سوال ۱

توضیح خطوط مربوط به استفاده از کتابخانه‌ها:

- `import cv2`: این خط کتابخانه OpenCV را وارد می‌کند که برای بینایی ماشین در زمان واقعی استفاده می‌شود. این شامل چندین صد الگوریتم بینایی ماشین است.
- `import numpy as np`: این خط کتابخانه NumPy را وارد می‌کند و به آن نام مستعار np می‌دهد. NumPy یک کتابخانه برای زبان برنامه نویسی Python است که پشتیبانی از آرایه ها و ماتریس های بزرگ چند بعدی را اضافه می‌کند، همراه با مجموعه ای بزرگ از توابع ریاضی سطح بالا برای عملیات روی این آرایه ها.
- `import matplotlib.pyplot as plt`: این خط ماژول pyplot را از کتابخانه Matplotlib وارد می‌کند و به آن نام مستعار plt می‌دهد. pyplot یک کتابخانه رسم نمودار است که برای گرافیک ۲ بعدی در زبان برنامه نویسی Python استفاده می‌شود. می‌توان از آن در اسکریپت های Python، پوسته، سرورهای برنامه های وب و سایر ابزارهای رابط کاربری گرافیکی استفاده کرد.
- `from PIL import Image`: این خط ماژول Image را از کتابخانه Pillow (PIL به معنای Python Imaging Library) وارد می‌کند. Pillow یک کتابخانه برای زبان برنامه نویسی Python است که پشتیبانی از باز کردن، دستکاری و ذخیره سازی انواع مختلف قالب های فایل تصویر را اضافه می‌کند.

در ادامه کدها توضیح داده می‌شوند.

```
img1_path = 'images/1.jpg'
img1 = cv2.imread(img1_path, cv2.IMREAD_COLOR)
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
plt.imshow(img1)
plt.axis('off')
```

در ابتدا تصویر را لود کرده و آن را از حالت دیفالت BRG به حالت RGB تبدیل می‌کنیم. سپس آن را نمایش می‌دهیم و با استفاده از خط آخر، نمودارهای x و y را حذف می‌کنیم.

در ادامه توابع به ترتیب توضیح داده شده‌اند:

```python
def RGB_to_CMYK(r, g, b, RGB_SCALE = 255, CMYK_SCALE = 100):
    """
    Convert RGB values to CMYK values.
    Args:
        r (numpy.ndarray): Red channel values.
        g (numpy.ndarray): Green channel values.
        b (numpy.ndarray): Blue channel values.
        RGB_SCALE (int): Scale factor for RGB values. Default is 255.
        CMYK_SCALE (int): Scale factor for CMYK values. Default is 100.
    Returns:
        c (numpy.ndarray): Cyan channel values.
        m (numpy.ndarray): Magenta channel values.
        y (numpy.ndarray): Yellow channel values.
        k (numpy.ndarray): Black channel values.
    """
    r = r.astype(np.float32) / RGB_SCALE
    g = g.astype(np.float32) / RGB_SCALE
    b = b.astype(np.float32) / RGB_SCALE

    # Extract out K [0,1]
    k = 1 - np.max(np.array([r, g, b]), axis=0)
    epsilon = 1e-8

    c = (1 - k - r) / (1 - k + epsilon)
    m = (1 - k - g) / (1 - k + epsilon)
    y = (1 - k - b) / (1 - k + epsilon)

    # Rescale to the range [0,CMYK_SCALE]
    c, m, y, k = np.array([c, m, y, k]) * CMYK_SCALE

    return c, m, y, k
```

این تابع با استفاده از کتابخانه NumPy جهت انجام عملیات روی آرایه‌ها، مقادیر RGB را به مقادیر CMYK که برای چاپ رنگی استفاده می‌شود، تبدیل می‌کند:

- ابتدا مقادیر (r, g, b) RGB به نوع داده float32 تبدیل می‌شوند تا محاسبات دقیق انجام شود.
- سپس بیشترین مقدار هر کانال رنگ (R, G, B) محاسبه می‌شود.
- این بیشترین مقدار از ۱ کم می‌شود تا کمترین مقدار هر کانال رنگ به دست آید. این کار جهت اختصاص دادن کم‌ترین مقدار موجود در کانال‌ها به رنگ سیاه است که از استفادهٔ مجدد آن در باقی رنگ‌ها جلوگیری شود.
- سپس مکمل هر کانال رنگ با کم کردن مقدار سیاه محاسبه می‌شود تا مقادیر CMYK به دست آیند. این عملیات طبق روابط موجود در اسلایدها انجام می‌شود. تنها تفاوت، تقسیم مقادیر بر مقدار $1 - k$ است تا تمامی سه رنگ را نرمالایز کنیم.
- در نهایت، مقادیر CMYK به محدوده [0, CMYK_SCALE] مقیاس می‌شوند و تمام این این مقادیر به صورت یک تاپل (c, m, y, k) برگردانده می‌شوند.

```python
def CMYK_to_RGB(c, m, y, k, RGB_SCALE = 255, CMYK_SCALE = 100):
    """
    Convert CMYK values to RGB values.
    Args:
        c (numpy.ndarray): Cyan channel values.
        m (numpy.ndarray): Magenta channel values.
        y (numpy.ndarray): Yellow channel values.
        k (numpy.ndarray): Black channel values.
        RGB_SCALE (int): Scale factor for RGB values. Default is 255.
        CMYK_SCALE (int): Scale factor for CMYK values. Default is 100.
    Returns:
        r (numpy.ndarray): Red channel values.
        g (numpy.ndarray): Green channel values.
        b (numpy.ndarray): Blue channel values.
    """
    #############
    c, m, y, k = np.array([c, m, y, k]) / CMYK_SCALE

    r = (1 - k) * (1 - c)
    g = (1 - k) * (1 - m)
    b = (1 - k) * (1 - y)

    r, g, b = np.array([r, g, b]) * RGB_SCALE
    #############

    return r, g, b
```

