Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Eletrônica – DAELN
Engenharia Eletrônica
Disciplina: IF69D – Processamento Digital de Imagens
Semestre: 2024.2
Prof.: Gustavo Borba e Humberto Gamba

RELATÓRIO
Análise e Aplicações do Thresholding Adaptativo em Processamento de Imagens

Alunos:
Christopher Akira França Maekawa / 1799355

Fevereiro 2025

1. Objetivo

O objetivo deste relatório é investigar o conceito de adaptive thresholding (thresholding adaptativo) no processamento de imagens, durante este estudo iremos analisar três abordagens: a aplicação padrão da técnica no MATLAB, o método de Wellner e uma versão modificada do método de Wellner utilizando imagens integrais. O estudo visa compreender o funcionamento de cada uma dessas abordagens, examinar suas implementações e avaliar as diferentes performances alcançadas por cada técnica.

2. Fundamentação Teórica

O adaptive thresholding é uma técnica avançada de segmentação de imagens utilizada para separar os objetos de interesse (foreground) do fundo (background), ajustando dinamicamente o valor do limiar em diferentes regiões da imagem. Ao contrário do thresholding tradicional, que aplica um único valor de limiar global, o adaptive thresholding calcula um limiar local para cada pixel com base nas características de sua vizinhança, o que permite uma segmentação mais precisa, especialmente em imagens com variações de iluminação, cores ou contrastes [1][2].

O método de adaptive thresholding é particularmente útil em condições em que o limiar global não seria eficaz, como em imagens com gradientes de iluminação fortes ou variações de intensidade ao longo da imagem. A técnica envolve a análise estatística dos valores de intensidade dos pixels em regiões locais, com a média (aritmética ou gaussiana) ou outras estatísticas (como a mediana) sendo frequentemente usadas para calcular o limiar para cada pixel.[1].

(a) *(b)* *(c)*

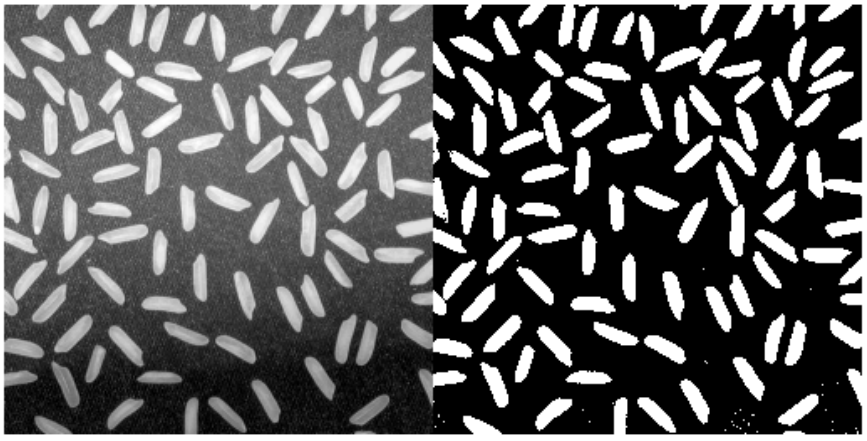
Figura 1 – (a) Imagem original. (b) Imagem com global thresholding aplicado (c) Imagem com adaptive thresholding aplicado

Esse método é crucial para melhorar a eficácia de tarefas de processamento de imagens, como a detecção de objetos em cenários com variações de iluminação, e é fundamental para a realização de segmentações precisas em áreas como visão computacional e reconhecimento de padrões [3].

2.1. Método padrão matlab

No MATLAB, a função adaptthresh é utilizada para calcular um limiar adaptativo local, baseado em estatísticas de primeira ordem (média local) nas vizinhanças dos pixels. Essa técnica é fundamental para segmentação de imagens em condições de iluminação não uniforme, onde o limiar global não seria eficaz. A função adaptthresh é aplicada a imagens em escala de cinza 2D ou volumes em 3D, e seu objetivo é calcular um limiar adaptativo local para cada pixel, considerando as características da sua vizinhança.[4]

O adaptthresh é uma ferramenta útil para imagens com gradientes de iluminação, como imagens de texto impresso ou objetos com fundo heterogêneo. Um exemplo de uso é a segmentação de grãos de arroz brancos de um fundo escuro, ou texto escuro de um fundo claro, onde a sensibilidade e o tipo de estatística podem ser ajustados para otimizar a segmentação de cada caso. O método estatístico utilizado pode ser a Média, Mediana e Gaussiana, é necessário também o ajuste do valor de sensibilidade para alcançar os melhores resultados além de outros parâmetros que podem melhorar a qualidade da imagem[4].



(a) *(b)*

Figura 2 – (a) Imagem 'rice.png' obtida no matlab. (b) Imagem com método adaptive threshold aplicado

2.2. Método de Wellner

O método de thresholding adaptativo de Wellner, proposto por Pierre D. Wellner em 1993, é uma técnica eficiente para a segmentação de imagens em tempo real. Diferente dos métodos de thresholding global, que aplicam um único limiar para toda a imagem, o método de Wellner utiliza uma média móvel aproximada dos pixels vizinhos para calcular um limiar local.[6]. A abordagem de Wellner se baseia no cálculo de uma média móvel dos últimos s pixels processados. O valor de cada pixel da imagem é comparado com essa média, e, se for menor que uma porcentagem predefinida (t) desse valor, ele é classificado como preto; caso contrário, é classificado como branco. A principal vantagem desse método é que ele requer apenas uma única passagem pela imagem, tornando sua execução extremamente rápida [5].

