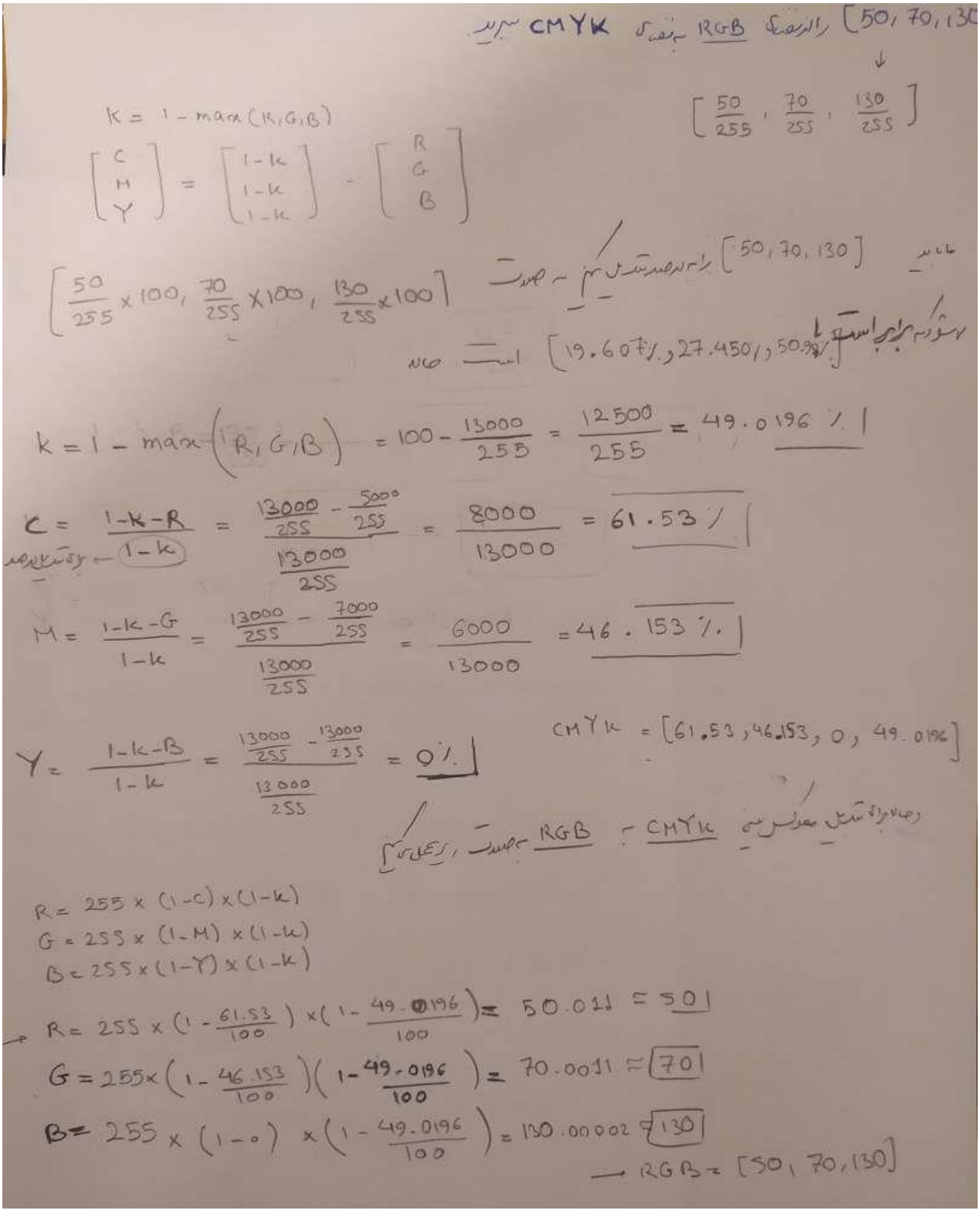
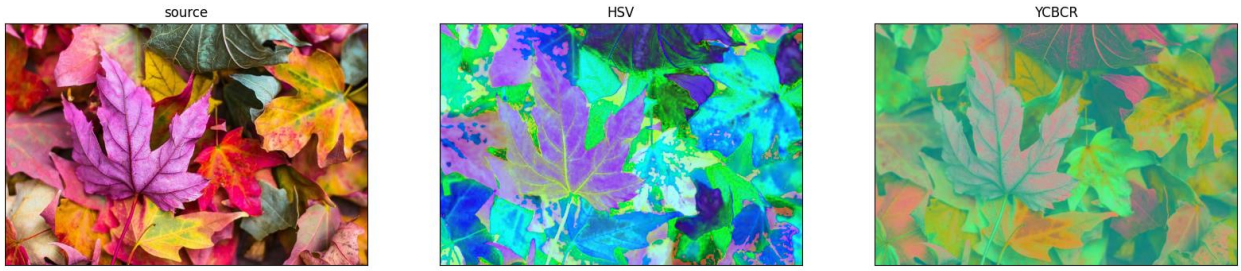
سوال اول

