Universidade Estadual de Campinas

Instituto de computação

Introdução ao processamento de imagem digital

MC920

Trabalho #3

Nome: Leonardo de Alencar Lopes **RA:** 171928

1. Introdução

O objetivo deste trabalho é identificar objetos em imagens a partir de técnicas de limiarização global e local. Para isso, foi utilizada a linguagem Python 3 e as bibliotecas OpenCV (tratamento das imagens) e Matplotlib (exibição de gráficos).

Junto a este documento está o arquivo comprimido **171928-Trabalho-3.zip**. Nele, há todos os scripts que foram implementados e serão citados ao longo deste texto, além das pastas **img**, que contém as imagens utilizadas pelos algoritmos, e **resultados**, que contêm as imagens de outputs.

2. Os programas

Técnicas de limiarização se baseiam na classificação dos pixels de uma imagem a partir da especificação de um ou mais limiares **T**. Assim, a partir do histograma é possível visualizar a forma como este limiar separa a imagem em grupos dominantes, necessários para fazer a identificação de diferentes objetos.

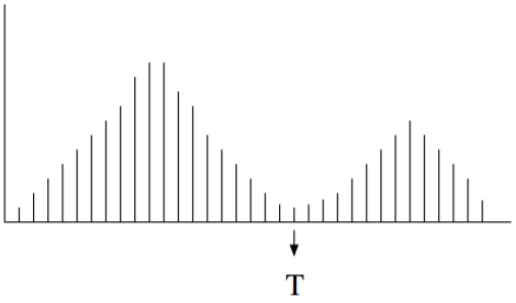


Figura 1: Limiarização global

A classificação de cada pixel é realizada comparando o seu nível de cinza com o valor do limiar. Assim, para cada ponto (x,y) tal que f(x,y)>T, este pixel é denominado como pertencente do objeto. Com isso, uma imagem g(x,y) pode ser obtida pela fórmula:

$$g(x,y) = \begin{cases} 0, se\ f(x,y) \leq T \\ 1, se\ f(x,y) > T \end{cases}$$

Neste caso, a limiarização é denominada binarização, pois a imagem resultante possui apenas dois valores de intensidade, 0 (preto) ou 1 (branco).

Para este relatório, foram desenvolvidos programas que realizam a limiarização de forma global e programas que realizam de forma local. No caso da limiarização global, é utilizado apenas um valor de limiar para segmentar a imagem toda, sendo muito suscetível erros por conta de iluminação não uniforme, ruído, parâmetros do dispositivo de aquisição não-uniformes, entre outros. No caso da limiarização local, o limiar é determinado com base nas características locais de cada região. Portanto, cada um dos programas desenvolvidos se baseia em dessas duas formas, variando entre si a maneira em que o limiar é calculado.

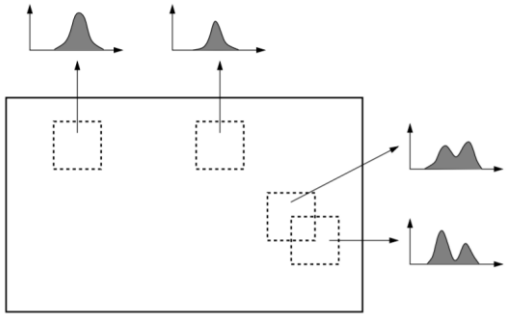


Figura 2: Limiarização local

Cada programa recebe como entrada uma imagem .**pgm**, que deve estar dentro da pasta **img**. Eles realizam o processamento da segmentação de acordo com os métodos propostos pela especificação deste relatório. No caso de métodos baseados em limiarização global, sua primeira saída será um gráfico do histograma da imagem, identificando o ponto em que a imagem foi segmentada pelo limiar T. Para métodos locais, será apresentado apenas o histograma. Em seguida, é apresentada a imagem de *input* f(x,y) e a imagem de *output* g(x,y). Por fim, é apresentado na saída padrão a fração dos pixels pretos para cada imagem de entrada. A imagem de saída será gravada na pasta **resultados**, especificando em seu nome os parâmetros utilizados, caso haja.

Método global

Para o método de limiarização global, foi utilizado o algoritmo de Ridler e Calvard para determinar T. Ele se baseia no cálculo de T de forma iterativa, a partir de um T_0 inicialmente definido. Para cada novo cálculo de T a imagem é particionada em duas regiões R_1 e R_2, em seguida calcula-se valores de intensidade média de pixel em cada uma dessas regiões. Com isso, o novo limiar é calculado pela fórmula:

$$T_{i+1} = \frac{média_{R1} + média_{R2}}{2}$$

Esse processo é repetido até que o valor de $|T_{i+1} - T_i| < 1$. O processo desse cálculo é realizado pela função *ridler_calvard(img)*, implementada no arquivo **limi_lib.py**.

