدا کنیم.

```
imgs=[]
for i in range(1,9):
    imgs.append(cv2.imread(f'images/Q2/{i}.jpg' ))
    imgs[i-1] = cv2.resize(imgs[i-1] ,(0,0) ,fx=0.5 ,fy=0.5)

    j = 0
fig, (ax1, ax2 , ax3,ax4,ax5,ax6,ax7,ax8) = plt.subplots(1, 8,figsize =(18, 18) ,
                                    sharex=True, sharey=True)
tuples = (ax1, ax2 , ax3,ax4,ax5,ax6,ax7,ax8)
for i in range(8):
        ax = tuples[i]
        ax.imshow(imgs[i])
        ax.axis('off')



plt.show()
```

سپس تابع sitcher را صدا می زنیم و flag آن را صفر(معادل حالت پاناروما) قرار می دهیم.سپس آن را stich می کنیم و خروجی آن در دو پارامتر است. اولی وضعیت بهم چسباندن عکس(true or false) و دومی ایمیج نهایی پاناروما که عکس ها کنار هم قرار گرفته شده اند. عملکرد آن هم به صورت پیدا کردن نقاط مشابه تصاویر کنار هم و چسباندن آنهاست به طوریکه عکس 360 درجه تشکیل شود.

نتیجه نهایی:

```
stitcher = cv2.Stitcher_create(0)
status, stitched = stitcher.stitch(imgs)
```

```
plt.imshow(stitched)
plt.axis('off')
```

سوال 3)

برای نگاشت ماسک به صورت ، ابتدا باید نقاط کلیدی صورت و ماسک را تعیین نماییم. برای صورت، از کتاب خانه ی موجود در سوال کمک میگیریم.

این کتاب خانه به این صورت عمل میکند که پس از تشخیص چهره ، 68 نقاط کلیدی را در صورت مشخص می کند. (به صورت ذکر شده در سوال)

ما برای نگاشت ماسک روی صورت ، چهار نقطه (3 ، 15 : گونه ، 9 :چانه ، 30: بینی) استفاده می کنیم.

https://medium.com/mlearning-ai/facial-mask-overlay-with-opencv-dlib-4d948964cc4d

```
allpoints = []
   p = "shape_predictor_68_face_landmarks.dat"
   detector = dlib.get_frontal_face_detector()
   predictor = dlib.shape_predictor(p)
   gray_img=cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
   h,w = gray_img.shape
   mask = cv2.resize(mask, (w,h), interpolation = cv2.INTER_AREA)
   #from https://pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/
   landmarks = predictor(gray_img, detector(face)[0])
   for n in range(0,68):
      if(n == 2 or n == 8 or n == 14 or n == 28):
         allpoints.append( (landmarks.part(n).x, landmarks.part(n).y))
```

برای یافتن نقاط کلیدی ماسک، بالا و پایین ماسک و گوشه های چپی و راستی را به عنوان نقاط کلیدی انتخاب میکنیم.

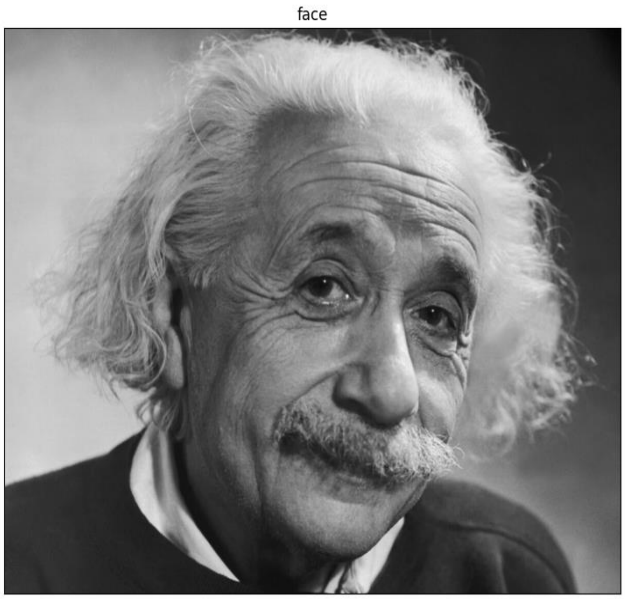
حال تنها لازم است که با کمک توابع تبدیل ، ماسک را به نقاط خواسته شده روی صورت نگاشت دهیم.

