Amirreza Vishteh

HW3

Operation System

(1

بخش اول

K=1-130/255=0.49

Cmy=[1-k,1-k,1-k]-rgb/255=[0.31,0.24,0]

```
# BGR to RGB
rgb = np.array([50,70,130])

# RGB to CMYK

k = np.round((1 - (np.max(rgb)/255)),2)
# c = (1 - rgb[0] - k) / (1 - k)
# m = (1 - rgb[1] - k) / (1 - k)
# y = (1 - rgb[2] - k) / (1 - k)
rgbn=np.round(rgb/255,2)
cmyk = 100*(np.ones(3,dtype=np.uint8) - k - rgbn)

print("BGR color:", rgb)
print("CMYK color:", cmyk)
```

بخش2

برای رفتن به فضای دیگر از تابع cvtColor استفاده میکنیم

```
OpenCV Python binary extension loader
#########
result = cv2.cvtColor(image, cv2.COLOR_BGR2YCrCb)
```

```
#############
result = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

بخش3

برای نشان دادن هر یک از کانال های hsv از تابع split استفاده میکنیم

```
# Convert BGR to HSV
hsv_img = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)

# Split the HSV image into three separate channels
h_channel, s_channel, v_channel = cv2.split(hsv_img)
image_list = [[h_channel, 'h', 'img'], [s_channel, 's', 'img'], [v_channel, 'v', 'img']]
# Display each channel separately
plotter(image_list,1 , 3, True, 20, 10, '2AA')
```

بخش4

در این بخش برای نشان دادن تفاوت ها هر تصویر سیاه و سفید میکنیما که مقادیر روشنایی ان از 0 تا 255 است را برابر یک کانال تصویر نهایی میگذاریم در واقع چون دو تصویر و 3 کانال داریم یکی را به دو کانال تصویر مقصد میبریم

```
result = np.zeros((916, 921, 3), dtype=np.uint8)
result[:, :, 1] = image1[0:916, 0:921]
result[:, :, 2] = image1[0: 916, 0:921]
result[:, :, 0] = image2[0: 916, 0:921]

return result
```

ه)

بسته به کاربرد دارد مثلا در صفحه نمایش که ما رنگ را نمایش میدهیم از rgb بهره میبریم ولی جایی که میخواهیم مثل پرینتر از ماده جاذب رنگ برای نمایش رنگ استفاده کنیم باید از فضای cmyk استفاده کنیمتاذ با جذب رنگ ها انها را rgb نمایش دهد

2)

با استفاده از تابع ها زیر تصاویر را به هم متصل میکنیم که لیست عکس ها را میگیرد و تصویر نهایی به همراه موفقیت امیز بودن کتر یا نه را برمیگرداند

```
stitchy=cv2.Stitcher.create()
(dummy,output)=stitchy.stitch(images)
```

3)ابتدا با استفاده از توابع کتابخانه dlib چهره های تصویر را شناسایی میکنیم

```
# Your code #
result=face.copy()
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("/content/drive/MyDrive/shape_predictor_68_face_landmarks.dat")
```

Detector در واقع چهره های داخل تصویر را برای ما می یابد و سپس با استفاده از predictor چهره را مانند انچه در داک امده به صورت نقاط کلیدی بازمیگرداند فقط چون نوع بازگشتی از np.arrray() نیست پس از تابع تبدیلی که در داک امده استفاده میکنیم(shape_to_np)

```
for (i, rect) in enumerate(rects):
    # determine the facial landmarks for the face region
    # convert the facial landmark (x, y)-coordinates to
    # array
    shape = predictor(result, rect)
    shape = shape_to_np(shape)
```

حال 4 نقطه را روی ماسک و تصویر انتخاب میکنیم و با استفاده از توابع prespective انها را منطبق میکنیم

صورت:30,15,9,3

در paint نقاط متناظر را مختصاتش را به دست اورده و می گذاریم