Além do algoritmo de Ridler e Calvard, também foi utilizado o valor médio de todos os pixels da imagem como limiar e outros valores dentro do intervalor [T-100, T+100], em que T foi calculado pela função *ridler_calvard(img)*.

Para verificar os resultados desse método, basta rodar o programa limi_global.py, passando como parâmetro a imagem de entrada (que deve estar na pasta img) e um parâmetro 1, caso queira analisar também os histogramas gerados. Ex:

python3 limi_global.py sonnet.pgm 1

Analisando as imagens de saída, é possível notar que métodos globais não conseguem prover resultados satisfatórios, principalmente em imagens que possuem sombreado bem acentuado, como na imagem de exemplo. Alterar o valor de T apenas modifica as pequenas partes dos objetos que o algoritmo consegue identificar.

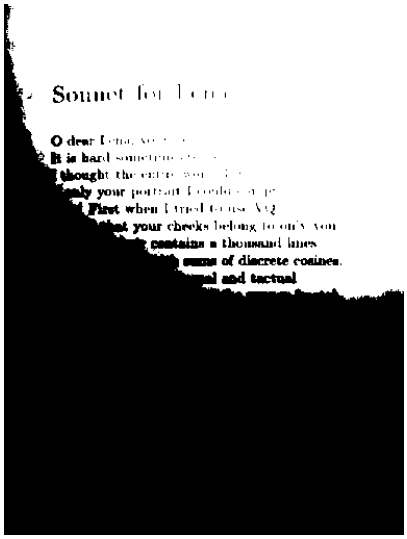


Figura 3: Imagem sonnet.pgm utilizando Ridler e Calvard

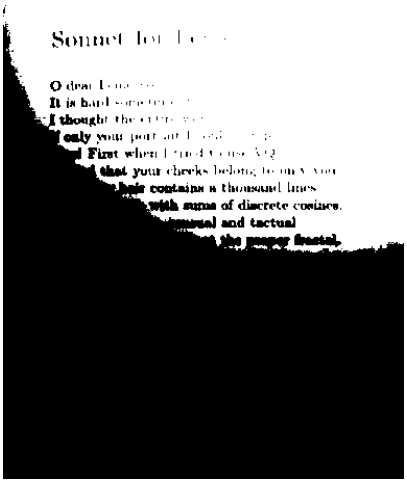


Figura 4: Imagem sonnet.pgm utilizando média

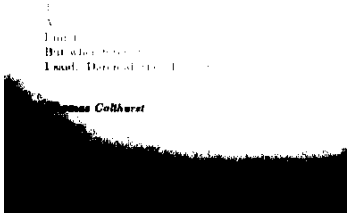


Figura 5: Imagem sonnet.pgm com T=59

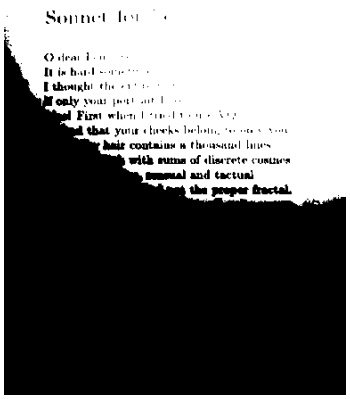


Figura 6: Imagem sonnet.pgm com T=129

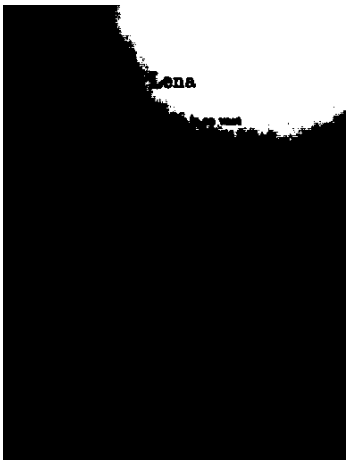


Figura 7: Imagem sonnet.pgm com T=219

Não foram notáveis as diferenças entre o método de Ridler e Calvard e o cálculo do limiar pela média. Em todos os testes, limiarização utilizando métodos globais não tiveram um efeito bom, como pode ser exemplificado nas imagens acima.

Método de Bersen

O método de Bersen e os métodos a seguir realizam a limiarização de forma local, calculando um valor de limiar para cada pixel baseado nos valores da sua vizinhança. Assim, o limiar é calculado como

$$T(x,y) = \frac{z_{min} + z_{max}}{2}$$

em que z_{min} e z_{max} são valores de níveis de cinza mínimo e máximo, respectivamente, em uma vizinhança nxn centrada em (x,y).