Além disso, Wellner propõe uma otimização baseada na redução do cálculo explícito da média móvel. Em vez de recalcular toda a soma dos últimos s pixels para cada novo ponto da imagem, ele sugere um método aproximado no qual a média é atualizada incrementalmente subtraindo uma fração do pixel mais antigo e adicionando o valor do novo pixel processado. Esse ajuste permite que o algoritmo mantenha sua eficiência, ao mesmo tempo que melhora a resposta às variações locais da imagem [6].

Comparado a outros métodos adaptativos, como o de Wall, que segmenta a imagem com base em histogramas locais e interpolação entre regiões, o método de Wellner é significativamente mais rápido, pois evita o cálculo de históricos de intensidade complexos e requer apenas uma passagem pela imagem. No entanto, o método de Wellner pode falhar em imagens com transições graduais de iluminação, onde o limiar adaptativo pode não se ajustar de maneira ideal [6].

Em resumo, o thresholding adaptativo de Wellner representa um compromisso eficiente entre velocidade e precisão, sendo amplamente utilizado em sistemas que necessitam de segmentação rápida. Uma das limitações encontradas no método de Wellner é sua dependência da ordem de varredura dos pixels, o que pode resultar em artefatos indesejados em imagens com iluminação desigual. Esse problema é particularmente evidente quando a imagem possui um gradiente de iluminação significativo, levando a resultados inconsistentes dependendo da direção de varredura utilizada. Para mitigar esse efeito, algumas variantes do método foram propostas, como a varredura alternada (boustrophedon), que processa linhas alternadas da imagem em direções opostas, reduzindo as distorções causadas pela iluminação desigual [6].

(a) *(b)*

Figura 3 – (a) Aplicação do método wellner com varredura da esquerda para direita. (b) Aplicação do método wellner com varredura da direita para esquerda

Apesar de suas limitações, suas otimizações e variantes permitem seu uso em uma ampla gama de aplicações de visão computacional, especialmente aquelas que exigem sua aplicação em ambientes microcontrolados com processamento em tempo real.

2.3. Método de imagens íntegras

O método de thresholding adaptativo baseado em imagens íntegras aprimora a segmentação proposta pelo algoritmo de Wellner. Sua inovação reside no uso da imagem integral para obter rapidamente a soma das intensidades dos pixels dentro de uma janela ao redor de cada ponto, permitindo uma adaptação local mais eficiente. Uma imagem integral, também conhecida como tabela de soma acumulada, é uma estrutura de dados que permite calcular a soma dos valores dentro de uma região retangular da imagem em tempo constante.

Para construir a imagem integral, cada pixel armazena a soma de todos os valores de intensidade dos pixels à esquerda e acima dele. Essa técnica elimina a necessidade de iterar sobre cada pixel da janela para somar suas intensidades, otimizando significativamente o processamento [6].

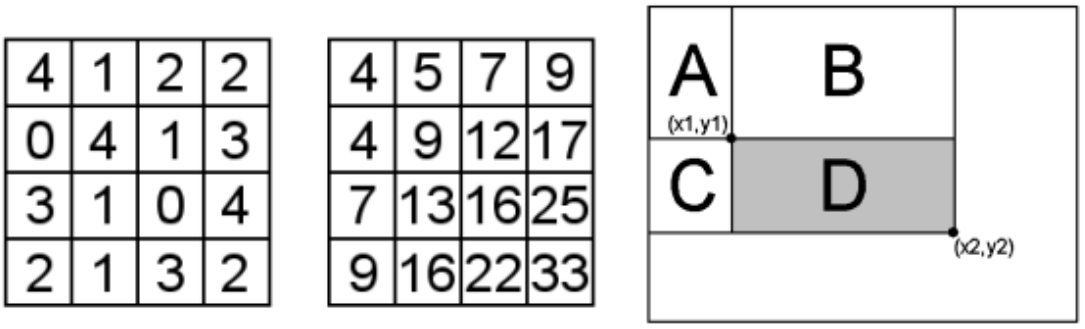


Figura 4 – Imagem explicativa da aplicação da imagem integral

Diferentemente do método original de Wellner, que emprega uma média móvel aproximada sobre os últimos pixels vistos, o método modificado calcula a média em uma janela quadrada de tamanho s ao redor de cada pixel. Isso proporciona uma distribuição mais uniforme dos pixels vizinhos, resultando em uma estimativa mais precisa da intensidade média local. A imagem integral simplifica o cálculo da média local ao eliminar a necessidade de iterar sobre cada pixel da janela. Em vez disso, permite que a soma de qualquer região retangular seja obtida em tempo constante, após uma única iteração para sua construção. [5].

O método modificado ocorre em duas etapas. Primeiro, constrói-se a imagem integral, onde cada pixel contém a soma das intensidades dos pixels à esquerda e acima dele. Em seguida, calcula-se a média da janela s × s ao redor de cada pixel usando a imagem integral e compara-se o valor do pixel atual com essa média. Se o valor do pixel for menor que uma porcentagem predefinida (t) da média, ele é classificado como preto; caso contrário, como branco. Essa abordagem garante segmentação adaptativa e eficiente, ajustando-se a variações de iluminação e contraste [5].

