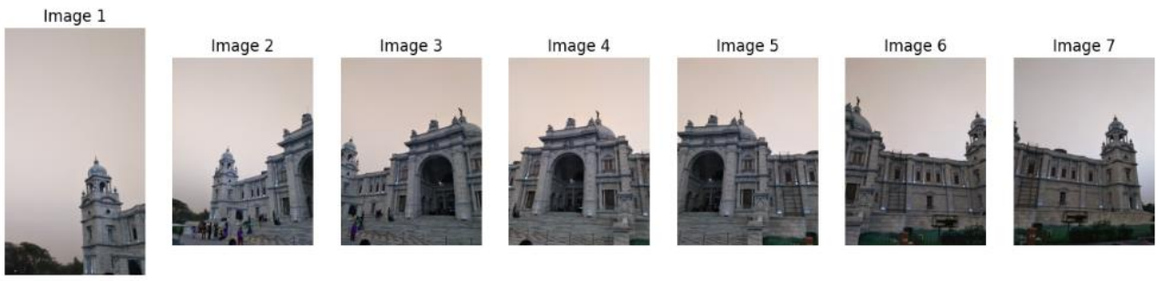
۱.

```
In [10]: # read input victoria images and show them in a row together
IMAGE_CNT = 7
FIG_SIZE = 16

fig = plt.figure(figsize=(FIG_SIZE, FIG_SIZE))

for i in range(1, IMAGE_CNT + 1):
    victoria_i = cv2.imread(f"images/victoria{i}.png")
    fig.add_subplot(1, IMAGE_CNT, i)
    plt.imshow(victoria_i)
    plt.axis('off')
    plt.title(f"Image {i}")
```



در ابتدا تصاویر داده شده را یکی‌یکی باز کردم و در یک ردیف کنار هم نمایش دادم.

```
In [7]: # initialize OpenCV's image sticher object and then perform the image stitching on input images

image_sticher = cv2.Stitcher.create()
status, output = image_sticher.stitch(tuple(cv2.imread(f"images/victoria{i}.png") for i in range(1, IMAGE_CNT + 1)))
```

```
In [10]: # show victoria panorama
plt.imshow(output)
plt.axis('off')
plt.title(f"victoria panorama")
plt.show()
```

همانطور که خواسته شده بود با استفاده از *OpenCV* آبجکت *sticher* را ساختم و ۷ تصویر داده‌شده را باز کردم و به متود *stitch* این آبجکت دادم و در نهایت با *plotlib* روی صفحه نمایش دادم.

۲.

$$x_2 = x_1\cos\theta - y_1\sin\theta$$
$$y_2 = x_1\sin\theta + y_1\cos\theta$$
$$cost = \sum (x_2^n - x_1^n\cos\theta + y_1\sin\theta)^2 + (y_2^n - x_1^n\sin\theta - y_1^n\cos\theta)^2$$
$$\frac{d\,cost}{d\theta} = 2\sum (y_1^n y_2^n + x_1^n x_2^n)\sin\theta + (x_2^n y_1^n - x_1^n y_2^n)\cos\theta = 0$$
$$\Rightarrow \Big(\sin\theta \underbrace{\sum y_1^n y_2^n + x_1^n x_2^n}_{S_1}\Big) + \Big(\cos\theta \underbrace{\sum x_2^n y_1^n - x_1^n y_2^n}_{S_2}\Big) = 0$$
$$\Rightarrow S_1\sin\theta + S_2\cos\theta = 0 \Rightarrow S_1\sin\theta = -S_2\cos\theta$$
$$\Rightarrow S_1\tan\theta = -S_2 \Rightarrow \tan\theta = -\frac{S_2}{S_1} \Rightarrow \theta = \tan^{-1}\left(\frac{S_2}{S_1}\right)$$
$$\Rightarrow \theta = \tan^{-1}\frac{\sum x_2^n y_1^n - x_1^n y_2^n}{\sum y_1^n y_2^n + x_1^n x_2^n}$$

۳.ت: پیداکردن رئوس مستطیل کاغذ با استفاده از یافتن کانتورها

در تابع *find_vertices* ابتدا از ورودی یک کپی تهیه کردم که در هنگام تست و نمایش کانتورها تصویر اصلی لبه‌ها در این تابع تغییر نکند.

