Pontifícia Universidade Católica de Minas Gerais
Instituto de Ciências Exatas e Informática
Departamento de Engenharia de Computação

Relatório: Trabalho Prático 1

Exercício 1: Detecção de Bordas

Professora: Ana Carolina Conceição de Jesus

Marcus Leandro Gomes Campos Oliveira

Belo Horizonte
Campus Coração Eucarístico

29 de abril de 2025

Conteúdo

1 Resumo

Este trabalho apresenta uma das possíveis soluções para o *Exercício 1 - Detecção de Bordas*, utilizando técnicas estudadas na disciplina de Processamento e Análise de Imagens. O objetivo foi resolver um problema comum dessa área de pesquisa por meio da aplicação prática de conceitos teóricos. O trabalho inclui a explicação teórica de cada técnica utilizada, sua respectiva implementação em Python, as decisões tomadas durante o desenvolvimento, a análise dos resultados obtidos e, por fim, uma comparação com um artigo publicado na mesma área. Os resultados mostram que a detecção de bordas é uma tarefa desafiadora, e que diferentes técnicas podem apresentar desempenhos variados dependendo do cenário. Este estudo contribui para o entendimento das técnicas mais simples de detecção de bordas, apontando possíveis melhorias e ajustes necessários conforme o contexto de aplicação.
Palavras-chave: Deteção de Bordas, Processamento e Análise de Imagens, Python.

2 Introdução

A detecção de bordas é uma etapa fundamental no processamento e análise de imagens, sendo amplamente utilizada em aplicações como reconhecimento de objetos, segmentação de imagens, visão computacional e sistemas autônomos. Essa técnica visa identificar transições abruptas de intensidade nos pixels de uma imagem, que geralmente correspondem às fronteiras dos objetos presentes nela.

Diversos algoritmos foram desenvolvidos para realizar a detecção de bordas, cada um com suas características e aplicações específicas. Entre os métodos baseados em gradiente, destacam-se os operadores de Sobel, Prewitt e Roberts, que calculam a derivada primeira da intensidade da imagem para identificar regiões com variações significativas . Já os métodos baseados na segunda derivada, como o Laplaciano do Gaussiano (LoG) e o Difference of Gaussians (DoG), detectam bordas localizando cruzamentos por zero na segunda derivada da imagem.

Este trabalho tem como objetivo explorar e implementar um algoritmo Laplaciano de detecção de bordas, aplicando-as a imagens digitais e analisando seus ajustes em diferentes cenários. Além disso, será realizada uma comparação entre esse trabalho realizado e outro artigo da área, visando aprofundar a compreensão sobre as diferentes formas de se detectar as bordas de uma imagem.

3 Detecção de Bordas

3.1 Abordagem adotada

O objetivo deste exercício consiste em detectar e demarcar as bordas de uma imagem qualquer. A forma mais simples de realizar essa tarefa é por meio da análise da diferença de intensidade entre pixels adjacentes. Para uma avaliação mais próxima de cenários reais, foram utilizadas imagens coloridas com diferentes tamanhos e resoluções.

Dentro desse conjunto de imagens, foram definidos alguns *presets* específicos para lidar com variações de tonalidade predominante, o que permitiu adaptar a abordagem para imagens mais claras ou mais escuras. Essa estratégia contribui significativamente para o aumento da eficiência do processo de detecção de bordas.

A metodologia adotada pode ser dividida em etapas sucessivas, que vão desde o carregamento das imagens até a geração da imagem final com as bordas destacadas. A seguir, cada uma dessas etapas será detalhada. Ressalta-se que, para a implementação, foram utilizadas exclusivamente as bibliotecas *NumPy* e *Pillow*, responsáveis respectivamente pelos cálculos matemáticos e pela manipulação dos arquivos de imagens.

3.2 Pré processamento

Antes da aplicação de filtros e ajustes, é necessário preparar a imagem adequadamente para receber a detecção de bordas. Essa etapa é muito importante porque a detecção de bordas tem maior efetividade em imagens na escala de cinza. Isso ocorre por conta do funcionamento da maioria dos algoritmos; esses algoritmos se baseiam em calcular a diferença entre dois (ou mais) pixels e, através de um valor de limiar, determinar se a diferença corresponde a uma borda.

A seguir temos o código utilizado nessa etapa e o seu funcionamento detalhado.

3.2.1 Código

```
# --- 1. Carregamento e pre-processamento da imagem ---
img = Image.open('./image1.jpg').convert('L')
matriz_original = np.array(img).astype(np.float32)
altura, largura = matriz_original.shape

# Calcula a media
media = np.mean(matriz_original)

# Indica os ajustes a serem feitos
# Media dos tons de cinza:
# Imagem de tons escuros
if media < 50:
    A = 10.0
    limiar_diferenca = 90
# Imagem de tons medios
elif media < 170:
    A = 2.0
    limiar_diferenca = 80
# Imagem de tons claros
else:
    A = 1.4
    limiar_diferenca = 9
```

3.2.2 Funcionamento

O processo de detecção de bordas inicia-se com o carregamento e o pré-processamento da imagem. O trecho de código apresentado realiza essas etapas utilizando as bibliotecas *PIL (Pillow)* e *NumPy*.

