

Universidade de Brasília

Faculdade de Tecnologia

Departamento de Engenharia Mecânica

Disciplina: VISÃO COMPUTACIONAL – PPMEC3617

Professor: José Maurício S. T. Motta

Aluno: Judah Holanda Correia Lima (252112485)

Período: 2025/2

1º Trabalho de Avaliação

▼ 1

▼ a)

Estime o ruído de seu sistema de aquisição de imagens (pode ser uma webcam ou qualquer outra) usando os passos propostos no algoritmo EST_NOISE e AUTO_COVARIANCE, expostos nos slides em sala de aula. Você precisará capturar diversas imagens da mesma cena estática, em um ambiente sem variações de iluminação (lâmpadas fluorescentes afetam a iluminação das imagens, bem como imagens externas de nuvens e árvores, sendo indicada cena interna e com fixação perfeita da câmera), plotando depois o gráfico da intensidade da imagem ao longo de uma linha e a variação produzida pelo ruído. Esta variação deverá ser pequena, da ordem de menos de 5% em geral, dependendo da qualidade da câmera. Discuta os resultados.

▼ ALGORITMO EST_NOISE

- Temos n imagens da mesma cena:

$$(E_0, E_1, \dots, E_{n-1})$$

Média em cada pixel:

$$\overline{E(i, j)} = \frac{1}{n} \sum_{k=0}^{n-1} E_k(i, j)$$

Desvio padrão em cada pixel:

$$\sigma(i, j) = \sqrt{\frac{1}{n-1} \sum_{k=0}^{n-1} (\overline{E(i, j)} - E_k(i, j))^2}$$

Média do desvio padrão para a imagem inteira:

$$\bar{\sigma} = \frac{1}{N \cdot N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \sigma(i, j)$$

Mundo Real

No entanto a imagem não necessariamente tem o mesmo tamanho na largura(W) e altura(H)

Então podemos reescrever como:

Para cada pixel ((i, j)), com (i = 0, ..., W - 1) e (j = 0, ..., L - 1):

Média do desvio padrão para toda a imagem:

$$\bar{\sigma} = \frac{1}{W \cdot L} \sum_{i=0}^{W-1} \sum_{j=0}^{L-1} \sigma(i, j)$$

AUTO_COVARIANCE:

- $c = 1/N^2$
- $(N_{i'} = N - i' - 1)$
- $(N_{j'} = N - j' - 1)$

For each (i', j' = 0, ..., N - 1):

$$C_{EE}(i', j') = c \cdot \sum_{i=0}^{N_{i'}} \sum_{j=0}^{N_{j'}} (E(i, j) - \overline{E(i, j)}) \cdot (E(i + i', j + j') - \overline{E(i + i', j + j')})$$

Mundo Real

No entanto a imagem não necessariamente tem o mesmo tamanho na largura(W) e altura(H)

Então podemos reescrever como:

- $c = 1/(L \cdot H)$
- $(W_{i'} = W - i' - 1)$
- $(H_{j'} = H - j' - 1)$

For each (i' = 0, ..., W - 1) and (j' = 0, ..., H - 1):

$$C_{EE}(i', j') = c \cdot \sum_{i=0}^{W_{i'}} \sum_{j=0}^{H_{j'}} (E(i, j) - \overline{E(i, j)}) \cdot (E(i + i', j + j') - \overline{E(i + i', j + j')})$$

Como

$$(E(i, j) - \overline{E(i, j)})$$

Não varia em relação a i' e j', podemos colocar de fora do somatório, pois o somatório de uma constante por cada elemento é o mesmo que o a constante vezes somatório de cada elemento.

$$C_{EE}(i', j') = c \cdot (E(i, j) - \overline{E(i, j)}) \cdot \sum_{i=0}^{W_{i'}} \sum_{j=0}^{H_{j'}} (E(i + i', j + j') - \overline{E(i + i', j + j')})$$

E também é possível escrever como:

$$C_{EE}(i', j') = c \cdot (E(i, j) - \overline{E(i, j)}) \cdot \left(\left(\sum_{i=0}^{W_{i'}} \sum_{j=0}^{H_{j'}} (E(i + i', j + j')) - \left(\sum_{i=0}^{W_{i'}} \sum_{j=0}^{H_{j'}} \overline{E(i + i', j + j')} \right) \right) \right)$$

Ou ainda: A variação das imagens:

$$V(i, j) = E(i, j) - \overline{E(i, j)}$$

A soma de todos os vizinhos de cada ponto:

$$N(i, j) = \sum_{i=0}^{W_{i'}} \sum_{j=0}^{H_{j'}} E(i + i', j + j')$$

A Média da soma de todos os vizinhos de cada ponto:

$$\overline{N(i, j)} = \sum_{i=0}^{W_{i'}} \sum_{j=0}^{H_{j'}} \overline{E(i + i', j + j')}$$