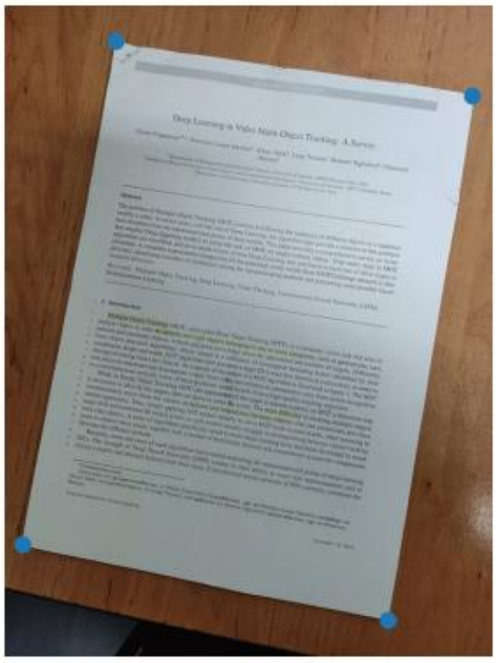
```
def find_vertices(im):
    edges = copy.deepcopy(im)
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    convexed_contours = np.array([cv2.convexHull(contour) for contour in contours])

    for contour in sorted(convexed_contours, key = cv2.contourArea, reverse = True):
        p = cv2.arcLength(contour, True) # perimeter aproximation
        polygon_vertices = cv2.approxPolyDP(contour, 0.02*p, True)

        if len(polygon_vertices) == 4:
            return polygon_vertices
```

سپس با تابع *findcountours* تمام کانتورهای تصویر لبه‌ها را پیدا و ذخیره کردم. آرگومان دوم این تابع پنج حالت مختلف برای تشخیص کانتورها در اختیارمان می‌گذارد. حالت انتخاب شده یعنی RETR_EXTERNAL در زمانی که چند کانتور درون هم قرار گرفته باشند فقط بیرونی‌ترین کانتور را برمی‌گرداند که با توجه به اینکه می‌خواهیم مرزهای کاغذ را پیدا کنیم و کانتورهای درون کاغذ برایمان اهمیتی ندارد بهترین حالت برای ما همین است. حالت پیش‌فرض انتخاب شده در داک *OpenCV* حالت RETR_TREE است که به صورت سلسله‌مراتبی تمام کانتورهای درونی را هم در کانتور اصلی قرار می‌دهد و ذخیره می‌کند که برای ما کاربردی ندارد. آرگومان سوم این تابع را طبق راهنمایی قرارداده شده در نوتبوک CHAIN_APPROX_SIMPLE گذاشتم.



در مرحله‌ی بعدی به ازای هریک از این کانتورها پوش محدب مربوط به آن (*convex hull*) را با تابع آماده *OpenCV* بدست آوردم. پوش محدب برای یک کانتور یعنی کوچکترین مجموعه محدبی که نقاط کانتور داخل آن قرار بگیرند. (مثل یک کش که دور تعدادی میخ قرار بگیرد)

بعد از بدست آوردن پوش محدب هریک از کانتورها، به ترتیبِ مساحت، آن‌ها را بررسی کردم. برای مرتب‌سازی بر اساس مساحت از تابع *sorted* پایتون استفاده کردم و *cv2.contourArea* را به عنوان آرگومان *key* به *sorted* دادم. برای اینکه ببینم کانتور موردبررسی همان برگه کاغذ هست یا نه از *approxPolyDP* استفاده کردم تا کانتور را تقریب بزنند، اگر بعد از تقریب زدن، این کانتور چهارراس داشته باشد یعنی احتمالا همان کاغذ است (چون به ترتیب مساحت کانتورها را بررسی کردیم بزرگترین چهارضلعی بین کانتورهاست) تابع *approxPolyDP* یک آرگومان اپسیلون دارد که دقت ساده‌کردن چندضلعی را مشخص می‌کند که طبق جستجوهایی که داشتم اگر این پارامتر را ضریبی از محیط چندضلعی بدهیم نتیجه بهتری خواهیم گرفت. برای تخمین زدن محیط چندضلعی از تابع *arclength* استفاده کردم. هم تابع *arclength* و هم تابع *approxPolyDP* یک آرگومان *boolean* دارند که به معنای این است آیا شکل کانتور موردنظر بسته است و نقاط کانتور تشکیل دور می‌دهد یا نه که چون ابتدا پوش محدب گرفتیم پس همه‌ی کانتورهایی که داریم بسته هستند و این پارامتر را *True* پاس دادم.

```
In [40]: print(vertices)
         print(vertices[:, 0])
```

```
[[[136  43]]

 [[576 110]]

 [[471 755]]

 [[ 21 662]]]
[[136  43]
 [576 110]
 [471 755]
 [ 21 662]]
```

در نوتبوک خواسته شده خروجی محاسبه شده *4x2* باشد تا در قطعه کد نوشته شده *scatter* بدرستی اجرا شود. خروجی تابع *find_vertices* یک بُعد اضافه‌تر دارد (خروجی *4x1x2* است) پس قبل از *scatter* بُعد اضافی را حذف کردم.