Inicialmente, a função `Image.open` carrega a imagem `image1.jpg`. Em seguida, o método `.convert('L')` converte a imagem para escala de cinza. Em uma imagem colorida no espaço RGB, cada pixel é representado por três componentes: vermelho (R), verde (G) e azul (B). A conversão para escala de cinza envolve a combinação desses três valores em um único valor de intensidade luminosa. A função utiliza a Transformação Luma ITU-R 601-2, conforme a equação:

$$L = 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B$$

Essa fórmula considera a sensibilidade do olho humano às diferentes cores, atribuindo maior peso ao canal verde, seguido pelo vermelho e, por último, o azul.

Após a conversão, a imagem em escala de cinza é transformada em uma matriz `NumPy` de ponto flutuante de 32 bits, onde cada elemento representa a intensidade de um pixel, variando

de 0 (preto) a 255 (branco). As variáveis `altura` e `largura` armazenam as dimensões da imagem, obtidas a partir do atributo `shape` da matriz.

O próximo passo envolve o cálculo da média das intensidades dos pixels da imagem em escala de cinza. Esse valor médio fornece uma estimativa do brilho geral da imagem. Com base nesse valor, o código ajusta os parâmetros `A` (fator de amplificação, que será apresentado mais tarde) e `limiar_diferenca`, que influenciam diretamente na detecção de bordas.

A lógica condicional classifica a imagem como escura, média ou clara, com base no valor da média de intensidade. Para imagens mais escuras (média < 50), são atribuídos valores mais altos aos parâmetros, visando realçar as bordas em áreas de baixa luminosidade. Para imagens mais claras (média ≥ 170), valores menores são utilizados, evitando a detecção excessiva de bordas em regiões já bem iluminadas. Essa adaptação dinâmica dos parâmetros melhora a eficácia da detecção de bordas em diferentes condições de iluminação.

3.3 Suavização Gaussiana

A suavização gaussiana é uma técnica de filtragem espacial amplamente utilizada no processamento digital de imagens, com o objetivo de reduzir ruídos e detalhes de alta frequência, resultando em uma imagem mais suave. Essa técnica é especialmente útil como etapa de pré-processamento em algoritmos de detecção de bordas, segmentação e reconhecimento de padrões.

Matematicamente, a suavização gaussiana consiste na convolução da imagem original com uma função Gaussiana bidimensional. A função Gaussiana contínua em duas dimensões é definida por:

$$w(s,t) = G(s,t) = Ke^{\left(-\frac{s^2+t^2}{2\sigma^2}\right)}$$

Onde K corresponde à altura da curva, t corresponde ao valor esperado (centro da curva) e σ^2 é a variância. Valores maiores de σ^2 resultam em uma suavização mais intensa.

Na prática, essa função contínua é discretizada para formar uma máscara (ou kernel) que será aplicada à imagem. A discretização envolve a amostragem da função Gaussiana em uma grade finita e a normalização dos valores para garantir que a soma total do kernel seja igual a 1, preservando o brilho da imagem.

O kernel utilizado nesse trabalho foi um modelo 3x3. Um kernel Gaussiano é uma aproximação discreta da função Gaussiana com um pequeno desvio padrão. O utilizado nesse trabalho foi:

$$K = \frac{1}{28}\begin{bmatrix} 1 & 4 & 1 \\ 4 & 8 & 4 \\ 1 & 4 & 1 \end{bmatrix}$$

Neste kernel, o peso central é o maior, e os pesos diminuem à medida que se afastam do centro, refletindo a distribuição Gaussiana. A aplicação desse kernel a cada pixel da imagem envolve a multiplicação dos valores dos pixels vizinhos pelos pesos correspondentes do kernel e a soma dos resultados, substituindo o valor do pixel central pela média ponderada obtida.

A suavização gaussiana é eficaz na redução de sombras, que é um fator problemático na detecção de bordas. Além disso, devido à sua propriedade de ser um filtro passa-baixas, ela preserva as estruturas de baixa frequência, como regiões homogêneas, enquanto suaviza transições abruptas.

3.3.1 Código

```
# --- 2. Aplica a suaviza   o Gaussiana 3x3 ---
kernel = np.array([[1, 4, 1],
                   [4, 8, 4],
                   [1, 4, 1]], dtype=np.float32)
```

```
kernel = kernel / kernel.sum()

suavizada = np.zeros_like(matriz_original)

for i in range(1, altura - 1):
    for j in range(1, largura - 1):
        regiao = matriz_original[i-1:i+2, j-1:j+2]
        suavizada[i, j] = np.sum(regiao * kernel)
```

3.3.2 Funcionamento

Inicialmente, define-se um kernel 3x3, representado por uma matriz *NumPy*. Este kernel é uma aproximação discreta de uma função Gaussiana bidimensional, onde os valores centrais possuem maior peso, diminuindo gradualmente em direção às bordas. A normalização do kernel, dividindo cada elemento pela soma total dos valores, garante que a intensidade média da imagem seja preservada após a aplicação do filtro. Cria-se uma matriz vazia, com as mesmas dimensões da imagem original, para armazenar os valores da imagem suavizada.

A convolução é realizada percorrendo cada pixel da imagem (exceto as bordas, para evitar acessos fora dos limites da matriz) e aplicando o kernel à vizinhança 3x3 do pixel atual. Para cada posição (i, j), extrai-se uma submatriz 3×3 da imagem original, centrada no pixel atual. Esta submatriz é multiplicada elemento a elemento pelo kernel, e a soma dos produtos é atribuída ao pixel correspondente na imagem suavizada. Este processo atenua variações abruptas de intensidade, resultando em uma imagem mais homogênea.

