Trabalho M2

Processamento de Imagens

Universidade do vale do Itajaí (UNIVALI)

Escola do Mar, Ciência e Tecnologia

Filtragem Espacial x Filtragem no Domínio da Frequência

Resumo: Este trabalho tem por objetivo consolidar o aprendizado sobre reconhecimento de imagem e processamento digital de imagens.

PROFESSOR: Felipe Viel

Jonathas De Oliveira Meine

Matheus Armando Timm Barbiéri

Itajaí, 03 de outubro de 2024

Sumário

1. INTRODUÇÃO

Este trabalho visa comparar diferentes técnicas de filtragem de imagens, utilizando as métricas de qualidade PSNR, RMSE e MSE. A análise envolve filtros espaciais e no domínio da frequência, comparando o filtro espacial de esmaecimento gaussiano com os filtros Passa Baixa Ideal e Gaussiano, além de comparar os filtros Sobel e Passa Alta Ideal e Gaussiano com o Canny. Serão avaliadas as semelhanças e diferenças nas imagens resultantes, além da implementação e do tempo de processamento, com base nas métricas selecionadas.

2. IMPLEMENTAÇÃO

Para este trabalho, utilizamos a linguagem de programação *Python* com as bibliotecas *OpenCV*, *NumPy* e *time*, para implementar as métricas de qualidade, os filtros espaciais Esmaecimento Gaussiano e Sobel; os filtros de domínio da frequência Passa Baixa e Passa Alta: ideais e gaussianos; e medir o tempo de processamento dos filtros.

2.1. Métricas de Qualidade

As métricas de qualidade *MSE*, *RMSE* e *PSNR* são utilizadas para identificar o quão diferente duas imagens são, suas implementações encontram-se a seguir:

2.1.1. *MSE (Mean Squared Error)*

O *MSE* mede a diferença média quadrática entre os pixels de duas imagens. Quanto menor o valor, mais semelhantes são as imagens. É sensível a grandes erros.

```
def MSE(img1, img2):
    return np.mean((img1- img2) ** 2)
```

2.1.2. *RMSE (Root Mean Squared Error)*

O *RMSE* é a raiz quadrada da *MSE*, proporcionando uma métrica de erro mais interpretável no mesmo nível das intensidades da imagem original. Valores menores indicam maior semelhança.

```
def RMSE(img1, img2):
    return np.sqrt(MSE(img1, img2))
```

2.1.3. *PSNR (Peak Signal-to-Noise Ratio)*

O *PSNR* mede a qualidade de uma imagem em relação a outra de referência, expressando o erro em decibéis. Valores mais altos indicam melhor qualidade e menos ruído.

```
def PSNR(img1, img2):
    mse = MSE(img1, img2)
    if (mse == 0):
        return 100

    max_pixel = 255.0
    psnr = 20 * np.log10(max_pixel / np.sqrt(mse))

    return psnr
```

2.2.Filtros Espaciais

2.2.1. Esmaecimento Gaussiano

O esmaecimento gaussiano suaviza uma imagem aplicando uma função gaussiana nos pixeis vizinhos. É amplamente usado para reduzir ruído e detalhes finos, preservando transições suaves nas bordas. Nesta implementação, foi utilizado um kernel 3x3.

```
def apply_gaussian_blur(img):
    kernel = np.array([
        [1, 2, 1],
        [2, 4, 2],
        [1, 2, 1]
        ]) / 16

    img_height, img_width = img.shape
    padded_img = np.pad(img, pad_width=1, mode='edge')
    output_img = np.zeros((img_height, img_width), dtype=float)

    for i_img in range(1, img_height + 1):
        for j_img in range(1, img_width + 1):
            output_img[i_img-1, j_img-1] = np.sum(
                padded_img[
                    i_img-1:i_img+2,
                    j_img-1:j_img+2
                ] * kernel
            )
    return np.array(output_img, dtype=np.uint8)
```

2.2.2. Sobel

O filtro Sobel detecta bordas em uma imagem calculando o gradiente de intensidade em direções horizontais e verticais. Ele realça áreas de transição brusca de intensidade, sendo útil para a detecção de bordas.

```
def apply_sobel(img):
    kernel_x = np.array([
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1]
        ])
    kernel_y = np.array([
        [-1, -2, -1],
        [0, 0, 0],
        [1, 2, 1]
        ])
    img_height, img_width = img.shape

    padded_img = np.pad(img, pad_width=1, mode='edge')
    output_img = np.zeros((img_height, img_width), dtype=float)

    for i_img in range(1, img_height + 1):
        for j_img in range(1, img_width + 1):
            output_img[i_img-1, j_img-1] = int(
                np.abs(np.sum(
                    padded_img[
                        i_img-1:i_img+2,
                        j_img-1:j_img+2
                    ] * kernel_x
                )) +
                np.abs(np.sum(
                    padded_img[
                        i_img-1:i_img+2,
                        j_img-1:j_img+2
                    ] * kernel_y
                ))
            )

    return np.array(output_img, dtype=np.float32)
```

2.3.Filtros de Domínio da Frequência

Na implementação dos filtros de Passa Baixa e Passa Alta, é necessário passar como parâmetro da função: o tamanho da imagem, coordenada central, raio de aplicação, e um valor booleano para determinar se será utilizado o filtro Ideal ou Gaussiano.