Para verificar os resultados desse método, basta rodar o programa limi_bersen.py, passando como parâmetro a imagem de entrada (que deve estar na pasta img). Ex:

python3 limi_bersen.py sonnet.pgm 1

O programa irá apresentar o resultado do método de Bersen utilizando valores **n** (3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43 e 47) para a vizinhança sobre os pixels. Analisando os resultados, é possível notar uma significativa melhora sobre a forma como as letras da imagem foram identificadas. Apesar de ainda não ser possível ler o texto, todo o espaço que contém caracteres foi destacado. Valores de n a partir de 19 tiveram resultados parecidos, enquanto valores menores tiveram deformações significativas no objeto.

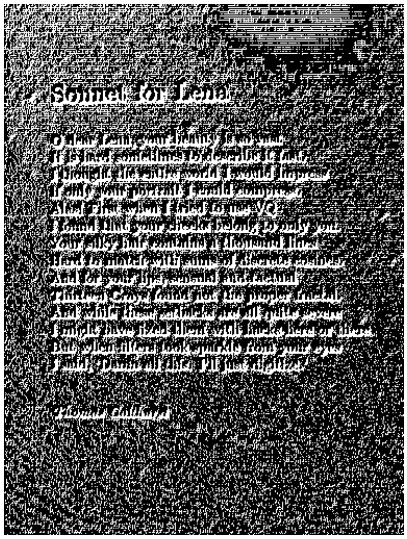


Figura 8: Imagem sonnet.pgm com Bersen e n=3

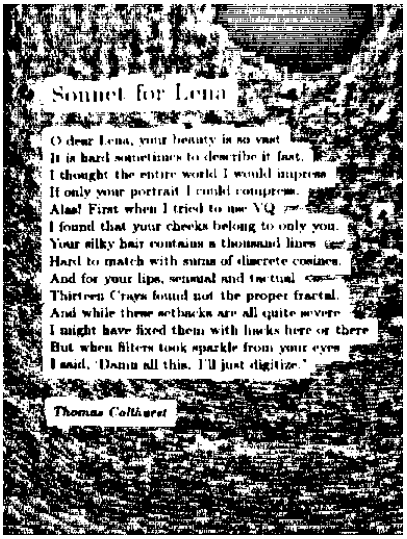


Figura 9: Imagem sonnet.pgm com Bersen e n = 19

Método de Niblack

O método de Niblack calcula o limiar a partir da média local μ(x,y) e do desvio padrão σ(x,y) a partir da vizinhança nxn centrada em um pixel (x,y). Assim, temos

$$T(x,y) = \mu(x,y) + k \cdot \sigma(x,y)$$

em que k é utilizado para ajustar a fração da borda do objeto a ser considerada.

Para verificar os resultados desse método, basta rodar o programa limi_niblak.py, passando como parâmetro a imagem de entrada (que deve estar na pasta img). Ex:

python3 limi_niblak.py sonnet.pgm 1

O programa irá apresentar os resultados do método de Niblack utilizando n=15 (valor sugerido) e variando o valor de k em 0.1 unidade no intervalor [-1,0]. As imagens obtidas tiveram um desempenho melhor em áreas sem objetos, mas ainda não é possível ler o texto. O melhor resultado obtido foi para k=-0.2, sendo possível identificar mais palavras.