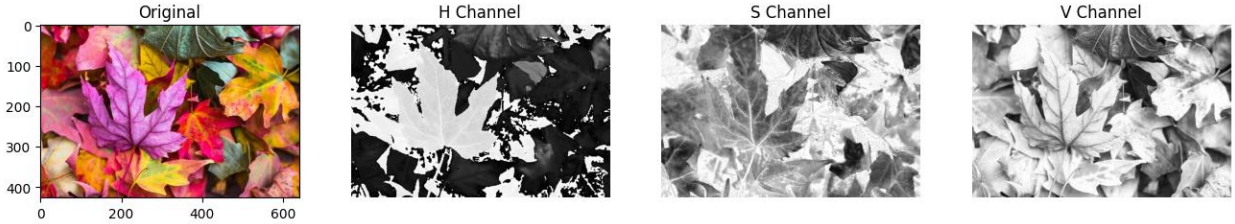
الف)هم در نوتبوک انجام شده است و هم به صورت دستی که جواب آن مطابق زیر است:



ب)این کار را در نوتبوک با تابع cvtcolor انجام دادم و خروجی به صورت زیر شد.



ج)با استفاده از split هر کدام از کانال های H,S,V را جدا می کنیم که تصویر به صورت زیر میشود

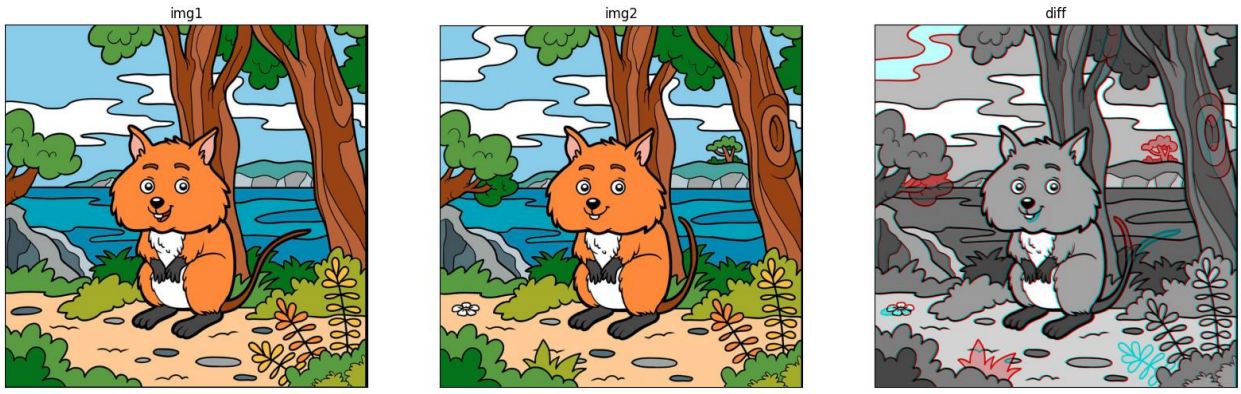


همانطور که مشاهده میکنید کانال v که مربوط به سطح روشنایی هست جاهایی که سطح روشنایی زیاد هست رنگ به سمت سفید هست و جاهایی که سطح روشنایی کم هست به سمت سیاه هست

برای کانال s که معیاری از رقیق شدگی رنگ خالص با نور سفید است مشاهده میکنیم مثلا برای گوشه سمت چپ که یک سمتی ان سفید هست در عکس سیاه هست یا قسمت های قرمز رنگ که s انها برابر با یک هست در عکس رو به سفیدی هست.

و کانال H که مربوط به اصل رنگ هست که اول یک زاویه بین ۰ تا ۳۶۰ درجه هست و بعد برای اینکه ۸ بیتی تصویر ما بشود در 255/360 ضرب می کنیم به طور کلی، مقادیر کمتر از ۳۰ و بیشتر از ۳۳۰ برای کانال H به رنگ قرمز نزدیک هستند، مقادیر بین ۳۰ تا ۹۰ برای کانال H به رنگ زرد نزدیک هستند، مقادیر بین ۹۰ تا ۱۵۰ برای کانال H به رنگ سبز نزدیک هستند، مقادیر بین ۱۵۰ تا ۲۱۰ برای کانال H به رنگ آبی نزدیک هستند، مقادیر بین ۲۱۰ تا ۲۷۰ برای کانال H به رنگ ارغوانی نزدیک هستند، و مقادیر بین ۲۷۰ تا ۳۳۰ برای کانال H به رنگ قرمز دوباره نزدیک هستند.بنابراین، با نمایش کانال H به عنوان تصویر خام، می‌توانید توزیع رنگی تصویر را در این رنگ‌های اصلی بررسی کنید. به عنوان مثال، اگر کانال H برای یک تصویر، مقادیری نزدیک به ۱۲۰ داشته باشد، می‌توان نتیجه گرفت که بیشترین رنگ تصویر به رنگ سبز نزدیک است.

