PS1 - Isaque Fernando Report

Question 1

Imports and Global Constants

```
import cv2 as cv
import sys
from matplotlib import pyplot as plt

drawing = False
point = (0, 0)
```

A

```
def step_1() -> None:
    """Carrega uma imagem e a exibe.

    Returns: None.
    """
    img = cv.imread('../imgs/lenna.png', )
    # verificando o carregamento da imagem
    if img is None:
        sys.exit("Não foi possível abrir a imagem")

    # exibindo a mensagem
    cv.imshow('Exemplo', img)
    key = cv.waitKey(0)

    if key == ord('s'):
        cv.imread('../imgs/lenna.png')
```

B

```
def step_2() -> None:
    """Calculando o histograma.

    Returns: None.
    """
    img = cv.imread('../imgs/lenna.png')
    colors = ('b', 'g', 'r')
    for i, col in enumerate(colors):
        hist = cv.calcHist([img], [i], None, [256], [0, 256])
        plt.plot(hist, color=col)
        plt.xlim([0, 256])
```

```
    plt.title('Histogramas')
    plt.show()
```

C

```
def step_3() -> None:
    """Obtendo dados da imagem em tempo real.

    Returns: None.
    """
    img = cv.imread('../imgs/lenna.png')

    # função de intensidade
    def intensity(red: int, green: int, blue: int) -> float:
        """Calcula a intensidade de um pixel.

        Args:
            red: Escala de vermelho.
            green: Escala de verde.
            blue: Escala de azul.

        Returns: Intensidade do pixel
        """
        return round(sum([red, green, blue])/3, 2)

    # callback
    def mouse_drawing(event, x, y, flags, params) -> None:
        global point, drawing
        if event == cv.EVENT_MOUSEMOVE:
            drawing = True
            point = (x, y)

    cv.namedWindow('Rectangle')
    cv.setMouseCallback('Rectangle', mouse_drawing)

    while True:
        frame = img  # nova referência ao objeto da imagem
        x = point[0]  # x do mouse
        y = point[1]  # y do mouse
        x_initial = x - 13
        y_initial = y - 13
        x_final = x + 13
        y_final = y + 13
        rect = frame[x_initial: y_initial+1, y_final: y_final + 1]  #
retângulo
        b = img[y, x, 0]  # obtendo o valor da escala blue do pixel
        g = img[y, x, 1]  # obtendo o valor da escala verde do pixel
        r = img[y, x, 2]  # obtendo o valor da escala vermelha do pixel
        # caso o evento tenha ocorrido
        if drawing:
            # desenhamos o retângulo e exibimos as informações
```

```
            cv.rectangle(frame, (x_initial, y_initial), (x_final, y_final),
(0, 255, 0), 0)
            print(f'Cursor at: {x, y}. RGB values: R: {r}, G: {g} and B:
{b}. Intensity: {intensity(r, g, b)}')
            print(f'Average gray level: {rect.mean()}, Std Deviation:
{rect.std()}')

        cv.imshow('Rectangle', frame)
        key = cv.waitKey(25)
        if drawing:
            img = cv.imread('../imgs/lenna.png')
        if key == 13:
            print('Terminado')
        elif key == 27:
            break

    cv.destroyAllWindows()
```

D - Discuss the question

Given the above exercises we tried defining the homogeneousity in terms of histograms and variances.

Using the histograms we should observe that homogeneousity is related with the constant distribution in all frequencies of the graph.

Using the standard deviation we should observe that homogeneousity is closely related with low values of variance. That represents that in mean the value is close to the average.

Portuguese Discuss

A partir dos exercícios acima, notamos que é plausível considerar a homogeneidade utilizando o histograma e a variância.

Utilizando o histograma podemos observar que a homogeneidade está relacionada a uma certa constancia na distribuição do histograma por todo o eixo x. Em outras palavras, um histograma que possui uma certa distribuição por todo o eixo x representa maior homogeneidade, uma vez que gráficos neste padrão possuem menores rupturas.

Em termos da variância, a homogeinidade está relacionda a valores de variância menores. Isso significa que os valores de cada píxel da imagem ou janela não estão tão distantes da média.