Embora exija duas passagens pela imagem, esse método supera as limitações do algoritmo de Wellner, oferecendo segmentação mais robusta e independente da ordem de varredura. Sua implementação é simples, eficiente para aplicações em tempo real e ideal para imagens dinâmicas, como vídeos ao vivo, onde alta precisão e velocidade são essenciais. A modificação baseada na imagem integral representa, assim, uma solução eficaz para o thresholding adaptativo em visão computacional [6].

3. Implementação

As ferramentas utilizadas no desenvolvimento do relatório foram o Matlab versão R2024b junto a biblioteca adaptthresh

Para a facilitar a implementação, os 3 métodos foram separados em 4 funções que recebem a imagem a ser transformada em formato grayscale, a sensibilidade a ser utilizada, o método estatístico (apenas para a função padrão do matlab) e o número de iterações que serão utilizadas no valor do tempo médio de execução dos métodos.

```
function img_output = matlab_adaptive_threshold(input_image, threshold_sensitivity, statistic_mode, num_iter)
    % MATLAB_ADAPTIVE_THRESHOLD - Adaptive thresholding using MATLAB's built-in function
    %
    % Parameters:
    % input_image - Grayscale input image (uint8)
    % threshold_sensitivity - Sensitivity percentage for adaptive thresholding
    % statistic_mode - Statistical method used (e.g., 'mean', 'median', 'gaussian')
    % num_iter - Number of iterations for benchmarking execution time
    %
    % Return:
    % img_output - Binary output image (logical)
```

Figura 5 – Função criada para aplicação do método padrão do matlab para utilização do adaptive threshold

```
function img_output = wellner_adaptive_threshold(input_image, window_size, threshold_sensitivity, num_iter)
    % WELLNER_ADAPTIVE_THRESHOLD - Adaptive thresholding using Wellner's method
    %
    % Parameters:
    % input_image - Grayscale input image (uint8)
    % window_size - Size of the local region for thresholding (recommended: 1/8th of image width)
    % threshold_sensitivity - Sensitivity percentage (e.g., 15 for 15%)
    % num_iter - Number of iterations for benchmarking execution time
    %
    % Return:
    % img_output - Binary output image (logical)
```

Figura 6 – Função criada para aplicação do método de Wellner para utilização do adaptive threshold

```
function img_output = integral_image_adaptive_threshold(input_image, window_size, threshold_sensitivity, num_iter)
    % INTEGRAL_IMAGE_ADAPTIVE_THRESHOLD - Adaptive thresholding using an integral image
    %
    % Parameters:
    % input_image - Grayscale input image (uint8)
    % window_size - Local window size (in pixels)
    % threshold_sensitivity - Threshold percentage (e.g., 15 for 15%)
    % num_iter - Number of iterations for benchmarking execution time
    %
    % Return:
    % img_output - Binary output image (logical)
```

Figura 7 – Função criada para aplicação do método de Wellner com imagem integral para utilização do adaptive threshold

```
function img_output = integral_image_adaptive_threshold_modified(input_image, window_size, threshold_sensitivity, num_iter)
    % INTEGRAL_IMAGE_ADAPTIVE_THRESHOLD - Adaptive thresholding using an integral image
    %
    % Parameters:
    % input_image - Grayscale input image (uint8)
    % window_size - Local window size (in pixels)
    % threshold_sensitivity - Threshold percentage (e.g., 15 for 15%)
    % num_iter - Number of iterations for benchmarking execution time
    %
    % Return:
    % img_output - Binary output image (logical)
```

Figura 8 – Função criada para aplicação do método de Wellner com imagem integral modificada para utilização do adaptive threshold

3.1. Método padrão matlab

Para esta implementação, utilizou-se como base o exemplo fornecido pelo Help Center do MATLAB [4]. A partir deste código, foram realizadas modificações para permitir a visualização do Adaptive Threshold, utilizando cálculos baseados na média, mediana e no método Gaussiano. Para cada abordagem estatística, a sensibilidade foi ajustada manualmente a fim de obter o melhor resultado possível para esta aplicação.

```
function img_output = matlab_adaptive_threshold(input_image, threshold_sensitivity, statistic_mode, num_iter)
    total_time = 0;
    for i = 1:num_iter
        tic;
        img_output = adaptthresh(input_image, threshold_sensitivity, 'Statistic', statistic_mode);
        img_output = imbinarize(input_image, img_output);
        total_time = total_time + toc;
    end
    avg_time = total_time / num_iter;
    fprintf('Tempo médio de execução (%s): %.6f s\n', statistic_mode, avg_time);
    fprintf('Tempo médio de execução (%s): %.6f ms\n', statistic_mode, avg_time*1000);

end
```

Figura 9 – Código para transformação de imagens utilizando método matlab

3.2. Método de Wellner

O código foi desenvolvido utilizando o artigo publicado por Wellner como base para o seu desenvolvimento [6]. A partir dessa referência, foi criado um algoritmo que realiza thresholding adaptativo em imagens em tons de cinza, segmentando-as em fundo e objeto com base em um threshold local para cada pixel.