د)برای اینکه تفاوت دو تصویر را نشان بدهم تصویر اول کانال قرمز result گذاشتم و تصویر دوم را کانال آبی و سبز result گذاشتم و خروجی به صورت زیر شد که تفاوت ها را به خوبی نشان میدهد



ه) چرا از چند فضای رنگی استفاده می کنیم؟

استفاده از چند فضای رنگی در پردازش تصویر به دلایل زیر انجام می‌شود:

۱ .توصیف رنگ‌ها: هر فضای رنگی به روش خاص خود رنگ‌ها را توصیف می‌کند. برای مثال، در فضای رنگی RGB، رنگ‌ها با ترکیبی از سه رنگ قرمز، سبز و آبی توصیف می‌شوند، در حالی که در فضای رنگیHSV ، رنگ‌ها با استفاده از سه پارامتر اصل رنگ، رقیق شدگی رنگ خالص با نورسفید و شدت روشنایی توصیف می‌شوند. با استفاده از چندین فضای رنگی، می‌توانیم رنگ‌ها را به چندین روش توصیف کنیم و برای هر کاربرد، فضای رنگی مناسب را انتخاب کنیم.

۲ .پردازش تصویر: برخی عملیات پردازش تصویر، در یک فضای رنگی بهتر عمل می‌کنند. برای مثال، در فضای رنگیHSV ، سطوح روشنایی را می‌توان به راحتی تشخیص داد و از آن‌ها برای انجام عملیات‌هایی مانند تشخیص لبه‌ها استفاده کرد. همچنین، در فضای رنگیLab ، تفاوت‌های رنگی در سطوح روشنایی و رنگ را به خوبی نشان می‌دهد و برای تشخیص الگوها و برجستگی‌ها مفید است.مثلا میخواهیم روشنایی یک تصویر بیشتر کنیم کافی است پارامتر L در HSL را بیشتر کنیم.

۳ .نمایش تصویر: برخی فضاهای رنگی برای نمایش تصاویر بهتر عمل می‌کنند. برای مثال، فضای رنگی RGB برای نمایش تصاویر در دستگاه‌های دیجیتالی استفاده می‌شود، در حالی که فضای رنگی CMYK برای چاپ تصاویر مناسب است.

بنابراین، استفاده از چندین فضای رنگی در پردازش تصویر، نه تنها بهبود عملکرد الگوریتم‌های پردازش تصویر را فراهم می‌کند، بلکه به ما امکان می‌دهد تصاویر را با روش‌های مختلف توصیف کرده و نمایش دهیم.

سوال دوم

الگوریتم پانوراما دارای چهار مرحله اصلی است:

۱.تشخیص نقاط کلیدی و استخراج توصیفگرهای ثابت محلی

۲.پیدا کردن توصیفگرهای منطبق بین تصاویر ورودی

۳.محاسبه ماتریس هموگرافی با استفاده از الگوریتم RANSAC

۴.ماتریس هموگرافی بر روی تصاویر اعمال میشود تا تصاویر را ادغام کند.

در ابتدا همه تصاویر را در یک سطر نمایش می دهیم.در واقع همه تصاویری که دارای پسوند jpg. در فولدر Q2 هستند نشان می دهیم.

```python
path="images/Q2/*.jpg"
images=[cv2.imread(image) for image in glob.glob(path)]
plt.figure(figsize=(30,4))
for i in range(1, 8):
    plt.subplot(1, 8, i)
    plt.imshow(cv2.cvtColor(images[i-1], cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title(f'image{i}')
```

✓ 4.9s Python

ودر ادامه از create_Stitcher استفاده میکنیم تا یک آبجکت پانوراما بسازیم برای این کار باید ورودی آن را ۰ بدهیم. اگر استتوس صفر باشد یعنی عملیات تشکیل تصویر پانوراما موفقیت آمیز بوده است. برای همین این شرط را چک میکنیم و اگر برابر با صفر بود تصویر تشکیل شده را نمایش میدهیم

سوال سوم

الف)در ابتدا با استفاده از dlib.get_frontal_face_detector() یک شی تشخیص دهنده صورت ایجاد میکنیم که قابلیت تشخیص صورت را دارد. سپس با استفاده از تابع dlib.shape_predictor، یک شیء پیش بینی کننده نقاط کلیدی صورت ایجاد میکنیم. این شیء می تواند با استفاده از یک مدل آموزش دیده شده ایجاد شودکه نقاط کلیدی صورت را در یک تصویر شناسایی می کند. فایل `shape_predictor_68_face_landmarks.dat` یک مدل پیش بینی کننده نقاط کلیدی صورت است که توسط DLIB آموزش داده شده است. این مدل شامل ۶۸ نقطه کلیدی مشخص در یک صورت است که در تصویر شناسایی می شوند.

```
# Initialize face detector and shape predictor from dlib
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
```

