Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Eletrônica – DAELN
Seu curso aqui
Disciplina: IF69D – Processamento Digital de Imagens
Semestre: 2023.2
Prof.: Gustavo Borba e Humberto Gamba

RELATÓRIO
Limiarização Adaptativa com Imagem Integral

Alunos:
Felipe Adonai de Moraes / 2004429

setembro.2024

1. Objetivo

Implementar a técnica de Limiarização Adaptativa com Imagem Integral em Python e comparar seu desempenho com outras técnicas de limiarização, como o método de Otsu e a limiarização adaptativa tradicional.

2. Fundamentação Teórica

2.1 Limiarização

A limiarização é uma técnica fundamental em processamento de imagens que classifica pixels como "escuros" ou "claros" com base em um limiar pré-definido. O objetivo é transformar uma imagem em tons de cinza em uma imagem binária, onde os pixels acima do limiar são considerados "claros" (usualmente representados por 255) e os pixels abaixo são considerados "escuros" (usualmente representados por 0). No entanto, em muitas aplicações reais, a iluminação da imagem pode variar significativamente, tornando a limiarização global ineficaz. Em imagens com variações de iluminação, um limiar fixo pode resultar em uma segmentação incorreta, classificando erroneamente pixels como "claros" ou "escuros" devido à diferença de iluminação.

Para lidar com esse problema, surge o método de Otsu, uma técnica de limiarização global que busca encontrar o limiar ótimo que maximiza a variância entre os dois grupos de pixels (foreground e background). Essa técnica assume que a imagem possui um histograma bimodal, o que nem sempre é o caso em imagens reais.

A limiarização de Otsu é um bom ponto de partida para imagens com contrastes bem definidos, mas para lidar com variações complexas de iluminação, métodos de limiarização adaptativa, como a técnica com imagem integral, são mais eficientes.

2.2 Limiarização Adaptativa

A limiarização é uma técnica fundamental no processamento de imagens que classifica pixels como "escuros" ou "claros" com base em um limiar pré-definido, transformando uma imagem em tons de cinza em uma imagem binária. No entanto, em imagens com variações de iluminação, um limiar fixo pode levar a uma segmentação incorreta. Para lidar com esse problema, métodos de limiarização adaptativa, como a limiarização adaptativa tradicional por média e o método de Wellner's, calculam um limiar diferente para cada pixel, considerando características locais da imagem.

A limiarização adaptativa tradicional por média calcula a média dos pixels em uma vizinhança definida ao redor de cada pixel e compara o valor do pixel atual com essa média para determinar se ele é "claro" ou "escuro". O método de Wellner's, por outro lado, utiliza uma

média móvel para determinar o limiar de cada pixel, considerando um número fixo de pixels anteriores e comparando o pixel atual com essa média móvel.

Em resumo, a limiarização adaptativa por média é mais robusta a variações de iluminação e oferece maior flexibilidade na definição da vizinhança, mas pode ser computacionalmente mais cara, especialmente em imagens grandes. O método de Wellner's é mais simples e eficiente, mas pode ser menos preciso em imagens com variações de iluminação abrupta ou complexas. A escolha do melhor método depende dos requisitos específicos da aplicação e das características da imagem.

2.3 Imagem Integral (Tabela de Área Somada)

A Imagem Integral, também conhecida como Tabela de Área Somada, é uma estrutura de dados que permite calcular a soma dos valores dos pixels dentro de qualquer região retangular de uma imagem em tempo constante. Essa técnica é extremamente útil para algoritmos de processamento de imagens que exigem o cálculo de médias de blocos, como a limiarização adaptativa.

A imagem integral é uma representação da imagem original, onde cada pixel contém a soma de todos os pixels acima e à esquerda dele, incluindo ele próprio. Para calcular a imagem integral, é utilizada a seguinte fórmula (1) [1]:

$$I(x,y) = f(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1) \quad (1)$$

Onde:

- $I(x,y)$ representa o valor da imagem integral no pixel (x, y)
- $f(x,y)$ representa a intensidade do pixel (x, y) na imagem original

Podemos verificar a expressão da fórmula de forma visual na Figura 1, onde à direita temos a matriz representando a imagem integral, e à esquerda a matriz representando a imagem em tons de cinza.