این بخش نیز عکس تابع قبلی است. به عنوان مثال فرمول مربوط به r به این صورت محاسبه می‌شود:

$$c = \frac{(1-k-r)}{(1-k)}$$
$$\rightarrow (1-k)c = 1-k-r$$
$$\rightarrow r = ck - c + 1 - k$$
$$\rightarrow r = c(k-1) - (k-1)$$
$$\rightarrow r = (c-1)(k-1)$$

به همین شکل سایر روابط نیز بدست می‌آیند و در پایان با استفاده از یک اسکیل کردن، تمام مقادیر بین ۰ تا ۲۵۵ قرار می‌گیرند.

```python
def RGB_to_HSI(r, g, b):
    """
    Convert RGB values to HSI values.
    Args:
        r (numpy.ndarray): Red channel values.
        g (numpy.ndarray): Green channel values.
        b (numpy.ndarray): Blue channel values.
    Returns:
        h (numpy.ndarray): Hue channel values. Range[0, 360]
        s (numpy.ndarray): Saturation channel values. Range[0, 255]
        i (numpy.ndarray): Intensity channel values. Range[0, 255]
    """

    #############
    r, g, b = r/255., g/255., b/255. # Normalize each channel
    epsilon = 1e-8

    # Calculate i: i = (r + g + b) / 3
    i = np.mean(np.array([r, g, b]), axis=0)
    i = np.round(i * 255) # [0, 1) -> [0, 255)

    # Calculate s: s = 1 - 3 * min(r, g, b) / sum(r, g, b)
    min_rgb = np.min(np.array([r, g, b]), axis=0)
    s = 1 - 3 * (min_rgb) / (r + g + b + epsilon)
    s = np.round(s * 255) # [0, 1) -> [0, 255)

    # Calculate h
    theta = np.arccos(((r - g) + (r - b)) / (2 * ((r - g)**2 + (r - b) * (g - b))**0.5 + epsilon))
    h = (180 / np.pi) * theta # convert rad to degree
    h[b > g] = 360 - h[b > g]

    return h, s, i
```

این تابع نیز مانند توابع پیشین، یک تصویر RGB را به عنوان ورودی می‌گیرد و این بار آن را به یک تصویر HSI (رنگ، اشباع، شدت) تبدیل می‌کند. این تابع سه پارامتر ورودی دارد: r، g و b که به ترتیب نمایانگر کانال‌های رنگی قرمز، سبز و آبی یک تصویر RGB هستند. این تابع مقادیر رنگ (h)، اشباع (s) و شدت (i) هر پیکسل در تصویر را خروجی می‌دهد.

- کد با نرمال‌سازی هر کانال رنگی با تقسیم آن بر ۲۵۵ شروع می‌شود. این کار باعث می‌شود که مقادیر ورودی در محدوده ۰ تا ۱ قرار بگیرند. متغیر epsilon نیز به منظور جلوگیری از خطای تقسیم بر صفر اضافه شده است.
- سپس، تابع شدت (i) را با گرفتن میانگین کانال‌های رنگی نرمال‌شده محاسبه می‌کند. پس از آن مقادیر شدت را به نزدیک‌ترین عدد صحیح بین ۰ و ۲۵۵ گرد می‌کند.
- اشباع (s) با گرفتن حداقل کانال‌های رنگی و تقسیم آن بر مجموع کانال‌های رنگی محاسبه می‌شود. این کار میزان تفاوت هر کانال رنگی از میانگین سه کانال را مشخص می‌کند. سپس مقادیر اشباع به نزدیک‌ترین عدد صحیح بین ۰ و ۲۵۵ گرد می‌شوند.
- رنگ (h) با استفاده از زاویه بین کانال‌های رنگی قرمز و آبی محاسبه می‌شود. از تابع arccos برای پیدا کردن زاویه بین کانال‌های قرمز و آبی استفاده می‌شود و سپس نتیجه از رادیان به درجه تبدیل می‌شود. مقادیر رنگ سپس تنظیم می‌شوند تا در محدوده ۰ تا ۳۶۰ قرار بگیرند.
- در نهایت، تابع مقادیر رنگ، اشباع و شدت را به صورت یک تاپل برمی‌گرداند.