در ادامه، با استفاده از شیء تشخیص دهنده صورتی که در مرحله قبل ایجاد شده است، به دنبال صورت در تصویر می گردد. با استفاده از این تابع، می توان اطلاعاتی مانند موقعیت و اندازه صورت در تصویر را دریافت کرد. خروجی این تابع، یک لیستی از شیء های مستطیلی است که هر کدام شامل موقعیت و اندازه یک صورت در تصویر هستند.در ادامه برنامه، با استفاده از شیء پیش بینی کننده نقاط کلیدی صورتی که در مرحله قبل ایجاد شده است، به دنبال نقاط کلیدی صورت در تصویر می گردد. برای این کار، این تابع باید به عنوان ورودی تصویر صورت مورد نظر و یک شیء مستطیلی که شامل موقعیت و اندازه صورت است، را دریافت کند. با استفاده از این تابع، می توان نقاط کلیدی مختلفی مانند نقاط چشم، بینی و دهان را در یک تصویر شناسایی کرد.

```
# Detect face landmarks
faces = detector(face)
landmarks = predictor(face, faces[0])
```

و در ادامه موقعیت های مربوط به ترتیب چانه و بینی و گونه سمت چپ و گونه سمت راست برای ماسک تصویر و خود تصویر صورت انخاب میکنیم برای ماسک را با توجه به ازمون خطا رو خود شکل ماسک به دست اوردم.

```
#position chin, noise, left and right cheeks on the mask
mask_points = np.array([(620, 700),
                        (620, 120),
                        (62, 300),
                        (1150,300)], dtype=np.float32)
# Select corresponding landmarks on the mask image
face_points = np.array([(landmarks.part(9).x, landmarks.part(9).y),
                        (landmarks.part(28).x, landmarks.part(28).y),
                        (landmarks.part(2).x, landmarks.part(2).y),
                        (landmarks.part(14).x, landmarks.part(14).y)], dtype=np.float32
```

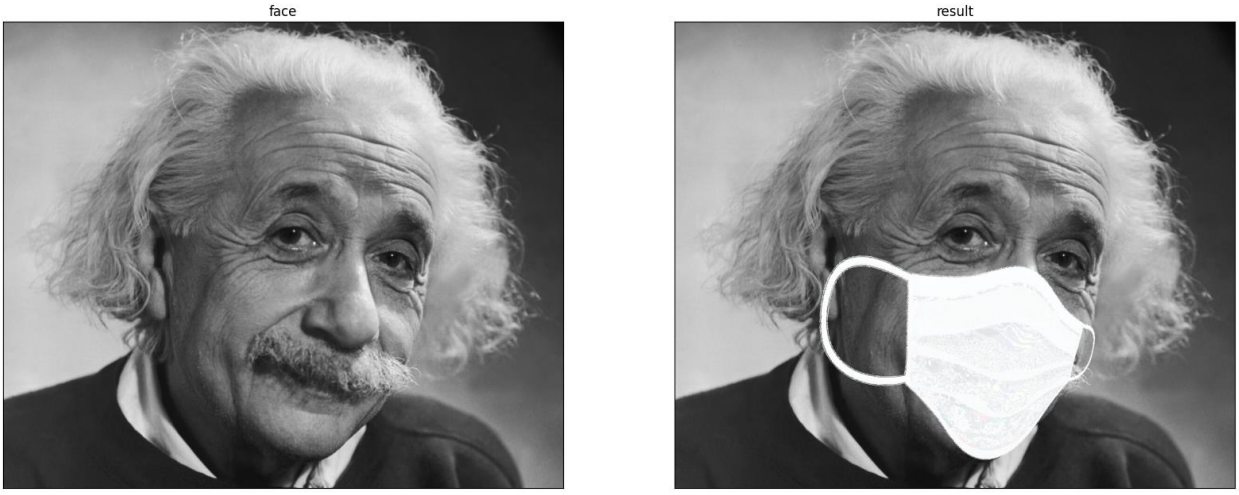
در ادامه باید ماتریسی که نقاط کلیدی ماسک را به نقاط کلیدی صورت تبدیل میکند به دست بیاوریم.

```
# Compute matrix
M = cv2.getPerspectiveTransform(mask_points, face_points)
```

و بعد با استفاده از ماتریس به دست امده تصویر خروجی ماسک را بر روی صورت مطابق کد زیر به دست می اوریم.

```
# Transform the mask image to fit onto a face using a perspective transformation matrix
mask_warped = cv2.warpPerspective(mask, M, (face.shape[1], face.shape[0]))
```

و در نهایت با استفاده از add یا bitwise_orدو تصویر را به هم میچسبانیم و خروجی به صورت زیر میشود



ب) درباره ی تسک detection landmark face در بینایی کامپیوتر و یکی از راهکارهای موجود توضیح دهید.