98	84	4	5	10
123	123	16	11	11
123	123	63	10	20
123	120	119	82	40
123	115	107	102	71

98	182	186	191	201
221	428	448	464	485
344	671	757	813	854
467	917	1119	1257	1338
590	1155	1464	1704	1856

Figura 1 - (esquerda) Imagem em tons de cinza e (direita) imagem integral

Para calcular a soma dos valores dos pixels dentro de um retângulo arbitrário na imagem original, basta utilizar a imagem integral e os quatro pontos do retângulo.
A fórmula para calcular a soma dos valores dos pixels dentro de um retângulo com canto superior esquerdo (x1, y1) e canto inferior direito (x2, y2) é (2) [1]:

$$\sum\sum f(x,y) = I(x2,y2) - I(x2,y1-1) - I(x1-1,y2) + I(x1-1,y1-1) \quad (2)$$

Onde:

- $\sum\sum f(x, y)$ representa a soma dos valores dos pixels dentro do retângulo
- $I(x,\ y)$ representa o valor da imagem integral no pixel (x, y)

A imagem integral é uma técnica poderosa e eficiente para calcular a soma de pixels em regiões retangulares, o que a torna uma ferramenta valiosa para vários algoritmos de processamento de imagens.

2.4 Técnica de Limiarização Adaptativa com Imagem Integral

O método apresentado no artigo "Adaptive Thresholding Using the Integral Image" combina a limiarização adaptativa com a imagem integral. Ele calcula a média dos pixels dentro de um bloco definido ao redor de cada pixel usando a imagem integral e compara o valor do pixel com a média do bloco. Se o valor do pixel estiver abaixo de um limiar definido (multiplicado pela média do bloco), o pixel é classificado como "escuro", caso contrário, é classificado como "claro".

Algumas vantagens do modelo incluem:

- Robustez à Iluminação: O uso da imagem integral torna o método mais robusto a alterações de iluminação.
- Eficiência Computacional: A técnica é mais eficiente computacionalmente do que outros métodos adaptativos, a imagem integral permite calcular a soma dos valores dos pixels dentro de qualquer retângulo em tempo constante, o que é significativamente mais eficiente do que realizar a soma iterando sobre todos os pixels individualmente.
- Independente da Ordem de Processamento: O método produz a mesma saída independentemente da ordem de processamento dos pixels.

Algumas desvantagens incluem:

- Menos flexível: O método é menos flexível para definir a forma da região de vizinhança, se limitando a regiões retangulares.
- Cálculo da Imagem Integral: O cálculo da imagem integral pode ser demorado para imagens grandes.

Esse modelo é amplamente utilizada em várias áreas de processamento de imagens, incluindo:

- Detecção de Objetos: A imagem integral é usada para calcular rapidamente a média de blocos de pixels para detectar objetos em imagens.
- Detecção de Características: A imagem integral é útil para identificar características locais em imagens, como bordas e cantos.
- Realidade Aumentada (RA): A imagem integral pode ser usada para detectar marcadores em tempo real em aplicações de RA.

3. Implementação

O algoritmo foi implementado em Python, utilizando a biblioteca OpenCV. O código implementa os seguintes métodos:

- Limiarização de Otsu: Calcula o limiar ótimo de Otsu para segmentar a imagem.
- Limiarização Adaptativa com Média: Calcula a média dos pixels vizinhos em um bloco definido ao redor de cada pixel para realizar a limiarização adaptativa.
- Limiarização Adaptativa com Imagem Integral: Implementa o algoritmo descrito no artigo "Adaptive Thresholding Using the Integral Image" utilizando a imagem integral para calcular a média dos pixels.
- Método de Wellner: Implementa o método de limiarização adaptativa de Wellner, que utiliza uma média móvel para determinar o limiar de cada pixel.

O algoritmo de limiarização adaptativa com imagem integral pode ser dividido em duas etapas principais: Cálculo da imagem integral e cálculo da limiarização.