Essa operação intermediária suaviza as sombras das imagens e torna-as mais planas, reduzindo variações de intensidade que confundem o algoritmo de detecção de borda. As sombras foram um grande desafio durante a execução desse trabalho, sendo mitigadas pelo uso desse filtro.

3.4 Filtro Mediana

O filtro mediana é uma técnica não linear de processamento de imagens amplamente utilizada para a redução de ruídos, especialmente eficaz contra o ruído impulsivo, também conhecido como ruído "sal e pimenta". Diferentemente dos filtros lineares tradicionais, como os filtros de média, o filtro mediana preserva melhor as bordas da imagem enquanto suaviza regiões homogêneas.

O funcionamento do filtro mediana baseia-se na análise local de uma vizinhança em torno de cada pixel da imagem. Para cada pixel, considera-se uma janela de tamanho ímpar (como 3×3, 5×5, etc.) centrada no pixel em questão. Todos os valores de intensidade dos pixels contidos nessa janela são então organizados em ordem crescente, e o valor central (mediana) é selecionado para substituir o valor original do pixel.

Formalmente, seja $I(x, y)$ a intensidade do pixel na posição (x, y) da imagem. Para uma janela W centrada em (x, y), a saída do filtro é definida como:

$$I'(x, y) = \text{mediana}\{I(s, t) \,|\, (s, t) \in W\}$$

onde $I'(x, y)$ representa a nova intensidade atribuída ao pixel. Esse processo é repetido para todos os pixels da imagem, excetuando-se, em geral, as bordas externas, que podem ser tratadas com estratégias específicas de extensão de borda.

A principal vantagem do filtro mediana é sua capacidade de eliminar valores atípicos (outliers) sem suavizar excessivamente os contornos dos objetos presentes na imagem, o que o torna particularmente útil em aplicações que exigem preservação de detalhes estruturais. No entanto, por se tratar de uma operação não linear, o filtro mediana é mais custoso computacionalmente em comparação aos filtros lineares simples.

3.4.1 Código

```
def filtro_mediana(imagem, tamanho_janela=5):
offset = tamanho_janela // 2
altura, largura = imagem.shape
resultado = np.copy(imagem)

for i in range(offset, altura - offset):
    for j in range(offset, largura - offset):
        vizinhos = imagem
        [i-offset:i+offset+1, j-offset:j+offset+1].flatten()
        resultado[i, j] = np.median(vizinhos)

return resultado.astype(np.uint8)
```

3.4.2 Funcionamento

O código apresentado implementa o filtro mediana de forma manual para o processamento de imagens digitais. Esta função recebe como entrada uma imagem, representada como uma matriz bidimensional de intensidades, e um parâmetro opcional que define o tamanho da janela de vizinhança utilizada no cálculo da mediana, cujo valor padrão é 5×5.

Inicialmente, o código calcula o valor do *offset*, que corresponde à metade do tamanho da janela, desconsiderando a parte fracionária. Este valor é utilizado para evitar o acesso a índices inválidos nas bordas da imagem durante a varredura. Em seguida, são extraídas as dimensões da imagem (`altura` e `largura`) e criada uma cópia da imagem original chamada `resultado`, que armazenará os novos valores de intensidade após a aplicação do filtro.

A aplicação do filtro é realizada por meio de dois laços `for` aninhados, que percorrem todos os pixels da imagem, excetuando-se as margens definidas pelo *offset*. Para cada pixel (i, j), é extraída uma submatriz centrada nesse pixel, de dimensão igual ao `tamanho_janela`. Esta submatriz, que representa os vizinhos locais do pixel, é então achatada em um vetor unidimensional utilizando o método `flatten()`.

Posteriormente, é calculada a mediana dos valores desse vetor utilizando a função `np.median()`. O valor obtido substitui a intensidade original do pixel no mesmo ponto da matriz `resultado`. Dessa forma, cada pixel passa a representar a mediana dos seus vizinhos locais, o que contribui para a atenuação de ruídos impulsivos, preservando ao mesmo tempo as bordas da imagem.

Após o processamento de todos os pixels válidos, a função converte a matriz `resultado` para o tipo `uint8`, garantindo que os valores estejam no intervalo adequado para imagens de 8 bits por pixel (0 a 255), e retorna a imagem suavizada.

Essa implementação é utilizada somente como apoio ao filtro *High-Boost*.

3.5 Filtro *High-Boost*

O filtro *high-boost* é uma técnica de realce de imagens que visa enfatizar os detalhes de alta frequência, como bordas e texturas, enquanto preserva as componentes de baixa frequência, como áreas homogêneas. Diferentemente dos filtros passa-alta tradicionais, que podem eliminar informações importantes de baixa frequência, o filtro high-boost permite um controle mais refinado sobre o grau de realce aplicado à imagem.

Matematicamente, o filtro high-boost é definido pela equação:

$$I_{HB} = A \cdot I - I_{LP}$$

onde:

- I_{HB} é a imagem resultante após a aplicação do filtro high-boost;

- A é o fator de amplificação, com $A \geq 1$;
- I é a imagem original;
- I_{LP} é a versão suavizada (passa-baixa) da imagem original.