A implementação começa convertendo a imagem para o formato double e aplicando um filtro gaussiano suave para reduzir o ruído. A imagem binarizada é inicializada como uma matriz lógica, com todos os pixels definidos inicialmente como fundo. Em seguida, um loop de iterações é executado para medir o tempo de execução, realizando o thresholding adaptativo.

```
input_image = double(input_image);
input_image = imgaussfilt(input_image, 1);

[rows, cols] = size(input_image);
img_output = false(rows, cols);
total_time = 0;
```

Figura 10 – Código de inicialização de variáveis

Para cada pixel, o código calcula uma média móvel local dentro de uma janela definida por window_size. Essa média é usada para calcular um threshold local, ajustado pela sensibilidade fornecida. Caso o valor do pixel seja inferior ao threshold calculado, ele é classificado como fundo, e se for superior, como objeto. Após a segmentação, a função bwareaopen é utilizada para remover objetos pequenos e isolados, refinando a segmentação.

```
for k = 1:num_iter
    tic;
    moving_avg = 127 * window_size;
    for i = 1:rows
        for j = 1:cols
            moving_avg = moving_avg - (moving_avg / window_size) + input_image(i, j);
            threshold = (moving_avg / window_size) * (100 - threshold_sensitivity) / 100;

            if input_image(i, j) < threshold
                img_output(i, j) = false;
            else
                img_output(i, j) = true;
            end
        end
    end
    img_output = bwareaopen(img_output, 10);
    total_time = total_time + toc;
end
avg_time = total_time / num_iter;
fprintf('Tempo médio de execução (Wellner): %.6f s\n', avg_time);
fprintf('Tempo médio de execução (Wellner): %.6f ms\n', avg_time*1000);
```

Figura 11 – Código de aplicação do método de Wellner

Por fim, o tempo médio de execução é calculado e a imagem binarizada resultante é retornada. O algoritmo oferece uma segmentação eficiente, ajustando dinamicamente o threshold de acordo com as variações locais da imagem.

3.3. Método de Wellner com imagens integrais

O código foi desenvolvido utilizando o artigo [5] como base em seu desenvolvimento. A partir dessa referência, foi criado um algoritmo que realiza o thresholding adaptativo utilizando imagens integrais, uma técnica eficiente para segmentação de imagens com o cálculo rápido do threshold local. O algoritmo começa convertendo a imagem para o formato double e aplicando um filtro gaussiano suave para reduzir o ruído. A imagem binarizada é inicializada com valores nulos, e uma imagem integral (integral_image) é calculada para armazenar as somas acumuladas dos pixels.

```
input_image = double(input_image);
input_image = imgaussfilt(input_image, 1);
[rows, cols] = size(input_image);
img_output = zeros(rows, cols, 'uint8');

integral_image = zeros(rows, cols);
```

Figura 12 – Código de inicialização de variáveis

O cálculo da imagem integral é feito somando cumulativamente os valores dos pixels de cada coluna e linha, permitindo obter rapidamente a soma de qualquer sub-região da imagem. Essa abordagem acelera o cálculo do threshold para cada pixel.

```
for k = 1:num_iter
    tic;
    for i = 1:cols
        sum_col = 0;
        for j = 1:rows
            sum_col = sum_col + input_image(j, i);
            if i == 1
                integral_image(j, i) = sum_col;
            else
                integral_image(j, i) = integral_image(j, i - 1) + sum_col;
            end
        end
    end

    half_s = floor(window_size / 2);
```

Figura 13 – Código da imagem integral

Com a imagem integral calculada, o algoritmo percorre a imagem e ajusta o threshold de acordo com a soma dos valores dentro de uma janela de tamanho definido por window_size. O valor do threshold é ajustado pela sensibilidade fornecida, e se a intensidade do pixel multiplicada pela área da janela for menor que a soma dos valores da janela ajustada, o pixel é classificado como fundo (preto); caso contrário, como objeto (branco). Após a segmentação, a função bwareaopen é utilizada para remover pequenos objetos isolados, refinando a binarização.

```
    half_s = floor(window_size / 2);
    for i = 1:cols
        for j = 1:rows
            x1 = max(i - half_s, 1);
            x2 = min(i + half_s, cols);
            y1 = max(j - half_s, 1);
            y2 = min(j + half_s, rows);

            count = (x2 - x1 + 1) * (y2 - y1 + 1);
            sum_window = integral_image(y2, x2);

            if x1 > 1
                sum_window = sum_window - integral_image(y2, x1 - 1);
            end
            if y1 > 1
                sum_window = sum_window - integral_image(y1 - 1, x2);
            end
            if x1 > 1 && y1 > 1
                sum_window = sum_window + integral_image(y1 - 1, x1 - 1);
            end

            if input_image(j, i) * count <= (sum_window * (100 - threshold_sensitivity) / 100)
                img_output(j, i) = 0;
            else
                img_output(j, i) = 255;
            end
        end
    end
    img_output = bwareaopen(img_output, 10);
    total_time = total_time + toc;
end
avg_time = total_time / num_iter;
fprintf('Tempo médio de execução (Integral Image): %.6f s\n', avg_time);
fprintf('Tempo médio de execução (Integral Image): %.6f ms\n', avg_time*1000);
```

Figura 14 – Código de aplicação do método de Wellner com imagem integral

Por fim, o tempo médio de execução é calculado após várias iterações e a imagem binarizada é retornada como saída. O uso da imagem integral permite que o processo de thresholding adaptativo seja realizado de maneira eficiente, reduzindo o tempo de cálculo e melhorando o desempenho, especialmente para imagens maiores.