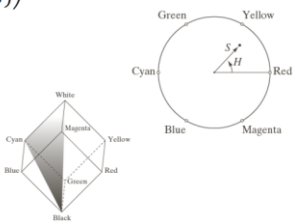
فرمول‌های استفاده شده مطابق اسلایدها هستند:

$$\theta = cos^{-1}\left(\frac{(R-G)+(R-B)}{2\sqrt{(R-G)^2+(R-B)(G-B)}}\right)$$

$$H = \begin{cases} \theta, & if\ B \le G \\ 360-\theta & if\ B > G \end{cases}$$

$$S = 1 - 3\frac{min(R,G,B)}{R+G+B}$$

$$I = \frac{R+G+B}{3}$$



کد از کتابخانه NumPy برای عملیات‌های آرایه‌ای استفاده می‌کند، به ویژه از توابع np.mean و np.min. NumPy یک کتابخانه محبوب برای محاسبات عددی در Python است که توابع مختلفی برای انجام عملیات‌های روی آرایه‌ها مانند پیدا کردن مقادیر حداقل یا میانگین ارائه می‌دهد.

```python
def show_images(T, cols=1):
  N = len(T)
  fig = plt.figure()
  for i in range(N):
      a = fig.add_subplot(int(np.ceil(N/float(cols))), cols, i+1)
      try:
        img,title = T[i]
      except ValueError:
        img,title = T[i], "Image %d" % (i+1)
      if(img.ndim == 2):
        plt.gray()
      plt.imshow(img)
      a.set_title(title)
      plt.xticks([0,img.shape[1]]), plt.yticks([0,img.shape[0]])
  fig.set_size_inches(np.array(fig.get_size_inches()) * N)
  plt.show()
```

این کد یک تابع پایتون به نام show_images را تعریف می‌کند. این تابع برای نمایش یک لیست از تصاویر همراه با عناوینشان در یک طرح شبکه‌ای طراحی شده است.

تابع show_images دو آرگومان می‌پذیرد: T که یک لیست از تاپل‌ها است که هر تاپل شامل یک تصویر و عنوان مربوطه است، و cols که تعداد ستون‌ها در طرح شبکه‌ای را مشخص می‌کند (مقدار پیش‌فرض ۱ است).

- تعداد کل تصاویر در لیست T را محاسبه کرده و آن را به متغیر N اختصاص می‌دهد.
- با استفاده از plt.figure از کتابخانه matplotlib، یک شکل جدید ایجاد می‌کند.
- برای هر تصویر در طرح شبکه‌ای، حلقه را اجرا می‌کند.
- درون حلقه، سعی می‌کند تاپل فعلی T[i] را به متغیرهای img و title باز کند. اگر باز کردن موفق نباشد به دلیل وجود یک ValueError (احتمالا به معنای این است که تاپل تنها شامل تصویر بدون عنوان است)، عنوان پیش‌فرض "تصویر ٪d" % (i+1) را به متغیر title اختصاص می‌دهد.
- اگر تصویر (img) خاکستری باشد (۲ بعدی)، با استفاده از plt.gray، نقشه رنگ را به خاکستری تنظیم می‌کند.
- تصویر را با استفاده از plt.imshow نمایش می‌دهد و عنوان زیرنمونه را با استفاده از a.set_title تنظیم می‌کند.

- تیک‌های x و y زیرنمونه را با استفاده از plt.xticks و plt.yticks به ابعاد تصویر تنظیم می‌کند. با این کار سایز تصویر مشخص می‌شود.
- پس از حلقه، اندازه شکل را بر اساس تعداد تصاویر (N) با استفاده از fig.set_size_inches تنظیم می‌کند تا مطمئن شود تصاویر خیلی کوچک نیستند.
- در نهایت، با استفاده از plt.show، شکل را نمایش می‌دهد.

کدها و نتایج مربوط به استفاده از توابع داخل نوتبوک مربوطه آورده شده است.

سوال ۲

در این سوال، از تابع show_images که در سوال اول توضیح داده شده، استفاده شده است.

در زیر توابع استفاده شده و نیز الگوریتم استفاده شده توضیح داده می‌شوند.