```
src= np.float32([[611,105], [1164,270],[617,665],[45,240]])
dest=np.float32([shape[29], shape[14], shape[8],shape[2]])
M = cv2.getPerspectiveTransform(src, dest)
result = cv2.warpPerspective(mask, M, (result.shape[1], result.shape[0]))
result= cv2.bitwise_or(face,result)
```

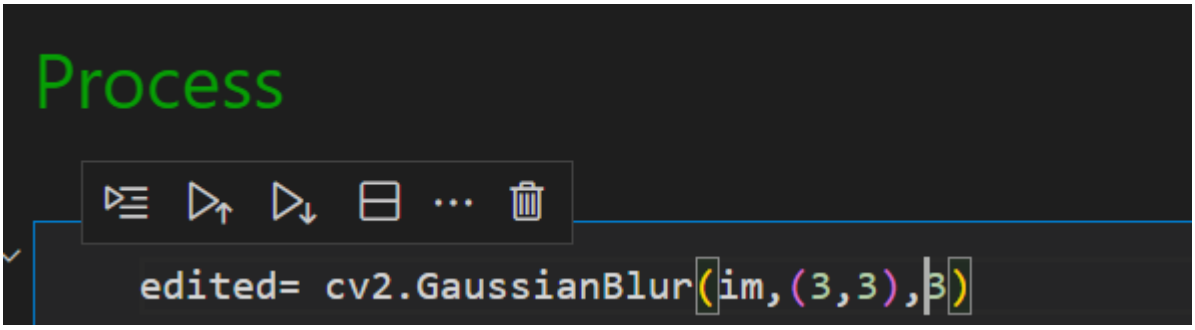
در واقع با prespective تصویر ماسک را به مکان درست میبریم و بعد با تابع bitwise_or تصویر ماسک را روی تصویر می اندازیم

ب)

روش بسیار مناسب برای تشخیص اجسام روش deep learning است که ویژگی ها را بسیار خوب تخمین میزند و با انها به خوبی و دقیق اجسام را تشخیص میدهند

4)

ابتدا بعد از خواندن عکس gray scale با استفاده از gaussianBlur یا یک فیلتر پایین گذر است نویز تصویر را میگیریم



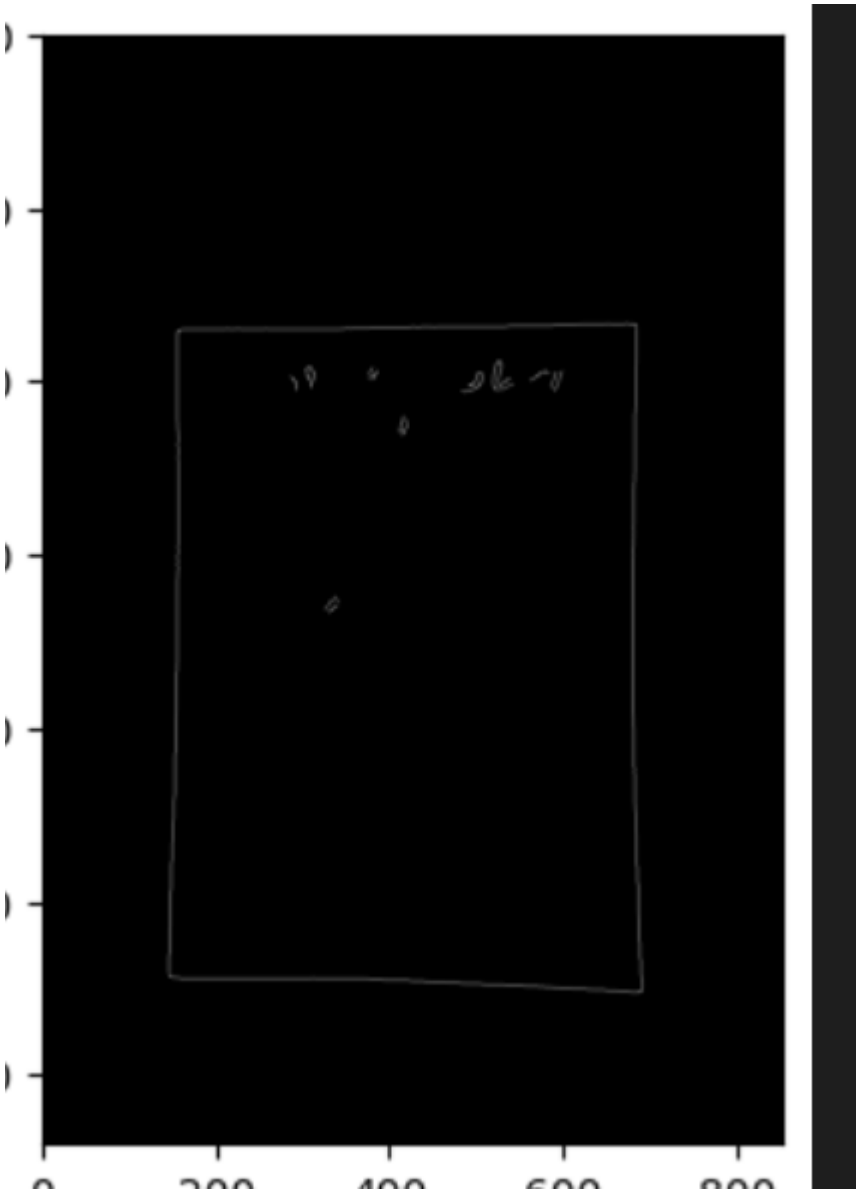
اگر از سایز بزرگتری برای پنجره استفاده کنیم مانند میانگین گیر تصویر تاری زیادی خواهد داشت.
در اینجا من به دو روش سوال را حل کردم یکی به کمک adaptiveThereshold برای یافتن لبه و یکی روش عادی در روش عادی بعد از یافتن لبه و contour درست که در واقع بزرگترین طول را دارد

حال با استفاده از تابع canny لبه های تصویر را می یابیم .