Question 2

Imports and Global Constants

```
import sys
import numpy as np
from typing import List, Tuple

import cv2 as cv
```

A

```
def step_1() -> None:
    """Realiza o passo A da questão 2.

    Returns: None.
    """
    cap = cv.VideoCapture('../videos/video.mp4')
    # verificando se o video foi carregado corretamente
    if cap is None:
        sys.exit('Erro ao ler o arquivo!')
    # quantidade de frames no vídeo
    length = int(cap.get(cv.CAP_PROP_FRAME_COUNT))
    print(length)
```

B

```
def step_2() -> Tuple[np.array, np.array, np.array]:
    """Realiza o passo B da questão 2.

    Returns: As três estatísticas calculadas da sequência de imagens.
    """
    # lendo o vídeo e iterando sobre os frames
    cap = cv.VideoCapture('../videos/video.mp4')
    # verificando se o vídeo foi carregado corretamente
    if cap is None:
        sys.exit('Erro ao ler o arquivo!')

    # iterando sobre cada frame
    success: bool = True
    # guardando as estatísticas selecionadas
    mean_per_frame = np.array([])
    std_per_frame = np.array([])
    contrast_per_frame = np.array([])
    while success:
        success, image = cap.read()
        if success:
            mean_per_frame = np.append(mean_per_frame, image.mean())
            std_per_frame = np.append(std_per_frame, image.std())
            contrast_per_frame = np.append(contrast_per_frame,
cv.cvtColor(image, cv.COLOR_BGR2GRAY).std())
    return mean_per_frame, std_per_frame, contrast_per_frame
```

C

```
def step_3() -> Tuple[List[float], List[float], List[float]]:
    """Realiza o passo C da segunda questão.

    Returns: Lista com as novas funções normalizadas
    """
    # usaremos o passo anterior para receber as funções f, g e h do vídeo
de teste
    f, g, h = step_2()  # mean per frame, std_per_frame e
contrast_per_frame
    # calculando o valor de alpha e beta para g_new
    alpha_g_new = (g.std() / f.std()) * (f.mean() - g.mean())
    beta_g_new = (f.std() / g.std())
    # calculando o valor de alpha e beta para h_new
    alpha_h_new = (h.std() / f.std()) * (f.mean() - h.mean())
    beta_h_new = (h.std() / g.std())
    # normalizando as funções
    f_new: List[float] = []
    g_new: List[float] = []
    h_new: List[float] = []
    # iterando sobre os valores para obter as novas funções
    for i, _ in enumerate(f):
        f_new.append(f[i])
        g_new.append(beta_g_new * (g[i] + alpha_g_new))
        h_new.append(beta_h_new * (h[i] + alpha_h_new))
    return f_new, g_new, h_new
```

D

```
def step_4() -> None:
    """Realiza o passo D da questão 2.

    Returns: None.
    """
    f_new, g_new, h_new = step_3()
    # para verificarmos a normalização precisamos calcular a distância
entre as funções
    d_1 = 0.0
    d_2 = 0.0
    # iterando sobre os valores das funções para obter a distância
    for i, _ in enumerate(f_new):
        d_1 += abs(f_new[i] - g_new[i])
        d_2 += abs(f_new[i] - h_new[i])
    d_1 = d_1 / len(f_new)
    d_2 = d_2 / len(f_new)
    print(f'Distance between f and g_new: {d_1}')
    print(f'Distance between f and h_new: {d_2}')
```

Discuss

After the conclusion of exercise using the example mp4 video it was possible to obtain the distance d_1 (f to g_new) and (f to h_new) close to 0. The values were `0.8635` and `0.466`. Although the values was obtained it was not possible to visualise the result of the operations.

Portuguese Discuss

Após a conclusão do exercício utilizando um vídeo mp4 como exemplo foi possível obter distâncias entre as funções normalizadas próximo de zero. Mesmo com esse cálculo sendo realizado, foi possível notar que sem a visualização do efeito na sequência original torna a operação um pouco abstrata ainda.

Dessa forma, o que podemos concluir de forma prévia é que essa operação pode ser um tipo de operação para aproximar uma imagem a um determinado padrão obtido em outra imagem.