در ابتدا نیاز است که تصاویر را بخوانیم. از آنجا که فرمت دو تصویر با یکدیگر تفاوت دارد، پس نیاز است که تصویر بزرگ‌تر را Resize کرد تا به اندازهٔ تصویر دیگر شود. برای عملیات درون‌یابی از INTER_AREA استفاده شده است، که طبق آنچه که در این وب‌سایت آمده، بهترین روش برای Resize کردن تصویر به تصویری کوچکتر است:

```
img2 = cv2.resize(img2, (img3.shape[1], img3.shape[0]), interpolation = cv2.INTER_AREA)
```

کد تابع اصلی نیز در زیر آورده شده است:

```
def diff(image1, image2):
    """
    Find the differences between two images.
    Args:
        image1 (numpy.ndarray): First image.
        image2 (numpy.ndarray): Second image.
    Returns:
        result (numpy.ndarray): Image showing the differences between the two input images.
    """

    #############
    gray_img2 = cv2.cvtColor(image1, cv2.COLOR_RGB2GRAY)
    gray_img3 = cv2.cvtColor(image2, cv2.COLOR_RGB2GRAY)
    result = np.zeros_like(image2)
    result[:, :, 0] = gray_img2[:, :]
    result[:, :, 1] = gray_img3[:, :]
    result[:, :, 2] = gray_img3[:, :]
    #############

    return result
```

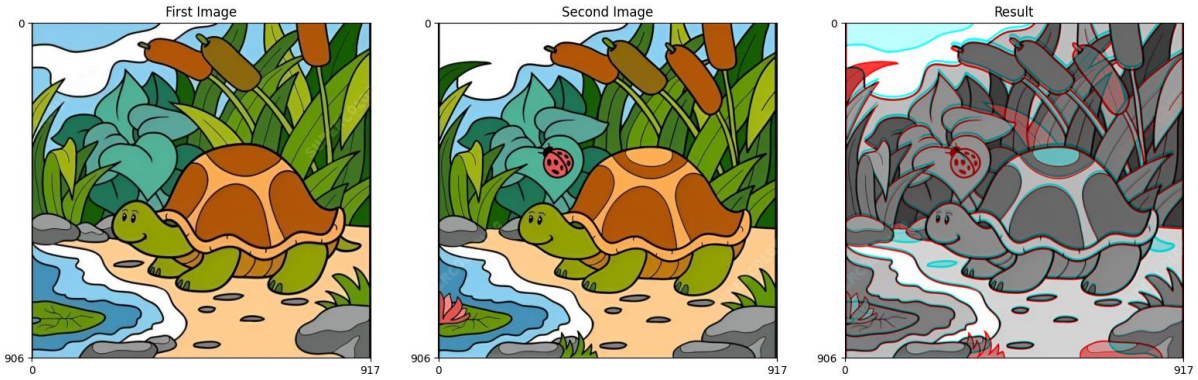
روشی که در کلاس تدریس شده است، در حوزهٔ رنگ‌های خاکستری عمل می‌کند. برای این منظور در ابتدا هر دو تصویر ورودی را به محدودهٔ رنگ‌های خاکستری می‌بریم. برای این کار از دستور *cv2.cvtColor* استفاده می‌کنیم با آرگومان *cv2.COLOR_RGB2GRAY*. بعد از این یک خروجی درست می‌کنیم که ساختار آن دقیقاً مطابق با *image2* است. بعد از این نیز کانال اول که مربوط به رنگ قرمز است را با مقادیر موجود در تصویر اول در حوزهٔ خاکستری پر می‌کنیم. همینکار را برای کانال‌های سبز و آبی انجام می‌دهیم اما با استفاده از مقادیر مربوط به تصویر دوم.

با استفاده از این روش، در صورتی که تفاوتی بین دو نقطه وجود نداشته باشد، هر سه کانال مقادیر یکسان خواهند گرفت و در تصویر نهایی سیاه و سفید خواهند بود. در غیر این صورت، اگر هر یک از تصاویر جزئیات متفاوتی نسبت به تصویر دیگر داشته

باشد، رنگ آبی فیروزه‌ای و یا قرمز بر روی تصویر نقش می‌بندد. همچنین این که چه رنگی توسط الگوریتم برای تغییرات انتخاب شود، بستگی به میزان روشنایی دو تصویر در ناحیهٔ تحت بررسی دارد.

در پایان نیز تصویر خاکستری‌ای که تفاوت‌های موجود در آن به صورت رنگی است را به عنوان خروجی بازمی‌گردانیم.

نتایج نهایی به صورت زیر است:



همانطور که مشخص است، الگوریتم به خوبی توانسته است که تفاوت‌ها را شناسایی کند.

کدها و نتایج مربوط به استفاده از توابع داخل نوتبوک مربوطه آورده شده است.