3.4. Método de Wellner com imagens integrais modificada

Este método modificado tem como objetivo acelerar a criação da imagem integral e melhorar a eficiência do processo de thresholding adaptativo. A implementação foi otimizada ao utilizar a função cumsum para calcular a imagem integral de forma mais eficiente, ao invés de calcular a soma acumulada de cada pixel manualmente.

A implementação começa convertendo a imagem para o formato double e aplicando um filtro gaussiano suave para reduzir o ruído. A variável img_output é inicializada como uma matriz lógica, com todos os pixels inicialmente classificados como fundo.

```
input_image = double(input_image);
input_image = imgaussfilt(input_image, 1);
[rows, cols] = size(input_image);
half_s = floor(window_size / 2);
img_output = false(rows, cols);

total_time = 0;
for k = 1:num_iter
    tic;
    integral_image = cumsum(cumsum(input_image, 1), 2);
```

Figura 15 – Código de inicialização de variáveis e criação da imagem integral

O cálculo da imagem integral é realizado de maneira eficiente usando a função cumsum, que calcula a soma acumulada das intensidades dos pixels de forma rápida, primeiro na direção das linhas e depois nas colunas. Esse cálculo permite acessar rapidamente a soma de qualquer sub-região da imagem, essencial para o thresholding adaptativo.

Para cada pixel da imagem, o código calcula a soma dos valores dentro de uma janela de tamanho definido por window_size e determina o valor do threshold adaptativo com base na média dos valores na janela. O valor do pixel é comparado com o threshold ajustado pela sensibilidade, e se for maior, o pixel é classificado como objeto (branco); caso contrário, é classificado como fundo (preto). Após a segmentação, a função bwareaopen é aplicada para remover pequenos objetos isolados, refinando a segmentação.

```
        for i = 1:rows
            for j = 1:cols

                x1 = max(i - half_s, 1);
                x2 = min(i + half_s, rows);
                y1 = max(j - half_s, 1);
                y2 = min(j + half_s, cols);
                sum_window = integral_image(x2, y2);
                if x1 > 1
                    sum_window = sum_window - integral_image(x1 - 1, y2);
                end
                if y1 > 1
                    sum_window = sum_window - integral_image(x2, y1 - 1);
                end
                if x1 > 1 && y1 > 1
                    sum_window = sum_window + integral_image(x1 - 1, y1 - 1);
                end

                count = (x2 - x1 + 1) * (y2 - y1 + 1);
                avg = sum_window / count;
                img_output(i, j) = input_image(i, j) > (avg * (1 - threshold_sensitivity / 100));
            end
        end
        img_output = bwareaopen(img_output, 10);
        total_time = total_time + toc;
    end
    avg_time = total_time / num_iter;
    fprintf('Tempo médio de execução (Integral Image Modified): %.6f s\n', avg_time);
    fprintf('Tempo médio de execução (Integral Image Modified): %.6f ms\n', avg_time*1000);
end
```

Figura 16 – Código de aplicação do método de Wellmer utilizando a imagem integral modificada

Por fim, o tempo médio de execução é calculado após várias iterações, e a imagem binarizada resultante é retornada. A otimização do cálculo da imagem integral, utilizando a função cumsum, torna a operação mais eficiente, acelerando a execução do algoritmo e permitindo o processamento de imagens maiores em menos tempo.

3.5. Geração das imagens

A execução das funções foi concentrada em três scripts, com o objetivo de carregar a imagem desejada, chamar as funções de transformação e exibir os resultados na tela. Esses scripts são o Jornal_Test.m, QR_Test.m e Rice_Test.m. Cada um deles utiliza uma imagem distinta, facilitando a avaliação dos algoritmos desenvolvidos.

```
input_image = imread('rice.png');

% Convert to grayscale if necessary
if size(input_image, 3) == 3
    input_image = rgb2gray(input_image);
end

original_image = input_image;

img_output_median = matlab_adaptive_threshold(input_image, 0.3, 'median', 1);
img_output_median = uint8(img_output_median * 255);

img_output_mean = matlab_adaptive_threshold(input_image, 0.3, 'mean', 50);
img_output_mean = uint8(img_output_mean * 255);

img_output_gaussian = matlab_adaptive_threshold(input_image, 0.5, 'gaussian', 50);
img_output_gaussian = uint8(img_output_gaussian * 255);


img_output_wellner = wellner_adaptive_threshold(input_image, round(size(input_image, 2) / 8), 0.1, 50);
img_output_wellner = uint8(img_output_wellner*255);


img_output_integral = integral_image_adaptive_threshold(input_image, round(size(input_image, 2) / 8), 2, 50);
img_output_integral = uint8(img_output_integral*255);

img_output_integral_modified = integral_image_adaptive_threshold_modified(input_image, round(size(input_image, 2) / 8), 2, 50);
img_output_integral_modified = uint8(img_output_integral_modified*255);
```

Figura 17 – Código de carregamento da imagem e chamada das funções de transformação