Reescrevendo a equação, temos:

$$I_{HB} = (A-1) \cdot I + (I - I_{LP})$$

Observa-se que $I - I_{LP}$ corresponde à aplicação de um filtro passa-alta na imagem original. Assim, o filtro high-boost pode ser interpretado como a soma da imagem original amplificada por $(A-1)$ com sua versão passa-alta. Quando $A = 1$, o filtro high-boost se reduz a um filtro passa-alta convencional. Para $A > 1$, parte da imagem original é preservada na saída, permitindo recuperar componentes de baixa frequência que seriam perdidas em uma filtragem passa-alta pura.

Na implementação prática, a versão suavizada I_{LP} pode ser obtida por meio de filtros passa-baixa, como o filtro da média ou o filtro gaussiano. A escolha do fator A influencia diretamente o grau de realce: valores maiores de A resultam em um realce mais pronunciado das altas frequências, enquanto valores próximos de 1 produzem um efeito mais sutil.

O filtro high-boost é amplamente utilizado em aplicações que requerem o realce de detalhes finos sem comprometer a integridade das regiões homogêneas da imagem, como em sistemas de reconhecimento de padrões, análise médica de imagens e processamento de fotografias digitais.

3.5.1 Código

```
# --- 3. Filtro High-Boost ---
matriz_original = filtro_mediana(matriz_original)
if media < 50:
    suavizada = filtro_mediana(suavizada, tamanho_janela=10)
else:
    suavizada = filtro_mediana(suavizada, tamanho_janela=9)
highboost = (A * matriz_original) - suavizada
highboost = np.clip(highboost, 0, 255).astype(np.uint8)
```

3.5.2 Funcionamento

Inicialmente, a matriz da imagem original é submetida a um processo de filtragem por mediana, utilizando a função `filtro_mediana`. O filtro mediana é essencial para a remoção de ruídos impulsivos (como o ruído sal e pimenta) sem comprometer significativamente a preservação das bordas da imagem. Este pré-processamento garante uma base mais limpa para as operações subsequentes.

Em seguida, a matriz suavizada, derivada da imagem original, também é submetida a uma filtragem por mediana. O tamanho da janela utilizado na filtragem é adaptativo, sendo escolhido com base na média de intensidades da imagem calculada anteriormente.

Finalmente, os valores da matriz resultante são limitados ao intervalo válido de intensidades para imagens de 8 bits (0 a 255) utilizando a função `np.clip`, garantindo que não ocorram valores inválidos ou distorções visuais. Após o ajuste, os dados são convertidos para o tipo `uint8`, compatível com a representação padrão de imagens digitais.

Este conjunto de operações assegura que a imagem final apresente um realce controlado dos detalhes, sendo importante para esse trabalho por conta da valorização de características sutis sem amplificação excessiva de ruídos.

3.6 Filtro Laplaciano

O filtro Laplaciano é uma técnica de processamento de imagens utilizada principalmente para a detecção de bordas. Ele se baseia na aplicação do operador Laplaciano, que é um operador diferencial de segunda ordem, calculado como a soma das segundas derivadas parciais da função de intensidade da imagem. Em termos discretos, o filtro Laplaciano é implementado por meio de uma convolução da imagem com uma máscara (ou kernel) que aproxima essas derivadas. O principal objetivo deste filtro é identificar regiões da imagem onde ocorrem mudanças rápidas de intensidade, que geralmente correspondem às bordas dos objetos.

Matematicamente, o operador Laplaciano ∇^2 aplicado a uma função de duas variáveis $I(x, y)$, que representa a intensidade da imagem em cada ponto, é definido como:

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Na prática, essa operação é aproximada utilizando uma máscara discreta, como por exemplo:

$$\text{Máscara clássica} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{ou} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Essas máscaras são convoluídas com a imagem, calculando em cada ponto a diferença entre a intensidade do pixel central e a soma ponderada das intensidades dos seus vizinhos. Como resultado, as regiões com variações abruptas de intensidade (bordas) apresentam valores altos, enquanto regiões homogêneas tendem a valores próximos de zero.

O filtro Laplaciano possui a característica de ser isotrópico, ou seja, responde igualmente a bordas em todas as direções. Além disso, devido ao fato de calcular segundas derivadas, ele é extremamente sensível ao ruído; portanto, é comum aplicar uma etapa de suavização na imagem, como um filtro Gaussiano, antes da aplicação do Laplaciano, resultando na técnica conhecida como *LoG* (Laplacian of Gaussian).

A principal vantagem do filtro Laplaciano é sua simplicidade e capacidade de destacar de forma clara e precisa as bordas presentes na imagem, sendo amplamente utilizado em sistemas de visão computacional, segmentação de imagens e pré-processamento para extração de características.

3.6.1 Código

```
# --- 4. Deteccao de bordas com Laplaciano ---
laplaciano = np.zeros((altura, largura), dtype=np.int32)
# Mascara Laplaciana 8-conectada
mascara = np.array([
    [-1, -1, -1],
    [-1,  8, -1],
    [-1, -1, -1]
])

# Aplicando o filtro
for i in range(1, altura - 1):
    for j in range(1, largura - 1):
        regiao = highboost[i-1:i+2, j-1:j+2]
        valor = np.sum(regiao * mascara)
        laplaciano[i, j] = valor

contorno = np.where(np.abs(laplaciano) >
limiar_diferenca, 255, 0).astype(np.uint8)
```

3.6.2 Funcionamento

Nesta etapa, é realizada a detecção de bordas na imagem previamente realçada pelo filtro *High-Boost*, utilizando o operador Laplaciano.