سوال ۳:

بخش الف)

سوال سوم

بخش الف) طبق تعریف برای ماتریس Harris از رابطه زیر استفاده می‌کنیم:

$$M = \sum_{x,y} g\,w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

برای سادگی ماتریس وزن‌ها (w) را به صورت زیر برای یک پنجره ۳×۳ تعریف می‌کنیم:

$$w = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

بنابراین برای بدست آوردن ماتریس هریس برای پنجره نقطه‌ی داده شده به صورت زیر عمل می‌کنیم:

$$I_x = \begin{bmatrix} 3 & 2 & 0 \\ 3 & 4 & 2 \\ 1 & 3 & 2 \end{bmatrix} \quad , \quad I_y = \begin{bmatrix} 3 & 2 & -1 \\ 4 & 4 & 1 \\ 0 & 3 & 2 \end{bmatrix}$$

$$\rightarrow I_x^2 = (3)^2 + (2)^2 + (0)^2 + (3)^2 + (4)^2 + (2)^2 + (1)^2 + (3)^2 + (2)^2$$
$$= 56$$

$$\rightarrow I_y^2 = (3)^2 + (2)^2 + (-1)^2 + (4)^2 + (4)^2 + (1)^2 + (0)^2 + (3)^2 + (2)^2$$
$$= 60$$

$$\rightarrow I_x I_y = 3\times3 + 2\times2 + 0\times(-1) + 3\times4 + 4\times4 + 2\times1 + 1\times0 + 3\times3 + 2\times2$$
$$= 56$$

بنابراین ماتریس هریس برای پنجره مورد نظر به صورت زیر است:

$$M = \begin{bmatrix} 56 & 56 \\ 56 & 60 \end{bmatrix}$$

بخش ب)

Subject:

بخش ب)

$$R = \det(M) - k[\text{trace}(M)]^2$$

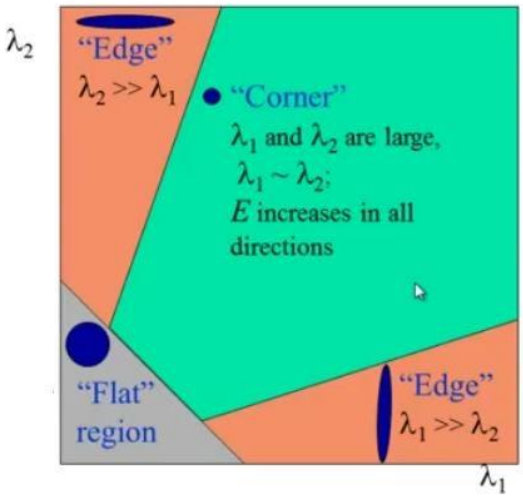
$$\det(M) = 56 \times 60 - 56 \times 56 = 56 \times (4) = 224$$

$$\text{trace}(M) = (56 + 60)^2 = 13456$$

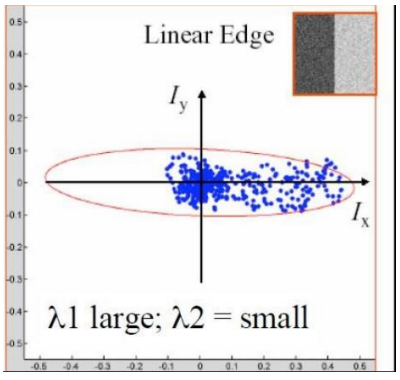
$$\Rightarrow R = 224 - 0.04 \times 13456 = -314.24$$

بخش ج) همانطور که محاسبه شد مقدار بدست آمده برای R منفی است. طبق آنچه که در اسلایدها آورده شده است این مقدار مشخص‌کنندهٔ لبه است. علت اینکه برای لبه‌ها مقدار R منفی بدست می‌آید آن است که برای این نقاط اگر چه مقدار دترمینان که به صورت ضرب دو مقدار ویژه است بزرگ بدست می‌آید، اما مجذور حاصل جمع دو مقدار ویژهٔ یک ماتریس از آن بزرگتر است که این خود مشخص کنندهٔ آن است که شعاع بیضی فیت شده به نمودار گرادیان در دو جهت x و y به صورتی است که یک طرف بزرگتر و طرف دیگر کوچکتر است.

این موضوع نیز در شکل زیر واضح است:

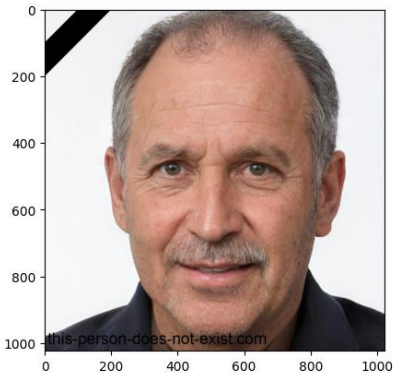


همچنین نمودار مربوط به این ماتریس نیز می‌تواند به صورت زیر می‌باشد:

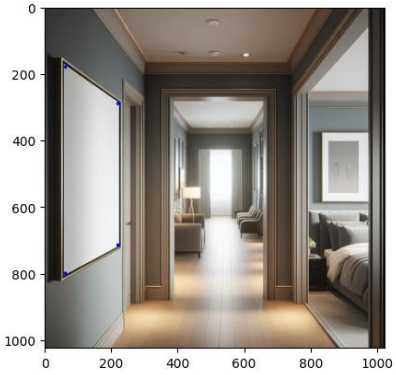


سوال چهارم

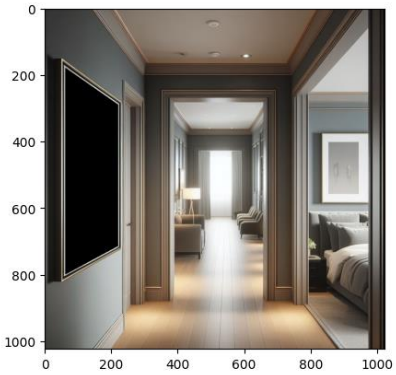
برای ربان مشکی از یک ذوزنقه استفاده می‌کنیم. به این صورت که یک ذوزنقهٔ مشکی را با استفاده از تابع *Polygon* به یک گوشه از تصویر مربوط به پدربزرگ اضافه می‌کنیم. برای بدست آوردن مختصات نقاط مربوط به چهارگوشهٔ این ذوزنقه مقادیر مختلف را امتحان کرده و در نهایت بهترین را انتخاب می‌کنیم.



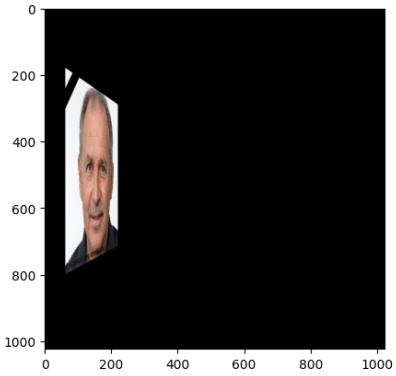
برای تصویر اصلی نیز در ابتدا مقادیر مختلف را امتحان می‌کنیم و بر روی تصویر نقطه می‌گذاریم تا چهارگوشهٔ مورد نظر را پیدا کنیم. در نهایت این چهار گوشه بدست می‌آیند.



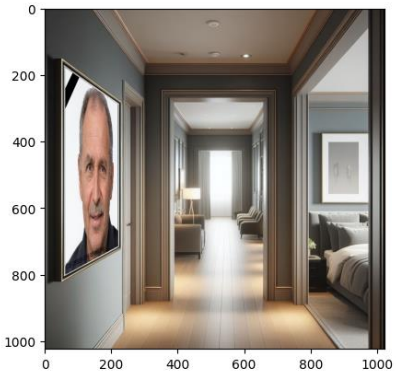
حال که این نقاط را بدست آوردیم، می‌توانیم باری دیگر از تابع *polygon* استفاده کنیم تا یک ذوزنقهٔ مشکی را بر روی این بخش از تصویر قرار دهیم.



در ادامه و با استفاده از *Perspective* تصویر مربوط به پدربزرگ را تبدیل می‌کنیم و تصویر زیر را بدست می‌آوریم:



در پایان تصویر محیط را با این تصویر بدست آمده جمع می‌کنیم تا ناحیهٔ مرتبط با تابلو که مشکی شده است، کاملاً با این تصویر مربوط به پدربزرگ جایگذاری شود.



سوال پنجم

بخش الف)

سوال پنجم

بخش الف) ابتدا تبدیل affine داریم:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} =$$

با توجه به وجود ۳ پارامتر و در نتیجه ۳ درجه آزادی، نیاز به ۳ نقطه داریم. بنابراین نقاط A، B و نقاط متناظر با آن‌ها انتخاب می‌کنیم

A: (0,0) , B: (1,0) D: (1,2)

A': (3,2) , B': (4,1) , D': (1,2)

داریم:

$$(A \to A'): \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \Rightarrow t_x = 3,\ t_y = 2 \quad (I)$$

$$(B \to B'): \begin{bmatrix} 4 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \overset{(I)}{\Rightarrow} \begin{bmatrix} 4 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & 3 \\ a_{21} & a_{22} & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 4 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} + 3 \\ a_{21} + 2 \end{bmatrix} \Rightarrow \begin{cases} a_{11} = 1 & (II) \\ a_{21} = -1 & (III) \end{cases}$$

$$(D \to D'): \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \overset{(I)}{\Rightarrow} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} a_{11} + 2a_{12} + 3 \\ a_{21} + 2a_{22} + 2 \end{bmatrix}$$

IDEA

$$\Rightarrow \begin{cases} a_{11} + 2a_{12} = -2 & (IV) \\ a_{21} + 2a_{22} = 0 & (V) \end{cases}$$

بخش ب)

II, III, IV, V $\Rightarrow$ $\begin{cases} a_{12} = -1.5 \\ a_{22} = 0.5 \end{cases}$

بنابراین تبدیل نهایی به‌دست آمده به صورت زیر است:

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} 1 & -1.5 & 3 \\ -1 & 0.5 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}$$

حال برای محاسبات اولیه C برابر با (2و1) است. تحت تبدیل بالا داریم:

$$C' = \begin{bmatrix} 1 & -1.5 & 3 \\ -1 & 0.5 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 + (-1.5) + 3 \\ (-2) + (0.5) + 2 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 0.5 \end{bmatrix}$$ → که مطابق شکل درست است

