Trabalho 3

Tiago de Paula Alves (187679)
tiagodepalves@gmail.com

31 de março de 2025

1 Introdução

Neste trabalho foram implementadas técnicas de limiarização de imagens monocromáricas. Esses processamento são comumente utilizados para destacar objetos na imagem, separando-os do fundo. Essa técnicas marcam o objeto encontrado como preto na imagem e o fundo como branco, por isso elas também são chamadas de técnicas de binarização.

Os oitos métodos de limiarização do trabalho envolvem limites em em toda a imagem, no método global, ou em apenas uma vizinhança do pixel, nos métodos locais. As técnicas foram comparadas em sete imagens distintas, apresentadas na seção 3.2, e vários parâmetros de controle de cada método, descritos junto com o método na seção 4.

2 O Programa

O corpo do programa foi desenvolvido em Python 3.6+, utilizando a biblioteca padrão em conjunto com as bibliotecas OpenCV, para leitura e escrita das imagens, NumPy e SciPy, para as técnicas de limiarização local.

Em especial, a limiarização local foi implementada com a função `generic_filter`[1] do SciPy. Para acelerar um pouco o processamento, as funções de cálculo do limiar que são passadas como argumento da `generic_filter` foram implementadas em C (padrão GNU11) e requerem o compilador GCC 4.7+ na primeira execução do programa.

2.1 Código Fonte

O código fonte foi separado nos seguintes arquivos, dentro da pasta `limiarizacao`:

__main__.py Processamento os comandos e as opções da linha de comando.

metodos Pacote interno com as operações de limiarização.

- **metodos/__init__.py** Classes e objetos para as técnicas de limiarização e tratamento dos parâmetros de cada técnica, pelos argumentos da linha de comando.
- **metodos/locais.py** Wrapper para a chamada das funções em C.
- **metodos/locais.c** Funções de cálculo do limiar da vizinhança do pixel.
- **metodos/ops.c** Operações de mínimo, máximo, média, desvio padrão e mediana.

inout.py Tratamento de entrada e saída do programa.

tipos.py Tipos para checagem estática com MyPy.

Além disso, existem outros arquivos junto com o código fonte. A pasta `imagens` contém as figuras base utilizadas neste relatório, apresentadas na seção 3.2. Foi disponibilizado também um *script* em `bash`, `run.sh`, que realiza todos os processamentos apresentados ao longo do relatório.

[1] *scipy.ndimage.generic_filter*. URL: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.generic_filter.html`.

2.2 Execução

A execução deve ser feita através do interpretador de Python, com as seguintes entradas obrigatórias: o caminho para a imagem PGM que será processada e o método de limiarização a ser aplicado. Ao final da execução, a imagem resultante será exibida na tela.

Os argumentos opicionais podem ser vistos com `$ python3 limiarizacao --help`. A primeira opção é `--output`, ou `-o`, que salva o resultado em um arquivo PNG ou PGM em vez de exibir na tela. Se é desejável tanto a exibição da imagem quanto o salvamento no arquivo, o argumento `--force-show` ou `-f` pode ser usado.

O método de limiarização deve vir após essas opções e pode ser uma das seguintes chaves: `global`, `bernsen`, `niblack`, `sauvola`, `phansalkar`, `contraste`, `media` ou `mediana`. Os parâmetros de cada método pode ser visto com `$ python3 limiarizacao METODO --help`, onde `METODO` é a técnica desejada.

Por exemplo, o comando abaixo apresenta executa o método de Bernsen com vizinhança de raio 6 em uma imagem `entrada.pgm`, salvando depois o resultado em `saida.png`.

```
$ python3 limiarizacao entrada.pgm -o saida.png bernsen -r 6
```

3 Implementação

3.1 Técnicas de Limiarização

A limiarização aplicada aqui se baseia em encontrar um limiar T, escolhendo cada pixel como objeto se o valor dele for maior que T ou como fundo, no caso contrário. Nesse trabalho, os pixels do objeto foram coloridos em preto e o fundo em branco.

Os métodos globais em geral escolhem um mesmo limiar T para toda a imagem, mas o método `global` implementado aqui escolhe, por padrão, T como o valor médio na imagem. Isso pode ser alterado com a *flag* `-T` na linha de comando, mudando o limiar global para o valor dado.