Resultando na covariância:

$$C_{EE}(i', j') = c \cdot V(i, j) \cdot (\overline{N(i, j)} - \overline{\overline{N(i, j)}})$$

```
import os
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import resize

def get_images(folder):
    files = [
        os.path.join(folder, f)
        for f in os.listdir(folder)
        if f.lower().endswith((".png", ".jpg", ".jpeg", ".bmp", ".npz"))
    ]

    if not files:
        raise ValueError("Nenhuma imagem encontrada na pasta.")

    first_file = files[0]

    if first_file.lower().endswith(".npz"):
        # Carrega o arquivo npz
        data = np.load(first_file)
        # Se soubermos que a chave é "resultado"
        if "resultado" in data:
            arr = data["resultado"]
        else:
            # caso contrário, pega a primeira chave disponível
            key = list(data.keys())[0]
            arr = data[key]

        height, width = arr.shape[:2]

    else:
        # Carregar como imagem comum
        img_ref = Image.open(first_file).convert("RGB")
        width, height = img_ref.size

    return {
        "files": files,
        "width": width,
        "height": height
    }

def image_to_array(file, width=None, height=None):
    if file.lower().endswith(".npz"):
        # Carregar npz
        data = np.load(file)
        # se tiver a chave "resultado", usa ela. Se não, pega a primeira existente
        arr = data["resultado"] if "resultado" in data else data[list(data.keys())]

        # garantir dtype float64
        np_img = np.array(arr, dtype=np.float64)
```

```

    if width is not None and height is not None:
        # Se precisar forçar resize (padrão igual ao das imagens PIL)
        if (np_img.shape[1], np_img.shape[0]) != (width, height):
            np_img = resize(
                np_img, (height, width, np_img.shape[2]), preserve_range=True
            ).astype(np.float64)

    else:
        # Imagem regular
        img = Image.open(file).convert("RGB")
        if width is not None and height is not None:
            img = img.resize((width, height))
        np_img = np.array(img, dtype=np.float64)

    return np_img


def save_image(output, name, img_type):
    name = f"{name}.{img_type}"
    if img_type.lower() == "npz":
        np.savez_compressed(name, resultado=output)
    else:
        output = np.clip(output, 0, 255).astype(np.uint8)
        # Salva a imagem média
        img_final = Image.fromarray(output, mode="RGB")
        img_final.save(name)
    print(f"Imagen salva como {name}")


def average_images(folder, output="average", img_type="png"):
    images = get_images(folder)
    files = images["files"]
    width = images["width"]
    height = images["height"]

    # Acumula as imagens em numpy arrays
    sum = np.zeros((height, width, 3), dtype=np.float64)

    index = 0
    for file in files:
        np_img = image_to_array(file, width, height)
        # print(f'Imagen-{index}')
        # print(np_img)
        sum += np_img
        index += 1

    # print('Soma')
    # print(sum)

    avg = sum / len(files)

    # print('Media')
    # print(avg)

    save_image(avg, output, img_type)


def variation_images(folder, avg="average", output="variation/", img_type="png"):

```

```

images = get_images(folder)
files = images["files"]
width = images["width"]
height = images["height"]
avg_name = f"{avg}.{img_type}"
# garantir tamanho
avg_img = image_to_array(avg_name, width, height)

index = 0
for file in files:
    np_img = image_to_array(file, width, height)
    # print(f'Imagen-{index}')
    # print(np_img)
    sub = np_img - avg_img

    name = f'{output}{index}'
    save_image(sub, name, img_type)
    index += 1


def neighbors_images(folder, output="neighbors/"):
    images = get_images(folder)
    files = images["files"]
    width = images["width"]
    height = images["height"]
    # garantir tamanho

    index = 0
    for file in files:
        # garantir tamanho
        img = image_to_array(file, width, height)
        # Soma total de todos os pixels (por canal)
        soma_total = np.sum(img, axis=(0, 1)) # vetor [R_total, G_total, B_total]

        # Para cada pixel, o novo valor é soma_total - pixel_atual
        resultado = soma_total - img

        name = f'{output}{index}'
        save_image(resultado, name, 'npz')
        index += 1


def deviation_images(folder, avg="average.png", output="deviation", image_type="p
images = get_images(folder)
files = images["files"]
width = images["width"]
height = images["height"]

# Acumula as imagens em numpy arrays
sum = np.zeros((height, width, 3), dtype=np.float64)
# garantir tamanho
avg_img = image_to_array(avg, width, height)

index = 0
for file in files:
    # garantir tamanho
    np_img = image_to_array(file, width, height)
    # print(f'Imagen-{index}')
    # print(np_img)

```

```

    sub = avg_img - np_img
    pow = sub ** 2
    sum += pow
    index += 1

# print('Soma')
# print(sum)

avg = sum / len(files)

# print('Media')
# print(avg)

root = avg ** 0.5

# print('Raiz')
# print(root)

save_image(root, output, image_type)

def covariance_image(variation_folder="variation", neighbors_folder="neighbors",
variation_images = get_images(variation_folder)
variation_files = variation_images["files"]
width = variation_images["width"]
height = variation_images["height"]
neighbors_images = get_images(neighbors_folder)
neighbors_files = neighbors_images["files"]
n_np_img_avg = image_to_array(neighbors_average, width, height)
c = 1 / (width * height)

s = len(variation_files)
if s != len(variation_files):
    raise Exception("Size Error")

index = 0
for index in range(0, s):
    variation_file = variation_files[index]
    neighbors_file = neighbors_files[index]

    v_np_img = image_to_array(variation_file, width, height)
    n_np_img = image_to_array(neighbors_file, width, height)

    result = n_np_img - n_np_img_avg
    result = result * v_np_img
    result = result * c
    name = f"{output}{index}"
    save_image(result, name, img_type)

def image_average(input="deviation.png"):
    img = image_to_array(input)
    avg = np.mean(img)
    print(f"Media da imagem {input}: {avg}")
    return avg