```
edited= cv2.GaussianBlur(im,(25,25),3)
plt.imshow(edited, cmap='gray')
plt.show()

edgeimage= cv2.Canny(edited,100,110)
kernel=np.ones((9,9))
edgeimage=cv2.dilate(edgeimage,kernel)
plt.imshow(edgeimage,cmap="gray")
plt.show()
```

با اعداد تابع canny انقدر بازی میکنیم تا نوشته های داخل تصویر را به عنوان contour شناسایی نکند

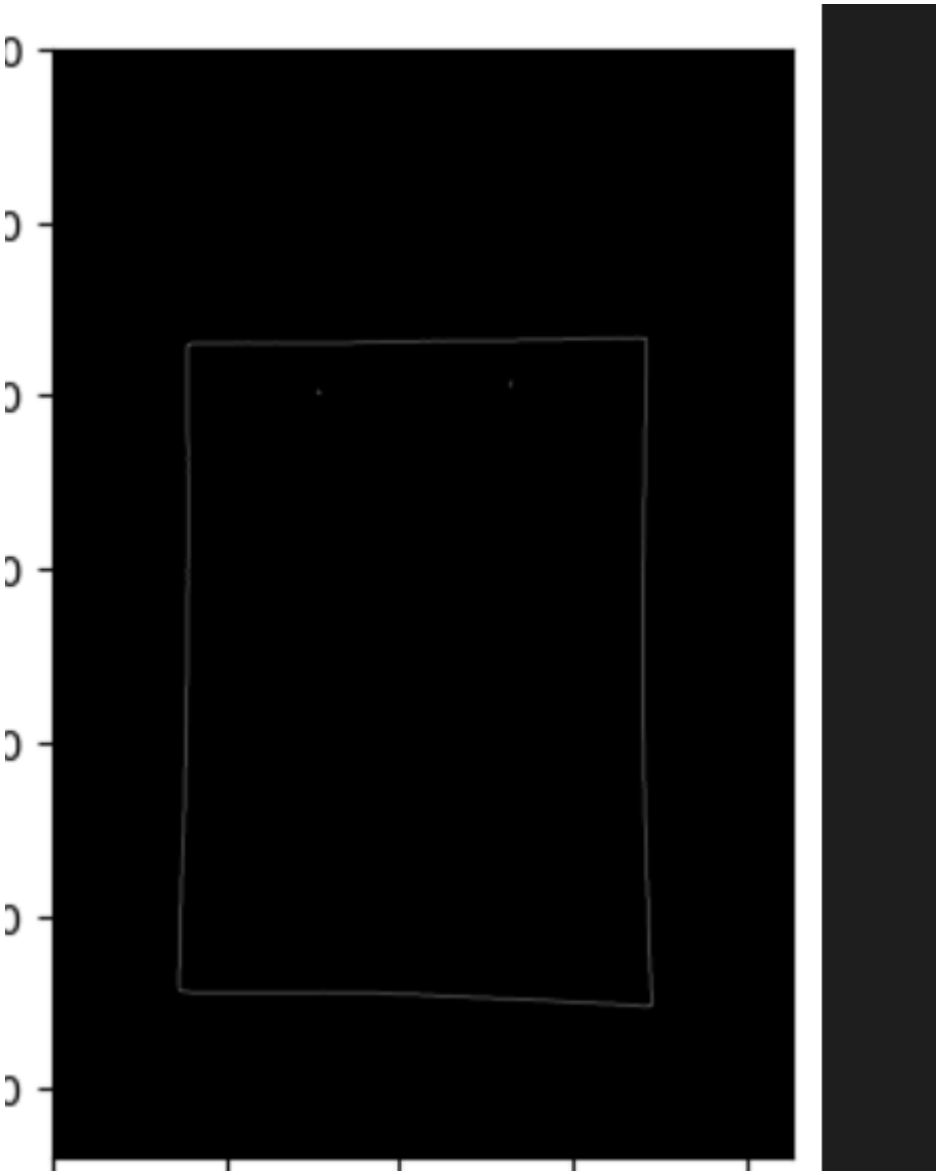


این عکس دارای نوشته ها است با این canny