تشخیص محل و نشانه‌های صورت (Face Landmark Detection) یکی از وظایف اصلی در حوزه بینایی کامپیوتر و پردازش تصویر است. در این وظیفه، هدف شناسایی موقعیت نشانه‌های مختلف صورت مانند چشم، بینی، دهان، گوش و... است. این تکنیک در بسیاری از برنامه‌های بینایی کامپیوتری مانند تشخیص احساسات، تشخیص جنسیت و سن، تشخیص هویت و غیره مورد استفاده قرار می‌گیرد.یکی از راهکارهای موجود برای انجام تشخیص محل و نشانه‌های صورت، استفاده از شبکه‌های عصبی کانولوشنی (Convolutional Neural Networks) است. در این راهکار، ابتدا یک شبکه عصبی کانولوشنی برای تشخیص محل صورت ایجاد می‌شود. سپس با استفاده از یک شبکه عصبی دیگر به نام شبکه تشخیص نشانه‌های صورت (Facial Landmark Detection Network)، موقعیت نشانه‌های مختلف صورت مانند چشم، بینی و دهان تشخیص داده می‌شود.برای آموزش شبکه تشخیص نشانه‌های صورت، از مجموعه داده‌هایی مانند AFLW و W۳۰۰ استفاده می‌شود. این مجموعه داده‌ها حاوی تصاویری از افراد با موقعیت نشانه‌های صورت مشخص شده هستند. در فرآیند آموزش، شبکه تشخیص نشانه‌های صورت با استفاده از مجموعه داده‌های آموزشی، به یادگیری موقعیت دقیق نشانه‌های صورت می‌پردازد.از دیگر راهکارهای موجود برای انجام تشخیص محل و نشانه‌های صورت، استفاده از الگوریتم‌های مبتنی بر ویژگی (Feature-based Algorithms) مانند آلگوریتم Active Shape Model (ASM) و آلگوریتم Active Appearance Model (AAM) است. در این راهکارها، از ویژگی‌های مختلفی مانند رنگ، شکل و بافت صورت برای تشخیص محل و نشانه‌های صورت استفاده می‌شود.

سوال چهارم

الف)تصویر را سیاه سفید کردم و بعد Bilateral filter را بر ان اعمال کردم چون همانطور در تمرین پیش داشتیم از فیلتر گوسی بهتر هست و تفاوت های رنگی بین پیکسل های همسایه را نیز در نظر میگیرد.پارامترهای bilateral را طبق زیر در نظر گرفتم

```
# Bilateral filter
bilateralimg = cv2.bilateralFilter(grayimg, 5, 10, 10)
imshow(bilateralimg)
```

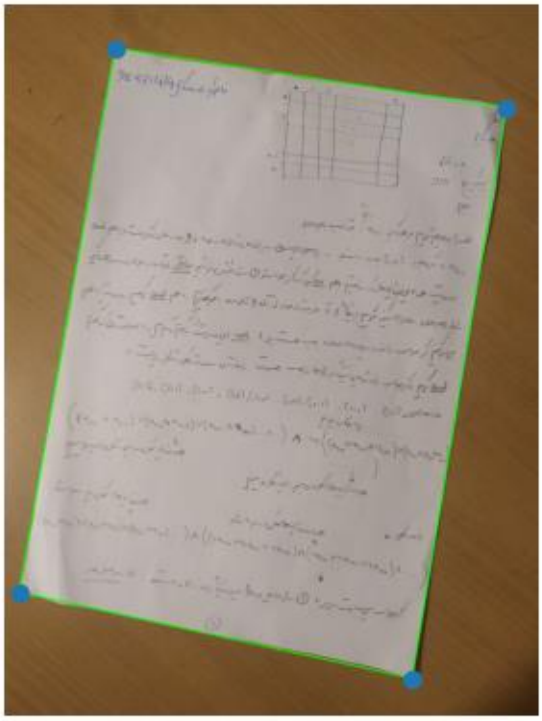
پارامتر مربوط به اندازه پنجره را ۵ گرفتم چون هر چقدر که سایز پنجره بیشتر باشد نویز بیشتری از عکس حذف میشود ولی تصویر تارتر میشود من با توجه به خروجی مرحله بعد یعنیcanny و ازمون خطا به نظرم ۵ عدد خوبی بود پارامتر مربوط sigmacolor را ۱۰ گذاشتم که ان را نیز با ازمون خطا گذاشتم و پارامتر sigmaspace را ۱۰ گذاشتم چون این مقدار هر چقدر کوچکتر باشد نویز کمتری میگیرد و مقادیر canny را نیز مطابق زیر گذاشتم

```
# canny
```

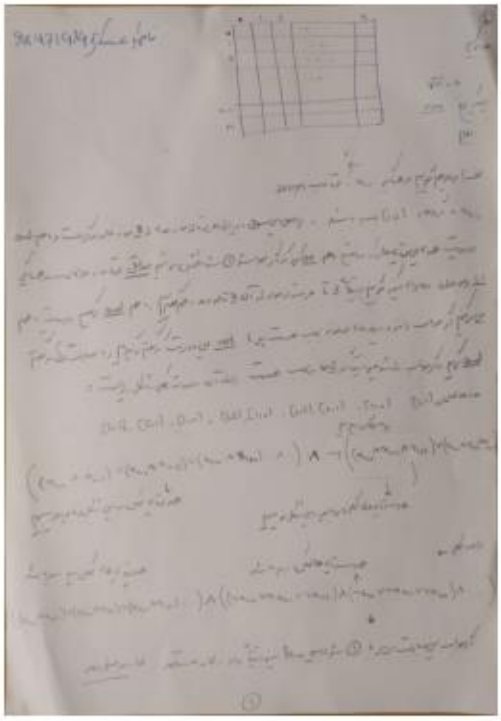
```
edgesimg = cv2.Canny(bilateralimg,10,150)
imshow(edgesimg)
```