```

```

def get_arrays(folder, add=["average.png"]):
    images = get_images(folder)
    files: list = images["files"]
    width = images["width"]
    height = images["height"]
    for e in add:
        files.append(e)
    a: list = []
    for file in files:
        print(file)
        np_image = image_to_array(file, width, height)
        a.append(np_image)
    return a

def plot_gray(imgs, output="graf.png"):
    """
    Plota em escala de cinza todas as imagens e a média.
    NÃO inclui o desvio.
    """
    idx_media = len(imgs) - 1 # média é última

    plt.figure(figsize=(12, 6))

    for i, img in enumerate(imgs):
        gray = img.mean(axis=2).flatten()

        if i == idx_media: # destaque para média
            lw, alpha, estilo, nome, cor = 0.5, 1, "-", "media", "red"
        else:
            lw, alpha, estilo, nome, cor = 0.5, 1, "--", f"img{i}", "black"

        plt.plot(
            gray,
            color=cor,
            linestyle=estilo,
            lw=lw,
            alpha=alpha,
            label=nome if i == idx_media else None,
        )

    plt.title("Curvas em escala de cinza (imagens + média)")
    plt.xlabel("Índice do pixel (flattened)")
    plt.ylabel("Intensidade (0-255)")
    plt.legend()
    plt.grid(True)
    plt.savefig(output, dpi=300, bbox_inches="tight")
    plt.close()
    print(f"Gráfico (sem desvio) salvo em '{output}'")

def plot_desvio(img, output="graf_desvio.png"):
    """
    Plota apenas a imagem de desvio em grayscale.
    """
    gray = img.mean(axis=2).flatten()

    plt.figure(figsize=(12, 6))
    plt.plot(

```

```

        gray,
        color="black",
        linestyle="-",
        lw=0.5,
        label="desvio"
    )

plt.title("Curva em escala de cinza (desvio)")
plt.xlabel("Índice do pixel (flattened)")
plt.ylabel("Intensidade (0-255)")
plt.legend()
plt.grid(True)
plt.savefig(output, dpi=300, bbox_inches="tight")
plt.close()
print(f"Gráfico do desvio salvo em '{output}'")

def pixel_p(value):
    return (value / 255) * 100

average_images("pictures")
deviation_images("pictures")

dev = image_average()
dev_p = pixel_p(dev)

variation_images("pictures")
neighbors_images("pictures")
average_images("neighbors", "neighbors_average", "npz")

covariance_image()

var = image_average("variation_average.png")
var_p = pixel_p(var)

print(f"% do devio médio: {dev_p}")
print(f"% da variação média: {var_p}")

arrays = get_arrays("pictures")

plot_gray(arrays)
plot_desvio(image_to_array("deviation.png"))
plot_desvio(image_to_array("variation_average.png"), "graf_variation.png")

```

▼ Media das imagens:



CAPTURED WITH
SAYCHEESE

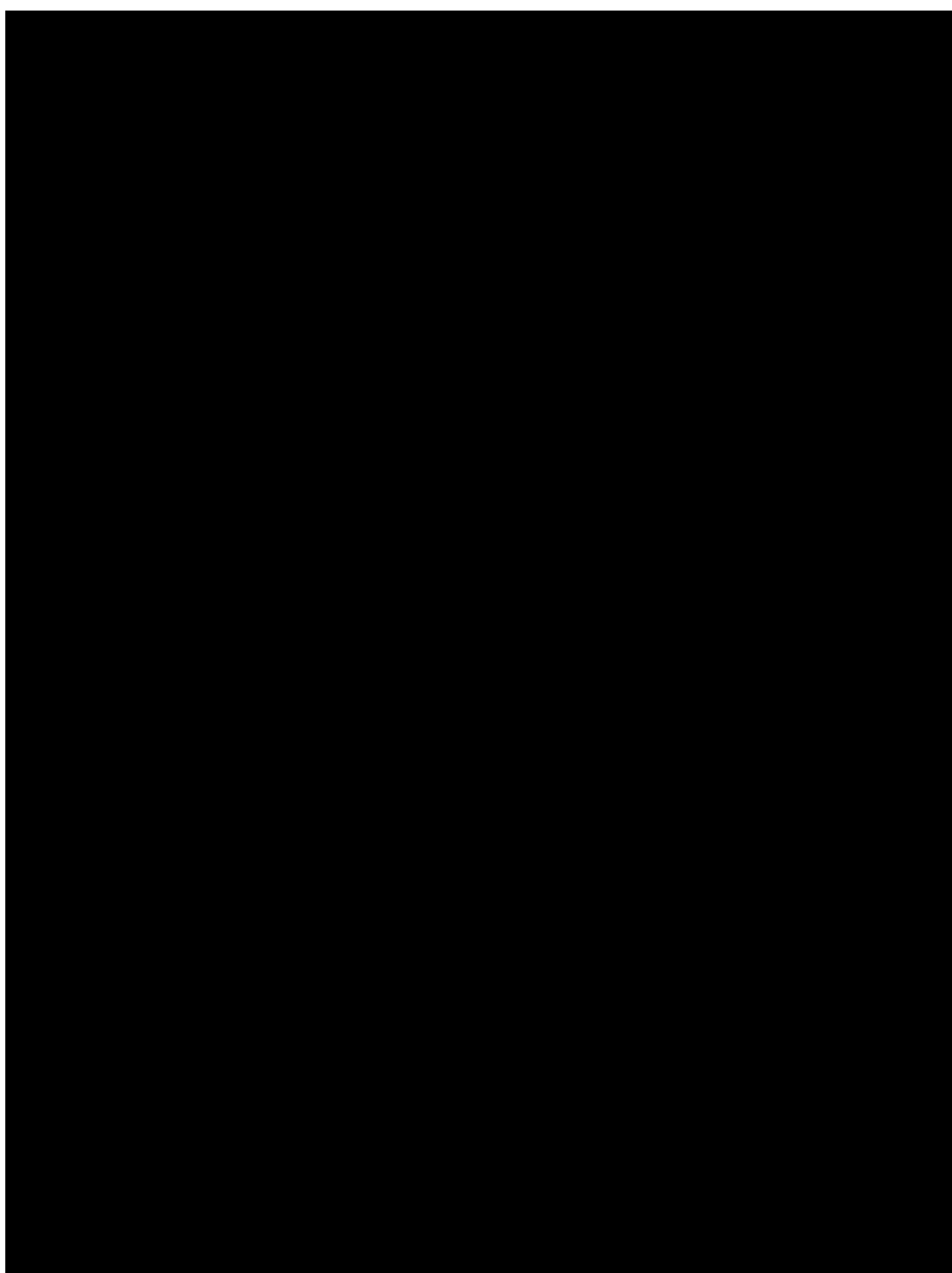
- ▼ Imagem da média do desvio:

Como é todo pequeno, é todo preto



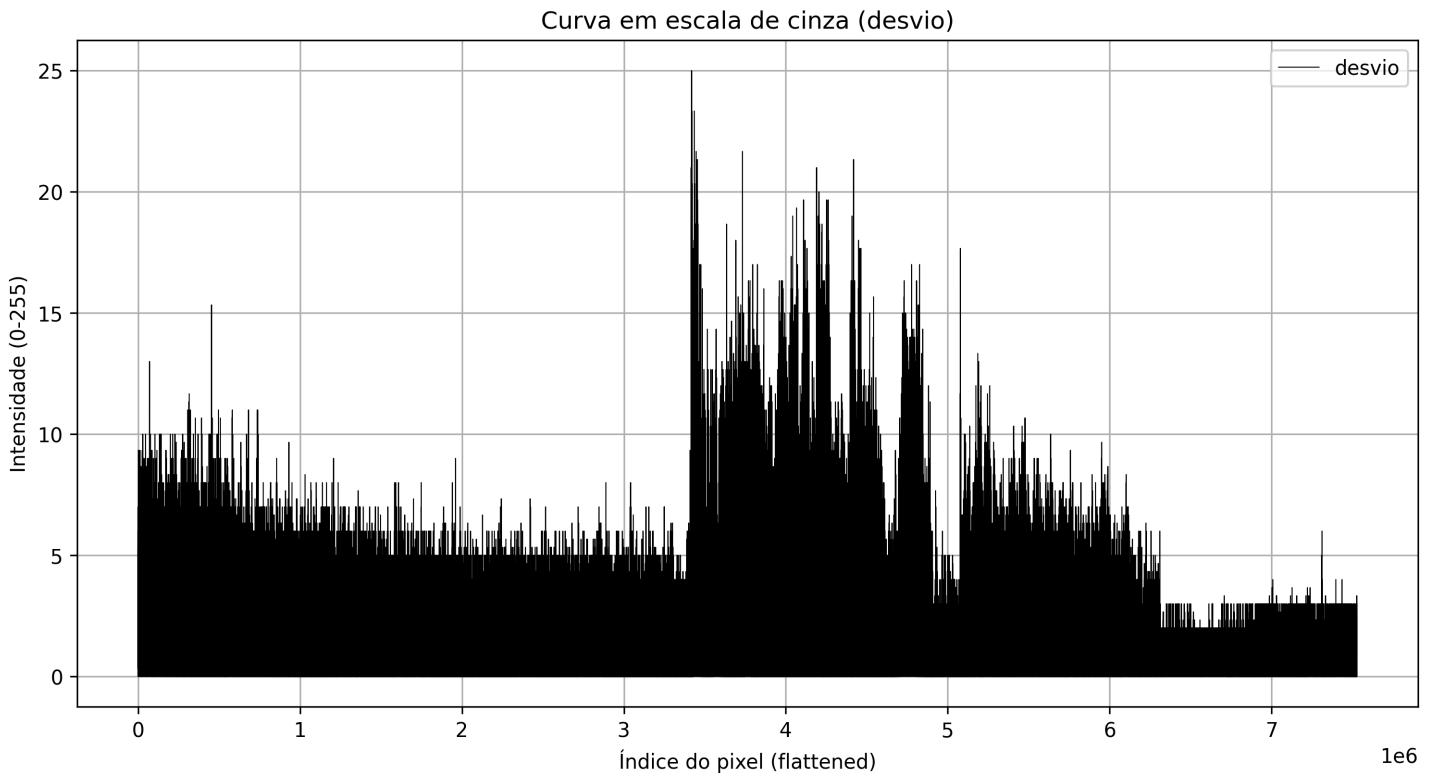
- ▼ Imagem da média do desvio:

Como é todo pequeno, é todo preto

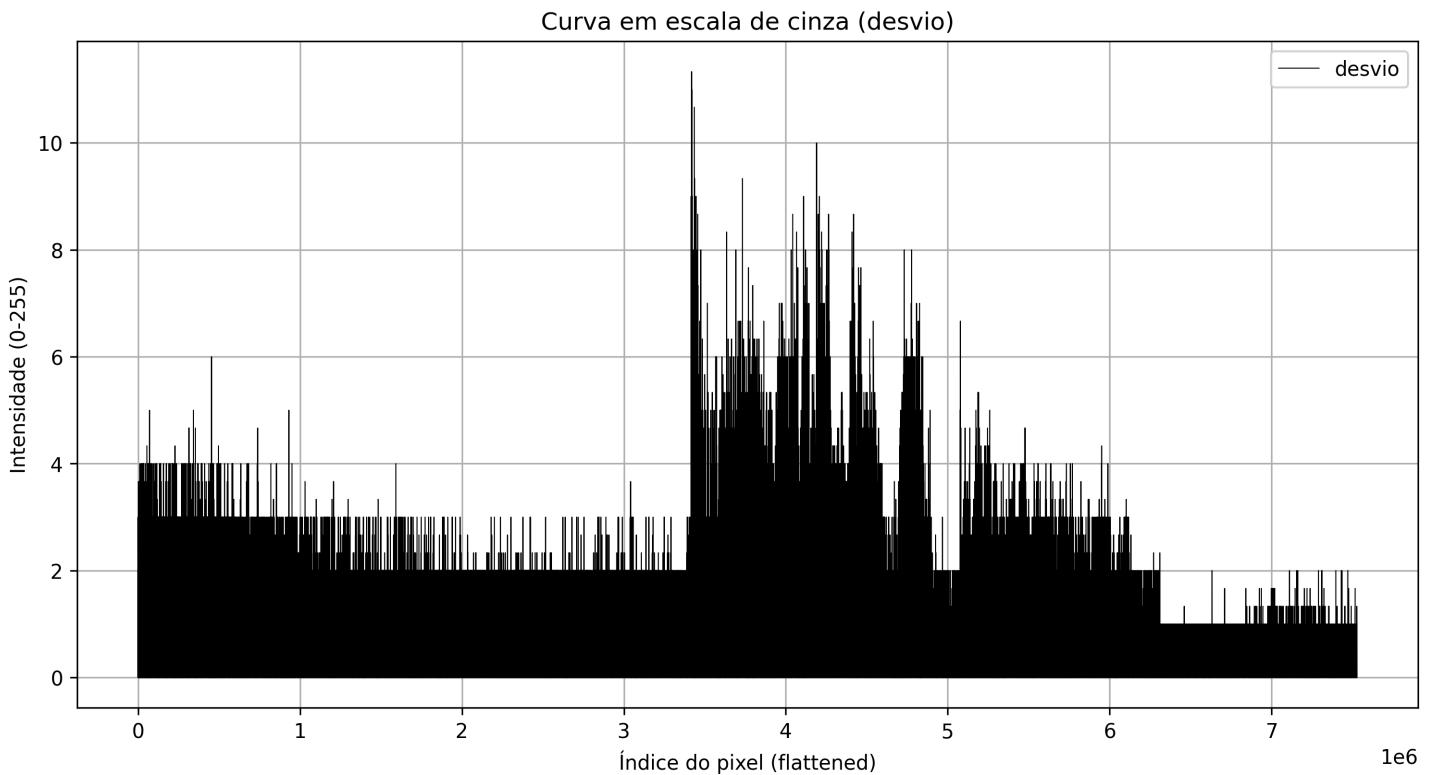


▼ Graficos:

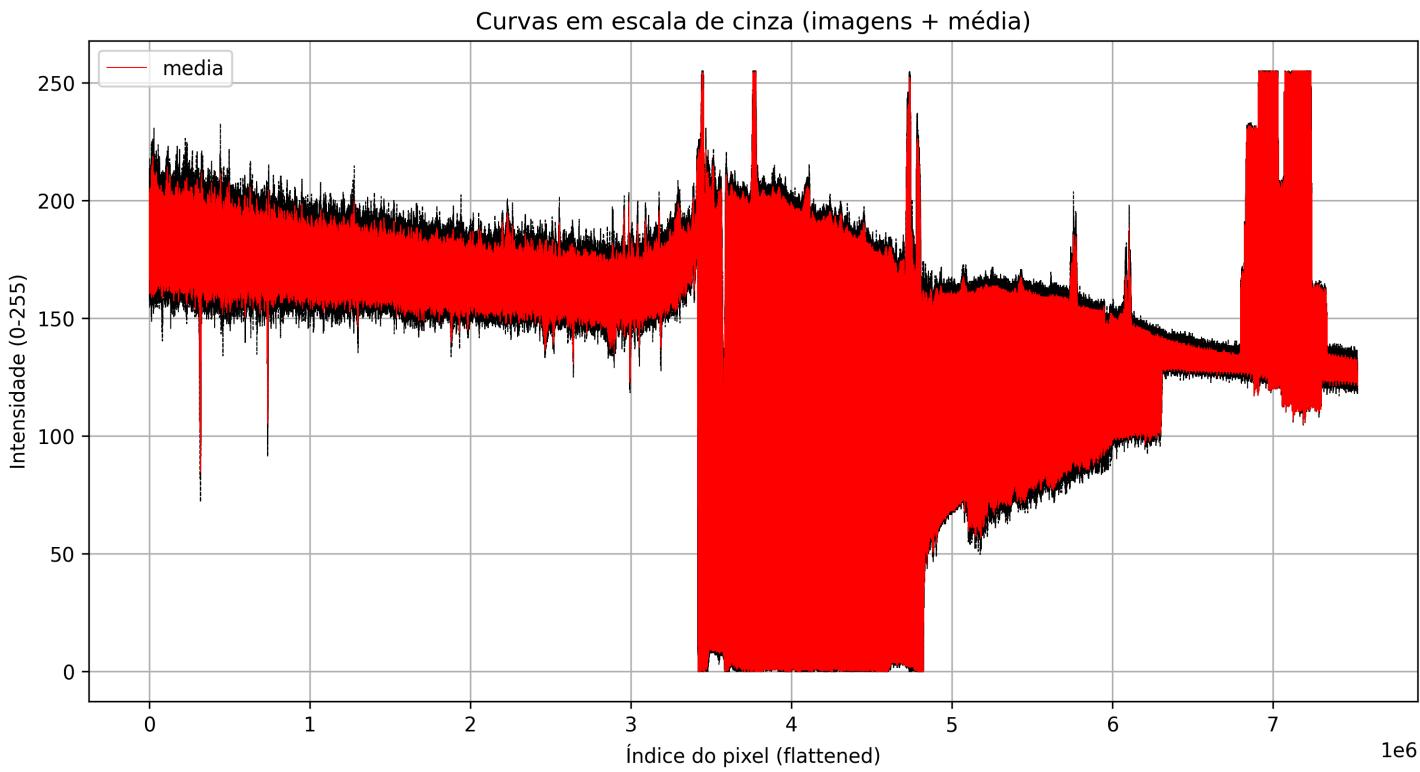
▼ Desvio:



▼ Variação:



▼ Total:



Resultados:

Media da imagem deviation.png: 1.3016877440935732

Media da imagem variation_average.png: 0.44390546351562765

% do devio médio: 0.5104657819974797

% da variação média: 0.17408057392769713

Comentários finais:

Média das imagens:

Ao calcular e vizualizar a média das imagens capturadas, obtivemos uma cena praticamente idêntica à original, assim é visível que o ruído não é tão alto ou tem um desvio alto.

Imagen da média do desvio:

A imagem resultante do desvio-padrão apresenta valores muito pequenos, o que a torna quase totalmente preta. Esse efeito mostra que a variação pixel a pixel entre as imagens é baixa, ou seja, o sensor da câmera apresenta ruído pequeno e estável.

Ruído médio:

Foi estimado que o desvio médio corresponde a aproximadamente a 1.3 ou 0.5% da intensidade máxima. E que a variação média é 0.4 ou 0.2% da intensidade máxima.

Gráficos da intensidade:

- A curva média acompanha bem o perfil das imagens originais, demonstrando que a média preserva os detalhes da cena.

- Já as curvas de desvio mostram pequenas flutuações ao longo da linha, indicando que o ruído se distribui de forma aleatória e sem padrão aparente (como esperado para ruído branco).

Covariância (AUTO_COVARIANCE):

Os resultados da covariância (imagens pretas) reforçam que não há correlação significativa entre os vizinhos, confirmando que o ruído é predominantemente aleatório e não estruturado (não apresenta um padrão espacial repetitivo). Isso é um bom indicador de que não há interferências externas sistemáticas na aquisição. Isso também pode mostrar a alta resolução da imagem onde não existem muitos pixels que "veem parcialmente um elemento entre ambos".

