به نام خدا

تمرین چهارم – بینایی کامپیوتر **نیایش خانی ۹۹۵۲۱۲۳۵**

سوال ۱)

الف) ابتدا تصویر را می خوانیم و از آنجایی که در opencv فرمت عکس ها BGR است، آن را به RGB تبدیل می کنیم.

```
#Read image and  convert from BGR to RGB
image1 = cv2.imread('images/1.jpg')
image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
```

سپس تابع RGB_to_CMYK را پیاده‌سازی می‌کنیم.

```
def RGB_to_CMYK(r, g, b, RGB_SCALE=255, CMYK_SCALE=100):
    # Normalize the RGB values to the range [0, 1]
    r /= RGB_SCALE
    g /= RGB_SCALE
    b /= RGB_SCALE

    # Calculate the K (black) value
    k = 1 - max(r, g, b)

    if k == 1:
        # If K is 1, C, M, and Y are all zero (since it's completely black)
        c = 0
        m = 0
        y = 0
    else:
        # Calculate the CMY values
        c = (1 - r - k) / (1 - k)
        m = (1 - g - k) / (1 - k)
        y = (1 - b - k) / (1 - k)

    c = round(c * CMYK_SCALE, 2)
    m = round(m * CMYK_SCALE, 2)
    y = round(y * CMYK_SCALE, 2)
    k = round(k * CMYK_SCALE, 2)

    return c, m, y, k
```

این تابع مقادیر RGB (از ۰ تا ۲۵۵) را به مقادیر CMYK (۰ تا ۱۰۰) تبدیل می کند. سپس مقادیر RGB را در مقیاس ۰ تا ۱ نرمال کرده، سیاه (K) را محاسبه می کند و سپس فیروزه ای، سرخابی و زرد را بر اساس مقدار سیاه بدست می‌آورد. اگر K=1

باشد، کاملاً سیاه است، بنابراین C، M و Y روی ۰ تنظیم می شوند. در غیر این صورت، آنها از مقادیر عادی RGB محاسبه می شوند. هنگام کار با تصاویر، مقادیر پیکسل معمولاً باید اعداد صحیح باشند بنابراین این مقادیر را گرد می‌کنیم.

سپس تابع CMYK_to_RGB را به این صورت پیاده سازی می‌کنیم:

```
def CMYK_to_RGB(c, m, y, k, RGB_SCALE=255, CMYK_SCALE=100):
    # Normalize the CMYK values to the range [0, 1]
    c /= CMYK_SCALE
    m /= CMYK_SCALE
    y /= CMYK_SCALE
    k /= CMYK_SCALE

    # Calculate the RGB values
    r = (1 - c) * (1 - k)
    g = (1 - m) * (1 - k)
    b = (1 - y) * (1 - k)

    # Scale to the RGB_SCALE
    r = round(r * RGB_SCALE)
    g = round(g * RGB_SCALE)
    b = round(b * RGB_SCALE)

    return r, g, b
```

این تابع مقادیر CMYK را در مقیاس ۰ تا ۱ نرمال کرده، سپس مقادیر RGB را بر اساس رابطه معکوس بین CMYK و RGB محاسبه می کند. این مقادیر را به محدوده استاندارد RGB (به طور پیش فرض ۰ تا ۲۵۵) کاهش می دهد و آنها را برمی‌گرداند.

ب) در قسمت بعدی برای تبدیل RGB به HSI به این صورت عمل می کنیم.

```
def RGB_to_HSI(r, g, b):
    r /= 255.0
    g /= 255.0
    b /= 255.0

    def calc_hue(r, g, b):
        numerator = 0.5 * ((r - g) + (r - b))
        denominator = np.sqrt((r - g)**2 + (r - b) * (g - b))
        theta = np.arccos(numerator / denominator)

        if b <= g:
            return theta
        else:
            return 2 * np.pi - theta

    def calc_saturation(r, g, b):
        min_val = min([r, g, b])
        saturation = 1 - 3 * min_val / (r + g + b)
        return saturation

    def calc_intensity(r, g, b):
        return (r + g + b) / 3

    h = calc_hue(r, g, b)
    s = calc_saturation(r, g, b)
    i = calc_intensity(r, g, b)

    return h, s, i
```

این تابع مقادیر RGB را می گیرد، آنها را در محدوده [۰، ۱] نرمال می کند و سپس مقادیر Hue، Saturation و Intensity (HSI) را محاسبه می کند. Hue با استفاده از تابع آرکوزین، اشباع به صورت یک منهای نسبت حداقل مقدار RGB به مجموع مقادیر RGB و شدت به عنوان میانگین مقادیر RGB محاسبه می شود. تابع مقادیر HSI محاسبه شده را برمی گرداند.