با کمک getperspectivetransform(src, dst)، ماتریس تبدیل را پیدا میکنیم و با استفاده از آن در فانکشن warperspective(img , m , (shape))، ماسک را نگاشت میدهیم.

در مرحله ی آخر، ماسک را به کمک addweighted(img1 , a , img2 , b , 0) به صورت اصلی اضافه کرده.

```
pst2 = np.float32([[allpoints[0][0],allpoints[0][1]],[allpoints[1][0],allpoints[1][1]],
[allpoints[2][0],allpoints[2][1]],[allpoints[3][0],allpoints[3][1]]])

pst1 = np.float32([[160,390],[615,840]و [1105,390]  ,[615,190]])

#perspective transform:

M = cv2.getPerspectiveTransform(pst1, pst2)

result = cv2.warpPerspective(mask, M, (w,h))

return cv2.addWeighted(face,1, result,0.9,0)
```

نتیجه ی حاصل:

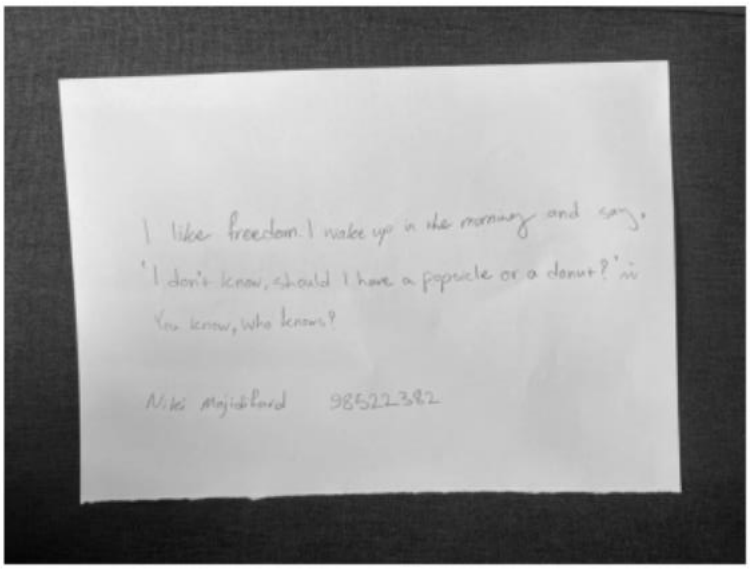


سوال 4

الف) grayscale

```
def to_grayscale(im):

    gray_img = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)

    return gray_img
grayscale = to_grayscale(im)
imshow(grayscale)
```

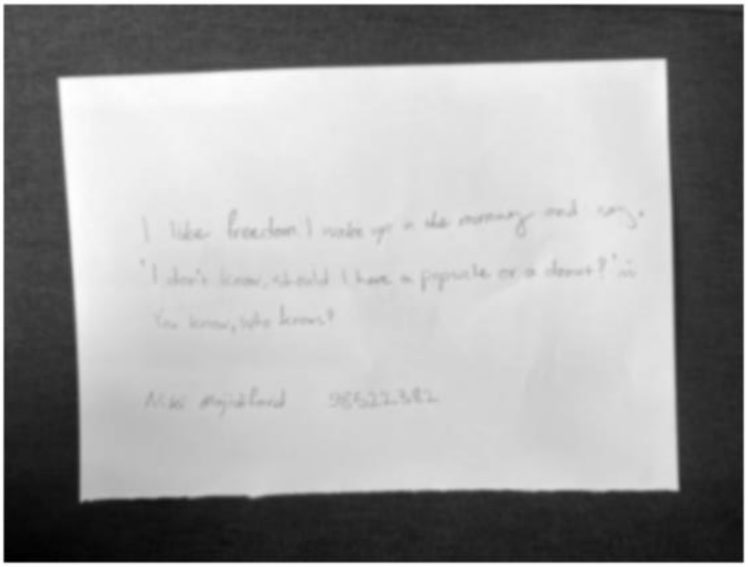
✓ 0.3s



blur

```
def blur(im):
    blur = cv2.GaussianBlur(im, (15,15), 0)
    return blur
blurred = blur(grayscale)
imshow(blurred)
```