```
edgeimage= cv2.Canny(edited,30,50)
```

حال با همان مقادیر خوب می ایییم و گوشه ها را میابیم :

دو حد ما یکی 100 و دیگری 110 میشود تا هر چیزی را لبه در نظر نگیرد



نتیجه لبه یابی همانطور که میبینیم خوب است و گوشه ها نیز به درستی به دست می ایند

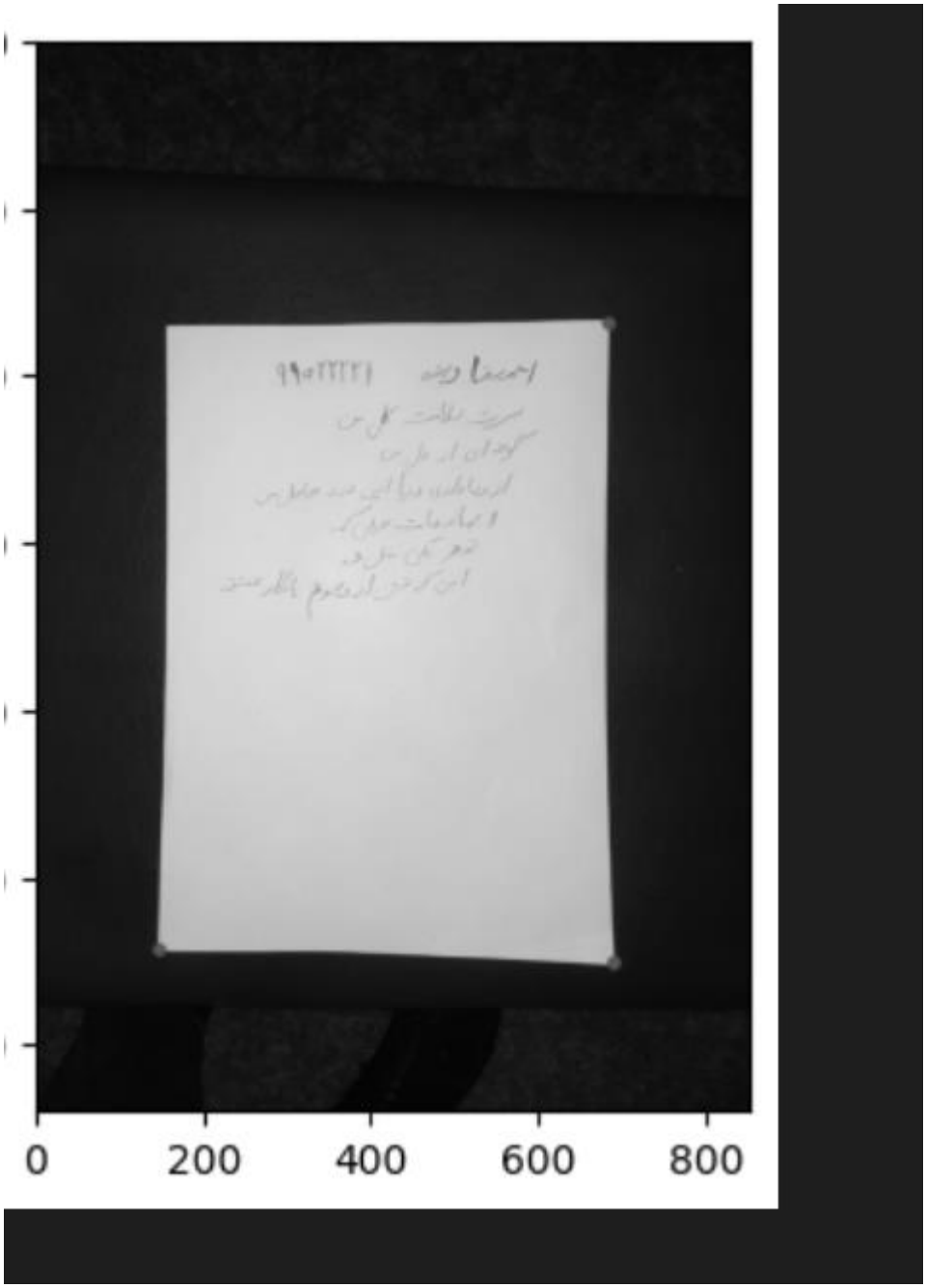
```
contours,_=cv2.findContours(edgeimage,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

max_con = contours[0]
max_con_A = len(max_con)
for cnt in contours:
    if len(cnt) > max_con_A:
        max_con_A = len(cnt)
        max_con = cnt
```

با استفاده از تابع findContours که ورودی اش یک عکس باینری است contour های تصویر را می یابیم و سپس همانطور که دیده میشود بزرگترین انها را می یابیم که در واقع برگه ما میشود