2.3.1. Filtro Passa Baixa

O filtro passa baixa remove as frequências altas, preservando detalhes de baixa frequência na imagem, como regiões suaves e transições gradativas. O Ideal tem uma transição abrupta entre frequências preservadas e removidas, resultando em

possíveis artefatos, já, o Gaussiano, tem uma transição suave entre as frequências, proporcionando suavização progressiva sem gerar artefatos visíveis.

```
def apply_low_pass(shape, center, radius, isIdeal):
    rows, cols = shape [:2]
    r, c = np.mgrid[0:rows:1, 0:cols:1]
    c -= center[0]
    r -= center[1]
    d = np.power(c, 2.0) + np.power(r, 2.0)
    lpFilter_matrix = np.zeros(shape, np.float32)
    if isIdeal:
        lpFilter = np.copy(d)
        lpFilter[lpFilter < pow(radius, 2.0)] = 1
        lpFilter[lpFilter >= pow(radius, 2.0)] = 0
    else:
        lpFilter = np.exp(-d/(2*pow(radius, 2.0)))
    lpFilter_matrix[:, :, 0] = lpFilter
    lpFilter_matrix[:, :, 1] = lpFilter
    return lpFilter_matrix
```

2.3.2. Filtro Passa Alta

O filtro passa alta ideal remove frequências baixas, realçando bordas e detalhes finos. O Ideal faz uma separação brusca entre as frequências, destacando detalhes, mas introduzindo artefatos visíveis, como o efeito de *ringing*, já, o Gaussiano, é menos agressivo que o filtro ideal, evitando artefatos bruscos e proporcionando um realce mais natural.

```
def apply_high_pass(shape, center, radius, isIdeal):
    rows, cols = shape [:2]
    r, c = np.mgrid[0:rows:1, 0:cols:1]
    c -= center[0]
    r -= center[1]
    d = np.power(c, 2.0) + np.power(r, 2.0)
    lpFilter_matrix = np.zeros(shape, np.float32)
    if isIdeal:
        lpFilter = np.copy(d)
        lpFilter[lpFilter < pow(radius, 2.0)] = 0
        lpFilter[lpFilter >= pow(radius, 2.0)] = 1
    else:
        lpFilter = 1.0 - np.exp(-d/(2*pow(radius, 2.0)))
    lpFilter_matrix[:, :, 0] = lpFilter
    lpFilter_matrix[:, :, 1] = lpFilter
    return lpFilter_matrix
```

2.3.3. Aplicação dos Filtros Passa Baixa e Passa Alta

Para utilizar os filtros de Passa Baixa e Passa Alta, é necessário realizar a Transformada de Fourier na imagem, para assim ser possível trabalhar no domínio da frequência. Após isso, é possível aplicar os filtros, e então é preciso realizar a Transformada Inversa de Fourier para trazer a imagem de volta ao domínio espacial, para deste modo normalizá-la. Isto é realizado pela seguinte função:

```
def apply_frequency_filter(img, radius, isHighPass, isIdeal):
  image_f32 = np.float32(img)
  dft = cv2.dft(image_f32, flags = cv2.DFT_COMPLEX_OUTPUT)
  dft_shift = np.fft.fftshift(dft)

  center = (img.shape[1]//2, img.shape[0]//2)

  if isHighPass:
    mask = apply_high_pass(dft_shift.shape, center, radius, isIdeal)
  else:
    mask = apply_low_pass(dft_shift.shape, center, radius, isIdeal)

  filtered_freq = dft_shift*mask
  f_ishift = np.fft.ifftshift(filtered_freq)
  img_back = cv2.idft(f_ishift)
  img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])

  img_back = np.array(img_back, dtype=np.float32)

  filtered_img = np.abs(img_back)
  filtered_img -= filtered_img.min()
  filtered_img = filtered_img*255 / filtered_img.max()
  return filtered_img.astype(np.uint8)
```

2.4. Calculadora de tempo médio de processamento

Foi criada uma função de auxílio para medir o tempo médio de execução das funções de filtro. Esta função recebe um número de iterações, para se obter o tempo médio da execução, e uma função lambda, que no qual, deve ser uma função que retorna uma imagem com filtro.