۳.ث: نگاشت دورنما و برش

```
def crop_out(im, vertices):
    # Your code goes here.
    vertices = reorder(vertices)
    top_left, top_right, bottom_right, bottom_left = vertices

    top_width = np.sqrt(    ((top_right[0] - top_left[0]) ** 2)   +   ((top_right[1] - top_left[1]) ** 2)    )
    bottom_width = np.sqrt(    ((bottom_right[0] - bottom_left[0]) ** 2)   +   ((bottom_right[1] - bottom_left[1]) ** 2)    )
    target_width = int(max(top_width, bottom_width))

    right_height = np.sqrt(    ((top_right[0] - bottom_right[0]) ** 2)   +   ((top_right[1] - bottom_right[1]) ** 2)    )
    left_height = np.sqrt(    ((top_left[0] - bottom_left[0]) ** 2)   +   ((top_left[1] - bottom_left[1]) ** 2)    )
    target_height = int(max(right_height, left_height))

    transform = cv2.getPerspectiveTransform(
        vertices,
        np.array(
            [[0, 0], [target_width - 1, 0], [target_width - 1, target_height - 1], [0, target_height - 1]], dtype="float32"
        )
    )  # get the top or bird eye view effect

    return cv2.warpPerspective(im, transform, (target_width, target_height))
```

```
cropped = crop_out(im, vertices)
imshow(cropped)
```

با تابع *reorder* که در نوتبوک داده شده بود ترتیب رئوس را تغییر دادم و سپس پهنای بالا و پایین کاغذ و همینطور ارتفاع چپ و راست کاغذ را با محاسبه‌ی فاصله‌ی رئوس آن محاسبه کردم. پهنای تصویر نهایی را ماکسیمم پهنای بالا و پایین و بطور مشابه ارتفاع آن را ماکسیمم ارتفاع چپ و راست قرار دادم. مختصات نقاط تصویر اصلی و مختصات مقصد را با ترتیب گفته‌شده در نوتبوک به *getPerspectiveTransform* دادم و *Transform* خروجی را در تابع *warpPerspective* استفاده کردم تا تصویر اصلی از محل رئوس چهارضلعی برش زده شود و در نهایت با ابعاد خواسته شده در خروجی تابع *crop_out* برود.

۳.ج: بهبود تصویر

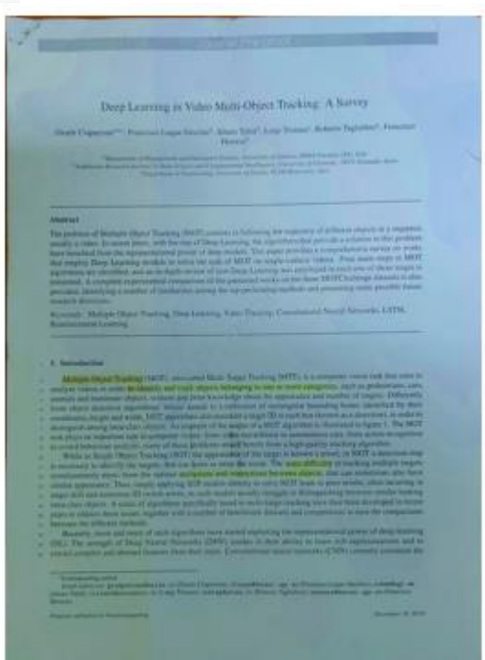
```
def enhance(im):
    im_HSV = cv2.cvtColor(im, cv2.COLOR_BGR2HSV).astype("float32")

    h, s, v = cv2.split(im_HSV)
    s = np.clip(s * 3, 0, 255)

    im_RGB = cv2.cvtColor(cv2.merge([h, s, v]).astype("uint8"), cv2.COLOR_HSV2BGR)
    return im_RGB
```

برای بهبود تصویر همانطور که پیشنهاد شده بود افزایش غلظت تصویر (*Saturation*) انجام دادم. برای این کار ابتدا فضای رنگی تصویر را به *HSV* تغییر دادم و بعد از ۳برابر کردن مقدار *S* تصویر را به فضای رنگی پیش‌فرض *BGR* برگرداندم.

افزایش غلظت رنگ (Saturation)



منابع:

سوال ۱: How to Display Multiple Images in One Figure Correctly in Matplotlib? - GeeksforGeeks

OpenCV Panorama Stitching - GeeksforGeeks

سوال ۳: OpenCV: Smoothing Images

OpenCV: Contours : Getting Started

OpenCV: Structural Analysis and Shape Descriptors

Create your own ‘CamScanner’ using Python & OpenCV | by Shirish Gupta | Level Up Coding (gitconnected.com)

پوش محدب - ویکی‌پدیا، دانشنامه‌ی آزاد(wikipedia.org)

python - cv2.approxPolyDP() , cv2.arcLength() How these works - Stack Overflow

python - What does cv2.approxPolydp() return? - Stack Overflow

How to change saturation values with opencv? - Stack Overflow