مشاهده میکنید threadshod ها را ۱۰و۱۵۰ گذاشتم که با ازمون خطا و با توجه به خروجی مقادیر بهتری بود تصویر خروجی به صورت زیر است:

ب)برای پیدا کردن مرزهای تصویر از cv2.findContours() استفاده می کنیم. و در ادامه بر روی کانترها یک حلقه میزنیم و با استفاده cv2.approxPolyDP() اگر چهارمرز پیدا کنه مرزها را رمز میکنیم و مختصات های گوشه را نیز در یک ارایه برای قسمت بعدی ذخیره میکنیم و در شکل نشان می دهیم. شکل ان مطابق زیر است

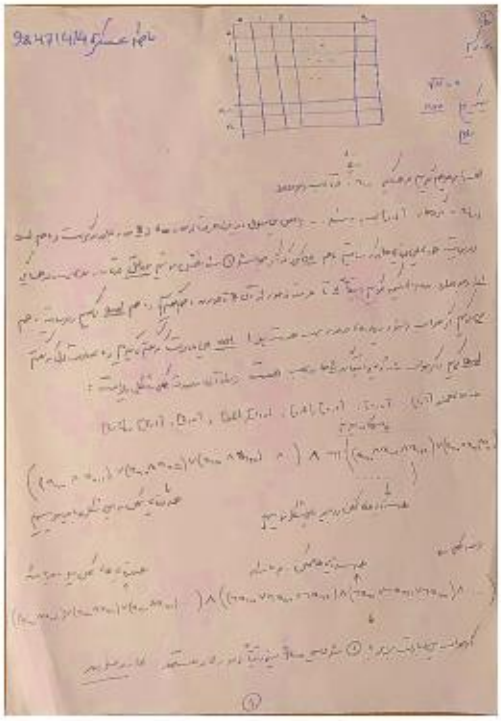


ج) با استفاده از نقاط گوشه پیدا شده هر دو طول کاغذ و هر دو عرض کاغذ را هم حساب میکنیم. سپس بیشترین مقداربین طول ها را و بیشترین عرض بین عرض ها را انتخاب میکنیم تا تصویر آسیب کمتری ببیند. سپس یک آرایه ی جدید از نقاط جدید با توجه به طول و عرض انتخاب شده میسازیم و به عنوان تارگت به getperspectivetrasformمیدهیم تا ماتریس تبدیل را پیدا کنیم و سپس به تابع wrapperspective میدهیم خروجی به صورت زیر است:



د)برای بهبود کیفیت تصوبر میتوان از مشتق تصویر استفاده کرد تا جاهایی که جهش رنگی داریم مشخصتر بشوند و هم چنین میتوانیم تصویر را به فضای رنگی HSV ببریم و برای اینکه تصویر روشن تر بشه پارامترV را در ان بیشتر کنیم یا پارامتر saturation را زیاد کنیم که با اینکار بیشتر به سمت رنگ های اصلی میرویم من هر دو روش برای پیاده سازی بهبود تصویر استفاده کردم

در ابتدا یک فیلتر sharpening که در جزوه هست تعریف کردم و با تصویر filter کردم در ادامه هر کدوم از کانال hsv را به دست اوردم و کانال های v,s را تقویت کردم تصویر خروجی به صورت زیر است:



سوال پنجم

الف) مراحل الگوریتم harris را به طور کامل توضیح دهید.

آشکارساز harris یک گوشه یاب هست و ما ابتدا یک پنجره که می تواند فیلتر گوسی باشد در نظر میگیریم و میزان اختلاف روشنایی را به ازای u,v طبق فرمول زیر حساب می کنیم.

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

و گوشه نقطه ای هست که عبارت بالا در ان بزرگ باشد می توانیم عبارت بالا را با استفاده از بسط تیلور ساده تر بنویسیم که می دانیم:

$$I(x+u,y+v) \approx I(x,y) + uI_x + vI_y$$

و با توجه به عبارت بالا فرمول harris به صورت زیر میشود:

$$E(u,v) \approx \sum_{x,y} w(x,y)\left[uI_x + vI_y\right]^2 = \sum_{x,y} w(x,y)\left(u^2I_x^2 + 2uvI_xI_y + v^2I_y^2\right)$$

$$= \begin{bmatrix} u & v \end{bmatrix} \left(\sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

که I_x, I_y مشتق نقطه x,y در جهت های افقی و عمودی هستند که با عملگر sobel می توانیم به دست اوریم.