Inicialmente, é criada uma matriz chamada `laplaciano`, com as mesmas dimensões da imagem de entrada e inicializada com zeros. Essa matriz armazenará o resultado da convolução da imagem com a máscara Laplaciana. A máscara utilizada é uma versão 8-conectada, representada por:

$$\text{Máscara} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Esta máscara avalia as diferenças de intensidade entre o pixel central e seus oito vizinhos, permitindo a detecção de bordas em todas as direções.

O processo de aplicação do filtro é realizado por meio de uma convolução manual. Para cada pixel da imagem (excetuando-se as bordas externas), é extraída uma região 3×3 centrada no pixel atual. Em seguida, é computado o somatório do produto elemento a elemento entre esta região e a máscara Laplaciana. O valor resultante é armazenado na matriz `laplaciano` na posição correspondente ao pixel central.

Após a aplicação do filtro, o próximo passo é gerar a matriz de contornos, denominada `contorno`. Para isso, é analisado o valor absoluto de cada elemento da matriz `laplaciano`. Se esse valor for superior a um limiar previamente definido (`limiar_diferenca`), considera-se que o pixel pertence a uma borda e atribui-se a ele o valor 255 (branco); caso contrário, o valor 0 (preto) é atribuído. Esta etapa gera uma imagem binária onde as bordas são destacadas.

A abordagem adotada nesse código assegura uma detecção eficiente das bordas, especialmente após a amplificação dos detalhes proporcionada pelo filtro *High-Boost*. A escolha da máscara 8-conectada é importante pois permite capturar bordas em múltiplas direções de maneira uniforme, enquanto a utilização de um limiar adaptável contribui para reduzir o impacto de pequenas variações ou ruídos residuais na imagem processada.

4 Metodologia

Este trabalho foi desenvolvido a partir da seleção de imagens de diferentes contextos, a fim de compor uma amostra representativa de situações reais de aplicação. Foram utilizadas cinco imagens de variados tamanhos e resoluções, incluindo: uma fotografia de um tigre, uma imagem de uma mesa com objetos dispostos sobre ela, duas imagens de desenhos digitais e uma imagem de uma renderização 3D.

Essas imagens foram empregadas na definição dos melhores *presets* de luminosidade, conforme descrito anteriormente.

4.1 Decisões de Implementação

Inicialmente, foi idealizado o uso exclusivo do filtro Laplaciano para a detecção de bordas. No entanto, esta abordagem mostrou-se pouco eficaz, apresentando resultados insatisfatórios. Diante disso, implementou-se o filtro Gaussiano, com o objetivo de suavizar as imagens e melhorar a qualidade antes da detecção de bordas. A inclusão da suavização gaussiana trouxe avanços consideráveis, embora os resultados ainda não fossem satisfatórios para todas as imagens analisadas.

Como alternativa, introduziu-se o uso do filtro High-Boost, visando realçar as bordas presentes nas imagens. A aplicação desse filtro produziu resultados visivelmente superiores, mas ainda exigia refinamentos adicionais para a obtenção de uma qualidade adequada.

Esse refinamento foi alcançado pela utilização do filtro Mediana, que contribuiu para a redução de ruídos residuais e para o aprimoramento da definição das bordas. Embora alguns desafios persistam, em todas as imagens processadas as bordas tornaram-se claramente visíveis, validando a abordagem adotada.

5 Resultados e Análise

A análise dos resultados obtidos não foi baseada em métricas quantitativas, mas sim em uma avaliação qualitativa da qualidade visual das imagens processadas, considerando a clareza e a definição das bordas realçadas.

Para cada imagem testada, foram considerados diferentes fatores a fim de determinar a qualidade do resultado final. A seguir, são descritas as observações referentes a cada imagem:

- **Imagem 1: Foto do tigre**: O *preset* utilizado foi o correspondente a imagens com tons médios. Nesta imagem, foi particularmente desafiador obter um resultado satisfatório, pois os pelos do tigre dificultam a delimitação das bordas. Se ignorados, há perda significativa de contorno; se considerados, há um excesso de bordas, comprometendo a clareza da imagem. Concluiu-se que o ideal seria obter um contorno mínimo do animal, realçando apenas seus limites mais relevantes. O resultado é apresentado a seguir:

Figura 1: Imagem 1 (Original): Tigre

Fonte: Sherry Christian / Unsplash.