محاسبات اولیه E برابر است با (0,1). تحت تبدیل affine به‌دست آمده داریم:

$$E' = \begin{bmatrix} 1 & -1.5 & 3 \\ -1 & 0.5 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 + (-1.5) + 3 \\ 0 + (0.5) + 2 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 2.5 \end{bmatrix}$$ → که مطابق شکل صحیح است

سوال ششم

نوع تبدیل	انتقال	*Rigid*	شباهت	*Affine*	تصویری
فاصله جفت نقاط ثابت میماند	صحیح	صحیح	غلط	غلط	غلط
زاویه بین جفت خط ثابت میماند	صحیح	صحیح	صحیح	غلط	غلط
خط ها، خط باقی مانند	صحیح	صحیح	صحیح	صحیح	صحیح
زاویه بین هر خط و محور ایکس ثابت میماند	صحیح	غلط	غلط	غلط	غلط
چهار ضلعی ها، چهار ضلعی باقی می‌مانند	صحیح	صحیح	صحیح	صحیح	صحیح
خطوط موازی، موازی باقی می‌مانند	صحیح	صحیح	صحیح	صحیح	غلط
دایره ها، دایره باقی می‌مانند	صحیح	صحیح	صحیح	غلط	غلط
نسبت بین مساحت دو شکل ثابت باقی می‌ماند	صحیح	صحیح	صحیح	صحیح	غلط

سوال هفتم

Subject:

سوال هفتم

با توجه به تبدیل هوموگرافی داریم:

$$S_2 \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

پس خواهیم داشت:

$$x_2 = \frac{x_1'}{S_2} \quad , \quad y_2 = \frac{y_1'}{S_2} \quad , \quad S_2 =$$

$$S_2 = h_{31} x_1 + h_{32} y_1 + h_{33} ;$$
$$x_2' = h_{11} x_1 + h_{12} y_1 + h_{33} ;$$
$$y_2' = h_{21} x_1 + h_{22} y_1 + h_{33}$$

بنابه فرض داریم: ① $h_{33} = 1$ ② $(0,0) = H(.(0,0))$

پس داریم:

$$S_2 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\Rightarrow \begin{cases} 0 = 0 + 0 + h_{13} \Rightarrow h_{13} = 0 \\ 0 = 0 + 0 + h_{23} \Rightarrow h_{23} = 0 \end{cases}$$

همچنین از طرفی بر اثر صورت ندن خطوط، نقاط تلاقی آن در بی نهایت قرار خواهد گرفت. بنابراین مختصات همگن نقاط به صورت $(a,b,0)$ در خواهد آمد.

بنابراین داریم:

۱- محاسبه نقاط تلاقی در هر دسته خط:

① : $\begin{cases} x = 5 \\ 2x = y \\ x + 5 = y \end{cases} \Rightarrow x = 5, y = 10 \Rightarrow$ نقطه تلاقی در مختصات همگن : $(5,10,1)$

② : $\begin{cases} x = -5 \\ x + y = 5 \\ 2x + y = \end{cases} \Rightarrow x = -5, y = 10 \Rightarrow$ نقطه تلاقی در مختصات همگن : $(-5,10,1)$

IDEA

Subject:

۲- یافتن تبدیل H که نقاط تلاقی را هم به نقاط تلاقی ببرد؟

نقطه ①:
$$\begin{bmatrix} x_2 \\ y_2 \\ 0 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & 0 \\ h_{21} & h_{22} & 0 \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 10 \\ 1 \end{bmatrix}$$

$$\Rightarrow 5h_{31} + 10h_{32} + 1 = 0 \quad (I)$$

نقطه ②:
$$\begin{bmatrix} x_2 \\ y_2 \\ 0 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & 0 \\ h_{21} & h_{22} & 0 \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} -5 \\ 10 \\ 1 \end{bmatrix}$$

$$\Rightarrow -5h_{31} + 10h_{32} + 1 = 0 \quad (II)$$

$$(I), (II) \Rightarrow 20h_{32} = -2 \Rightarrow h_{32} = -0.1 \Rightarrow h_{31} = 0$$

طبق مطالبی که در اسلایدها آمده شده است از چهار درجه آزادی ماتریس تبدیل معکوس پذیری آن است.
بنابراین دترمینان این ماتریس نباید صفر باشد.
ماتریس H به شکل آمده با این فرض به صورت زیر است:

$$H = \begin{bmatrix} h_{11} & h_{12} & 0 \\ h_{21} & h_{22} & 0 \\ 0 & -0.1 & 1 \end{bmatrix}$$

داریم:

$$|H| \neq 0 \Rightarrow h_{11} \times (h_{22} - 0) + h_{12} \times (h_{21}) + 0 \times (-0.1 \times h_{21}) \neq 0$$
$$\Rightarrow h_{11}h_{22} - h_{12}h_{21} \neq 0$$
$$\Rightarrow h_{11}h_{22} \neq h_{12}h_{21}$$