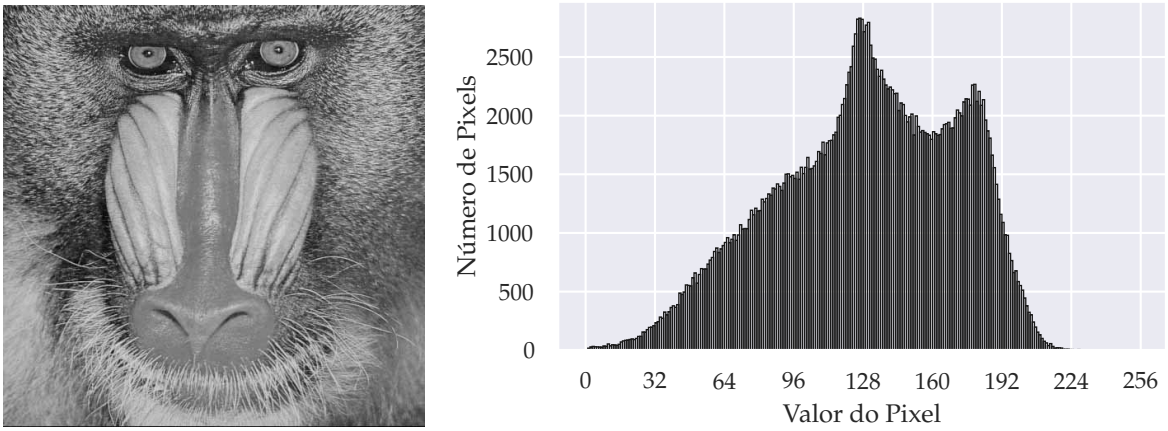
Por outro lado, os métodos locais escolhem um limiar $T(x, y)$ para cada pixel (x, y) aplicando uma função na vizinhança do pixel. Essa vizinhança é escolhida a partir de um raio $r = 10$, que pode ser alterado na linha de comando (opção `-r`). Existe a opção também de alterar a medida de distância, usando vizinhança-4, com a *flag* `-v4`, em vez da vizinhança-8, que é o padrão. Para o método `phansalkar`, foi necessário também normalizar a imagem para o intervalo $[0, 1]$ antes do processamento.

Todos os métodos locais foram implementados em C, como discutido anteriormente, para uso com a função `scipy.ndimage.generic_filter`. As bordas da imagem são extendidas com a opção `"mirror"`, que reflete os pixels mais próximos da fronteira, mas sem repetir o pixel exatamente na fronteira.

3.2 Imagens Base

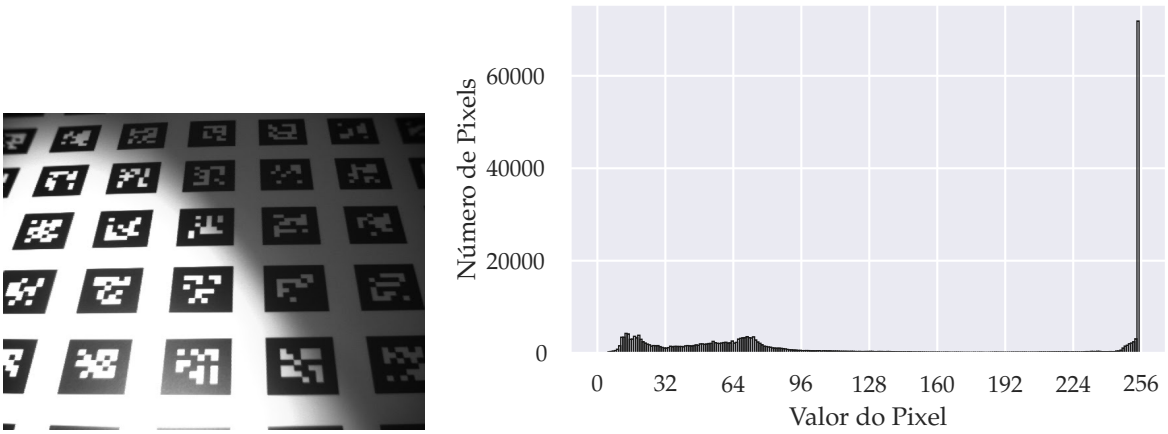
A seguir temos as sete imagens usadas como base de comparação dos métodos de limiarização. As imagens podem ser encontradas na pasta `imagens` do código fonte, a partir do nome apresentado no rótulo de cada uma.

Para cada imagem também está apresentado o histograma dos pixels, mostrando a quantidade de vezes que cada valor de pixel aparece na imagem. Podemos ver de forma geral nos histogramas o quão escura uma imagem é em média e o quão distribuído são os valores. Isso pode ajudar algumas separações de objetos, que aparecem como picos no histograma, como acontece na `peppers.pgm` (figura 4).



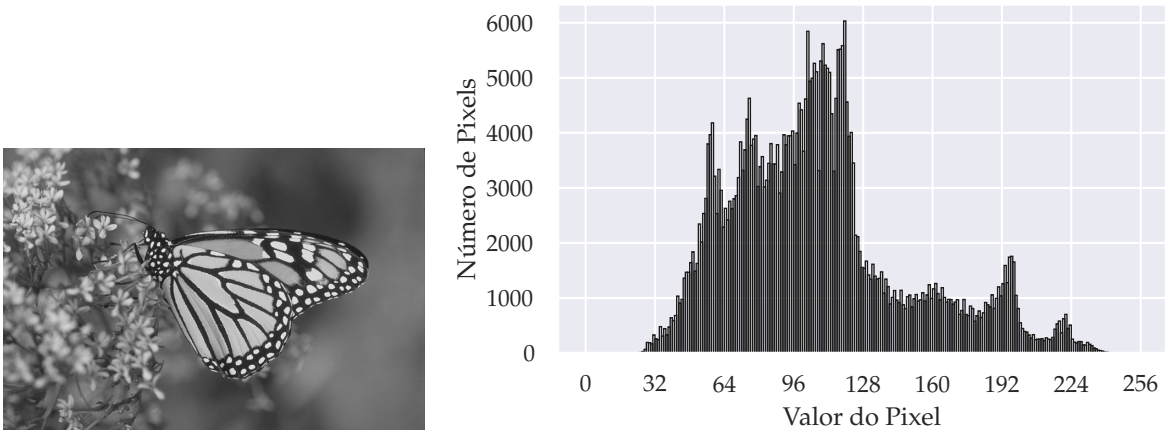
(a) Imagem. (b) Histograma.