✓ 0.7s



canny

```
def to_edges(im):

    edges = cv2.Canny(im, 100, 150)
    return edges
edges = to_edges(blurred)
imshow(edges)
```

✓ 0.3s

ب)contour

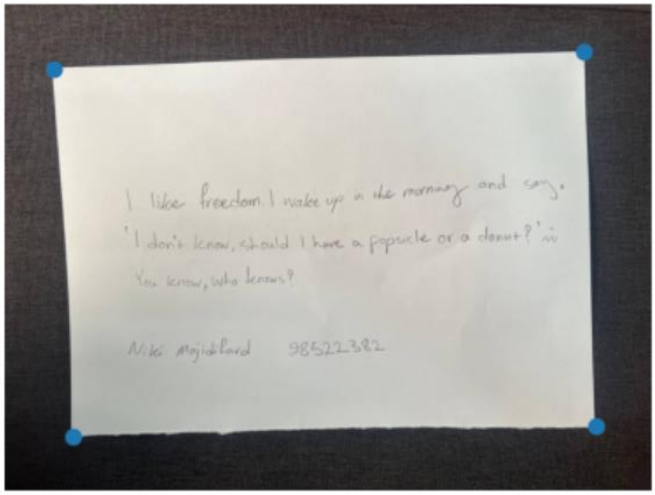
First: cv2.findcontour(imge, threshold[canny pic], method)

این دستور آرایه ای را از کانتور های عکس به ما می دهد. می خواهیم چهار گوشه عکس را پیدا کنیم. برای همین روی تک تک المان های کانتور یک فور می زنیم. Cv2.contourarea(c) مساحت کانتور را مشخص می کند. با کمک دو تابع cv2.arclen(c,bool) و cv2.approxPolyDP(c, perimeter, bool) به ترتیب به دست اوردن محیط و کمتر کردن تعداد نقاط تعیین کننده می پردازیم.

برای هر کانتور شکل به دنبال بزرگترین مساحت ناحیه بسته هستیم و با کمک دو تابع بالا نقاط را به دست می آوریم. اگر نقاط 4 تا باشند و بزرگترین مساحت به دست بیاید، آنگاه ما 4 نقطه گوشه را پیدا کرده ایم.

```python
def find_vertices(im):
    #contour

    contours, hierarchy = cv2.findContours(im, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    final = []
    maxx = 0
    for c in contours:
        area = cv2.contourArea(c)
        perimeter =  cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c,  0.02 * perimeter, True)
        if len(approx) == 4:
            if area > maxx:
                final = approx
                maxx = area

    vertices = []
    for item in final:
        vertices.append(item[0])

    vertices = np.float32([vertices[0],vertices[1],vertices[2],vertices[3]])
    return vertices
```