A exibição das imagens foi realizada utilizando a função subplot

```
figure;
subplot(2,2,1); imshow(original_image);                 title('Imagem original');
subplot(2,2,2); imshow(img_output_mean);                title('Matlab mean');
subplot(2,2,3); imshow(img_output_median);              title('Matlab median');
subplot(2,2,4); imshow(img_output_gaussian);            title('Matlab gaussian');

figure;
subplot(2,2,1); imshow(original_image);                 title('Imagem original');
subplot(2,2,2); imshow(img_output_wellner);             title('Wellner');
subplot(2,2,3); imshow(img_output_integral);            title('Integral image');
subplot(2,2,4); imshow(img_output_integral_modified);   title('Modified integral image');

figure;
subplot(2,3,1); imshow(img_output_mean);                title('Matlab mean');
subplot(2,3,2); imshow(img_output_median);              title('Matlab median');
subplot(2,3,3); imshow(img_output_gaussian);            title('Matlab gaussian');
subplot(2,3,4); imshow(img_output_wellner);             title('Wellner');
subplot(2,3,5); imshow(img_output_integral);            title('Integral image');
subplot(2,3,6); imshow(img_output_integral_modified);   title('Modified integral image');
```

Figura 18 – Código de plotagem dos dados resultantes

4. Resultados e conclusões

4.1 Imagens obtidas utilizando teste QR codes

Resultados obtidos utilizando a imagem de uma série de códigos QR, imagem obtida no artigo publicado [5]

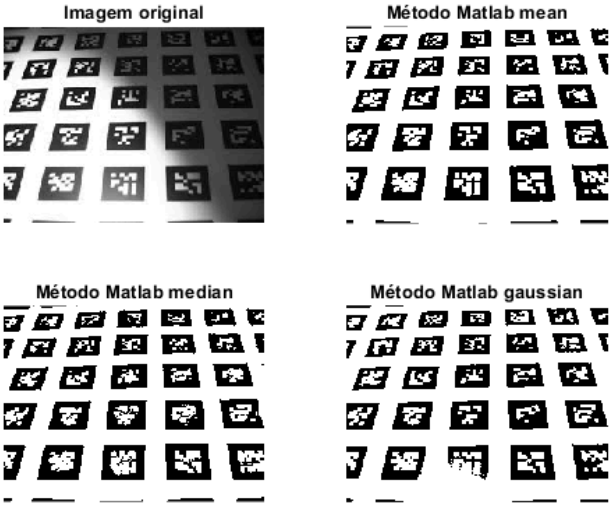


Figura 19 – (Imagem Original) Imagem obtida no artigo [5] (Método Matlab mean) Imagem ebitda ao aplicar o método mean do matlab a imagem original (Método Matlab median) Imagem ebitda ao aplicar o método median do matlab a imagem original (Método Matlab gaussian) Imagem ebitda ao aplicar o método gaussian do matlab a imagem original

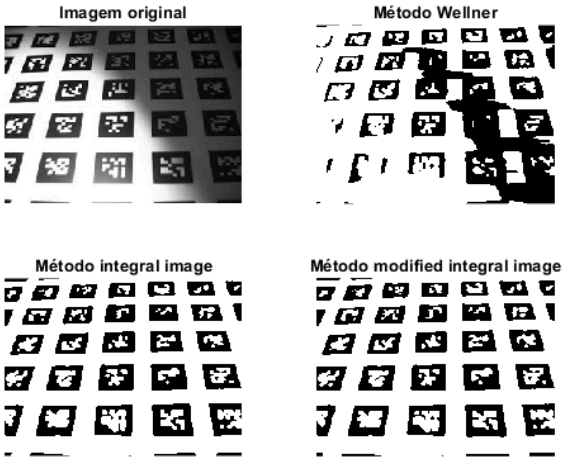


Figura 20 – (Imagem Original) Imagem obtida no artigo [5] (Método Wellner) Imagem ebitda ao aplicar o método Wellner a imagem original (Método Integral image) Imagem ebitda ao aplicar o método Wellner com imagem integral a imagem original (Método modified Integral image) Imagem ebitda ao aplicar o método Wellner com imagem integral modificada a imagem original