Figura 1: `imagens/baboon.pgm`



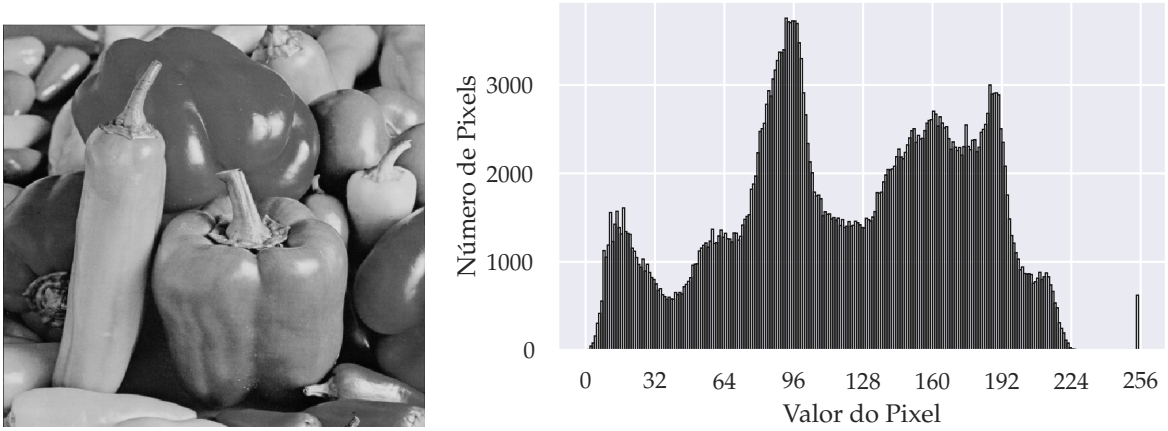
(a) Imagem. (b) Histograma.

Figura 2: `imagens/fiducial.pgm`



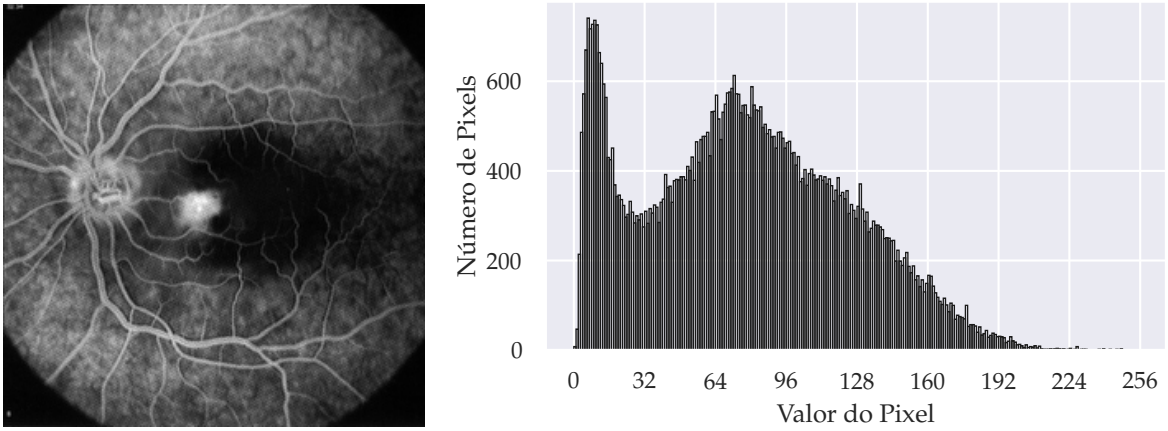
(a) Imagem. (b) Histograma.

Figura 3: `imagens/monarch.pgm`



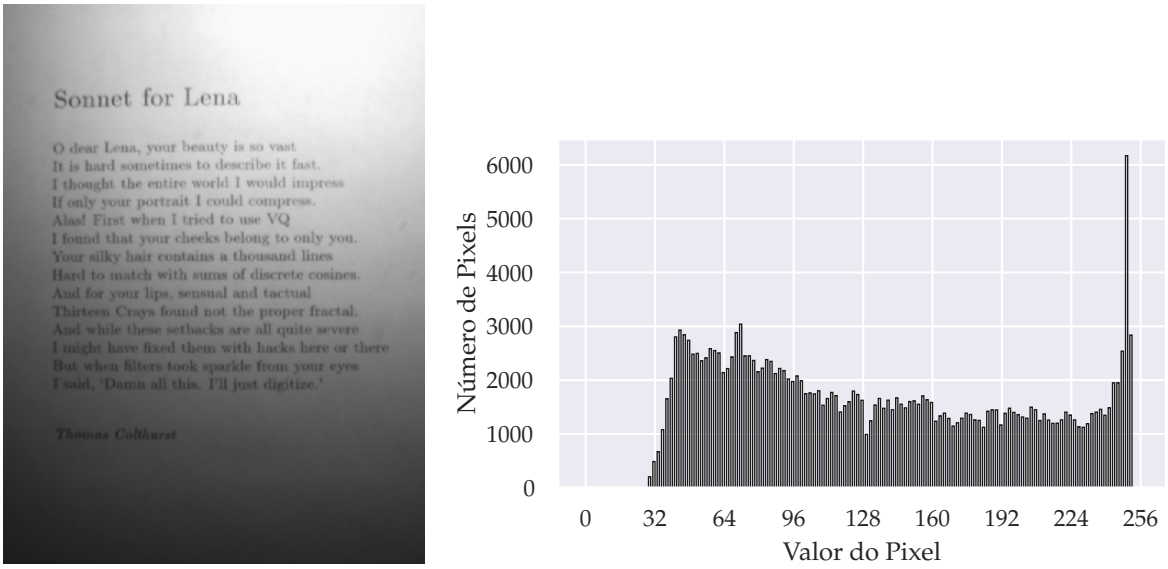
(a) Imagem. (b) Histograma.