✓ 0.1s



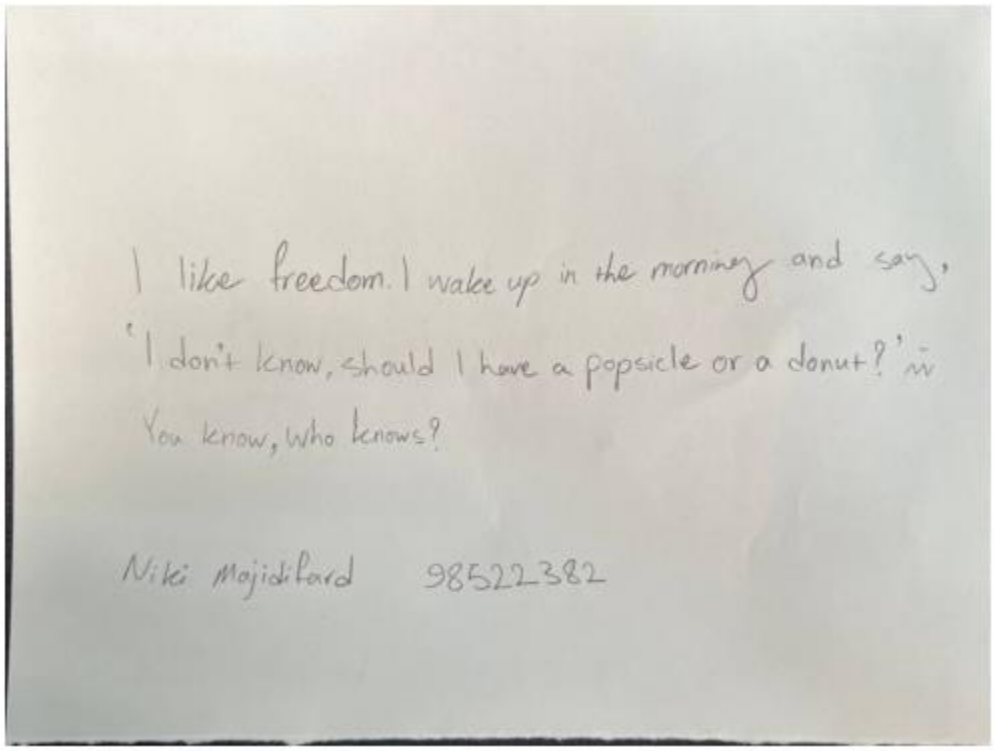
ج) perspective و تبدیل تصویری

این کار با کمک دو تابع انجام می شود. اول با کمک تابع cv2.getPerspecttiveTransform سورس و دستینیشن می دهیم. سورس نقاط گوشه و دستینیشن نقاط چهار گوشه تصویر به حالت معمولی است. (اندازه خود تصویر) و در این صورت تصویر ما به تصویر جدید مپ می شود.

```
def crop_out(im, vertices):
    # Your code goes here.

    target = np.float32([[0,0],[0,height],[width, height],[width,0]])


    transform = cv2.getPerspectiveTransform(vertices, target)  # get the top or
    return cv2.warpPerspective(im, transform, (width, height))
```

✓ 0.1s

خروجی :



ه) بهبود کیفیت تصویر :

بعد از کراپ کردن تصویر، برای بهبود می توانیم از توابع opencv برای بهبود استفاده کنیم.

مانند افزایش کنتراست و brightnessتصویراست و برای آن از این دستور استفاده می کنیم.

```
adjusted = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)
```

```
alpha = 1.2
```

```
beta = 1.4
```

آلفا مقدار کنتراست است. برای کاهش کنتراست، الفا بین 0 و 1 و برای کنتراست بالاتر از الفای بزرگ تر از 1 استفاده می شود. بتا مقدار روشنایی است. یک محدوده خوب برای مقدار روشنایی [−127, 127] است.

از دستور

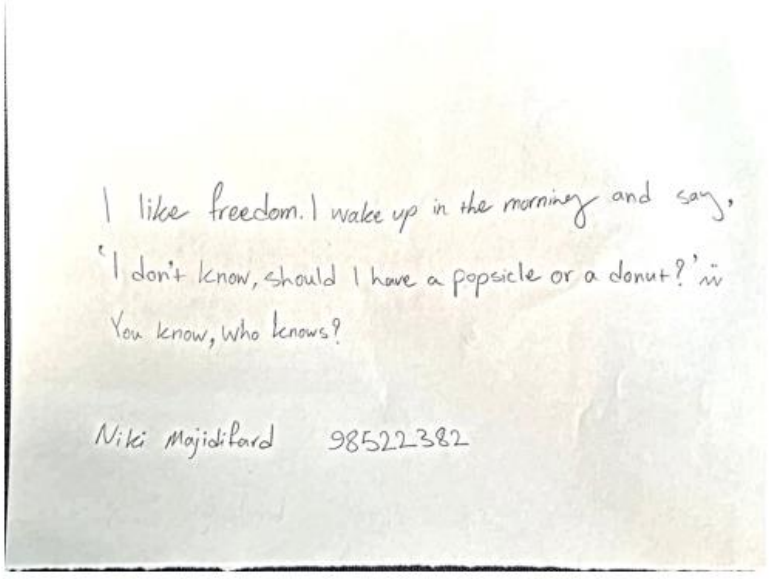
```
cv2.detailEnhance(adjusted, sigma_s=10, sigma_r=0.15)
```