4.2 Imagens obtidas utilizando teste rice.png

Resultados obtidos utilizando a imagem rice.png incluída nativamente ao Matlab

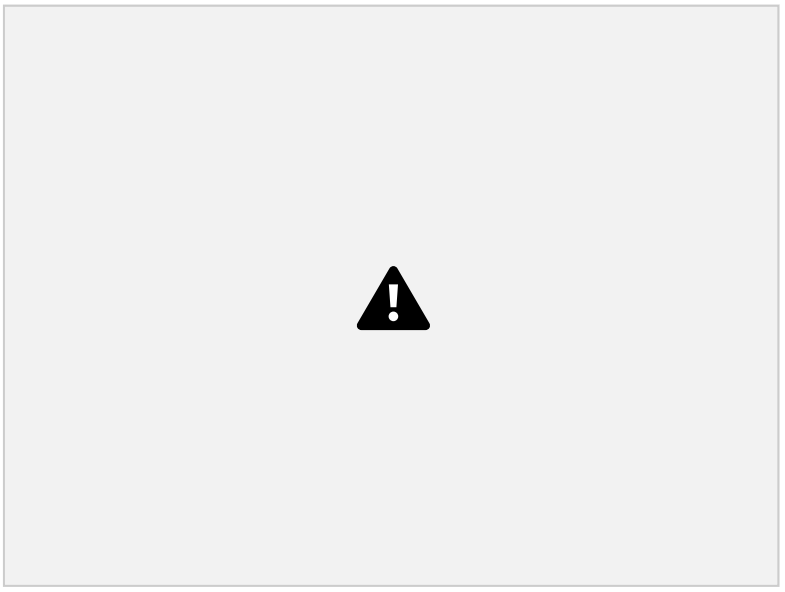


Figura 21 – (Imagem Original) Imagem 'rice.png' obtida no Matlab (Método Matlab mean) Imagem ebitda ao aplicar o método mean do matlab a imagem original (Método Matlab median) Imagem ebitda ao aplicar o método median do matlab a imagem original (Método Matlab gaussian) Imagem ebitda ao aplicar o método gaussian do matlab a imagem original

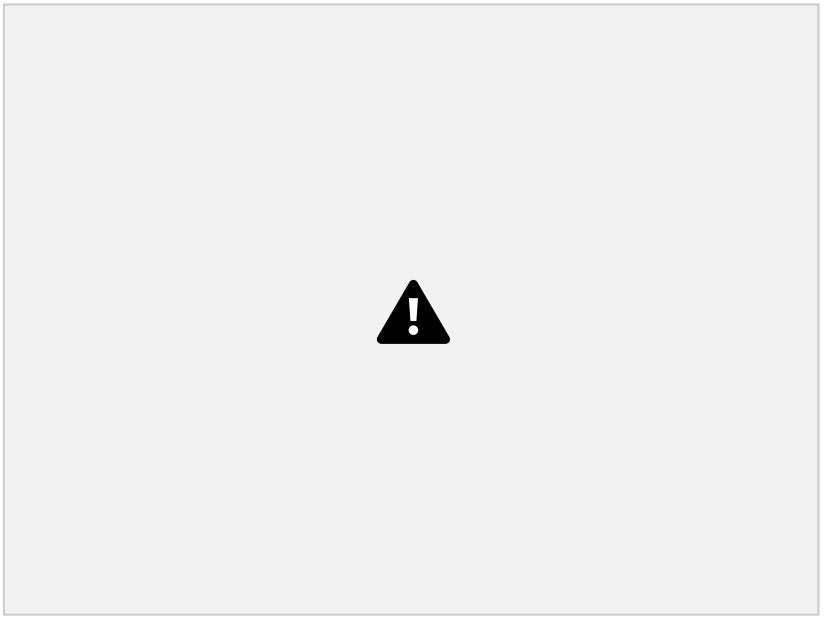


Figura 22 – (Imagem Original) Imagem 'rice.png' obtida no Matlab (Método Wellner) Imagem ebitda ao aplicar o método Wellner a imagem original (Método Integral image) Imagem ebitda ao aplicar o método Wellner com imagem integral a imagem original (Método modified Integral image) Imagem ebitda ao aplicar o método Wellner com imagem integral modificada a imagem original

4.3 Análise de eficiência

Para conseguir avaliar o tempo de execução de cada algoritmo, os valores médios de execução obtidos na execução a partir das funções de transformação, incluídos em uma planilha do excel e então plotados como gráfico de barras

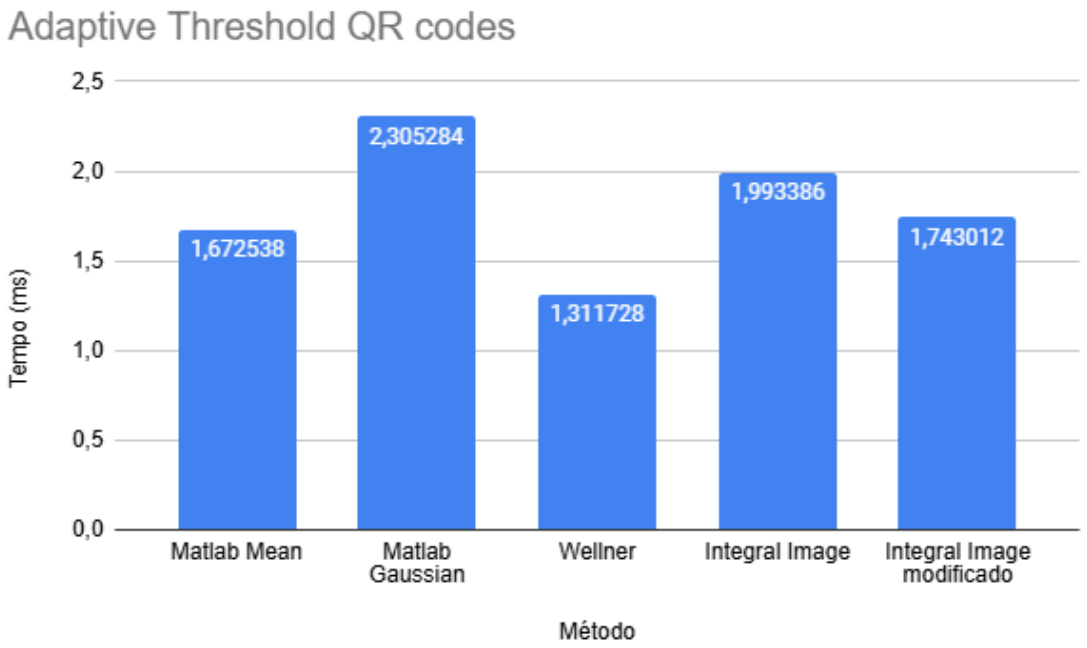


Figura 23 – Gráfico de tempo de execução imágem código QR obtida no artigo [5]