Figura 4: `imagens/peppers.pgm`



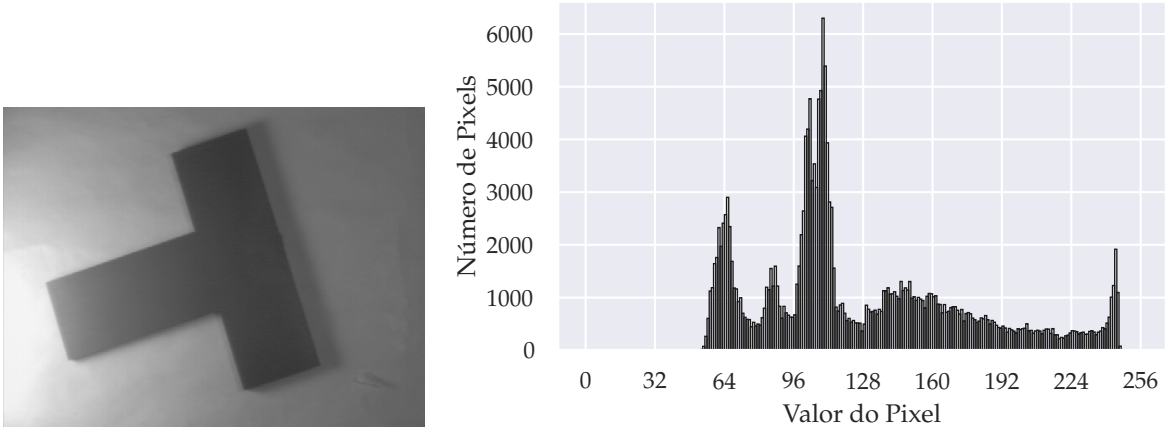
(a) Imagem. (b) Histograma.

Figura 5: `imagens/retina.pgm`



(a) Imagem. (b) Histograma.

Figura 6: `imagens/sonnet.pgm`



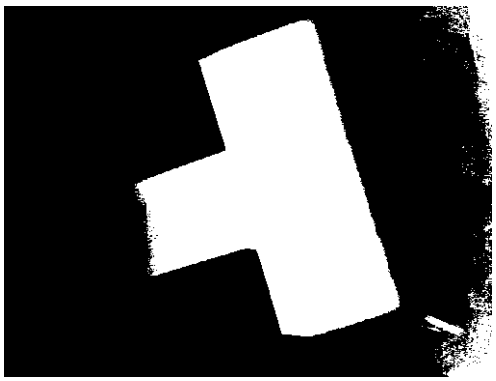
(a) Imagem. (b) Histograma.

Figura 7: `imagens/wedge.pgm`

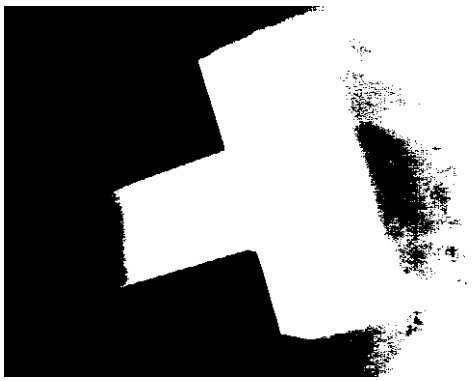
4 Resultados

4.1 Método Global

(a) `retina.pgm` com $T = 128$. 15% de pixels pretos.



(b) `wedge.pgm` com $T = 100$. 72% de pixels pretos.



(c) `wedge.pgm` com $T = 110$. 49% de pixels pretos.

Figura 8: Exemplos de aplicação do método `global`.

Esse é o método mais simples, definindo um limiar fixo T para toda a imagem. Ele funciona razoavelmente bem para imagens com iluminação regular, como pode ser visto na figura 8a.

Por outro lado, figuras com iluminação mais natural ou irregular podem ter uma grande variação da intensidade dos pixels ao longo do objeto. Isso faz com a separação do objeto de forma global seja praticamente impossível, levando a problemas como os das figuras 8b e 8c.

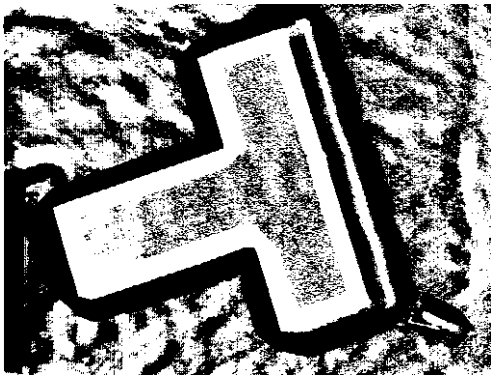
4.2 Método de Bernsen

Para o método de Bernsen, o limiar é calculado como a média da menor (z_{min}) e da maior ($z_{\text{máx}}$) intensidade na vizinhança de raio r, ou seja, $T(x,y) = (z_{\text{min}} + z_{\text{máx}})\ /2$. Esse método funciona melhor com imagens com alta variação de intensidade como as figuras 9a e 9b.