در آخر برای نشان دادن نتایج هر یک از این تبدیل ها از این کد استفاده می‌کنیم :

```
original_data = np.array(image1)
def process_image(image_data):
    vectorized_cmyk = np.vectorize(RGB_to_CMYK, signature='(),(),()->(),(),(),()')
    vectorized_rgb = np.vectorize(CMYK_to_RGB, signature='(),(),(),()->(),(),()')
    vectorized_hsi = np.vectorize(RGB_to_HSI, signature='(),(),()->(),(),()')
    # RGB to CMYK
    cmyk = vectorized_cmyk(image_data[...,0], image_data[...,1], image_data[...,2])
    cmyk_image = np.stack(cmyk, axis=-1)
    # CMYK to RGB
    rgb_from_cmyk = vectorized_rgb(cmyk_image[...,0], cmyk_image[...,1],
cmyk_image[...,2], cmyk_image[...,3])
    rgb_from_cmyk_image = np.stack(rgb_from_cmyk, axis=-1)
    # RGB to HSI
    hsi = vectorized_hsi(image_data[...,0], image_data[...,1], image_data[...,2])
    hsi_image = np.stack(hsi, axis=-1)
    return cmyk_image, rgb_from_cmyk_image, hsi_image
cmyk_image, rgb_from_cmyk_image, hsi_image = process_image(original_data)
# Display images
plt.figure(figsize=(18, 9))
# Original Image
plt.subplot(1, 4, 1)
plt.imshow(image1)
plt.title('Original RGB Image')
plt.axis('off')
# RGB to CMYK
plt.subplot(1, 4, 2)
plt.imshow(cmyk_image)
plt.title('RGB to CMYK')
plt.axis('off')
# CMYK to RGB
plt.subplot(1, 4, 3)
plt.imshow(rgb_from_cmyk_image)
plt.title('CMYK to RGB')
plt.axis('off')
# RGB to HSI
plt.subplot(1, 4, 4)
plt.imshow(hsi_image)
plt.title('RGB to HSI')
plt.axis('off')
plt.show()
```

ابتدا تصویر اصلی را به یک آرایه numpy تبدیل می‌کنیم. سپس فانکشن های تبدیل ها را برای استفاده بر روی آرایه‌های numpy به فرمت برداری درمی‌آوریم. در مرحله بعد تصویر RGB را به فضای رنگی CMYK تبدیل کرده و کانال‌های CMYK را به یک آرایه numpy تبدیل می‌کنیم. مراحل مشابهی برای تبدیل CMYK به RGB و RGB به HSIانجام می‌شود.

در نهایت، این تابع سه تصویر تبدیل شده را برمی‌گرداند. این فانکشن را کال کرده و تصاویر تبدیل شده را ذخیره می‌کنیم. سپس تصاویر تبدیل شده را نمایش می‌دهیم.

تصویر اصلی، تصویر تبدیل شده بهCMYK ، تصویر تبدیل شده از CMYK به RGB و تصویر تبدیل شده به HSI در زیر مشاهده می‌شود :