در جبر خطی مقادیر ویژه نشان می دهد که در یک راستا چه مقدار انرژی وجود دارد و بردار ویژه جهت آنها را نشان میدهد مقادیر ویژه ما لاندا۱ولاندا۲ هستند و اگر هر دوتای این مقادیرشان بزرگ باشد می توانیم نتیجه بگیریم که ان نقطه گوشه است برای فهمیدن این موضوع یک R تعریف میکنیم که به صورت زیر است:

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

دلیل اینکه فقط ضرب لاندا۱ولاندا۲ را نمیگیریم این هست که ممکن یکی از این دو مقدار بزرگ باشد و دیگری کوچک که باعث شود ضرب این دو مقدار بزرگ شود در حالی که این پیکسل جزو نقاط لبه است و نه گوشه.اثبات میشود که ضرب لاندا۱ولاندا۲ برابر با دترمینال ماتریس و جمع دو مقدار لاندا۱و لاندا۲ به توان ۲ برابر با trace ماتریس که برابر است جمع مقادیر روی قطر اصلی به توان دو است.پس فرمول R به صورت زیر میشود:

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

پس برای آشکارساز Harris مراحل زیر را طی میکنیم:

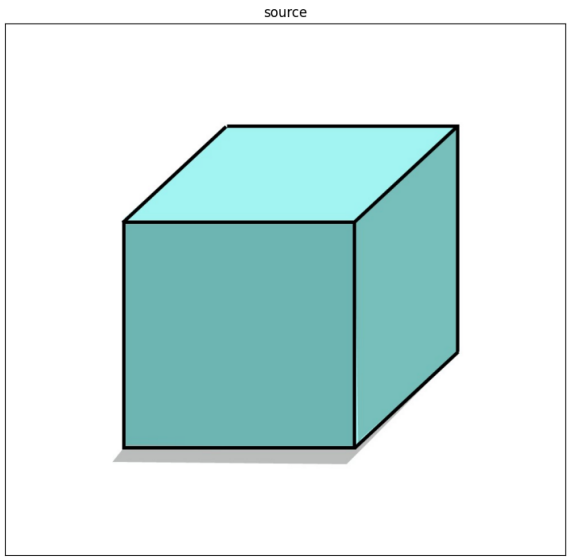
۱. محاسبه مشتق عمودی و افقی که همان I_y, I_x در ماتریس هستند که می توانیم با استفاده از عملگر Sobel انجام دهیم.
۲. مربع مشتق ها و حاصل ضربشان را محاسبه میکنیم.
۳. اعمال پنجره W که میتواند یک فیلتر گوسی باشد
۴. محاسبه مقدار R طبق فرمول
۵. در هر پیک از همسایه های مثلا ۲*۲ می اییم و مقدار غیر بیشینه را حذف میکنیم

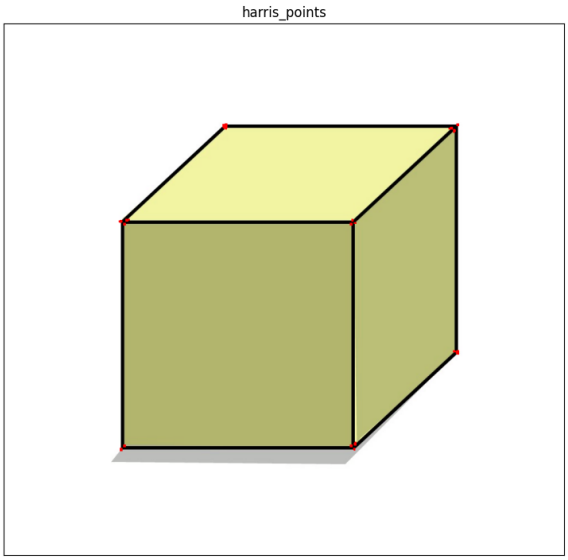
۶. یک threadshod میگذاریم و مقادیر بزرگ را به دست می اوریم مقادیر بزرگ گوشه های ما هستند.

ب) قسمتی که خودم پیاده سازی کردم دقیقا طبق مراحلی رفتم که در قسمت اول گفتم در ابتدا یک Sobel زدم و بعد روی آن یک فیلتر گوسی زدم سایز سوبل و فیلتر گوسی را 3*3 گرفتم مطابق زیر:

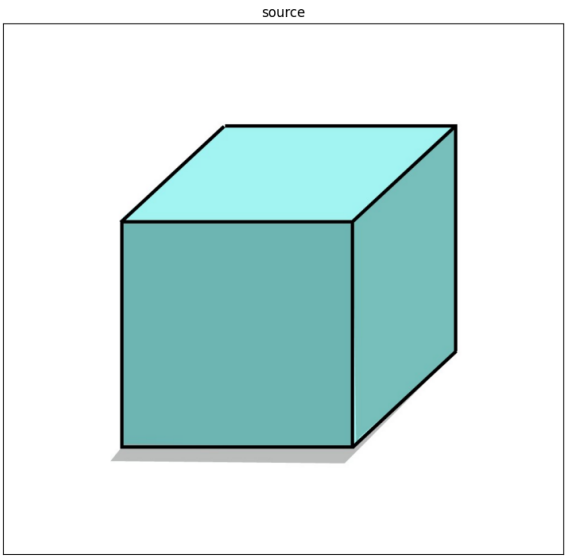
```
Ix = cv2.Sobel(grayimg, cv2.CV_64F, 1, 0, ksize=3)
Iy = cv2.Sobel(grayimg, cv2.CV_64F, 0, 1, ksize=3)

#This step gives the product of the gradient componen
Ix_2 = cv2.GaussianBlur(np.multiply(Ix, Ix),(3,3),3)
Iy_2 = cv2.GaussianBlur(np.multiply(Iy, Iy),(3,3),3)
IxIy = cv2.GaussianBlur(np.multiply(Ix, Iy),(3,3),3)
```