```
epsilon = 0.01 * cv2.arcLength(
                                max_con, True)
approx = cv2.approxPolyDP(max_con, epsilon, True)
```

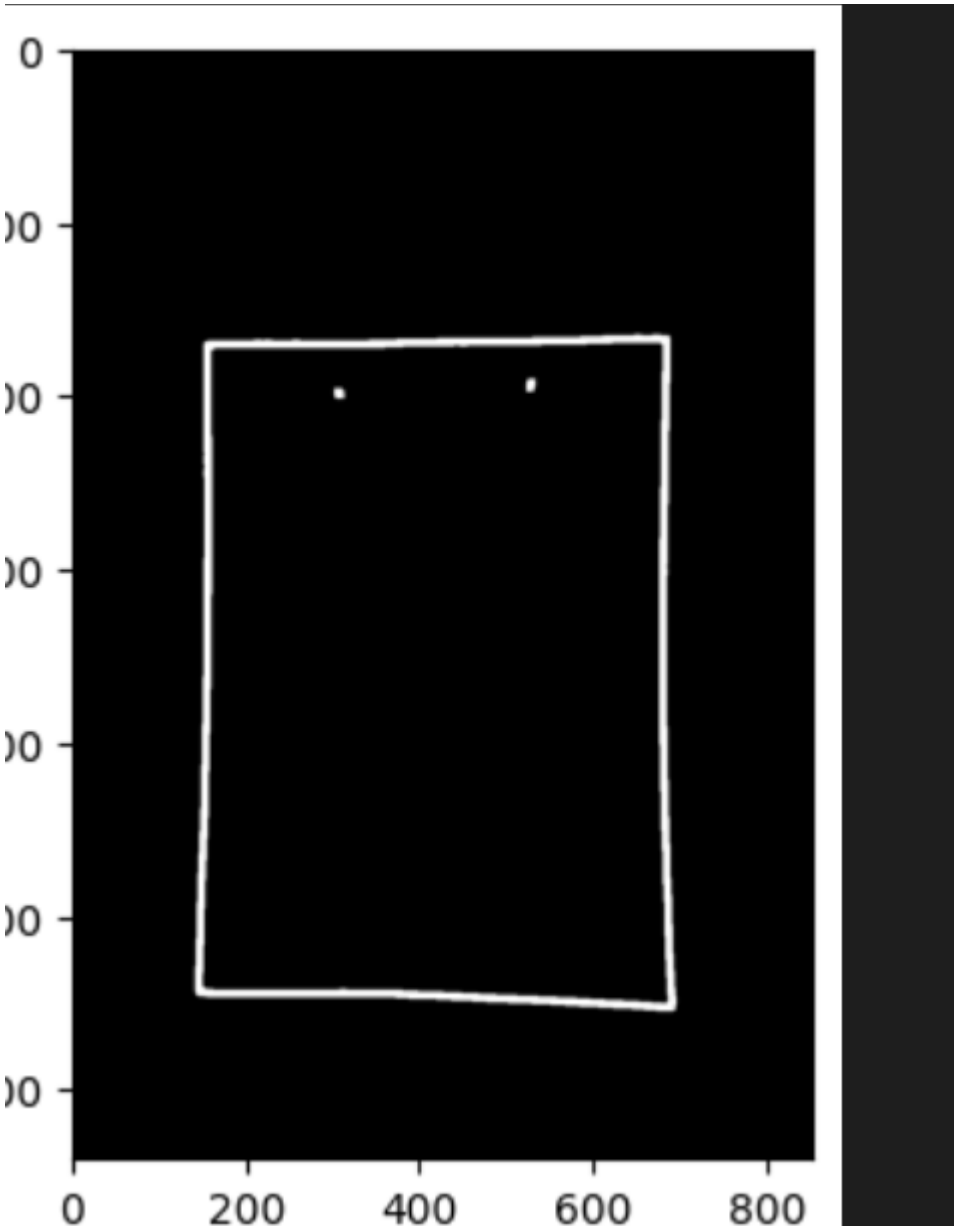
```
for point in approx:
    cv2.circle(res, tuple(point[0]), 5, (80, 0, 0), 5)
```

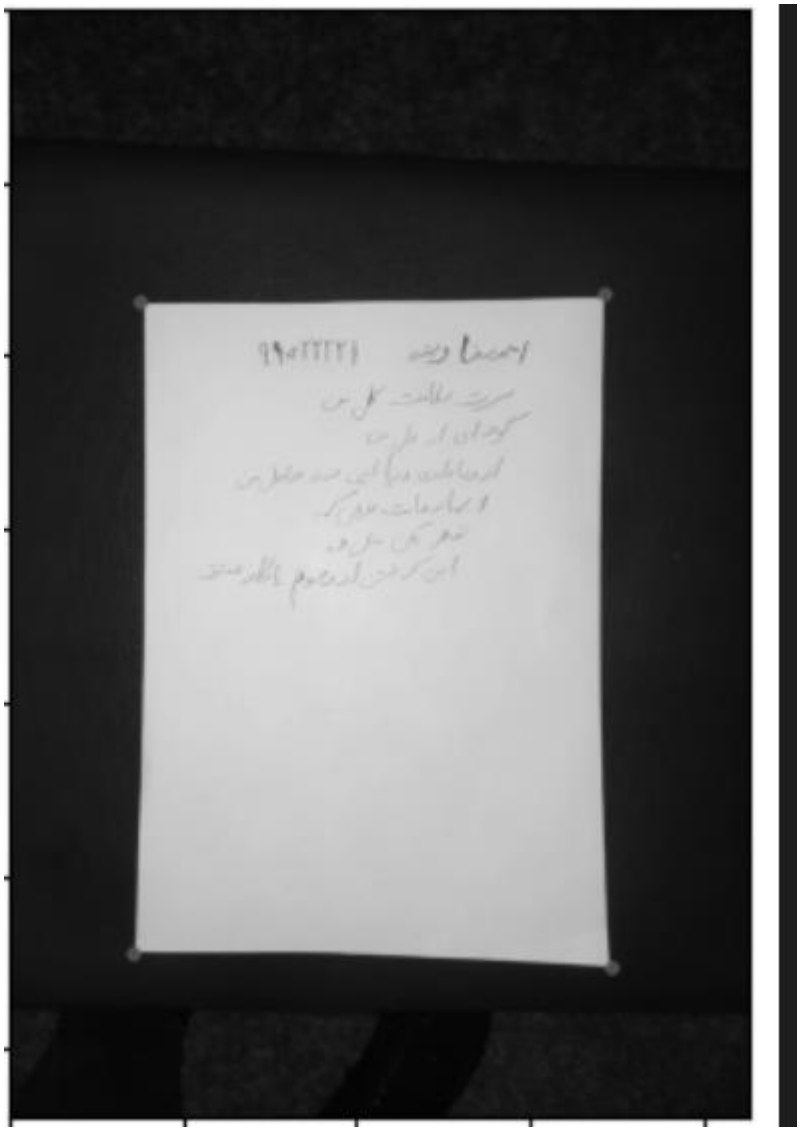


ابتدا فقط 3 گوشه پیدا میکند

بنابراین ما ابتدا کمی لبه ها را کلفت تر میکنیم و بعد گوشه ها را می یابیم