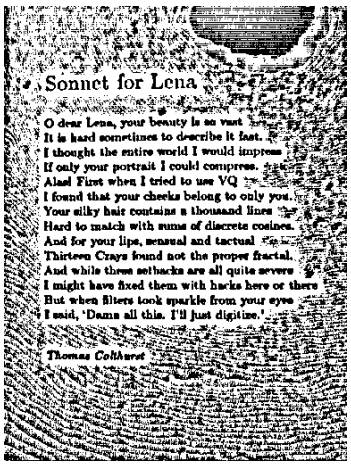


Figura 10: Imagem sonnet.pgm com Niblak e k=-0.2

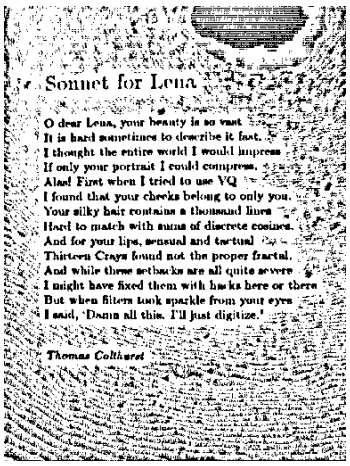


Figura 11: Imagem sonnet.pgm com Niblak e k=-0.5

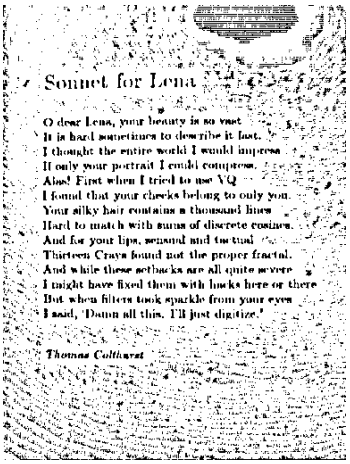


Figura 12: Imagem sonnet.pgm com Niblak e k=-0.8

Método de Sauvola e Pietaksinen

O método de Sauvola e Pietaksinen se propõe a melhorar o método de Niblak, principalmente para imagens de documentos com má iluminação. O cálculo é feito da seguinte forma:

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

Para verificar os resultados desse método, basta rodar o programa limi_sauvola.py, passando como parâmetro a imagem de entrada (que deve estar na pasta img). Ex:

python3 limi_sauvola.py sonnet.pgm 1

O programa irá apresentar os resultados do método de Sauvola e Pietaksinen utilizando n=15 e R=128 (conforme foi sugerido pelos autores). Porém, o valor de k é

variado em 0.1 unidade dentro do intervalor [0,1], para identificar alterações causadas por este parâmetro.

Nas imagens de saída é possível notar uma melhora muito significativa na identificação da área que contém objetos, deixando praticamente sem nenhum tipo de realce as partes vazias. Entretanto, para valores maiores de k, é notável que o método vai perdendo sua sensibilidade de detecção. Valores de k menores do que 0.2 se mostraram muito eficientes.

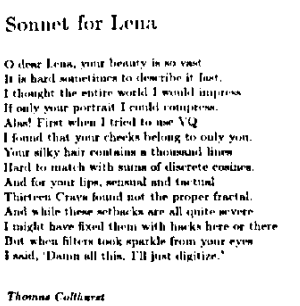


Figura 13: Imagem sonnet.pgm com Sauvola e Pietaksinen com k=0.1

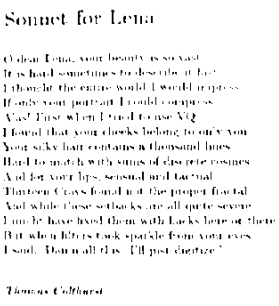


Figura 14: Imagem sonnet.pgm com Sauvola e Pietaksinen com k=0.2

Figura 15: Imagem sonnet.pgm com Sauvola e Pietaksinen com k=0.4

Método de Phansalskar, More e Sabale

O método de Phansalskar, More e Sabale procura ser uma variação do método de Sauvola e Pietaksinen, para lidar com imagens de baixo contraste. O cálculo é feito da seguinte forma:

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

Para verificar os resultados desse método, basta rodar o programa limi_phansalskar.py, passando como parâmetro a imagem de entrada (que deve estar na pasta img). Ex:

python3 limi_phansalskar.py sonnet.pgm 1

O programa irá apresentar o as imagens de saída para p=2 e q=10 (valores sugeridos pelos autores). Apesar dos autores sugerirem um valor de R como sendo 0.5, resultados melhores foram obtidos com R=25, portanto será este valor utilizado. Serão apresentadas imagens para diferentes valores de k, indo de 0.05 até 0.5.

É possível notar que valores próximos de 0.2 (sugerido pelos autores) para k fornecem imagens muito boas, identificando toda a parte da imagem que faz parte do texto e

clareando totalmente áreas pertencentes a espaços vazios. Para este valor de k, é possível identificar algumas palavras, porém ainda não é possível realizar uma leitura razoável.