Ao final, esta função retorna a imagem filtrada e o tempo médio em milissegundos.

```
def apply_and_measure_filter_mean_time(iterations, func):
  ms_spent = 0
  for i in range(iterations):
    start = time.time()
    img_with_filter = func()
    ms_spent += time.time() - start
  return (img_with_filter, ms_spent * 1000 / iterations)
```

3. RESULTADOS E ANÁLISES

Agora, abordaremos comparações e análises das aplicações dos filtros, evidenciando as diferenças entre o Esmaecimento Gaussiano e o Passa Baixa, e entre o Sobel e o Passa Alta.

Para análise, foi utilizada a seguinte imagem de tamanho 400 x 236 pixels durante o desenvolvimento desta atividade, que, no qual, possui um leve ruído (Figura 1):

Figura 1 – Imagem escolhida

Observação: Para calcular a média de tempo, foram utilizadas 5 iterações em todos os filtros.

3.1.Esmaecimento Gaussiano vs. Passa Baixa

3.1.1. Aplicação dos filtros

Aplicação do filtro de Esmaecimento Gaussiano na imagem escolhida, com a função implementada e especificada em **2.2.1**:

Figura 2 - Imagem com esmaecimento gaussiano

Com a imagem retornada (Figura 2), já é possível perceber que a quantidade de ruído na imagem já diminuiu, devido ao efeito de *blur* do filtro.

Aplicação dos filtros Passa Baixa Ideal e Gaussiano utilizando 70 pixels de raio para ambos, com a função desenvolvida em **2.3.1**:

Figura 3 - Imagem com Passa Baixa Ideal

Após aplicação do Passa Baixa Ideal (Figura 3), podemos identificar menos ruído na imagem, entretanto, é visível a presença de artefatos conhecidos como *ringing*, que é este efeito de onde ao redor dos objetos na imagem.

Figura 4 - Imagem com Passa Baixa Gaussiano

Já, com o Passa Baixa Gaussiano (Figura 4), conseguimos identificar bem menos ruído na imagem, e, diferente do Ideal, não apresenta artefatos.

3.1.2. Análise visual e métricas

Visualmente, o Passa Baixa Ideal não apresentou resultados muito bons, principalmente devido à adição de artefatos. Porém, podemos considerar que o Esmaecimento Gaussiano teve um bom resultado, removendo consideravelmente o ruído leve da imagem, e podemos dizer o mesmo do Passa Baixa Gaussiano.

Com os valores das métricas, é possível dizer mais precisamente que o Passa Baixa Gaussiano teve mais semelhança com o Esmaecimento Gaussiano, apresentando um *PSNR* de 38,31 dB, do que o Passa Baixa Ideal com 31,81 dB (Tabela 1).

Entretanto, se formos comparar o tempo médio de execução dos filtros, podemos observar que o Esmaecimento Gaussiano, com 790,74 milissegundos, leva aproximadamente 46 vezes o tempo de execução dos filtros do domínio da frequência, que apresentaram um tempo de aproximadamente 17 milissegundos, demonstrando uma vantagem considerável na escolha do Passa Baixa em relação ao Esmaecimento Gaussiano (Tabela 2).

Comparação com Esmaecimento Gaussiano			
Filtro	MSE	RMSE	PSNR (dB)
Passa Baixa Ideal	42,82	6,54	31,81
Passa Baixa Gaussiano	9,58	3,10	38,31

Tabela 1 - Métricas de Qualidade aplicadas entre Esmaecimento Gaussiano e Passa Baixa

Filtro	Tempo (ms)
Esmaecimento Gaussiano	790,74
Passa Baixa Ideal	16,93
Passa Baixa Gaussiano	17.50

Tabela 2 - Tempos médios de execução dos filtros Esmaecimento Gaussiano e Passa Baixa

3.2. Canny vs. Sobel e Passa Alta

3.2.1. Aplicação dos filtros

Aplicação do filtro Sobel na imagem escolhida, com a função implementada e especificada em **2.2.2**:

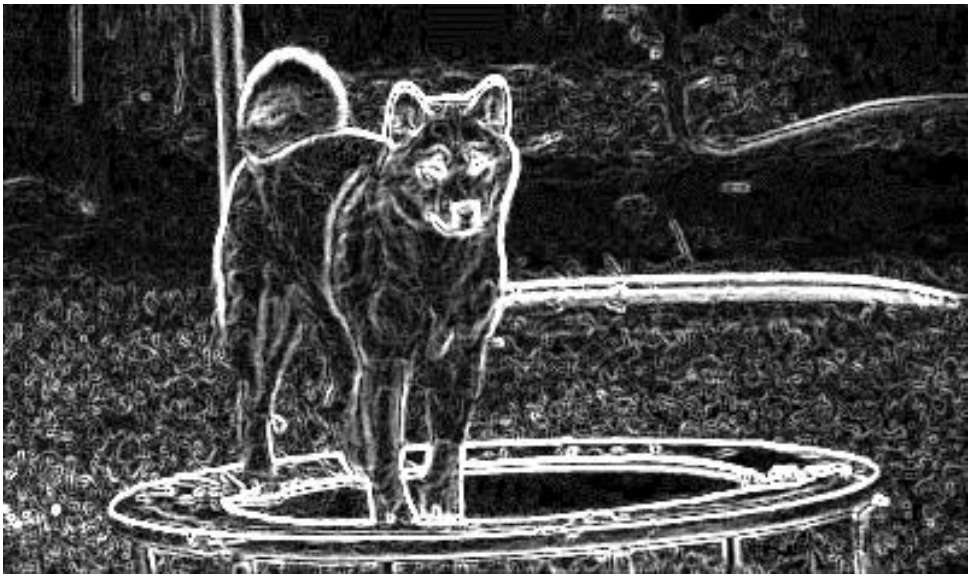

Figura 5 - Imagem com filtro Sobel

Após aplicação do filtro (Figura 5), podemos observar sem dificuldades as bordas dos objetos na imagem.

Aplicação dos filtros Passa Alta Ideal e Gaussiano utilizando 15 pixels de raio para ambos, com a função desenvolvida em **2.3.2**:

Figura 6 - Imagem com Passa Alta Ideal

Após aplicação do Passa Alta Ideal (Figura 6), também podemos identificar as bordas dos objetos, entretanto, é mais difícil identificar objetos no fundo da imagem, e é notável a presença de artefatos, assim como no Passa Baixa Ideal (Figura 3).

Figura 7 - Imagem com Passa Alta Gaussiano

Já, com o Passa Alta Gaussiano (Figura 7), temos um resultado bem similar ao Passa Alta Ideal, a diferença mais marcante é a ausência dos artefatos, assim, deixando a imagem com mais foco no propósito do filtro: realçar bordas.

Aplicação do Canny, utilizando a função da biblioteca *OpenCV* com *threshold* inferior de 400 e superior de 800:

Figura 8 - Imagem com Canny

Diferentemente dos outros, com o Canny, obtemos somente o que é borda, sem tons de transição, o que deixa a imagem mais simples, mas também mais difícil de identificar os objetos presentes (Figura 8).

3.2.2. Análise visual e métricas

A comparação dos tempos de processamento destes filtros não é relevante, já que o Sobel e Passa Alta foram realizados em Python e o Canny da *OpenCV* é altamente otimizado e implementado com C/C++.

Visualmente, podemos dizer que o Sobel tem um resultado melhor que o Canny, porque com o Sobel é mais fácil identificar profundidade e todos os objetos da imagem, enquanto com o Canny, muita informação é omitida, e as bordas identificadas não são suficientes para entender a imagem. As métricas apontam uma diferença muito grande, apresentando um *PSRN* de 5,95 dB (Tabela 3).

Agora, para o Passa Alta e o Canny, é visível uma maior semelhança nas imagens, já que o Passa Alta destaca as bordas de modo mais leve que o Sobel, entretanto, são nítidos os objetos da imagem com o Passa Alta, mas, com o Canny, não apresenta todo esse detalhe para com os objetos presentes. Diferente das métricas da comparação com o Sobel, as métricas do Passa Alta com o Canny informam que existe uma semelhança significativa entre as imagens, com um *PSNR* de aproximadamente 29 dB (Tabela 3).

Comparação com Canny			
Filtro	MSE	RMSE	PSNR (dB)
Sobel	16.540,97	128,61	5,95
Passa Alta Ideal	86,54	9,30	28,76
Passa Alta Gaussiano	75,28	8,68	29,36

Tabela 3 - Métricas de Qualidade aplicadas entre Canny, Sobel e Passa Alta

4. REFERÊNCIAS

MEINE, Jonathas de Oliveira; BARBIÉRI, Matheus Armando Timm; MUDAR TÍTULO. 2023. Disponível em: https://github.com/jonhymeine/spatial-and-frequence-filters. Acesso em: 09 out. 2024.

TENOR. Dog Doge GIF. 2016. Disponível em: https://tenor.com/pt-BR/view/dog-doge-trampoline-play-jumping-gif-6141197. Acesso em: 09 out. 2024.