بنابراین برای برقرار بودن ماتریس نیاز است که شرط لازم زیر برقرار باشد:

$$h_{11}h_{22} \neq h_{12}h_{21}$$

سوال هشتم

سوال هشتم

بخش الف) برای انتقال نقاط کانن X به مختصات ناهمگن از همان Scaling Factor همان عضو آخر بردار X است، استفاده می‌کنیم، یعنی داریم:

$$X = \begin{bmatrix} 0 \\ 2 \\ 2 \\ 1 \end{bmatrix} \Rightarrow S=1 \Rightarrow X_u = \begin{bmatrix} 0/1 \\ 2/1 \\ 2/1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix}$$

حال برای محاسبه مختصات پس از انتقال داریم:

$$X_1 = \%PX = \begin{bmatrix} 5 & -14 & 2 & 17 \\ -10 & -5 & -10 & 50 \\ 10 & 2 & -11 & 19 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 2 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 5\times0 + 2\times(-14) + 2\times2 + 1\times17 \\ -10\times0 + (-5)\times2 + (-10)\times2 + 50\times1 \\ 10\times0 + 2\times2 + (-11)\times2 + 19\times1 \end{bmatrix}$$

$$= \begin{bmatrix} -7 \\ 20 \\ 1 \end{bmatrix} \rightarrow \text{مختصات ناهمگن پرسپکتیو}$$

پس دوباره نیاز است که بر Scaling Factor تقسیم کنیم تا مختصات دکارتی پرسپکتیو به دست آید:

$$\rightarrow X = \begin{bmatrix} -7/1 \\ 20/1 \end{bmatrix} = \begin{bmatrix} -7 \\ 20 \end{bmatrix}$$

بخش ب) محاسبه ماتریس داخلی دوربین.

طبق اسلایدها ماتریس M به صورت زیر است:

$$M = \begin{bmatrix} f/p_u & 0 & c_x \\ 0 & f/p_v & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

که در این رابطه p_u و p_v همان فاصله هر پیکسل در راستای u و v هستند.

Subject:

با توجه به اطلاعات و اعداد داده شده، داریم:

$$M = \begin{bmatrix} \frac{5}{0.02} & 0 & 500 \\ 0 & \frac{5}{0.02} & 500 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 250 & 0 & 500 \\ 0 & 250 & 500 \\ 0 & 0 & 1 \end{bmatrix}$$

محاسبه ماتریس خارجی: از آنجایی که مبدا کانونی دوربین و مبدا در نظر گرفته شده یکی هستند بنابراین ماتریس Rotation برابر با ماتریس همانی شده و بردار انتقال به صورت $\vec{t} = (0,0,0)$ در خواهد آمد. بنابراین ماتریس extrinsic (بیرونی) دوربین به صورت زیر خواهد شد:

$$[RIt] = \begin{bmatrix} I_{3\times3} & \vec{0}_{3\times1} \\ 0_{1\times3} & 1_{1\times1} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

حال جهت به دست آمدن ماتریس P به صورت زیر ماتریس M را گسترش می دهیم:

$$M = \begin{bmatrix} 250 & 0 & 500 & 0 \\ 0 & 250 & 500 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

محاسبه افکشن:

$$\begin{bmatrix} u \\ v \\ 3 \end{bmatrix} = M \cdot [RIt] \cdot \begin{bmatrix} x \\ 1 \end{bmatrix} \rightsquigarrow x = \begin{bmatrix} 100 \\ 150 \\ 800 \end{bmatrix}$$

حال داریم:

$$M \cdot [RIt] = \begin{bmatrix} 250 & 0 & 500 & 0 \\ 0 & 250 & 500 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 250 & 0 & 500 & 0 \\ 0 & 250 & 500 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

IDEA

$$M \cdot [R|t] \cdot \begin{bmatrix} x \\ 1 \end{bmatrix} = \begin{bmatrix} 250 & 0 & 500 & 0 \\ 0 & 250 & 500 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 100 \\ 150 \\ 300 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 100\times250 + 0\times150 + 500\times300 + 0\times1 \\ 0\times100 + 250\times150 + 500\times300 + 0\times1 \\ 0\times100 + 0\times150 + 1\times300 + 0\times1 \end{bmatrix}_{3\times1}$$

$$= \begin{bmatrix} 65000 \\ 77500 \\ 300 \end{bmatrix} \rightarrow u = 65000,\ v = 77500,\ s = 300$$

$$\Rightarrow x = \frac{u}{s} = \frac{65000}{300} = 81.25,\quad y = \frac{v}{s} = \frac{77500}{300} = 96.375$$

از آنجایی که نیاز است در اینجا عدد نهایی عدد صحیح باشد، کف عدد نهایی را به دست می‌آوریم. پس:

$$x = \lfloor 81.25 \rfloor = 81 \quad , \quad y = \lfloor 96.375 \rfloor = 96$$