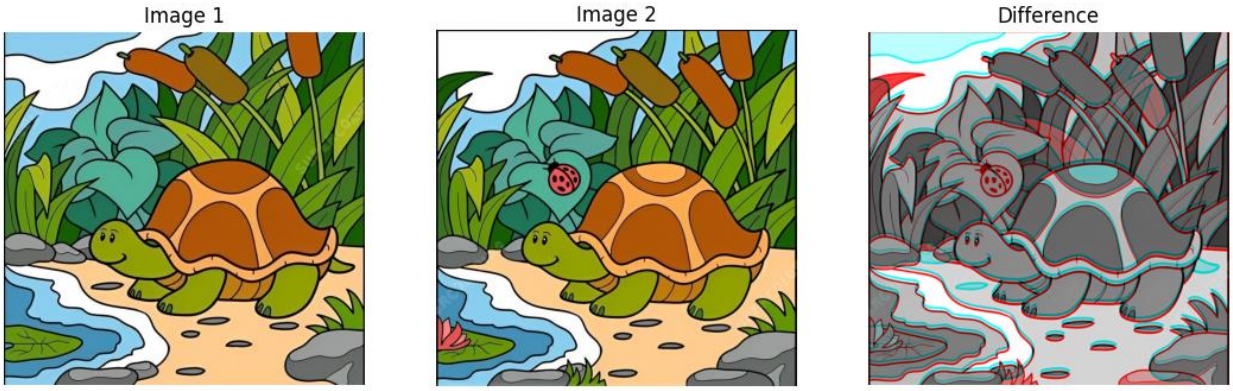
سوال ۲) ابتدا عکس ها را می خوانیم و مطابق اسلاید برای پیدا کردن تفاوت، به خاکستری تبدیل می کنیم.

```
# Read images and convert them to gray scale

image1 = cv2.imread('images/2.jpg')
image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
image1_gray = cv2.imread(('images/2.jpg'), cv2.IMREAD_GRAYSCALE)

image2 = cv2.imread('images/3.jpg')
image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
image2_gray = cv2.imread(('images/3.jpg'), cv2.IMREAD_GRAYSCALE)
```

سپس با استفاده از کد زیر تابعی را تعریف می کنیم که با ترکیب دو تصویر ورودی یک تصویر جدید ایجاد می کند. ابتدا اطمینان حاصل می کنیم که تصویر خروجی دارای عرض و ارتفاع یکسان با کوچکتر از دو تصویر ورودی است. سپس یک تصویر را با سه کانال رنگی (که نشان دهنده قرمز، سبز و آبی است) مقداردهی اولیه می کنیم و سپس این کانال ها را با داده های پیکسلی از دو تصویر ورودی پر می کنیم. کانال قرمز را از تصویر اول و کانال های سبز و آبی را از تصویر دوم می گیریم. سپس تابع تصویر ترکیبی حاصل را برمی گرداند که پیکسل هایی که فقط در یکی از تصاویر وجود داشته است با رنگ تصویر دیگر نمایان می شود.

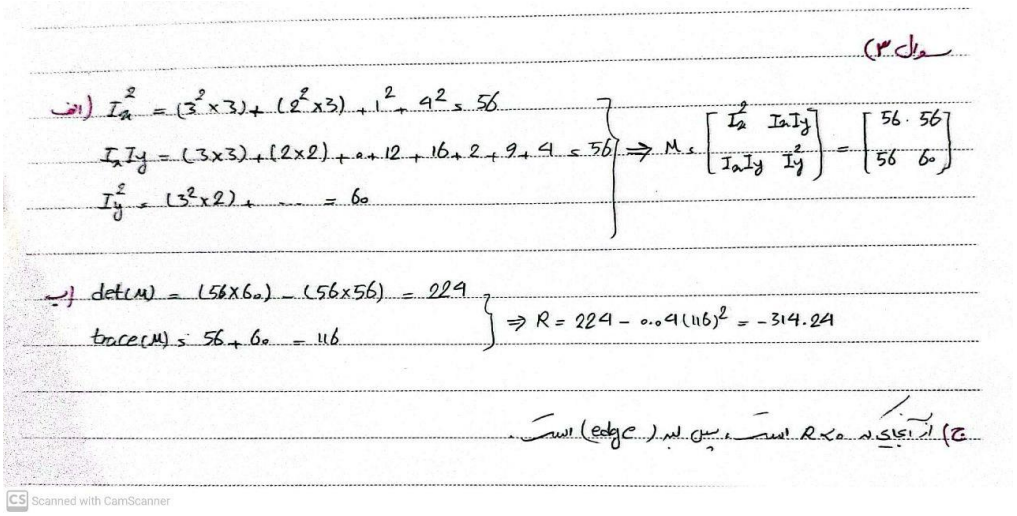
```python
def dif(image1, image2):
    # Ensure both images have the same minimum width and height
    min_width = min(image1.shape[1], image2.shape[1])
    min_height = min(image1.shape[0], image2.shape[0])
    # Create a result image with the same width, height, and three color channels
    result_shape = (min_height, min_width, 3)  # (Height, Width, Channels)
    result = np.zeros(result_shape, dtype=np.uint8)  # Initialize with zeros
    # Copy the red channel from the first image
    result[:, :, 0] = image1[:min_height, :min_width]  # Red
    # Copy the green channel from the second image
    result[:, :, 1] = image2[:min_height, :min_width]  # Green
    # Copy the blue channel from the second image
    result[:, :, 2] = image2[:min_height, :min_width]  # Blue
    return result
```