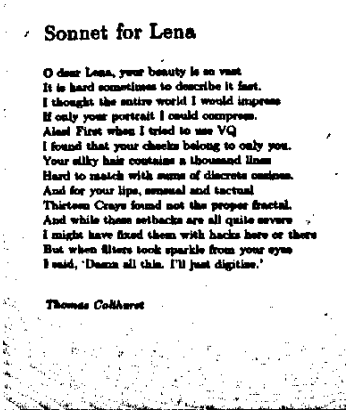


Figura 16: Imagem sonnet.pgm com Phansalskar e k=0.05

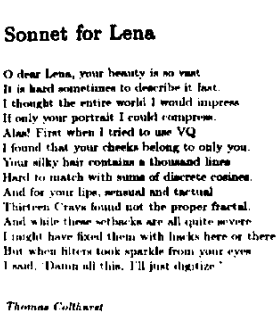


Figura 17: Imagem sonnet.pgm com Phansalskar e k=0.2

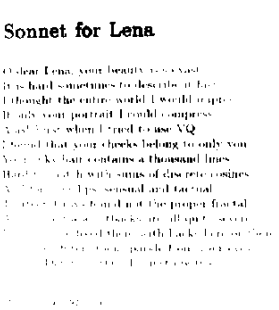


Figura 18: Imagem sonnet.pgm com Phansalskar e k=0.5

Método do Contraste

O método do Contraste atribui o valor de um pixel baseado nos valores de máximo e mínimo dos pixels da sua vizinhança. Assim, para uma vizinhança *nxn*, o pixel da imagem de saída assume o valor 0 se o valor do pixel na imagem de entrada está mais próximo do mínimo da sua vizinhança e assume 1, caso contrário.

Para verificar os resultados desse método, basta rodar o programa limi_contraste.py, passando como parâmetro a imagem de entrada (que deve estar na pasta img). Ex:

python3 limi_contraste.py sonnet.pgm 1

O programa irá apresentar as imagens de saída, variando o tamanho da vizinhança de 5 até 20, pulando de 3 em 3. Os resultados obtidos não são tão satisfatórios quanto os métodos locais vistos recentemente, se assemelhando muito ao método de Bernsen.

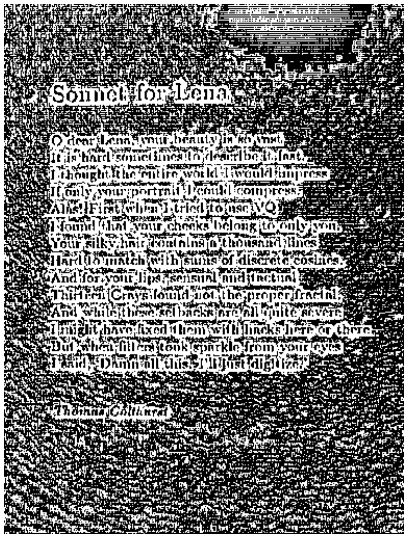


Figura 19: Imagem sonnet.pgm com Contraste e n=5

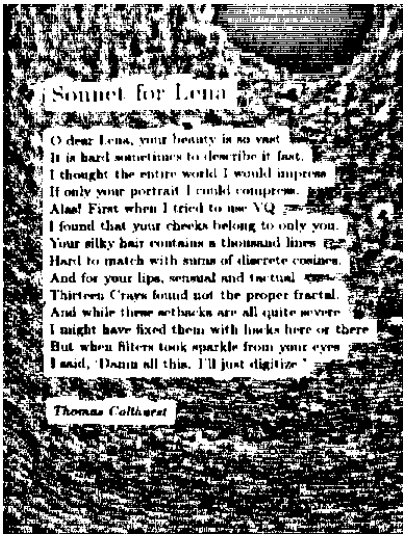


Figura 20: Imagem sonnet.pgm com Contraste e n=17

Método da Média

O método da Média seleciona como limiar para um pixel f(x,y) a média dos valores da sua vizinhança nxn. Assim, para facilitar implementação, foi desenvolvido como sendo o caso particular do método de Niblak para quando k=0.

Para verificar os resultados desse método, basta rodar o programa limi_media.py, passando como parâmetro a imagem de entrada (que deve estar na pasta img). Ex:

python3 limi_media.py sonnet.pgm 1

O programa irá apresentar as imagens de saída, variando o tamanho da vizinhança de 5 até 20, pulando de 3 em 3. Assim como o método do Contraste, os resultados não foram tão satisfatórios quanto métodos anteriores, também se assemelhando ao método de Bersen.