Figura 2: Imagem 1 (Resultado): Tigre

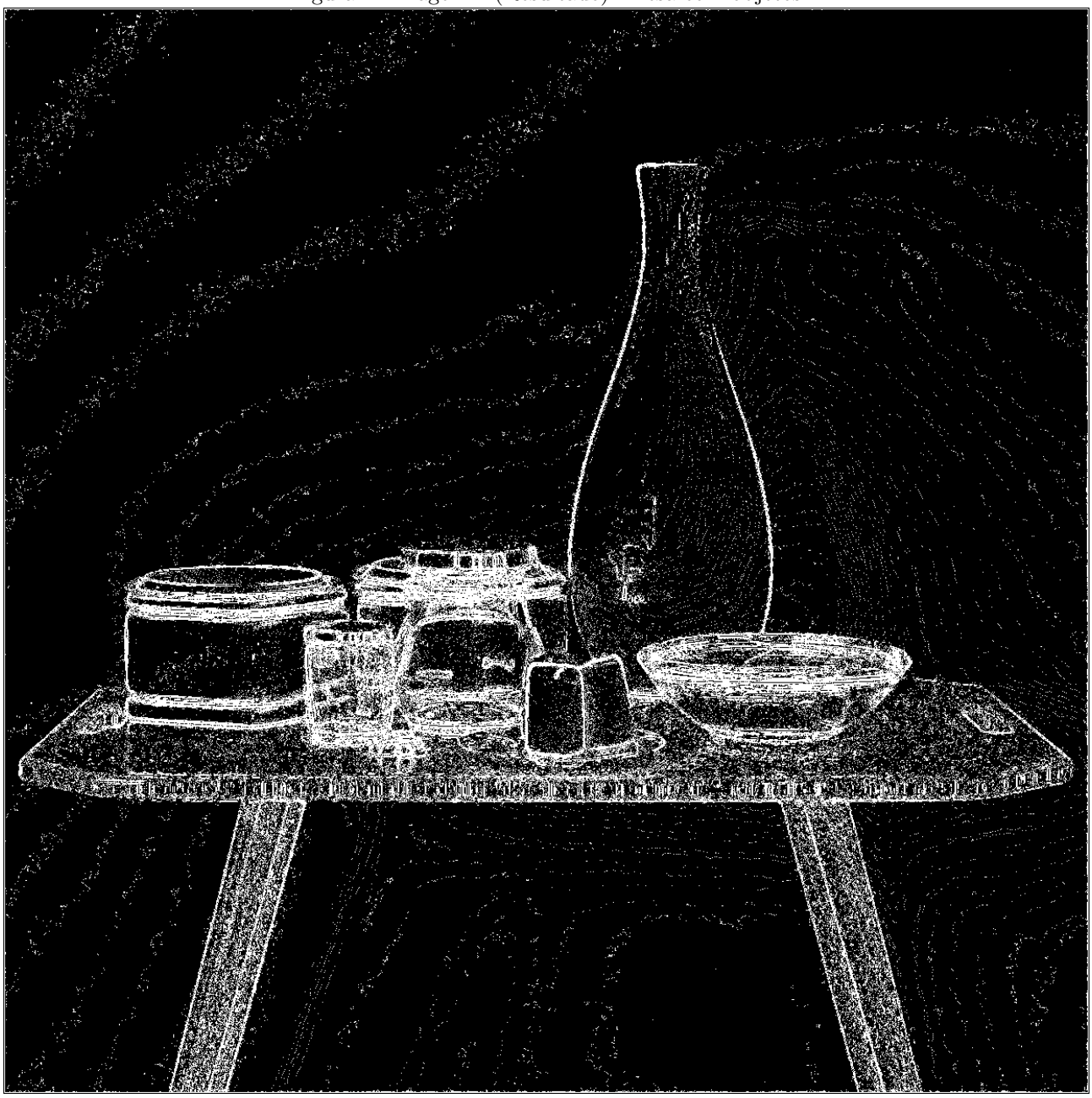
Fonte: Elaboração do autor a partir de Sherry Christian / Unsplash.

- **Imagem 2: Foto de uma mesa com objetos dispostos**: Foi utilizado o *preset* para imagens claras. Esta imagem evidenciou a necessidade da utilização de filtros adicionais ao Laplaciano, devido à presença de objetos translúcidos e com baixa definição de contorno, como um vaso de cor semelhante ao fundo à direita da composição. Além disso, essa imagem possui algumas manchas que ficam evidenciadas após a escala da imagem ser convertida para cinza. Somente após a aplicação combinada dos filtros mencionados anteriormente foi possível obter um resultado satisfatório. O resultado é mostrado abaixo:

Figura 3: Imagem 2 (Original): Mesa com objetos

Fonte: Autor desconhecido / 3dLancer.net.

Figura 4: Imagem 2 (Resultado): Mesa com objetos



Fonte: Elaboração do autor a partir de autor desconhecido / 3dLancer.net.

- **Imagem 3: *Frame* de uma animação**: Também foi utilizado o *preset* para tons médios. Esta imagem foi fundamental para a concepção da ideia dos *presets*, já que apresentava um defeito específico: camadas de manchas, que não pertenciam ao *frame*, mas que se tornavam muito evidentes após a conversão para escala de cinza. O uso do filtro Gaussiano foi crucial para suavizar esse artefato e permitir a extração adequada das bordas. O resultado está a seguir:

Figura 5: Imagem 3 (Original): *Frame* de animação

Fonte: Steins;Gate

Figura 6: Imagem 3 (Resultado): *Frame* de animação

Fonte: Elaboração do autor a partir de Steins;Gate

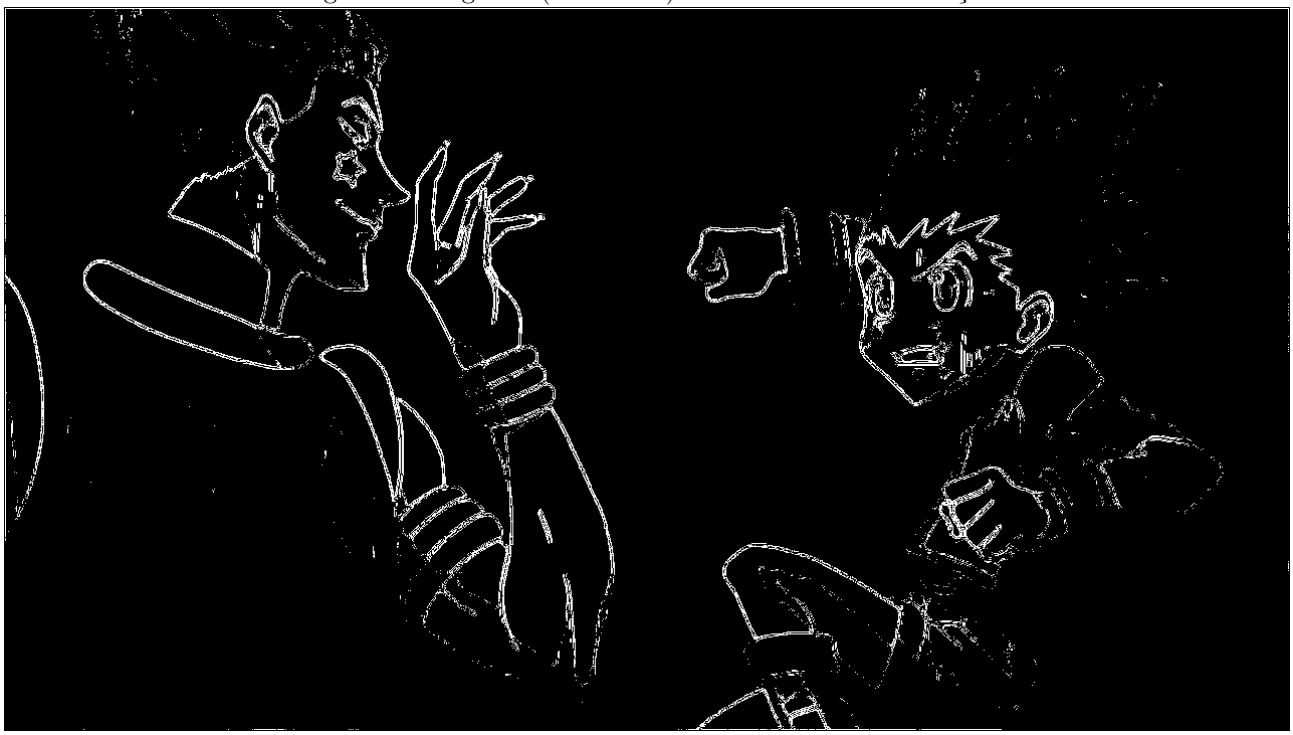
- **Imagem 4: *Frame* de outra animação**: Utilizou-se novamente o *preset* para tons médios. Esta imagem serviu de referência para esse tipo de iluminação, pois apresenta uma divisão visível entre regiões iluminadas (lado direito) e sombreadas (lado esquerdo). A principal dificuldade foi ajustar os parâmetros de forma a evidenciar as vestimentas do personagem à esquerda sem comprometer a definição do lado direito. O resultado final,

apesar de algumas limitações, foi satisfatório:

Figura 7: Imagem 4 (Original): *Frame* de outra animação

Fonte: Hunter x Hunter 2011

Figura 8: Imagem 4 (Resultado): *Frame* de outra animação



Fonte: Elaboração do autor a partir de Hunter x Hunter 2011

- **Imagem 5: *Frame* de uma renderização 3D**: Utilizou-se o *preset* para tons escuros. Esta imagem motivou o uso do filtro Mediana na entrada do filtro *High-Boost*. Sem essa etapa adicional, havia excesso de ruído nas regiões de folhagem, pouca definição nos pássaros e perda de detalhes nos objetos do canto inferior esquerdo. A dificuldade maior

foi o balanceamento das configurações, dada a existência de áreas levemente iluminadas contrastando com zonas bastante escuras. O resultado é mostrado a seguir:

Figura 9: Imagem 5 (Original): Renderização 3D

Fonte: Resident Evil 4 Remake

Figura 10: Imagem 5 (Resultado): Renderização 3D

Fonte: Elaboração do autor a partir de Resident Evil 4 Remake

5.1 Comparação com o artigo escolhido

O artigo *"An Enhanced Edge Detection Using Laplacian Gaussian Filtering Method from Different Denoising Images"*, publicado em 2024 na revista IJISAE, apresenta uma proposta voltada à detecção de bordas em imagens médicas, com ênfase em raios-X. A abordagem consiste na aplicação sequencial de dois filtros clássicos: o Gaussiano, utilizado como técnica de remoção de ruídos (*denoising*), e o Laplaciano, responsável pela detecção das bordas. A proposta visa melhorar a eficiência computacional e a precisão da detecção, sendo avaliada por métricas objetivas como RMSE, MSE, PSNR e tempo de execução. O algoritmo sugerido é testado em nove imagens distintas, demonstrando superioridade em relação aos métodos tradicionais tanto na qualidade das bordas detectadas quanto no desempenho computacional.

Em comparação, o presente trabalho também utiliza o filtro Laplaciano como base para a detecção de bordas, demonstrando uma afinidade teórica com o artigo analisado. Ambos os estudos reconhecem a sensibilidade do operador Laplaciano ao ruído e, por isso, o precedem de técnicas de suavização para atenuar esse efeito. Entretanto, enquanto o artigo da IJISAE utiliza apenas o filtro Gaussiano como etapa de pré-processamento, este trabalho adota uma abordagem mais elaborada e adaptativa, incorporando sucessivamente os filtros Gaussiano, Mediana e *High-Boost*, permitindo uma amplificação seletiva de detalhes relevantes e a atenuação de ruídos sem comprometer a nitidez das bordas.