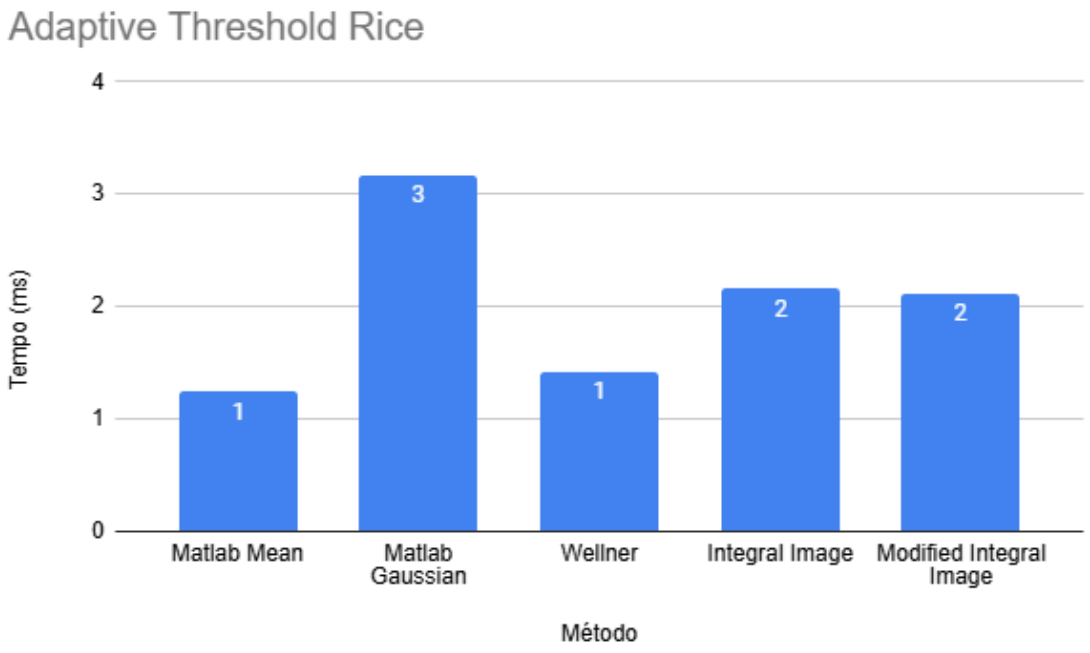


Figura 24 – Gráfico de tempo de execução da imagem 'rice.png' obtida no Matlab

4.4 Comentários finais

A técnica utilizada pelo método de Wellner se mostrou uma alternativa eficaz quando a prioridade é a velocidade de execução e as imagens não apresentam grandes variações de iluminação. No entanto, para contornar essa limitação, a introdução da técnica de imagens integrais trouxe uma melhoria significativa na qualidade da imagem, pois eliminou a dependência da ordem de varredura dos pixels e garantiu uma boa qualidade, mesmo em condições de iluminação inadequadas. Além disso, a modificação proposta na criação da imagem integral demonstrou ser uma melhoria quando comparada à abordagem manual. Por fim, foi possível validar que a implementação padrão do Matlab se configura como uma alternativa eficaz para a aplicação de thresholding adaptativo, oferecendo resultados de alta fidelidade.

Com base nos resultados, concluímos que o uso de thresholding adaptativo, especialmente com o auxílio de imagens integrais, é uma técnica robusta para segmentação de imagens em condições variadas de iluminação e contraste.

Referências

[1] Chaudhary, V. “Adaptive Thresholding.” Naukri Code 360 Library. Disponível em: https://www.naukri.com/code360/library/adaptive-thresholding. Acesso em: 10 fev. 2024.

[2] Fisher, R., Perkins, S., Walker, A., Wolfart, E. “Adaptive Thresholding.” HIPR2 – Hypermedia Image Processing Reference. University of Edinburgh, 2003. Disponível em: https://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm. Acesso em: 10 fev. 2024.

[3] Buhl, N. “Image Thresholding in Image Processing.” Encord Blog, 12 set. 2023. Disponível em: https://encord.com/blog/image-thresholding-image-processing/. Acesso em: 10 fev. 2024.

[4] MathWorks. “Adaptive Image Threshold Using Local First-Order Statistics (adaptthresh).” MATLAB Documentation. Disponível em: https://www.mathworks.com/help/images/ref/adaptthresh.html. Acesso em: 10 fev. 2024..

[5] Bradley, D., Roth, G. “Adaptive Thresholding Using the Integral Image.” Carleton University, Canada; National Research Council of Canada.

[6] P.D. Wellner, Adaptive thresholding for the digitaldesk, Technical Report EPC-1993–110, Rank Xerox Ltd, Cambridge, UK, 1993.