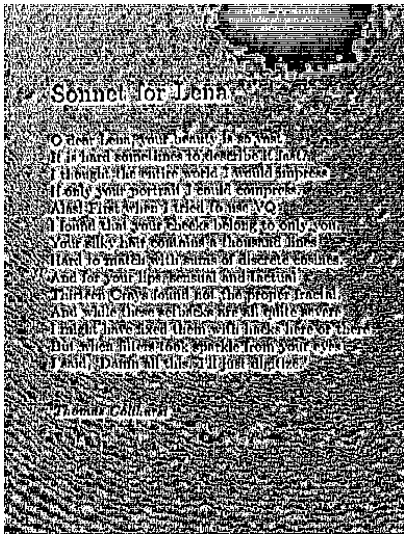


Figura 21: Imagem sonnet.pgm com Média e n=5

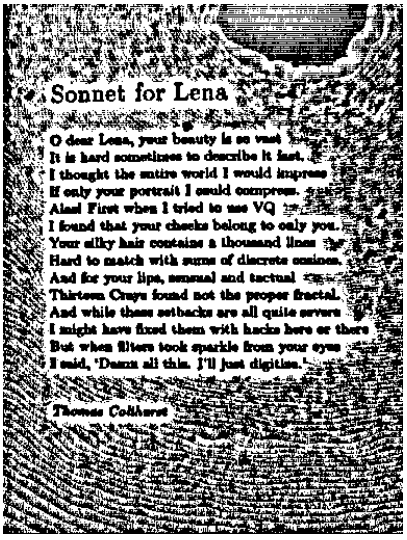


Figura 22: Imagem sonnet.pgm com Média e n=17

Método da Mediana

O método da Mediana seleciona como limiar para um pixel f(x,y) a mediana dos valores da sua vizinhança nxn.

Para verificar os resultados desse método, basta rodar o programa limi_mediana.py, passando como parâmetro a imagem de entrada (que deve estar na pasta img). Ex:

python3 limi_mediana.py sonnet.pgm 1

O programa irá apresentar as imagens de saída, variando o tamanho da vizinhança de 15 até 30, pulando de 3 em 3. De todos os métodos de limiarização local implementados para esta atividade, este foi o que obteve o pior desempenho. Nas suas saídas produzidas com a imagem *sonnet.pgm* não é possível identificar praticamente nenhuma letra, além de identificar de uma forma precária o espaço ocupado pelo texto. Nem em imagens mais simples o resultado foi satisfatório, não sendo possível identificar objetos que outros métodos identificam na maior parte dos *ranges* de seus parâmetros.