Além disso, destaca-se que o artigo analisado foca exclusivamente em imagens médicas, com propriedades e desafios bastante específicos, como contraste limitado e presença de ruído estrutural. Já este trabalho aplica a metodologia a imagens de natureza diversa, como fotografias de objetos, animações e renderizações tridimensionais, abrangendo um espectro visual mais amplo. Essa escolha amplia a aplicabilidade da solução proposta e exige maior flexibilidade da técnica adotada.

Outra distinção importante diz respeito à forma de avaliação dos resultados. O artigo da IJISAE fundamenta-se em métricas objetivas para medir o desempenho da detecção de bordas, o que confere um rigor quantitativo à análise. Por outro lado, este trabalho adota uma abordagem qualitativa, baseada na percepção visual das bordas geradas, o que é adequado ao caráter exploratório e didático da proposta, embora possa dificultar comparações diretas com outros métodos.

Por fim, um diferencial importante deste estudo é a introdução de *presets* de parâmetros baseados na média de luminância das imagens. Essa adaptação dinâmica não está presente no artigo comparado, mas mostrou-se fundamental aqui para ajustar o comportamento do algoritmo conforme o perfil de brilho da imagem, garantindo melhores resultados em contextos variados.

Em síntese, ambos os trabalhos partem de uma base teórica semelhante — o uso do operador Laplaciano suavizado por um filtro Gaussiano —, mas seguem caminhos diferentes na implementação e aplicação. O artigo da IJISAE destaca-se pela simplicidade e foco clínico, enquanto este trabalho explora um pipeline mais versátil e adaptativo, que pode ser especialmente útil em ambientes de estudo e prototipação. As diferenças metodológicas, contextuais e avaliativas revelam a complementaridade entre as abordagens e reforçam a importância de adaptar a técnica ao domínio específico de aplicação.

5.2 Possíveis melhorias

Apesar da aplicação de múltiplas etapas de pré-processamento e filtragem, os resultados obtidos com o algoritmo apresentado ainda deixam a desejar em certos aspectos. Foram identificados resquícios de ruído em algumas imagens, bem como falhas na definição contínua das bordas detectadas, que por vezes se apresentam de forma fragmentada ou pontilhada, comprometendo a legibilidade estrutural das imagens processadas.

Um dos principais fatores que podem ter contribuído para esses resultados inconsistentes é o ajuste não totalmente calibrado dos filtros utilizados. A parametrização empírica dos valores de

amplificação (`A`) e do `limiar_diferenca`, ainda que adaptada com base na média de intensidade da imagem, pode não ser suficiente para lidar com todas as variações possíveis de tonalidade e contraste, principalmente em imagens com regiões mistas de iluminação. Uma possível melhoria seria a substituição desses critérios fixos por um mecanismo de ajuste automático baseado em histogramas ou em análise de contraste local.

Além disso, o filtro de mediana, embora eficaz na remoção de ruídos impulsivos, pode atenuar detalhes finos quando utilizado com janelas muito grandes, como as adotadas nesta implementação (9×9 ou 10×10). Como alternativa, poderia ser utilizado os filtros de preservação de bordas mais sofisticados, como o *Bilateral Filter* ou o *Non-Local Means*, que tendem a apresentar melhor desempenho em contextos com ruídos estruturais e variações texturais.

Outro ponto relevante está na etapa de detecção de bordas em si. A aplicação da máscara Laplaciana 8-conectada, embora sensível a variações em múltiplas direções, pode gerar respostas discretas em áreas onde a transição de intensidade é mais suave, resultando em bordas descontínuas. Avaliar a combinação da resposta do Laplaciano com operadores de gradiente (como Sobel ou Prewitt) ou a utilização do operador *Laplacian of Gaussian (LoG)* diretamente pode proporcionar resultados mais robustos, unindo a sensibilidade direcional à suavização prévia integrada.

Por fim, ressalta-se que a etapa de avaliação dos resultados foi predominantemente qualitativa. A inclusão de métricas objetivas como *PSNR*, *MSE* ou *Edge Preservation Index* pode fornecer uma análise mais rigorosa da qualidade dos contornos detectados, auxiliando na calibração fina dos parâmetros envolvidos.

Em síntese, embora a estrutura geral do algoritmo seja sólida, a melhoria da qualidade dos resultados pode ser alcançada por meio de ajustes dinâmicos de parâmetros, substituição de filtros por versões mais adaptativas e integração de métricas quantitativas para guiar as decisões de implementação.

6 Conclusão

Referências

[1] MATHUR, Sumeet; GUPTA, Sandeep. An Enhanced Edge Detection Using Laplacian Gaussian Filtering Method from Different Denoising Images. *International Journal of Intelligent Systems and Applications in Engineering (IJISAE)*, v. 12, n. 18s, p. 313–323, 2024. Disponível em: `https://www.ijisae.org/index.php/IJISAE/article/view/4975`. Acesso em: abr. 2025.

[2] GONZALEZ, Rafael C.; WOODS, Richard E. *Digital Image Processing*. 3. ed. Upper Saddle River: Pearson, 2009.

[3] PARKER, James R. *Algorithms for Image Processing and Computer Vision*. 2. ed. Wiley Publishing, 2011.

[4] CANNY, John. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. PAMI-8, n. 6, p. 679–698, 1986.

[5] BRADSKI, Gary. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. Disponível em: `https://opencv.org`. Acesso em: abr. 2025.

[6] SOBEL, Irwin. Camera Models and Machine Perception. *Stanford Artificial Intelligence Project*, SAIL Technical Report, 1970.