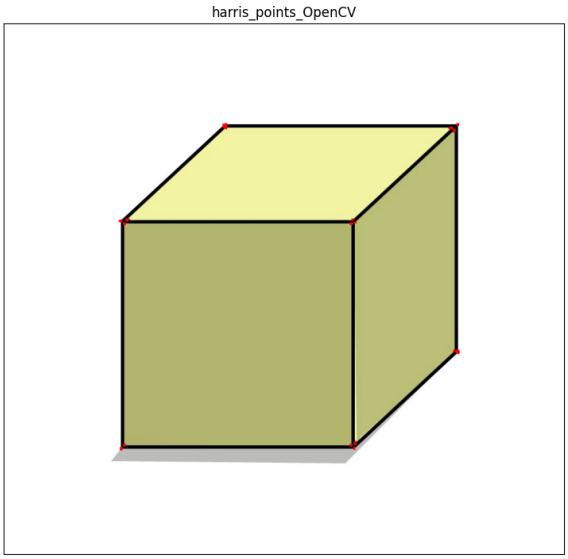
و در ادامه R را به دست می اورم و یک Threadshod می گذارم و گوشه ها را رنگ قرمز میکنم تصویر ان مطابق زیر است:





و در قسمت بعد از خود کتابخانه opencv یعنی cornerHarris استفاده میکنم که شکل آن مطابق زیر میشود:





سوال ششم

راجع به ۳ روش SIFT ، SURFو ORB برای استخراج ن قاط کلیدی این لینک را مطالعه کرده و خلاصه ای از مقایسه ی این ۳ روش ارائه نمایید

در ابتدا خلاصه ای از هر سه روش را می گویم.

SIFT

این الگوریتم یکی از دقیق‌ترین و پایدارترین روش‌های استخراج نقاط کلیدی از تصاویر است. این الگوریتم از یک سری مراحل تشکیل شده است که شامل اصلاح گاما، کاهش نویز، تشخیص لبه‌ها، پیدا کردن نقاط کلیدی و توصیف آن‌ها است. یکی از ویژگی‌های برجسته این روش، قابلیت مقاومت در برابر تغییرات مقیاسی است. با این حال، یکی از مشکلات این روش کند بودن آن است.

SURF

این الگوریتم برای سرعت بیشتر در استخراج نقاط کلیدی طراحی شده است. این الگوریتم با استفاده از فیلترهای دو بعدی و جستجوی تکراری، نقاط کلیدی را پیدا می‌کند. از ویژگی‌های این روش، حساسیت کمتر آن در برابر تغییرات مقیاسی و جهتی است. با این حال، در برخی موارد، دقت این روش پایین‌تر از SIFT است.

ORB

این الگوریتم یک روش ساده و سریع برای استخراج نقاط کلیدی از تصاویر است. این الگوریتم با استفاده از ترکیب دو الگوریتم FAST و BRIEF ایجاد شده است. الگوریتم FAST برای پیدا کردن نقاط کلیدی سریع استفاده می‌شود

و الگوریتم BRIEF برای توصیف نقاط کلیدی استفاده می‌شود. این الگوریتم با سرعت بالا و مقاومت در برابر تغییرات شدید شناخته شده است. با این حال، دقت آن کمتر از SIFT و SURF است.

مقایسه سه روش:

۱.سرعت:

ORBسریع‌ترین روش برای استخراج نقاط کلیدی است و با سرعت ۳ تا ۴ برابر سرعت SIFT و SURF عمل می‌کندSURF در مقایسه با SIFT سریع‌تر است.

۲.دقت:

SIFTدقیق‌ترین روش برای استخراج نقاط کلیدی است. در آزمایشات انجام شده، SIFTنتایج بهتری در مقایسه با SURF و ORB داشته است SURF نیز نتایج خوبی دارد و از لحاظ دقت بهتر از ORB است.

۳.مقاومت در برابر تغییرات شدید:

ORBبیشترین مقاومت در برابر تغییرات شدید از جمله تغییرات شدید در شدت روشنایی و گرادیان دارد.

با توجه به مقایسه‌ی انجام شده، برای مواردی که نیاز به دقت بالا است، می‌توان از روش SIFT استفاده کرد. اگر سرعت برای مواردی که مهم است، مهم است، می‌توان از روش ORB استفاده کرد. اما اگر نیاز به تعادل بین دقت و سرعت و مقاومت در برابر تغییرات شدید است، روش SURF گزینه‌ی بهتری است.