Conclusão:

Portanto, os experimentos comprovam que o ruído do sistema de aquisição é baixo, aleatório e corresponde a um valor médio inferior a 5% da intensidade máxima. A metodologia aplicada (média, desvio e covariância) mostrou-se eficaz para caracterizar o comportamento do sensor e confirmar a estabilidade da câmera utilizada.

▼ b)

Escreva ou utilize rotinas para inserção de ruído gaussiano e do tipo sal-e-pimenta e, após, implemente um filtro de ruídos Gaussiano e Mediano (não é filtro média) e aplique sobre as imagens, mostrando o resultado da aplicação dos dois filtros sobre as duas imagens. O código ou a função utilizada deve permitir a especificação da largura do filtro, para percepção dos resultados. Mostre as imagens e discuta os resultados.

```

import cv2
import numpy as np
from PIL import Image
from skimage.transform import resize

# -----
# Função para adicionar ruído Gaussiano
# -----
def add_gaussian_noise(image, mean=0, var=20):
    row, col, ch = image.shape
    sigma = var ** 0.5
    gauss = np.random.normal(mean, sigma, (row, col, ch))
    noisy = image + gauss
    noisy = np.clip(noisy, 0, 255).astype(np.uint8)
    return noisy

# -----
# Função para adicionar ruído Sal-e-Pimenta
# -----
def add_salt_pepper_noise(image, prob=0.02):
    noisy = np.copy(image)
    rnd = np.random.rand(image.shape[0], image.shape[1])

```

```

noisy[rnd < prob / 2] = 0           # pixels pretos (pimenta)
noisy[rnd > 1 - prob / 2] = 255    # pixels brancos (sal)
return noisy

def median_filter(I, n=3):
    """
    Aplica o filtro mediano em uma imagem em escala de cinza
    :param I: imagem de entrada (numpy array 2D)
    :param n: tamanho da janela (ímpar)
    :return: imagem filtrada
    """

    # Verificar se n é ímpar
    if n % 2 == 0:
        raise ValueError("O tamanho da máscara deve ser ímpar!")

    # Dimensões da imagem
    rows, cols = I.shape
    # Saída inicializada
    Im = np.zeros_like(I)

    # Quanto andar para cada lado
    pad = n // 2

    # Preenche as bordas da imagem (padding)
    padded = np.pad(I, pad, mode="edge")

    for i in range(rows):
        for j in range(cols):
            # Extrair vizinhança n x n
            window = padded[i : i + n, j : j + n]
            # Calcular a mediana dos elementos da janela
            m = np.median(window)
            # Atribuir ao pixel de saída
            Im[i, j] = m

    return Im

def gaussian_kernel_1d(size: int, sigma: float):
    """
    Cria um kernel gaussiano 1D
    """

    ax = np.linspace(-(size // 2), size // 2, size)
    kernel = np.exp(-(ax**2) / (2 * sigma**2))
    kernel /= np.sum(kernel)  # normaliza
    return kernel

# SEPAR_FILTER
def separable_gaussian_filter(I, size=5, sigma=1.0):
    """
    Aplica convolução gaussiana usando separabilidade (2x convolução 1D).
    :param I: imagem em escala de cinza (numpy array 2D)
    :param size: tamanho da máscara gaussiana (ímpar)
    :param sigma: desvio padrão da gaussiana
    :return: imagem filtrada
    """

    g = gaussian_kernel_1d(size, sigma)
    pad = size // 2

```

```

# Padding (replicação das bordas)
I_padded = np.pad(I, ((0, 0), (pad, pad)), mode="edge")
rows, cols = I.shape

# 1ª etapa: convolução horizontal
I_r = np.zeros_like(I, dtype=np.float32)
for i in range(rows):
    for j in range(cols):
        region = I_padded[i, j : j + size]
        I_r[i, j] = np.sum(region * g)

# Padding vertical
I_r_padded = np.pad(I_r, ((pad, pad), (0, 0)), mode="edge")

# 2ª etapa: convolução vertical
I_out = np.zeros_like(I_r, dtype=np.float32)
for i in range(rows):
    for j in range(cols):
        region = I_r_padded[i : i + size, j]
        I_out[i, j] = np.sum(region * g)

return np.clip(I_out, 0, 255).astype(np.uint8)

# -----
# Função para aplicar filtros
# -----
def apply_filters_cv2(image, ksize=5):
    # filtro Gaussiano
    gauss_filtered = cv2.GaussianBlur(image, (ksize, ksize), 0)
    # filtro Mediano
    median_filtered = cv2.medianBlur(image, ksize)
    # ambos
    both_filtered = cv2.medianBlur(gauss_filtered, ksize)
    return gauss_filtered, median_filtered, both_filtered

def apply_filters(image, ksize=5):
    gauss_filtered = np.zeros_like(image)
    median_filtered = np.zeros_like(image)
    both_filtered = np.zeros_like(image)

    for c in range(3): # para cada canal
        gauss_filtered[:, :, c] = separable_gaussian_filter(
            image[:, :, c], ksize, sigma=1.0
        )
        median_filtered[:, :, c] = median_filter(image[:, :, c], ksize)
        both_filtered[:, :, c] = median_filter(
            gauss_filtered[:, :, c], ksize
        )

    return gauss_filtered, median_filtered, both_filtered

def image_to_array(file, width=None, height=None):
    if file.lower().endswith(".npz"):
        # Carregar npz
        data = np.load(file)
        # se tiver a chave "resultado", usa ela. Se não, pega a primeira existente
        arr = data["resultado"] if "resultado" in data else data[list(data.keys())[0]]
        # garantir dtype float64