A imagem integral é calculada iterando sobre cada pixel da imagem original e somando os valores dos pixels acima e à esquerda dele, incluindo ele próprio. Essa operação é realizada linha por linha da imagem, armazenando o resultado em uma nova matriz, chamada de "intImgl". Veja o trecho de código que ilustra o cálculo da imagem integra na Figura 2l:

```
intImg = np.zeros((w, h))
  for i in range(w):
    sum = 0
    for j in range(h):
      sum += in_img[i, j]
      if i == 0:
        intImg[i, j] = sum
      else:
        intImg[i, j] = intImg[i-1, j] + sum
```

Figura 2 - Bloco de código para calculo da imagem integral

No código, intImg é a matriz que armazena a imagem integral, in_img é a imagem original, e w e h são a largura e a altura da imagem, respectivamente.
Após calcular a imagem integral, o algoritmo itera sobre cada pixel da imagem original. Para cada pixel, a média dos pixels em um bloco (retângulo) definido ao redor do pixel é calculada utilizando a imagem integral. O tamanho do bloco é um parâmetro que pode ser ajustado. O valor do pixel atual é então comparado com a média do bloco. Se o valor do pixel for menor do que a média do bloco multiplicada por uma porcentagem (limiar), o pixel é classificado como "escuro" (preto) e recebe o valor 0; caso contrário, o pixel é classificado como "claro" (branco) e recebe o valor 255. Veja o trecho de código que ilustra a limiarização na Figura 3.

```
out_img = np.zeros((w, h))
  # Itera sobre cada pixel da imagem.
  for i in range(w - 1):
    for j in range(h - 1):
      # Define as coordenadas do bloco.
      x1 = max(0, i - block_size // 2)
      x2 = min(w - 1, i + block_size // 2)
      y1 = max(0, j - block_size // 2)
      y2 = min(h - 1, j + block_size // 2)
      # Calcula a soma dos pixels no bloco.
      block_sum = intImg[x2, y2] - intImg[x1, y2] - intImg[x2, y1] + intImg[x1, y1]
      # Calcula a quantidade de pixels no bloco.
      block_count = (x2 - x1) * (y2 - y1)
      # Compara o pixel com a média do bloco.
      if in_img[i, j] * block_count <= (block_sum * (100 - threshold) / 100):
        out_img[i, j] = 0
      else:
        out_img[i, j] = 255
```

Figura 3 - Bloco de código para limiarização

Neste código, block_size define o tamanho do bloco e threshold é o limiar, definido como uma porcentagem.

A imagem integral é utilizada para calcular a soma dos pixels dentro do bloco de forma eficiente, evitando percorrer todos os pixels do bloco. O algoritmo itera sobre cada pixel da imagem, calcula a média do bloco, compara o pixel com a média do bloco e, então, classifica o pixel como "claro" ou "escuro" com base no limiar.

4. Resultados e conclusões

Foram realizados testes com diferentes imagens, variando o tamanho do bloco (block_size) e o limiar (threshold).

A escolha do tamanho do bloco (block_size) na limiarização adaptativa com imagem integral impacta diretamente a qualidade da segmentação. Observar a Figura 4, que ilustra os resultados da limiarização com diferentes valores de block_size para a mesma imagem, permite entender como esse parâmetro influencia o resultado:

- block_size pequeno (ex: 5 - Figura 4 à direita): Um bloco pequeno, como 5, torna a limiarização mais sensível a variações locais de iluminação. O resultado é uma segmentação mais detalhada, porém, pode ser suscetível a ruído e artefatos, resultando em bordas irregulares e imprecisões na segmentação.
- block_size médio (ex: 89 - Figura 4 à esquerda): Um bloco de tamanho médio, como 89, oferece um bom equilíbrio entre precisão e robustez. A segmentação se torna mais suave, com menos ruído e artefatos, mas pode perder alguns detalhes finos.
- block_size grande (ex: 500 - Figura 4 no centro): Com um bloco grande, como 500, a limiarização se torna mais robusta a variações de iluminação, mas perde detalhes. Os padrões se tornam mais "borrados", e as bordas podem perder nitidez, especialmente em regiões com variações abruptas de iluminação.

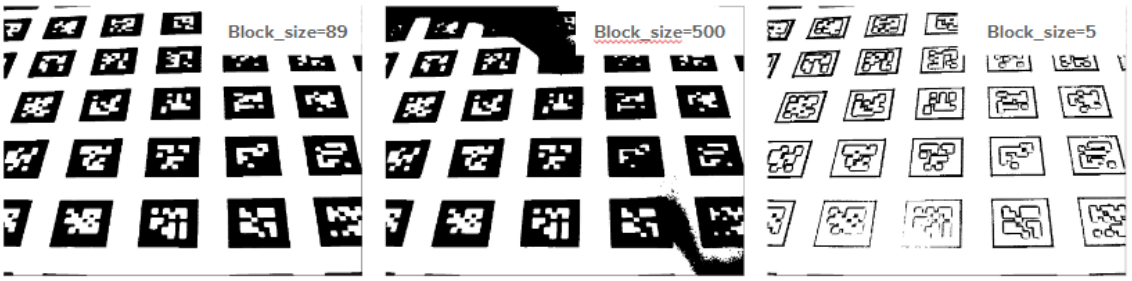

Figura 4 - Exemplos de limiarização variando o parâmetro block_size

O parâmetro threshold define a tolerância na comparação entre a intensidade do pixel atual e a média do bloco. Observar a Figura 5, que ilustra os resultados da limiarização com diferentes valores de threshold para a mesma imagem, permite entender como esse parâmetro influencia o resultado:

- threshold baixo (ex: 5 - Figura 5 do centro): Um threshold baixo, como 5, torna a limiarização mais sensível à diferença entre a intensidade do pixel e a média do bloco. A segmentação se torna mais rigorosa, com detalhes finos sendo preservados, mas o resultado pode ser suscetível a ruído.
- threshold médio (ex: 15 - Figura 5 à esquerda): Um threshold médio, como 15, oferece um bom equilíbrio entre precisão e tolerância a ruídos. A segmentação se torna mais suave, com menos ruído e artefatos, mas pode perder alguns detalhes finos.
- threshold alto (ex: 50 - Figura 5 à direita): Um threshold alto, como 50, torna a limiarização menos sensível, e mais tolerante a variações de iluminação e ruídos. A segmentação é mais robusta, mas pode perder detalhes e tornar a imagem "borrada"

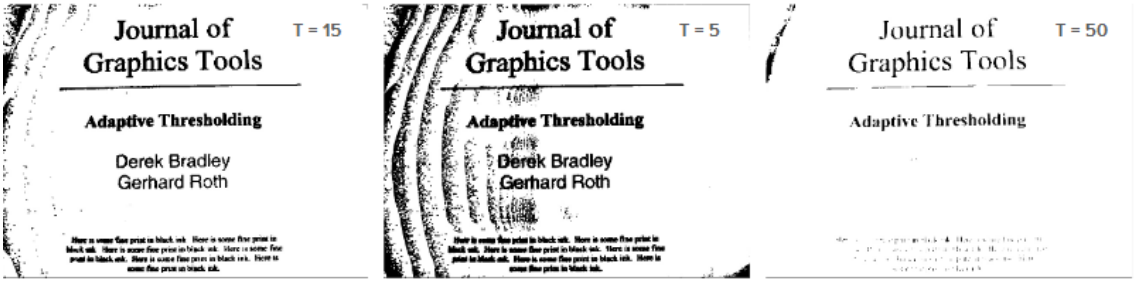

Figura 5 - Exemplos de limiarização variando o parâmetro threshold

Observar a Figura 6, que ilustra os resultados da limiarização com diferentes níveis de luminosidade para a mesma imagem, permite entender como a luminosidade influencia a segmentação. A imagem superior (alta luminosidade)apresentou falhas na segmentação. A imagem inferior (baixa luminosidade) apresentou uma segmentação mais completa e precisa do texto e da seta.