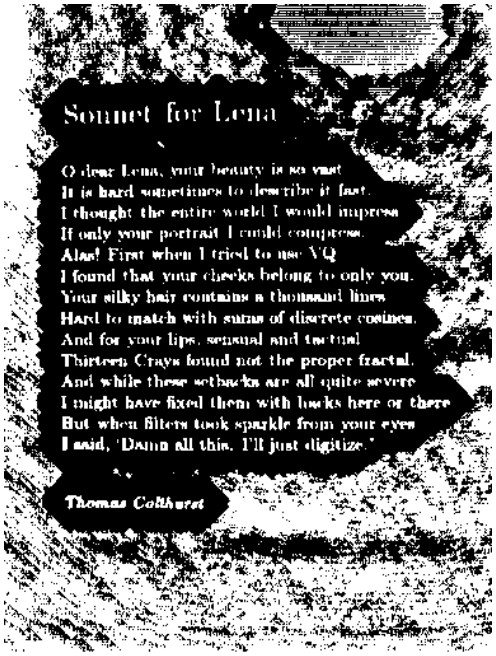
Para regiões de baixa variação de intensidade, como acontece nas imagens `wedge.pgm` e `sonnet.pgm`, o método de Bernsen gera artefatos inesperados, seguindo o gradiente da imagem. No entanto, o método funciona bem para evidenciar bordas dos objetos, como pode ser visto em 9c e 9d.

(a) `baboon.pgm` com $r = 50$.
57% de pixels pretos.

(b) `peppers.pgm` com $r = 50$.
53% de pixels pretos.



(c) `wedge.pgm` com $r = 20$.
49% de pixels pretos.



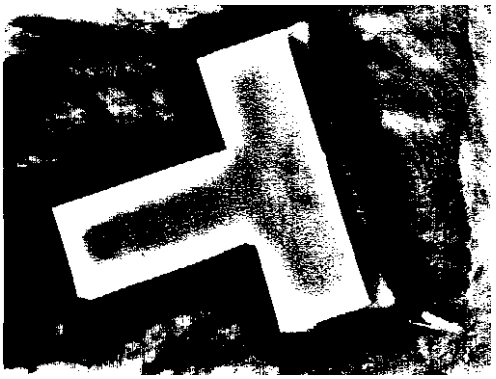
(d) `sonnet.pgm` com $r = 20$.
61% de pixels pretos.

Figura 9: Exemplos de aplicação do método `bernsen`.

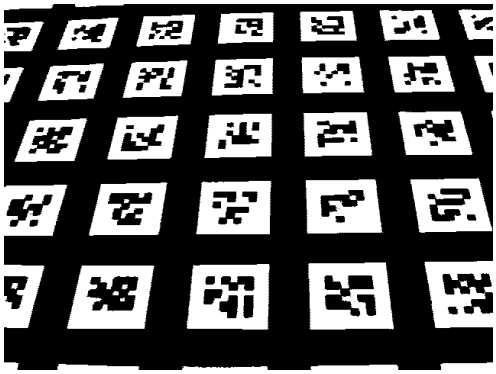
4.3 Método de Niblack

O limiar desse método é encontrado pela média da vizinhança $\mu(x,y)$, com mais uma parcela k do desvio padrão $\sigma(x,t)$, de forma que $T(x,y) = \mu(x,y) + k\sigma(x,y)$. Isso faz com o que o modelo seja mais robusto para reduzir o ruído, especialmente em regiões de baixa variação de intensidade. Podemos ver isso nas imagens figura 10a e figura 10c.

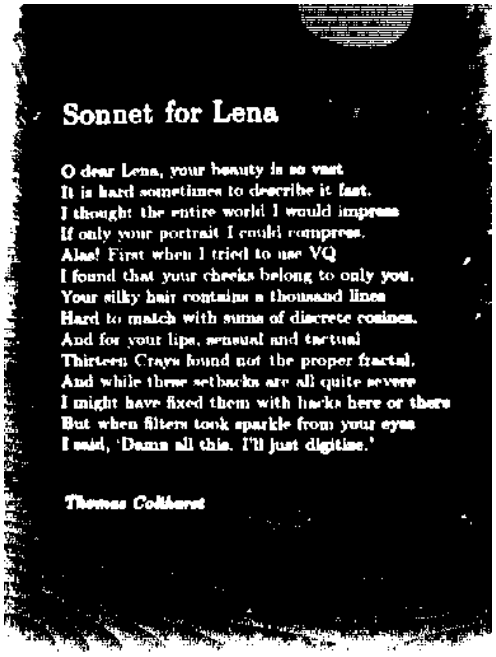
Por outro lado, o método pode acabar reduzindo os detalhes, no caso da vizinhança sem muito grande. Isso fica visível na diferença das letras do soneto da figura 10c e da 10d. Entretanto, para imagens com detalhes maiores, como a figura 10b, o método de Niblack pode funcionar bem.