برای smooth کردن تصویر به طوری که جزییات تصویر از بین نرود ، استفاده می شود.

- sigma_s میزان هموار شدن تصویر را کنترل می کند – هر چه مقدار آن بزرگتر باشد، تصویر صاف تر می شود

- sigma_r برای حفظ لبه ها مهم است. Sigma_r کوچک باعث می‌شود که فقط رنگ‌های بسیار مشابه به طور میانگین (به عنوان مثال صاف) شوند، در حالی که رنگ‌هایی که بسیار متفاوت هستند دست نخورده باقی می‌مانند.

خروجی نهایی:



سوال 5)

الف)گوشه یاب هریس ، الگوریتمی است که با کمک مشتق اول و مربعات آن در هر نقطه و همسایگی اش ، گوشه ها و لبه ها را تشخیص می دهد. در واقع این آشکار ساز میزان تغییرات در هر پیکسل را در هر پنجره برای یک پیکسل تعریف میکند.

فرمول کلی آشکار ساز هریس به صورت زیر است:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2$$

با کمک بسط تیلور ، معادله را می توان به صورت زیر نوشت:

$$E(u,v) \approx [u \quad v]\left(\sum \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix}\right)\begin{bmatrix} u \\ v \end{bmatrix}$$

حال می توان به تغییراتی به معادلات زیر رسید:

$$M = \sum w(x,y)\begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix} \quad E(u,v) \approx [u \quad v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$

چیزی که برای ما اهمیت دارد جهت تفاوت های مربع مشتق در اطراف هر پیکسل است که باید در همه ی جهات بزرگ باشد.

این مقدار را با بردار ویژه تعریف می کنیم.مقادیر ویژه مربوطه مقدار واقعی این افزایش ها را به ما می دهد. برای هر پنجره یک امتیاز R محاسبه می شود:

$$R = \det M - k(trace\, M)^2$$

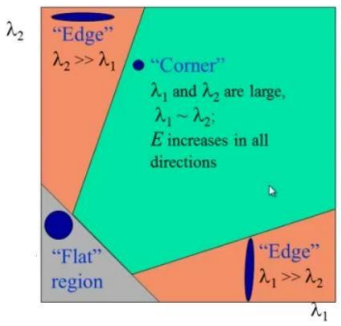
$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$

λ1 و λ2 مقادیر ویژه M هستند. بنابراین مقادیر این مقادیر ویژه تصمیم می‌گیرند که یک ناحیه گوشه، لبه یا مسطح باشد.

به طور کلی

- ✓ وقتی |R| کوچک است، که زمانی اتفاق می افتد که λ1 و λ2 کوچک هستند، منطقه مسطح است.
- ✓ وقتی R<0، که در λ2<<λ1 یا بالعکس اتفاق می افتد، منطقه یک یال است.
- ✓ وقتی R بزرگ باشد، زمانی که λ1 و λ2 بزرگ و λ1~λ2 باشند، این ناحیه یک گوشه است.



مراحل الگوریتم به طور خلاصه :

1- محاسبه ی مشتق افقی و عمودی :
مشتق های عمودی و عمودی عکس در هر نقطه را می توان باکمک عملگر هایی مانند عملگر sobelو یا prewitt در هر دو جهت x, y، محاسبه کرد.

2- محاسبه ی مربع مشتق ها : به علتی که در فرمول نهایی محاسبه ی هریس به آن احتیاج داریم.