```
edgeimage= cv2.Canny(edited,100,110)
kernel=np.ones((9,9))
edgeimage=cv2.dilate(edgeimage,kernel)
plt.imshow(edgeimage,cmap="gray")
plt.show()
```

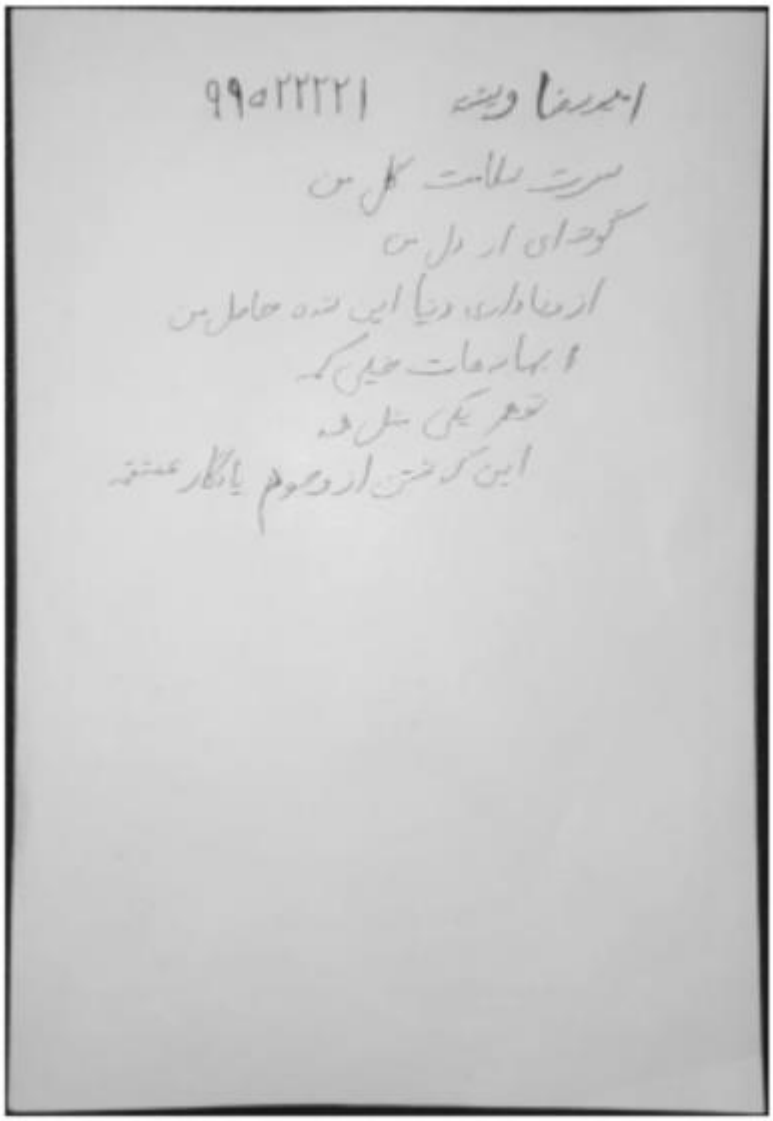




حال به سراغ برش تصویر میریم با استفاده از تابع گفته شده میرویم