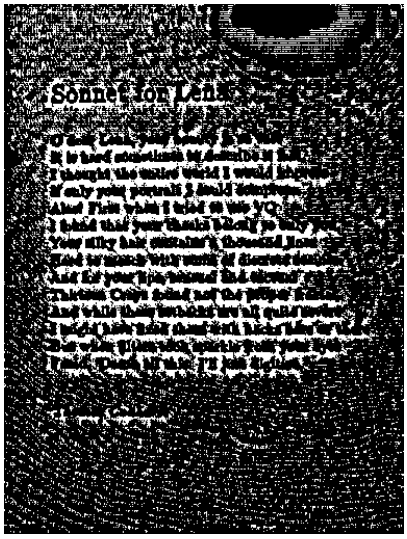


Figura 23: Imagem sonnet.pgm com Mediana e n=15

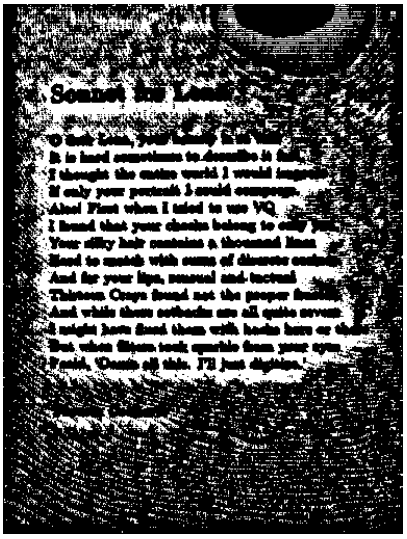


Figura 24: Imagem sonnet.pgm com Mediana e n=30

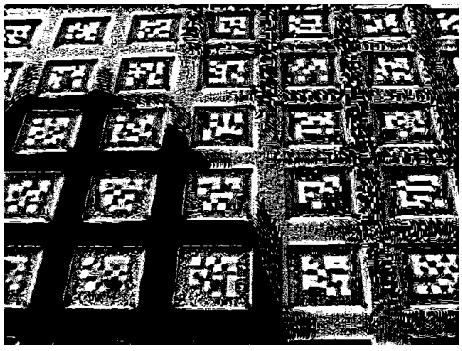


Figura 25: Imagem fiducial.pgm com Mediana e n=15

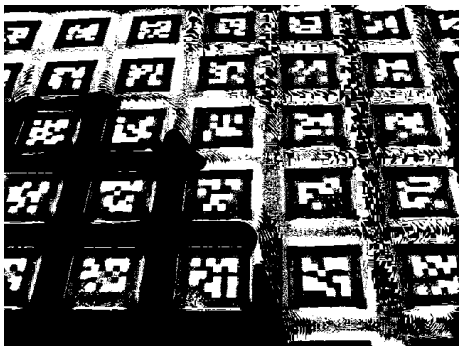


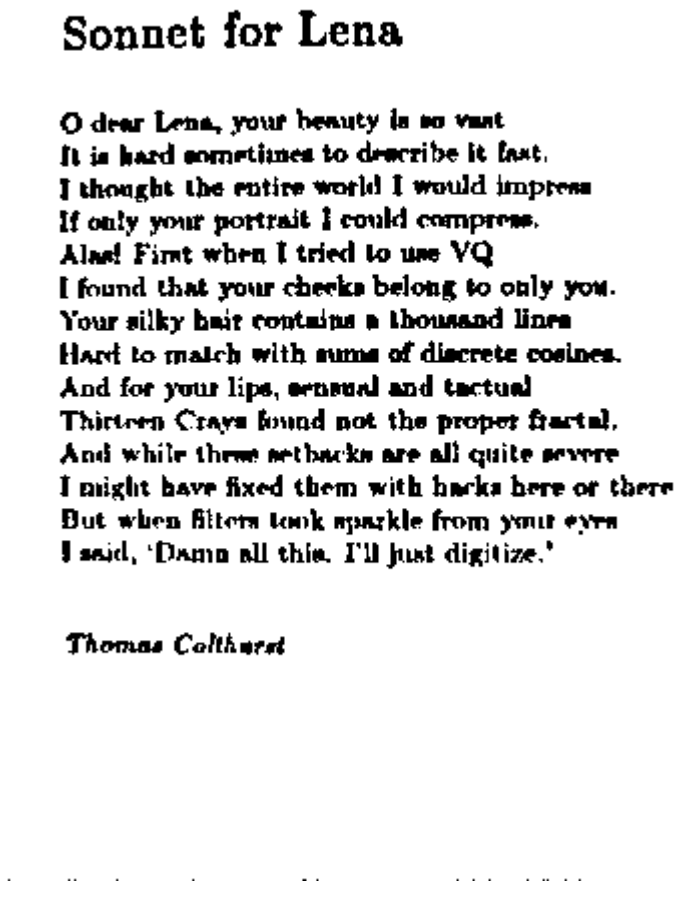
Figura 26: Imagem fiducial.pgm com Mediana e n=30

3. Conclusões

As técnicas de segmentação por limiarização são uma boa alternativa para encontrar objetos em imagens. Em imagens mais simples, foi possível notar que conseguem identificar espaços pertencentes a um mesmo componente com precisão, como pode ser notado nas imagens no final deste documento. Na principal imagem utilizada neste relatório (*sonnet.pgm*), foi possível identificar palavras inteiras e remover totalmente o efeito produzido pelo sombreamento heterogêneo, apenas modificando parâmetros do cálculo do limiar.

Por outro lado, a escolha do método a se utilizar não deve ser feita de forma superficial. O método de limiarização global se mostrou muito suscetível a erros provenientes de sombreamentos, independente do limiar escolhido. Os métodos locais tiveram um desempenho excelente, podendo ser possível até ler a imagem *sonnet.pgm* de forma razoável, bastante encontrar os parâmetros corretos no método de Phansalskar, More e Sabale. Apesar disso, os métodos do Contraste, da Média e da Mediana não conseguiram resultados satisfatórios em imagens mais complexas.

Melhores resultados obtidos



Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Figura 27: Melhor resultado de sonnet.pgm, utilizando Phansalskar com k = 0.1