3- اعمال اثر پنجره ی w، میزان تفاوت باید در همسایگی های نقاط پیکسل بررسی شود. به این منظوربرای هر پیکسل ، در یک پنجره ، برای هر یک از مربع مشتق ها مجموعشان را محاسبه می کنیم.

4- محاسبه ی مقادیر R : با جاگذاری در فرمول این مقدار به دست می آید.

5- حذف مقادیر غیر بیشینه و آستانه گذاری: یک local maximum به دست می آوریم و مقادیری که

6- بزرگ تر از مقدار آستانه ی ما هستند را نمایش می دهیم.

ب) پیاده سازی :

- پیش پردازش و smooth کردن تصویر :

```
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  h,w = gray.shape
  smoothed = cv2.GaussianBlur(gray,(5,5),0)
```

- مرحله اول و دوم: محاسبه مشتق و مربعات مشتق:

```
dx = cv2.Sobel(smoothed, cv2.CV_64F, 1, 0, ksize=3)
  dy = cv2.Sobel(smoothed, cv2.CV_64F, 0, 1, ksize=3)
  dx2=dx**2
  dy2=dy**2
  dxy=dx*dy
```

- مرحله سوم و چهارم:

با نوشتن الگوریتمی شبیه به کانولوشن:

```
for y in range(border, h-border):
    for x in range(border, w-border):
      Sx2 = np.sum(dx2[y-border:y+1+border, x-border:x+1+border])
      Sy2 = np.sum(dy2[y-border:y+1+border, x-border:x+1+border])
      Sxy = np.sum(dxy[y-border:y+1+border, x-border:x+1+border])
      I = np.array([[Sx2,Sxy],[Sxy,Sy2]])
      det=np.linalg.det(I)
      trace=np.matrix.trace(I)
      R=det-0.04*(trace**2)
      M[y-border, x-border]=R
```

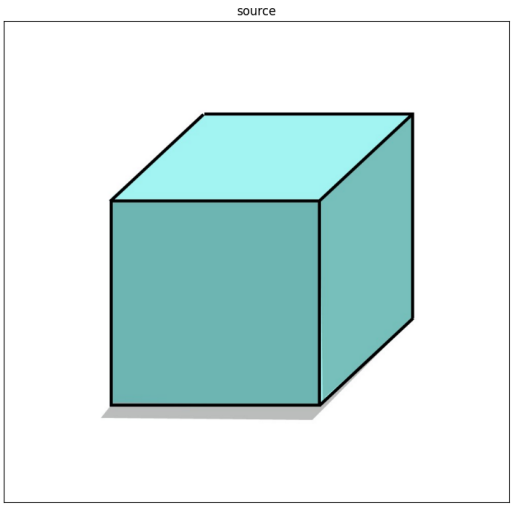
- مرحله پنجم:

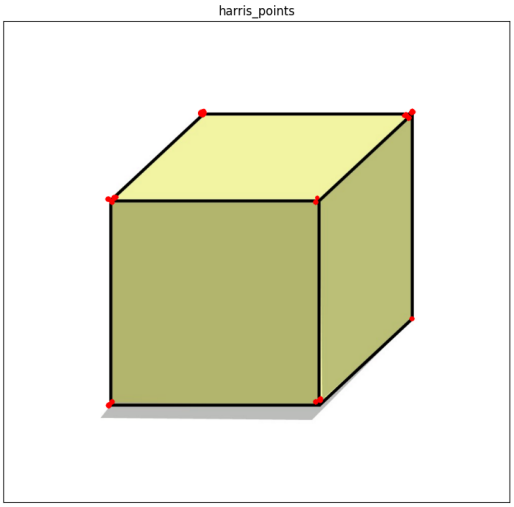
```
cv2.normalize(M, M, 0, 1, cv2.NORM_MINMAX)
  for y in range(border, h-border):
    for x in range(border, w-border):
      value=M[y, x]
```

```
if value> 0.20:
    cv2.circle(img,(x,y),4,(255,0,0) , -1)
```

در قسمت اخر نقاط را روی شکل اصلی با رنگ قرمز نمایش میدهیم.