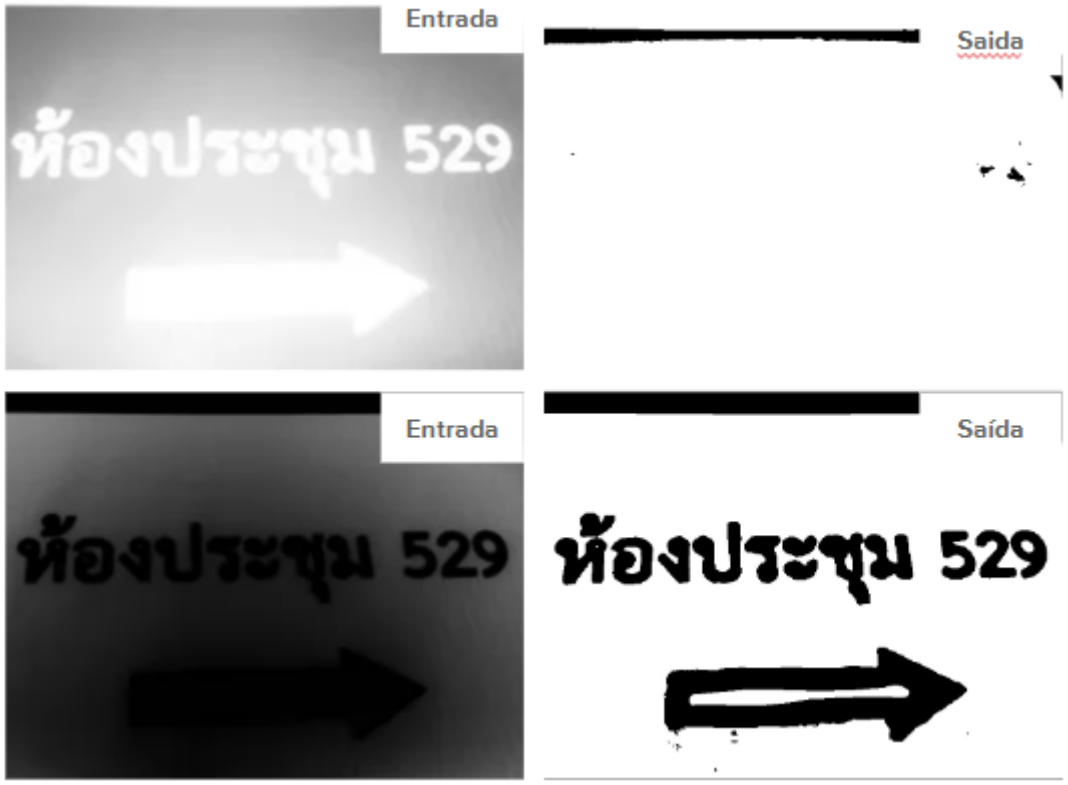

Figura 6 - Resultados limiarização com imagens com alta (cima) e baixa (baixo) luminosidade

Os resultados mostraram que a técnica de limiarização adaptativa com imagem integral é uma opção eficiente e robusta a variações de iluminação, especialmente em cenários com alta variação de intensidade.

Os métodos de limiarização adaptativa com média apresentaram bom desempenho, porém com tempos de execução mais elevados. O método de Otsu teve o menor tempo de execução, mas apresentou resultados menos precisos em imagens com variações de iluminação. Foi realizado a análise das vantagens, desvantagens e quando usar para cada um dos métodos analisados na Tabela 1.

Algoritmo	Vantagens	Desvantagens	Quando Usar
Otsu	Fácil implementação Baixo custo	Problemas ao lidar com iluminação	Histogramas com distribuição bimodal de intensidade Fundo uniforme
Wellners	Fácil implementação Adapta-se melhor com variação da iluminação	Depende da ordem de leitura dos pixels Baixa representação dos vizinhos	Pequenas variações de iluminação
Medium	Grande representação dos vizinhos Robusto quanto a variação de iluminação Grande parametrização	Custo computacional	Imagens com grande variação de iluminação
Integral	Eficiência computacional	Menos flexível Necessita de uma iteração a mais	Necessidade de alta performance: performance real-time

Tabela 1 - Comparativo entre os algoritmos de Otse, Wellners, Limiarização por média e Limiarização com imagem integral

Foram observados algumas limitações no modelo de limiarização adaptativa com imagem integral:

- O método de limiarização adaptativa com imagem integral apresenta uma baixa taxa de recall quando o objeto de interesse está em áreas de alta intensidade.
- O método pode classificar erroneamente objetos maiores do que o tamanho do bloco como background.
- O método é menos eficiente para imagens grandes devido ao cálculo linear da imagem integral e ao acesso à memória.

Para futuras pesquisas, seria interessante investigar a otimização do algoritmo para lidar com imagens grandes, utilizando técnicas de paralelização conforme propostas em [1] ou algoritmos mais eficientes para calcular a imagem integral, e alternativas para lidar com imagens de alta luminosidade conforme propostas em [2].

Conclui-se que a limiarização adaptativa com imagem integral é uma técnica eficiente e robusta para segmentar imagens com variações de iluminação. O método é especialmente interessante para aplicações em tempo real e com restrições de recursos computacionais. No entanto, é preciso estar atento às limitações da técnica, especialmente para imagens grandes.

Referências

[1] Bradley, D., & Roth, G. (2004). Adaptive thresholding using the integral image. Journal of Graphics Tools, 9(1), 11-17.

[2] Pintaric, T. (2023, 14 de junho). Concept of Modified Adaptive Thresholding Using Integral Image to Decompose Text Under Illumination. Kittipop-P. https://kittipop-p.medium.com/concept-of-modified-adaptive-thresholding-using-integral-image-to-decompose-text-under-illumination-f8ced99e271