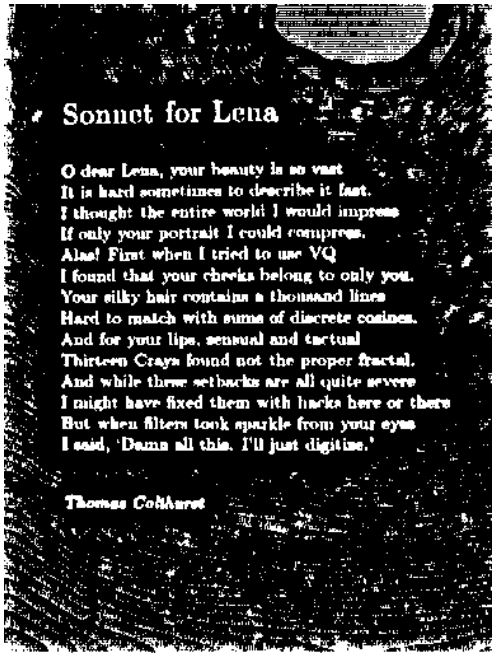
(a) `wedge.pgm` com $r = 50$ e $k = -0.3$.
71% de pixels pretos.



(b) `fiducial.pgm` com $r = 50$ e $k = -0.5$.
67% de pixels pretos.



(c) `sonnet.pgm` com $r = 50$ e $k = -0.5$.
86% de pixels pretos.



(d) `sonnet.pgm` com $r = 20$ e $k = -0.5$.
84% de pixels pretos.

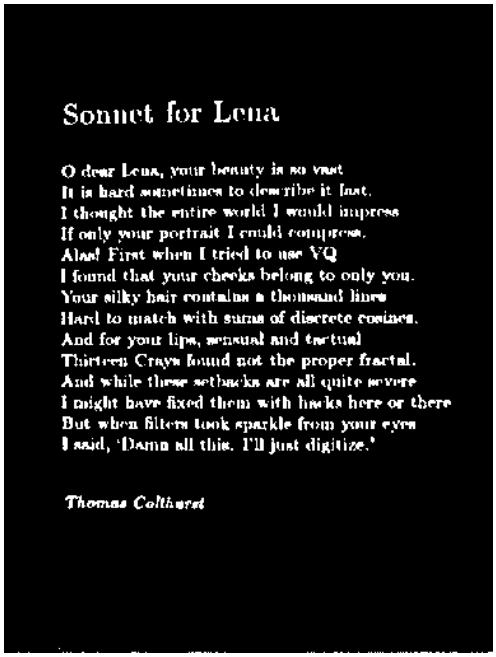
Figura 10: Exemplos de aplicação do método `niblack`.

4.4 Método de Sauvola e Pietikäinen

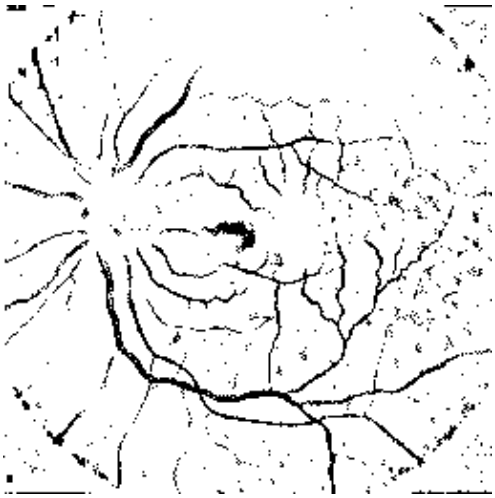
Assim como no método de Niblack, esse método se baseia na média e desvio padrão dos pixels da vizinhança, definindo o limiar como $T(x,y) = \mu(x,y)\left[1 + k\left(\frac{\sigma(x,y)}{R} - 1\right)\right]$. Essa fórmula faz o método funcione melhor com os anteriores em regiões de iluminação irregular, já que a baixa variância da região mantém o limiar da figura baixo.

Por exemplo, a figura 11a consegue reduzir ainda mais o ruído, mas ao mesmo tempo manter mais detalhes do que na figura 10. Isso era esperado, já que o método foi desenvolvido principalmente para o tratamento de imagens textuais.

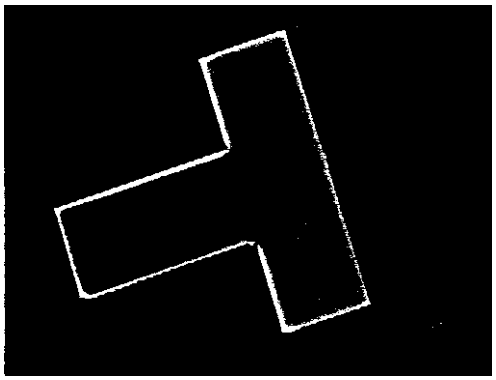
Apesar disso, o método ainda não consegue separar objetos em imagens rica em detalhes, como a `monarch.pgm` (figura 11d). Em imagens como a `retina.pgm`, ele consegue remover as manchas que existiam na figura 8, mas aumenta o ruído na imagem.