نتیجه کد بالا با آستانه ی 0.2 و k = 0.04 :





در opencvبا کمک دستور harris :

Harris Corner Detector in OpenCV

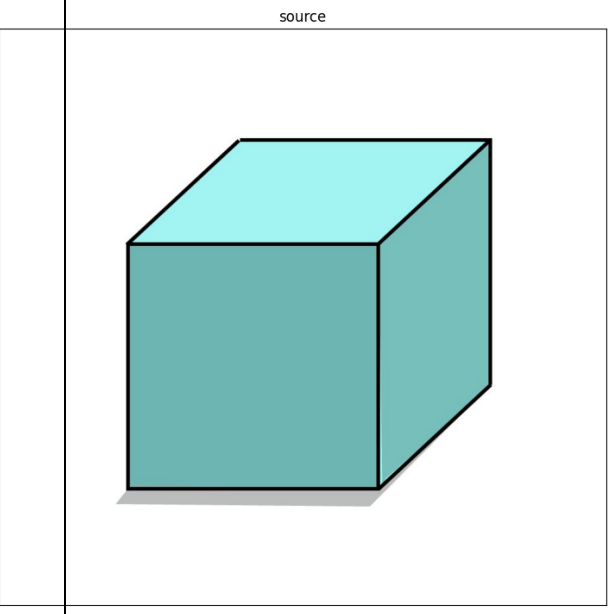
OpenCV has the function **cv.cornerHarris()** for this purpose. Its arguments are:

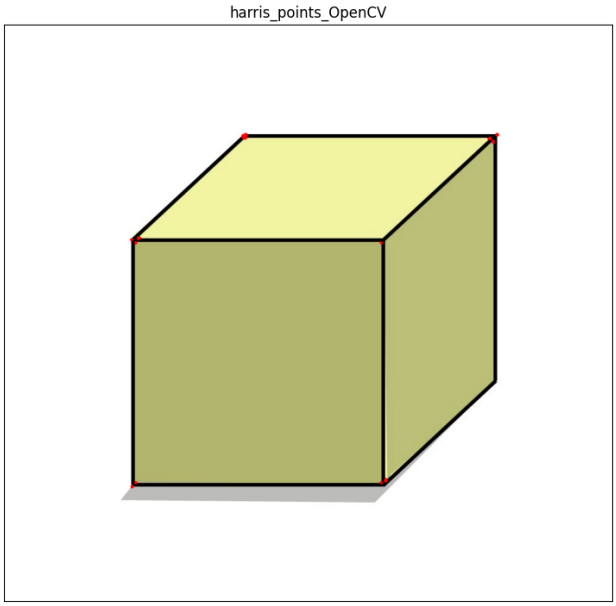
- **img** - Input image. It should be grayscale and float32 type.
- **blockSize** - It is the size of neighbourhood considered for corner detection
- **ksize** - Aperture parameter of the Sobel derivative used.
- **k** - Harris detector free parameter in the equation.

کد زده شده:

```
gray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
harris = cv2.cornerHarris(gray,2,5,0.04)
harris = cv2.dilate(harris,None)
thresh = 0.1*harris.max()
for j in range(0, image.shape[0]):
    for i in range(0, image.shape[1]):
        if(harris[j,i] > thresh):
            cv2.circle( image ,(i, j), 1, (255,0,0), -1)
```

نتیجه :





سوال 6) به صورت جدا گانه نوشته شده است.

لینک های استفاده شده :

https://www.tutorialspoint.com/how-to-change-the-contrast-and-brightness-of-an-image-using-opencv-in-python

/https://learnopencv.com/non-photorealistic-rendering-using-opencv-python-c

https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html

https://medium.com/data-breach/introduction-to-harris-corner-detector-32a88850b3f6

https://medium.com/mlearning-ai/facial-mask-overlay-with-opencv-dlib-4d948964cc4d

https://www.quora.com/Why-do-we-need-so-many-different-color-spaces