```
pts1 = np.float32([approx[0], approx[1], approx[2], approx[3]])
# print(approx[0],approx[1],approx[2],approx[3])
# print("---------")
width, height = 2400, 3500
pts2 = np.float32([[width, 0], [0, 0], [0, height], [width, height]])
M = cv2.getPerspectiveTransform(pts1, pts2)
output_img = cv2.warpPerspective(res, M, (width, height))
```

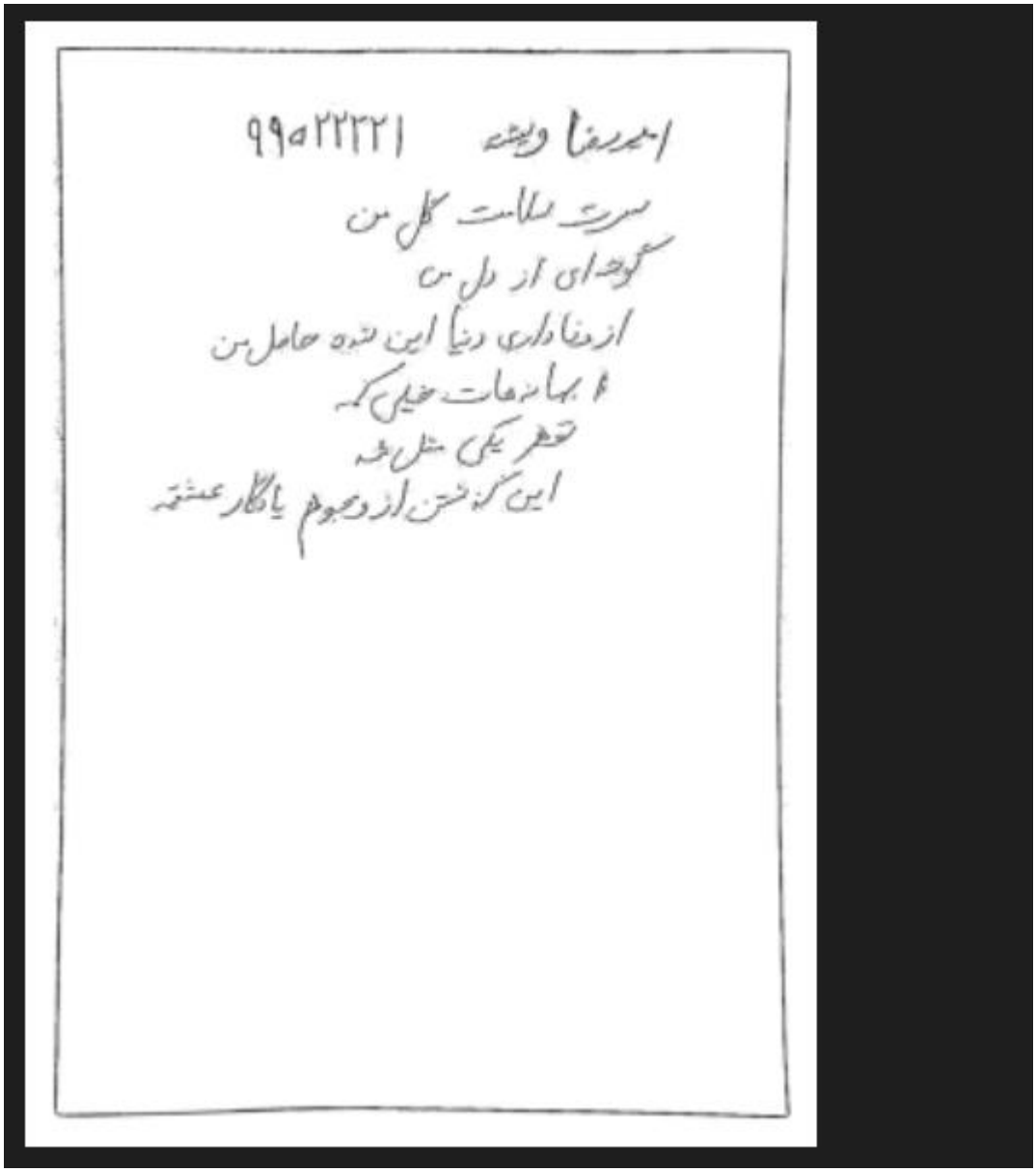
در اینجا ابتدا نقاط گوشه ای را پرینت کردم تا بفهمم هر کدام معادل کدام گوشه تصویر اصلی اند و سپس با استفاده از تابع های getPerspectiveTransform که ماتریس تبدیل 9 عضوی داخل اسلاید را میدهد را می یابیم warpPerspective با استفاده از ماتریس به دست اماده تصویر را در واقع معادل تصویر برگه میکنیم



حال برای بخش امتیازی یا همون د:

می ایم و با adaptivetreshold کیفیت تصویر را بالا میبریم