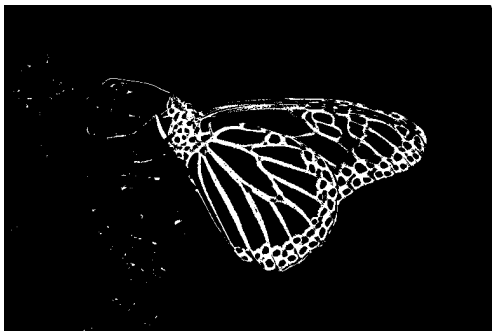
(a) `sonnet.pgm` com $k = 0.08$ e $R = 100$. 93% de pixels pretos.



(b) `retina.pgm` com $k = -0.5$ e $R = 100$. 7% de pixels pretos.



(c) `wedge.pgm` com $k = 0.08$ e $R = 200$. 98% de pixels pretos.



(d) `monarch.pgm` com $k = 0.5$ e $R = 300$. 94% de pixels pretos.

Figura 11: Exemplos de aplicação do método `sauvola` com $r = 10$.

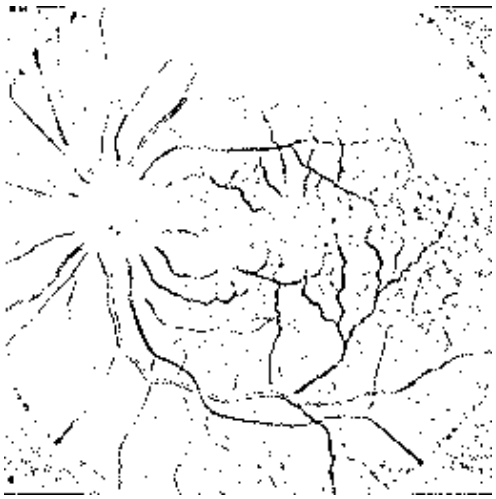
4.5 Método de Phansalkar, More e Sabale

O método desenvolvido por Phansalkar et al. teve forte influência do método de Sauvola, apresentado na seção anterior. A principal diferença é um fator exponencial na fórmula, o que faz com que o limiar não caia muito rapidamente em regiões de baixa intensidade. Essa técnica foi pensanda com foco em imagens celulares, com tratamento para cores e usando a intensidade normalizada em $[0, 1]$. Nessa técnica, a fórmula do limiar é dada por:

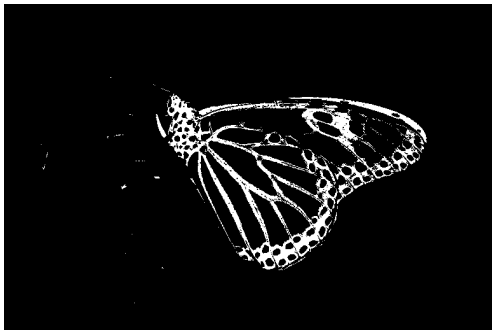
$$T(x,y) = \mu(x,y)\left[1 + pe^{-q\mu(x,y)} + k\left(\frac{\sigma(x,y)}{R} - 1\right)\right]$$

Podemos ver que o método consegue reduzir a força da borda da retina na figura 12a,

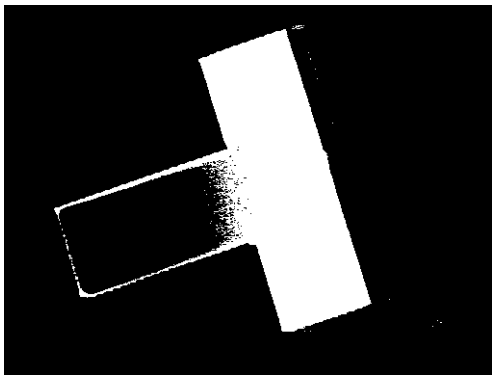
mas mantendo os vasos sanguíneos. Além disso, na `monarch.pgm`, a borboleta consegue ser mantida removendo ainda boa parte da folhagem, mas alguns detalhes são perdido. Esse método consegue também começar a marcar o centro do objeto na `wedge.pgm` (figura 12c).



(a) `retina.pgm` com $r = 5$, $k = 0.2$, $R = 4$, $p = 0.2$ e $q = 0.5$. 5% de pixels pretos.



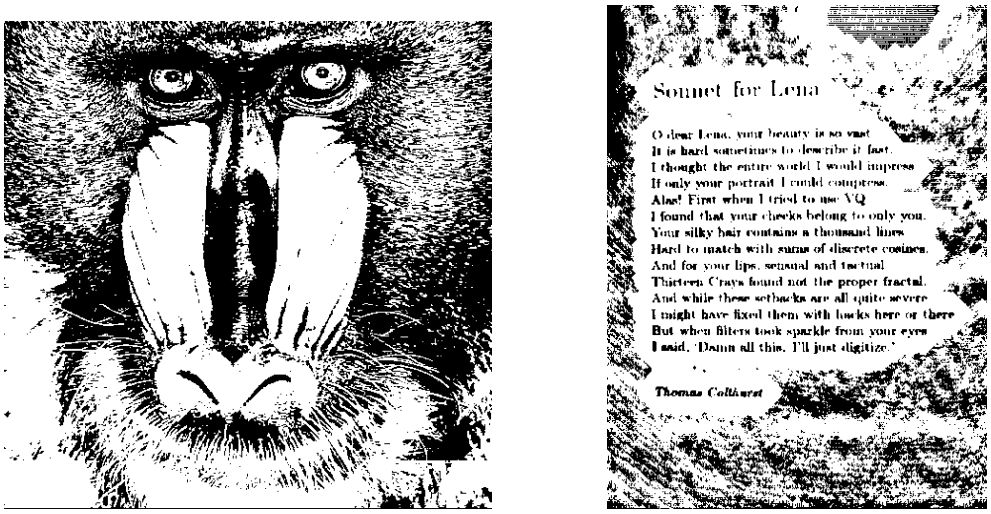
(b) `monarch.pgm` com $r = 10$, $k = 0.9$, $R = 0.7$, $p = 5$ e $q = 10$.
95% de pixels pretos.