با اعمال این تابع بر روی دو تصویر، خروجی به این صورت خواهد بود :



همانطور که مشاهده می شود تفاوت ها مشخص شده اند.

سوال ۳)



سوال ۴) ابتدا دو تصویر پدربزرگ و اتاق را می‌خوانیم و آنها را نمایش می دهیم. سپس برای اضافه کردن روبان سیاه این کار را انجام می دهیم :

```
output_image = image1.copy()
# Define the ribbon's thickness
ribbon_thickness = 50
# Define start and end points for the ribbon
start_point = (0, output_image.shape[0] // 3)
end_point = (output_image.shape[1] // 3, 0)
# Draw the ribbon
cv2.line(
    output_image,
    start_point,
    end_point,
    (0, 0, 0),  # Black color
    ribbon_thickness  # Line thickness)
# Save the result image to a file
output_filename = "images/grandpa_with_ribbon.jpeg"  # You can specify any file name and
format
cv2.imwrite(output_filename, cv2.cvtColor(output_image, cv2.COLOR_RGB2BGR))  # OpenCV saves in
BGR format
# Display the resulting image
plt.imshow(output_image)
plt.axis('off')
plt.show()
```

ابتدا یک کپی از تصویر تهیه می کنیم تا این کارها را برروی آن انجام دهیم. در ادامه ضخامت روبان و نقاط شروع و پایان دلخواه خودمان را مشخص می کنیم. سپس از cv2.line برای کشیدن یک روبان سیاه از start_point تا end_point با ضخامت مشخص شده استفاده می کنیم. در آخر تصویر حاصل را در یک فایل دیگر ذخیره کرده و آن را نمایش می دهیم که خروجی به صورت زیر خواهد بود :

در ادامه برای قرار دادن عکس پدربزرگ روی قاب موجود در تصویر اتاق اینگونه عمل می کنیم :

```
final_image = output_image.copy()
# Manually specify the coordinates of the four corners of the frame on the base image
frame_points = np.array([
    [60, 170],  # Top-left
    [220, 290],  # Top-right
    [220, 710],  # Bottom-right
    [60, 800],  # Bottom-left
], dtype=np.float32)
# Get the original dimensions of the grandpa image
overlay_height, overlay_width, _ = final_image.shape
# Define the original frame's coordinates (full size)
original_points = np.array([
    [0, 0],  # Top-left
    [overlay_width, 0],  # Top-right
    [overlay_width, overlay_height],  # Bottom-right
    [0, overlay_height],  # Bottom-left
], dtype=np.float32)
```

مانند قبل یک کپی از تصویر تهیه می کنیم. سپس مختصات نقاط اطراف قاب را که می خواهیم روی تصویر پایه قرار دهیم، به صورت دستی تعریف می‌کنیم.