```
out=output_img.copy()
out = cv2.adaptiveThreshold(out ,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY,21,6)
imshow(out)
```



در نوع دوم حل من از

Adaptive tereshold هم استفا ه کردم

5)

الف)

در روش harris ما باید ماتریس زیر زا بسازیم که هر عضو ان خود یک ماتریس است

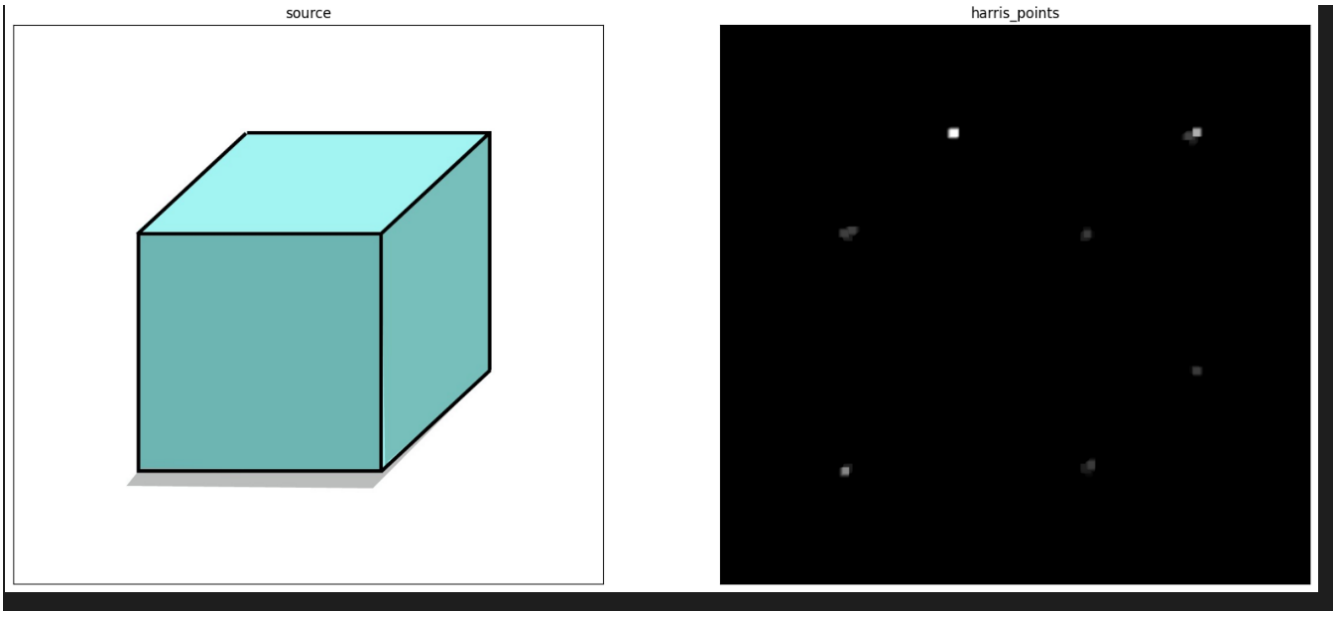
$$M = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

برای ساخت ان ابتدا مشتق افقی و عمودی را با استفاده از sobelx,sobely بدست می اوریم که تابع اماده انها وجود دارد نیاز به convolve انها برای به دست اوردن مشتق ها نیست

```
ix=cv2.Sobel(resultgray,cv2.CV_64F,1,0,ksize=3)
iy=cv2.Sobel(resultgray,cv2.CV_64F,0,1,ksize=3)
```

سپس مقادیر توان 2 مشتق عمودی و افقی و حاصل ضرب انها را به دست اورده و در ماتریس میگذاریم حال میدانیم که اگر این ماتریس دو مقدار ویژه بزرگ داشته باشد انگاه یا لبه یا گوشه داریم که برای تشهیص گوشه میتوان این کار را کرد که اگر حاصل ضرب مقادیر ویژه را از توان 2 حاصل جمع انها کم کنیم و مقدار بزرگی باقی بماند انگاه حتما گوشه است ولی اگر تنها توان حاصل ضرب انها مقدار بزرگی بود انگاه فقط یک مقدار ویژه دارد و لبه است

بنابراین بعد از به دست اوردن مقادیر ماتریس روی انها یک فیلتر گاووسی میزنیم و بعد ماتریس R را محاسبه میکنیم در اخر برای بهتر شدن نقاط گوشه میتوان یک local maximum گرفت که عملا هر گوشه را به یک نقطه تبدیل میکند که برای بزرگتر کردنش ما یک dilation میزنیم



6)هر 3 روش برای یافتن نقاط کلیدی اند

Sift:

در این الگوریتم از difference-of-Gaussian استفاده میشود که اکسترمو های محلی را بیابد و با لاپلاسین نقاط کلیدی را به دست می اورد و نسبت به scale و چرخش نقاط کلیدی ان به درستی عمل میکنند اما order محاسباتی بالایی در کنار دقت بالا دارد

SURF:

این روش از فیلترجعبه ای در یافتن نقاط کلیدی خود استفاده میکند و سرعت بیشتری دارد اما مانند روش قبل نسبت به scale شدن مقاوم نیست و دقت کمتری دارد

ORB:

این الگوریتم از FAST, BRIEF برای یافتن نقاط کلیدی خود استفاده میکند نسبت به روش SURF دقت بیشتر و نسبت به SIFT دقت بالا تری دارد و نسبت به چرخش و scale شدن مقاوم است

FAST برای یافتن نقاط کلیدی و BRIEF الگوی پیکسل‌ها را در patch مقایسه می‌کند