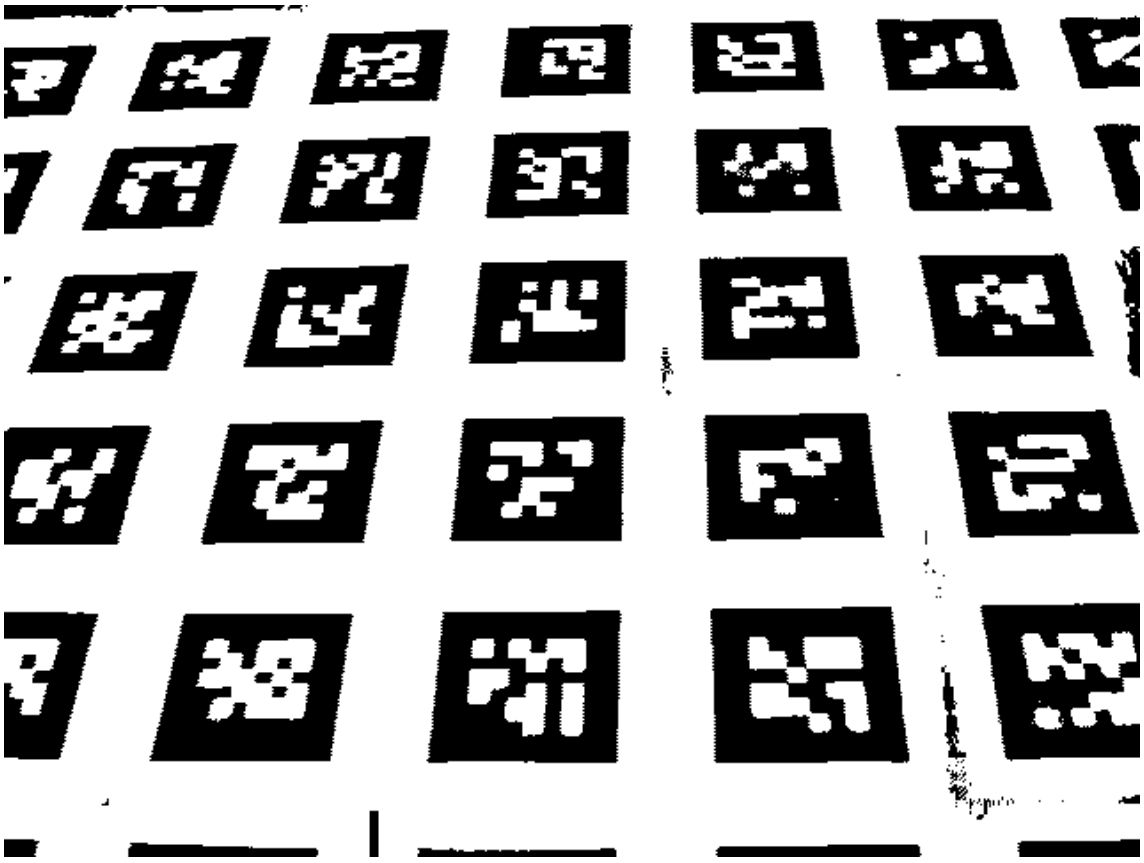


Figura 28: Melhor resultado fiducial.pgm, utilizando Niblak com k=-0.3 e n=45

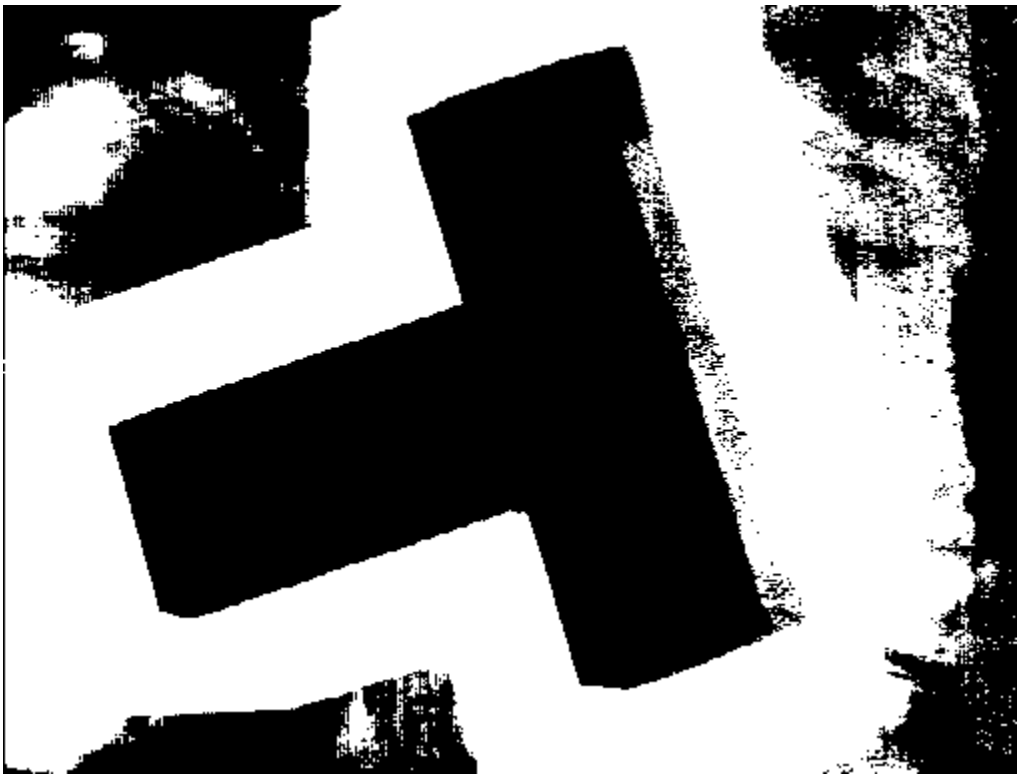


Figura 29: Melhor resultado wedge.pgm, utilizando Bersen com n=99