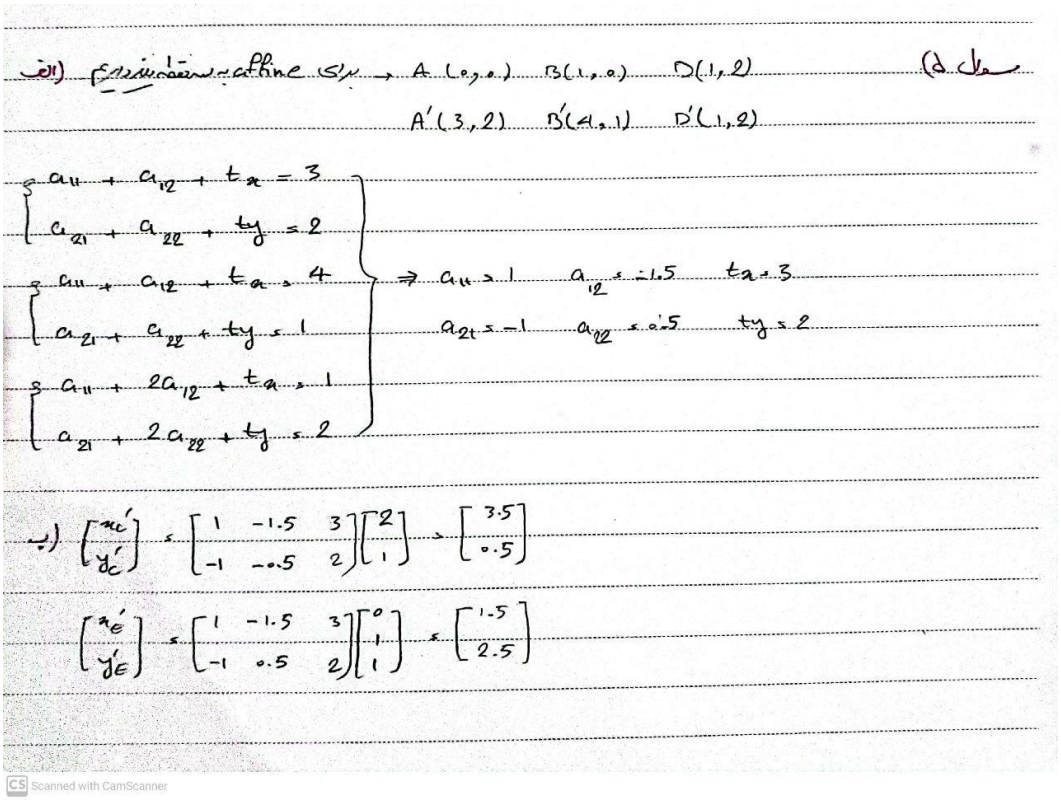
```
# Compute the transformation matrix
transformation_matrix = cv2.getPerspectiveTransform(original_points, frame_points)
```

سپس ماتریس تبدیل را با استفاده از cv2.getPerspectiveTransform() دریافت می کنیم تا نقاط اصلی را به نقاط قاب مشخص شده نگاشت کنیم.

```
# Apply the transformation to the overlay image
transformed_overlay = cv2.warpPerspective(final_image, transformation_matrix,
(image2.shape[1], image2.shape[0]))
# Create a mask to determine where to overlay the transformed image
overlay_mask = np.zeros_like(image2)
cv2.fillPoly(overlay_mask, [frame_points.astype(np.int32)], (255, 255, 255))
# Use the mask to overlay the transformed image onto the background
final_image = cv2.bitwise_and(image2, cv2.bitwise_not(overlay_mask))  # Clear the region for
the overlay
final_image = cv2.bitwise_or(final_image, transformed_overlay)  # Overlay the transformed
image
# Display the resulting image
plt.imshow(final_image)
plt.axis('off')
plt.show()
```

حال این تبدیل را با استفاده از cv2.warpPerspective() روی تصویر همپوشانی اعمال می کنیم. یک mask برای شناسایی محل قرار دادن همپوشانی تبدیل شده ایجاد کرده و سپس آن ناحیه را روی تصویر پایه پاک می کنیم و تصویر تبدیل شده را روی آن قرار می دهیم. در آخر تصویر ترکیبی نهایی را نمایش می دهیم. خروجی به این صورت خواهد بود :

سوال ۵)



سوال ۶)

نوع تبدیل	انتقال	rigid	شباهت	affine	تصویری
فاصله جفت نقطات ثابت میماند	✓	✓	✗	✗	✗
زاویه بین جفت خط ثابت میماند	✓	✓	✓	✗	✗
خط ها، خط باقی مانند	✓	✓	✓	✓	✗
زاویه بین هر خط و محور ایکس ثابت میماند	✓	✗	✗	✗	✗
چهار ضلعی ها، چهار ضلعی باقی می مانند	✓	✓	✓	✓	✓
خطوط موازی، موازی باقی می مانند	✓	✓	✓	✓	✗
دایره ها، دایره باقی می مانند	✓	✓	✓	✓	✓
نسبت بین مساحت دو شکل ثابت باقی می ماند	✓	✓	✗	✗	✗

سوال ۸)