(c) `wedge.pgm` com $r = 10$, $k = 0.1$, $R = 0.9$, $p = 12$ e $q = 16$.
83% de pixels pretos.

Figura 12: Exemplos de aplicação do método `phansalkar`.

4.6 Método do Contraste



(a) `baboon.pgm` com $r = 50$.
42% de pixels pretos.

(b) `sonnet.pgm` com $r = 20$.
26% de pixels pretos.

Figura 13: Exemplos de aplicação do método `contraste`.

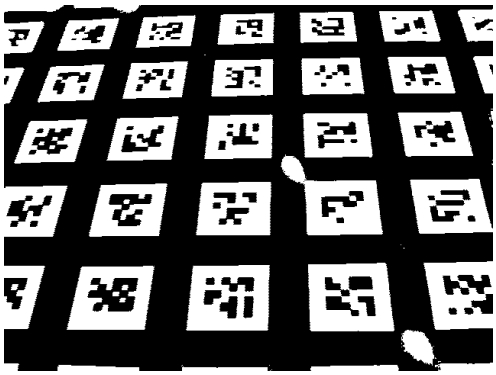
Esse método define objeto como pixels mais próximos do mínimo local do que do máximo, deixando o restante como fundo. Na prática, ele acaba funcionando como o método de Bernsen com o resultado invertido, por isso, o que foi discutido na seção 4.2 também vale aqui.

No entanto, note que as porcentagens do `sonnet.pgm` (figuras 9d e 13b) não soman 100%. Isso se deve aos pixels que se encontram exatamente na média da sua vizinhança e acaba sumindo em raios maiores, como da figura 13a.

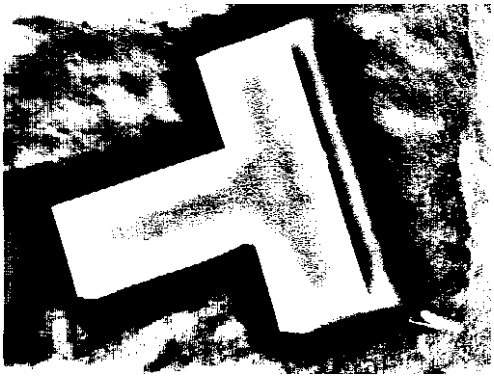
4.7 Método da Média

O método da média define o limiar apenas como a média da vizinhança, isto é, $T(x, y) = \mu(x, y)$. Isso funciona razoavelmente bem para imagens onde o objeto tem apenas uma cor, como é caso das figuras 14a e 14b.

No entanto, imagens pequenas como `retina.pgm`, onde o raio não pode ser muito grande, o método acaba deixando resultados pouco interessantes.



(a) `fiducial.pgm` com $r = 50$. 64% de pixels pretos.



(b) `wedge.pgm` com $r = 50$. 52% de pixels pretos.

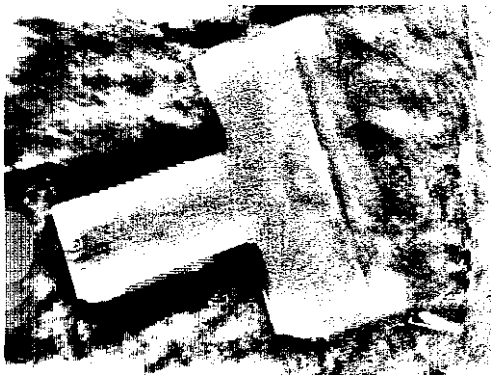
(c) `retina.pgm` com $r = 30$. 43% de pixels pretos.

Figura 14: Exemplos de aplicação do método `media`.

4.8 Método da Mediana

O último método seleciona o limiar como a mediana da vizinhança. Isso faz com que a probabilidade do pixel de ser maior ou menor que o limiar fique em torno de 50%, gerando muito ruído na imagem. Podemos ver esse efeito em objetos simples, de apenas um cor, como na figura 15a.

Em objetos detalhados (figura 15b), a técina acaba mantendo o aspecto da figura original, em vez de enfatizar os objetos da figura.



(a) `wedge.pgm` com $r = 50$.
42% de pixels pretos.

(b) `peppers.pgm` com $r = 60$.
50% de pixels pretos.

Figura 15: Exemplos de aplicação do método `mediana`.

5 Conclusão

Podemos ver que o método de Phansalkar et al., apesar de ter mais parâmetros, também é a técnica com mais controle do resultado. Além disso, com os parâmetros corretos, esse método pode funcionar exatamente como as outras técnicas baseadas na média da vizinhança.

Apesar disso, para objetos mais complexos, com grande variações de intensidade dentro do próprio, mas com boa iluminação em todo o objeto, o método de Bernsen e o método do contraste podem funcionar bem. O método global, também serve em caso específicos.