```

```

np_img = np.array(arr, dtype=np.float64)

    if width is not None and height is not None:
        # Se precisar forçar resize (padrão igual ao das imagens PIL)
        if (np_img.shape[1], np_img.shape[0]) != (width, height):
            np_img = resize(
                np_img, (height, width, np_img.shape[2]), preserve_range=True
            ).astype(np.float64)

    else:
        # Imagem regular
        img = Image.open(file).convert("RGB")
        if width is not None and height is not None:
            img = img.resize((width, height))
        np_img = np.array(img, dtype=np.float64)

return np_img


def save_image(output, name, img_type):
    name = f"{name}.{img_type}"
    if img_type.lower() == "npz":
        np.savez_compressed(name, resultado=output)
    else:
        output = np.clip(output, 0, 255).astype(np.uint8)
        # Salva a imagem média
        img_final = Image.fromarray(output, mode="RGB")
        img_final.save(name)
    print(f"Imagen salva como {name}")

# Carregar imagem (exemplo com imagem colorida)
image = cv2.imread("average.png")
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # converter pra RGB

# add noise
noisy_gaussian = add_gaussian_noise(image)
noisy_sp = add_salt_pepper_noise(image)

# save images with noise
save_image(noisy_gaussian, "noisy_gaussian", "png")
save_image(noisy_sp, "noisy_sp", "png")

# Aplicar filtros
gauss_filt_g, median_filt_g, both_filt_g, = apply_filters(noisy_gaussian)
gauss_filt_sp, median_filt_sp, both_filt_sp = apply_filters(noisy_sp)
gauss_filt_cv_g, median_filt_cv_g, both_filt_cv_g, = apply_filters_cv2(noisy_gaus
gauss_filt_cv_sp, median_filt_cv_sp, both_filt_cv_sp = apply_filters_cv2(noisy_sp

# save images with filter
save_image(gauss_filt_g, "gaussian_ruido_gaussiano", "png")
save_image(median_filt_g, "mediana_ruido_gaussiano", "png")
save_image(both_filt_g, "gaussian+mediana_ruido_gaussiano", "png")

save_image(gauss_filt_sp, "gaussian_ruido_sal_pimenta", "png")
save_image(median_filt_sp, "mediana_ruido_sal_pimenta", "png")
save_image(both_filt_sp, "gaussian+mediana_ruido_sal_pimenta", "png")

# versões OpenCV
save_image(gauss_filt_cv_g, "cv2_gaussian_ruido_gaussiano", "png")

```

```
save_image(median_filt_cv_g, "cv2_medianana_ruido_gaussiano", "png")
save_image(both_filt_cv_g, "cv2_gaussian+medianana_ruido_gaussiano", "png")

save_image(gauss_filt_cv_sp, "cv2_gaussian_ruido_sal_pimenta", "png")
save_image(median_filt_cv_sp, "cv2_medianana_ruido_sal_pimenta", "png")
save_image(both_filt_cv_sp, "cv2_gaussian+medianana_ruido_sal_pimenta", "png")

# Aplicar filtros
gauss_filt_g, median_filt_g, both_filt_g, = apply_filters(noisy_gaussian, 3)
gauss_filt_sp, median_filt_sp, both_filt_sp = apply_filters(noisy_sp, 3)
gauss_filt_cv_g, median_filt_cv_g, both_filt_cv_g, = apply_filters_cv2(noisy_gaus
gauss_filt_cv_sp, median_filt_cv_sp, both_filt_cv_sp = apply_filters_cv2(noisy_sp

# save images with filter
save_image(gauss_filt_g, "gaussian_ruido_gaussiano_3", "png")
save_image(median_filt_g, "medianana_ruido_gaussiano_3", "png")
save_image(both_filt_g, "gaussian+medianana_ruido_gaussiano_3", "png")

save_image(gauss_filt_sp, "gaussian_ruido_sal_pimenta_3", "png")
save_image(median_filt_sp, "medianana_ruido_sal_pimenta_3", "png")
save_image(both_filt_sp, "gaussian+medianana_ruido_sal_pimenta_3", "png")

# versões OpenCV
save_image(gauss_filt_cv_g, "cv2_gaussian_ruido_gaussiano_3", "png")
save_image(median_filt_cv_g, "cv2_medianana_ruido_gaussiano_3", "png")
save_image(both_filt_cv_g, "cv2_gaussian+medianana_ruido_gaussiano_3", "png")

save_image(gauss_filt_cv_sp, "cv2_gaussian_ruido_sal_pimenta_3", "png")
save_image(median_filt_cv_sp, "cv2_medianana_ruido_sal_pimenta_3", "png")
save_image(both_filt_cv_sp, "cv2_gaussian+medianana_ruido_sal_pimenta_3", "png")
```

Resultados: