MC920 - Trabalho 2

Tobias Conran Zorzetto, RA: 166214

April 2023

1 Introdução

Esse trabalho tem como objetivo realizar uma série de experimentos a respeito de métodos de limiarização em imagens monocromáticas. Entre eles serão abordados métodos globais e locais. Esses métodos trazem, a partir do retorno de uma imagem binária, diferentes formas de determinar os componentes da imagem como fundo (pixels em branco) ou objeto (pixels em preto). Nesse contexto, esse relátorio tem como objetivo apresentar alguns desse métodos, trazendo com si análises a respeito dos algoritmos utilizados e seus resultados.

Assim, o texto está dividindo em 3 seções: **Introdução**, **Execução**, onde será explicado de maneira geral como foram feitos os programas, além das bibliotecas e ferramentas utilizadas, e **Métodos e Resultados**, onde serão apresentadas os resultados da aplicação de cada método específicado no enunciado do trabalho.

2 Execução

Para a resolução dos métodos enunciados no trabalho 2, alguns padrões foram adotados. Primeiramente, todos os códigos que implementam os algoritmos soluções foram feitos em python. Além disso, foram utilizadas 3 bibliotecas em geral para resolução.

A biblioteca *numpy* foi utilizada para manipular e fazer operações nas imagens em forma de vetorização. A biblioteca *imageio* foi utilizada para ler as imagens originais, salvando-as em vetores do *numpy*. Por fim, *matplotlib.plt* foi utilizada para apresentar as imagens resultantes de cada método na tela.

Para cada método foi feito um programa, que o nome será apresentado na seção seguinte. Todos os programas utilizam a foto **Fiducial.pgm**, figura 1, que tem seu histograma apresentado na figura 2, e por final apresentam primeiramente a imagem binária resultante na tela e depois o histograma da imagem resultante, com níveis 0 (quantidade de pixels pretos) e 1 (quantidade de pixels brancos). Além disso, é apresentado no terminal a fração de pixels pretos em relação à quantidade de pixels da imagem resultante.

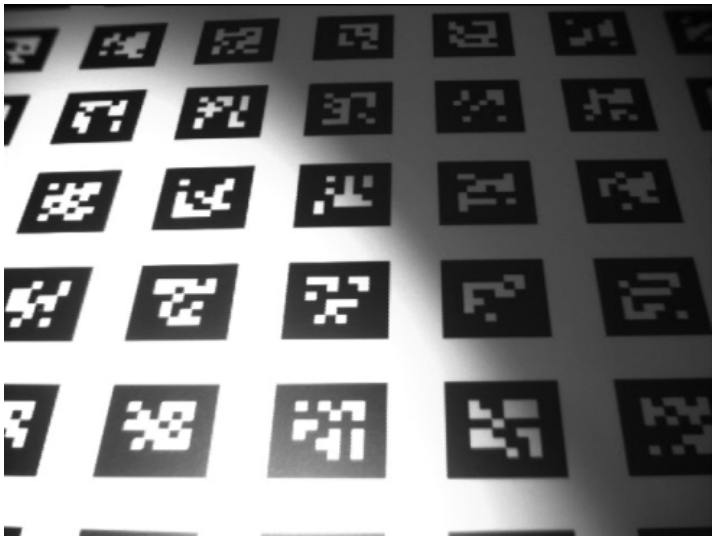

Figura 1: fiducial.pgm

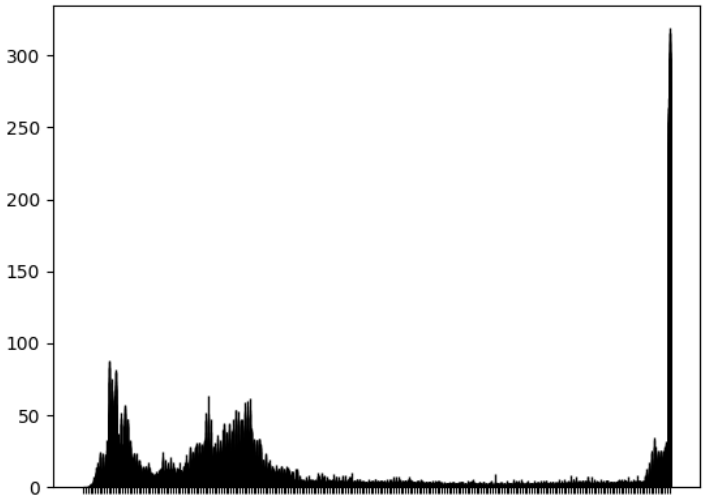

Figura 2: HIstograma da imagem fiducial.pgm

Para a resolução dos métodos, um algoritmo foi utlizado para os métodos globais e outro para os métodos locais. Para os métodos globais, cada um foi implementado de forma específica, assim cada forma será explicada na próxima secção.

Já para os métodos locais, o algoritmo consistiu em fazer um padding na imagem original dependendo do tamanho da máscara de vizinhança passada para o problema, depois iterar sobre os pontos originais da imagem. Para cada ponto é calculado o novo valor dele através da função que descreve tal método dentro da vizinhança parametrizada. Esse processo pode se tornar demorado dependendo do método utilizado e do tamanho da máscara, caso necessária a iteração sobre cada valor de vizinhança para cada ponto da imagem.

3 Métodos e Resultados

3.1 Método Global

O método Global, como já indica o nome, faz uma limirização global da imagem a partir da escolha arbitrária de um único valor T como limiar. A escolha de $T = 128$ foi feita a partir da análise do histograma da imagem original (figura 2) e avaliação do ponto em que parece dividir as duas áreas de maiores quantidades de valor, indicando que os valores anteriores a esse ponto representam objetos e

os maiores que ele representam fundo.

Assim, foi feito o programa `Global.py` que realiza por operações de vetorização o que foi descrito anteriormente, transformando pixels de valores menores que 128 em 0 e de valores maiores em 1.

A imagem binária resultante pode ser vista na figura 3, seu histograma resultante pode ser visto na figura 4 e a sua fração de pixels pretos foi de 0.62. Pode-se ver que para a parte mais clara da imagem o método funcionou muito bem, distinguindo cada um dos objetos sem muita perda de detalhe. Onde o método apresentou problemas foi na parte com sombra da imagem, na qual o conteúdo geral dos pixels é mais escuro em comparação à imagem toda, assim todos os pixels da área foram considerados objetos.

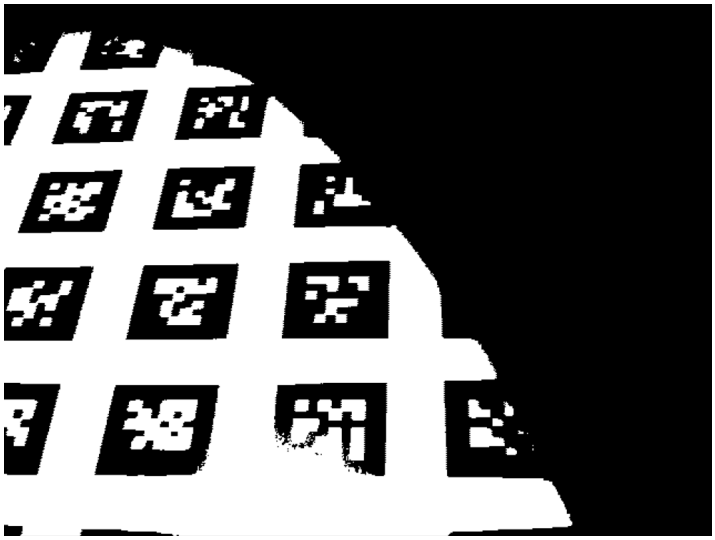

Figura 3: Imagem binária obtida a partir do método global

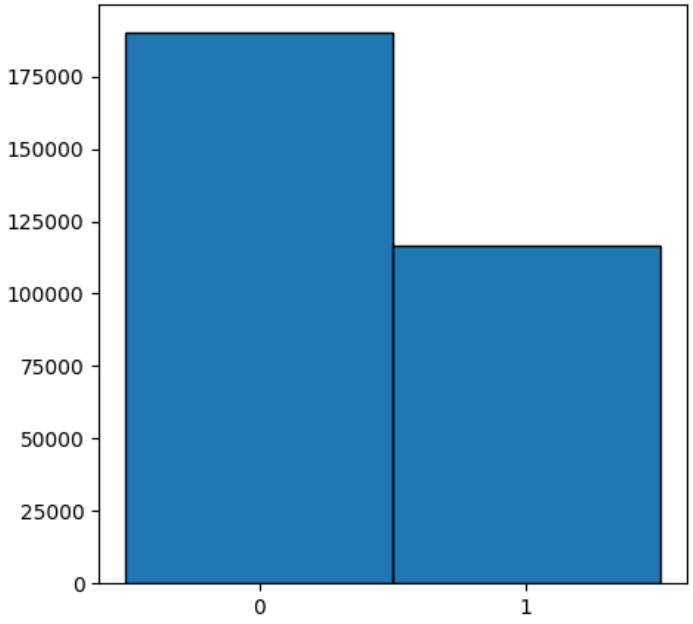

Figura 4: Histograma da imagem binária a partir do método global

3.2 Método de Otsu

O método de Otsu também é um método de limiariazação global. Porém, diferentemente do método anterior, a escolha do limiar T não é feita arbitráreamente, mas através da minimização da variância interclasse da imagem.

Assim, o programa `Otsu.py` implementa esse método através da função pronta da biblioteca *opencv*. Com isso, a imagem binária resultante pode ser vista na figura 5 e seu histograma na figura 6. Além disso, a fração de pixels pretos na imagem binária foi de 0.64, um pouco maior que com o método anterior.

A partir da Análise das imagens e da fração de pixels pretos, vê-se que por ser um método global, o método teve resultados parecidos ao anterior, na qual a parte sombreada foi toda considerada objeto. Porém, a imagem parece ter mais definição e menos erros em detalhes menores como pode ser visto no quadrado central inferior, na qual foi possível reconhecer sua totalidade, mesmo a parte mais clara, ignorada pelo método anterior.

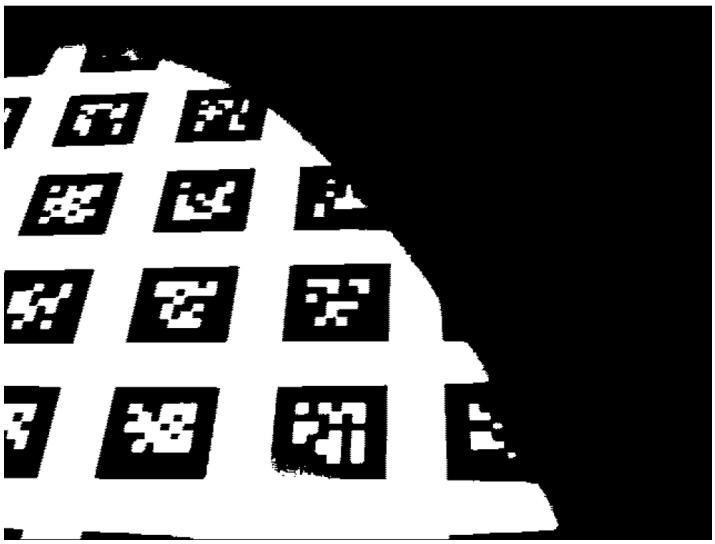

Figura 5: Imagem binária obtida a partir do método de Otsu

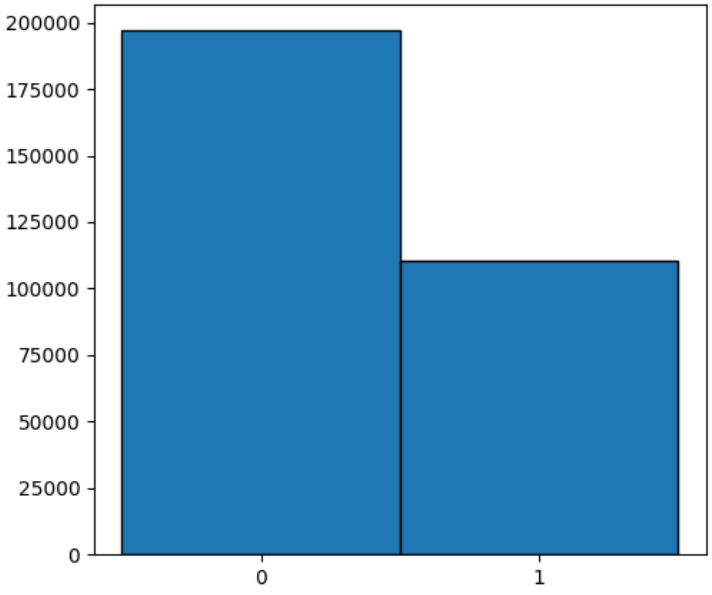

Figura 6: Histograma da imagem binária pelo método de Otsu

3.3 Método de Bernsen

O método de Bernsen é o primeiro método local apresentado nesse trabalho. A partir de uma vizinhaça nxn ao redor de cada pixel, o valor especifico T do limiar para esse ponto é calculado. Para esse método em específico, isso é feito encontrando o valor máximo e o valor mínimo dos pixels nessa vizinhança.

Assim, o limiar T é a média desses 2 valores. Com isso, o método de Bernsen foi implementado no programa `Bernsen.py` com uma vizinhança de $n = 5$.

A imagem binária resultante pode ser vista na figura 7, seu histograma resultante pode ser visto na figura 8 e a sua fração de pixels pretos foi de 0.28. A partir da análise das imagens e da fração de pixels pretos, pode ser visto que, primeiramente, com um método local agora pode ser vista uma distinção dos quadrados na parte sombreada do fundo. Apesar disso, o método apresentou dificuldades em distinguir fundos e objetos em situações em que existe uma concentração de um ou de outro. Por exemplo, o preenchimento dos quadrados está ruídoso, assim como o fundo na parte sombreada da imagem, no qual o método parece ter dificuldade de associar a fundo ou objeto, formando o padrão ruídoso visto na figura.

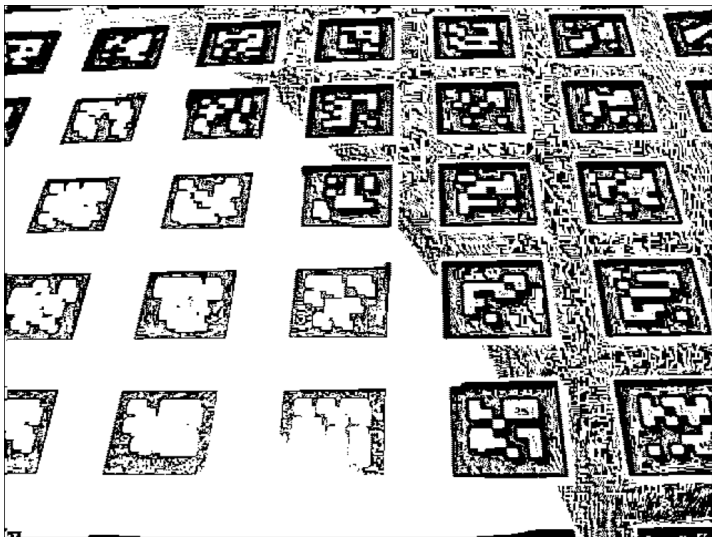

Figura 7: Imagem binária obtida a partir do método de Bernsen

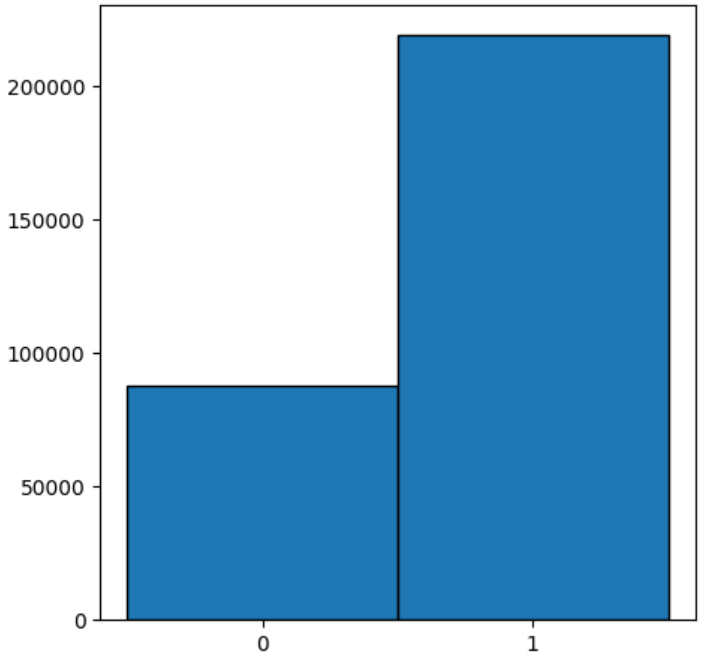

Figura 8: Histograma da imagem binária a partir do método de Bernsen

3.4 Método de Niblack

O Método de Niblack também é um algoritmo local, no qual são calculadas a média μ e o desvio padrão σ da vizinhança nxn ao redor de cada pixel. Assim calcula-se o limiar para esse ponto como $T = \mu + k\sigma$, sendo $k = 0.2$ uma constante abitrária.

O métod de Niblack foi implementado no programa `Niblack.py` com uma vizinhança de $n = 15$. Com isso, a imagem binária resultante pode ser vista na figura 9 e seu histograma na figura 10. Além disso, a fração de pixels pretos na imagem binária foi de 0.52. A patir dessas informações, vê-se que o resultado obtido foi melhorado em relação ao método anterior, no sentido em que todos os quadrados estão visiveis claramente e quase totalmente preenchidos, com poucos ruídos¿ O problema que persiste é a dificuldade na identificação do fundo na parte mais escura da imagem, além do surgimento de uma "segunda borda"nos quadrados da parte clara da imagem.

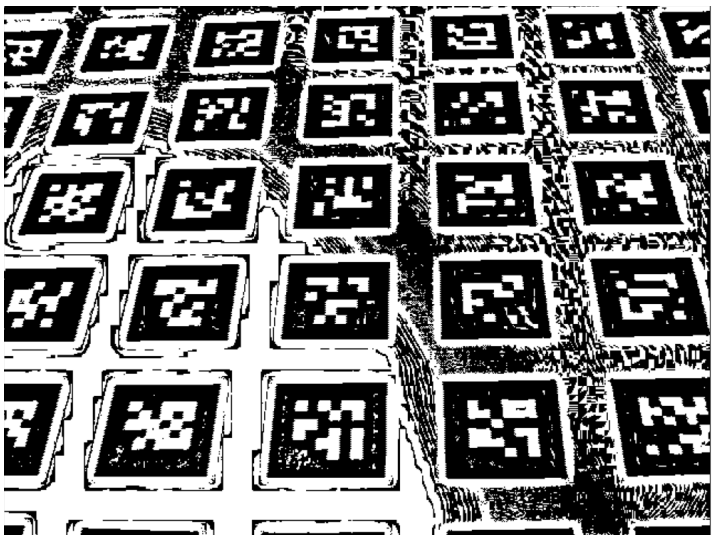

Figura 9: Imagem binária obtida a partir do método de Niblack

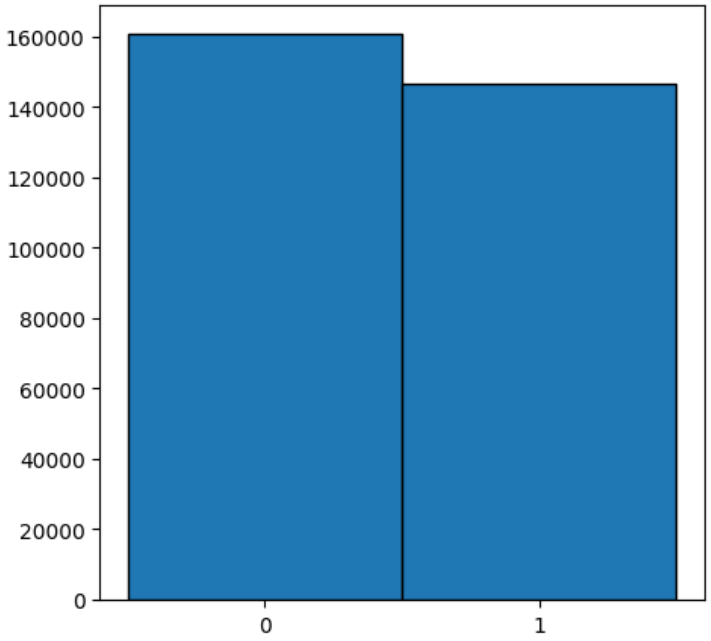

Figura 10: Histograma da imagem binária a partir do método de Niblack

3.5 Método de Sauvola e Pietaksin

O método de Sauvola e Pietaksin utiliza as mesmas propriedades do método de Niblack, mudando a apenas a fórmula para cálculo do limiar: $T = \mu(1+k(\frac{\sigma}{R})-1)$, em que $k = 0.5$ e $R = 128$ são constantes arbitrárias. Assim, a implementação desse método pode ser vista no programa `Sauvola.py`, com uma vizinhança de $n = 29$. Vale ressaltar que o tamanho grande da vizinhança torna o programa muito custoso, apesar disso, a escolha foi feita em prioridade a qualidade da resposta gerada em detrimento do tempo de execução.

A imagem binária resultante pode ser vista na figura 11, seu histograma resultante pode ser visto na figura 12 e a sua fração de pixels pretos foi de 0.28. A partir da análise desses resultados, vê-se que esse método se adapta melhor a situações de menos luz, sendo agora muito mais dificil de distinguir entre a parte mais clara e a parte sombreada da imagem original em questão de qualidade da resposta gerada. Apesar disso, Vê-se uma perda de detalhamento das regiões de fundo dentro dos quadrados principalmente.

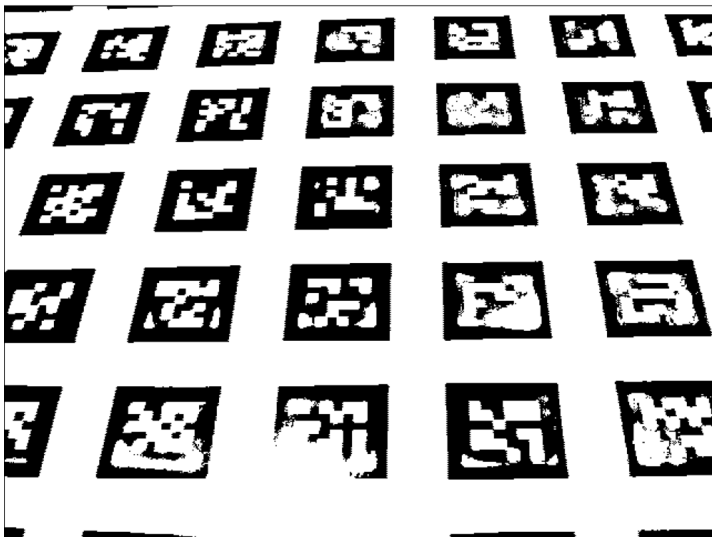

Figura 11: Imagem binária obtida a partir do método de Sauvola e Pietaksin

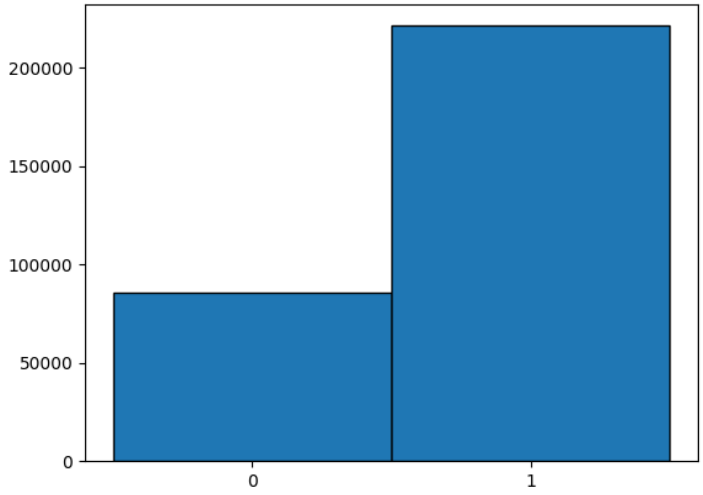

Figura 12: Histograma da imagem binária a partir do método de Sauvola e Pietaksin

3.6 Método de Phanlaskar, More e Sabale

Esse método também utilizada das mesmas propriedades que os 2 métodos anteriores, porém com a seguinte formula para o limiar: $T = \mu(1 + p \cdot exp(-q\mu) + k(\frac{\sigma}{R} - 1))$, sendo $k = 0.25$, $R = 128$, $p = 2$ e $q = 10$ constantes arbitrárias de ajuste. Assim, o método foi implementado no programa `Phanlaskar.py`, com uma vizinhança de $n = 29$.

O resultado da aplicação desse método pode ser visto nas figuras 13 e 14, a primeira mostrando a imagem binária resultante e a segunda mostrando o seu histograma correspondente. Além disso, a fração de pixels pretos pelos pixels da imagem foi de 0.32. A partir da análise desses resultados, vê-se que é um método muito bom para o caso da imagem, com objetos e fundos bem determinados, apenas com limitações no preenchimento dos objetos em locais específicos, que se misturam com o fundo.

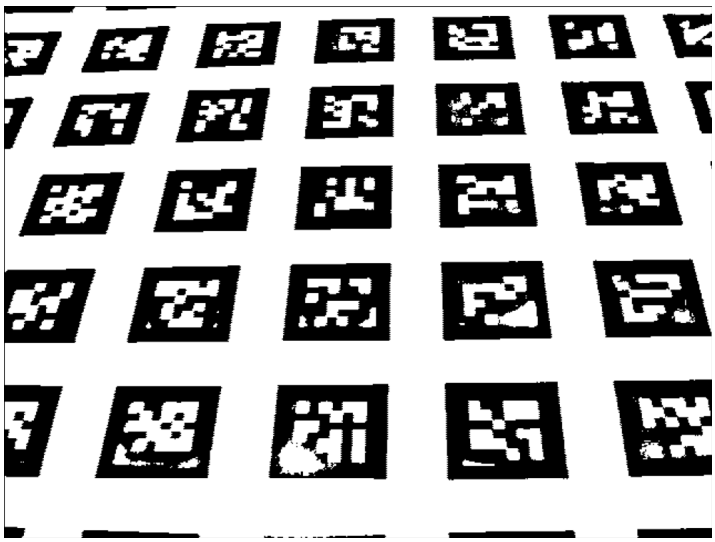

Figura 13: Imagem binária obtida a partir do método de Phanlaskar, More e Sabale

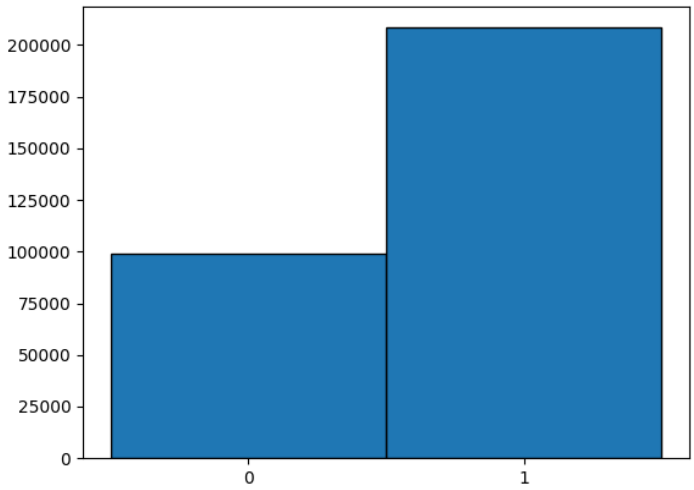

Figura 14: Histograma da imagem binária a partir do método de Phanlaskar, More e Sabale

3.7 Método do Contraste

Esse método, também local, encontra os valores minimos e máximos locais. Diferentemente dos outros métodos, no lugar de setar um limiar, o pixel é definido da seguinte forma: se o valor está mais próximo do máximo ele recebe o valor 1, caso mais perto do mínimo recbe o valor 0. Dessa forma o método foi implementado no programa `Constraste.py`, com uma vizinhança de $n = 15$.

Assim, a imagem binária resultante pode ser vista na figura 15 junto com seu histograma, figura 16. Sua fração de pixels pretos em função da totalidade de pixels da imagem foi de 0.39. A partir da análise desses resultados, vê-se um bom resultado, com objetos preenchidos e detalhes nos seus interiores. Apesar disso, ainda houve detecção errada excessiva de objetos nas áreas de fundo principalmente da área escura da imagem.

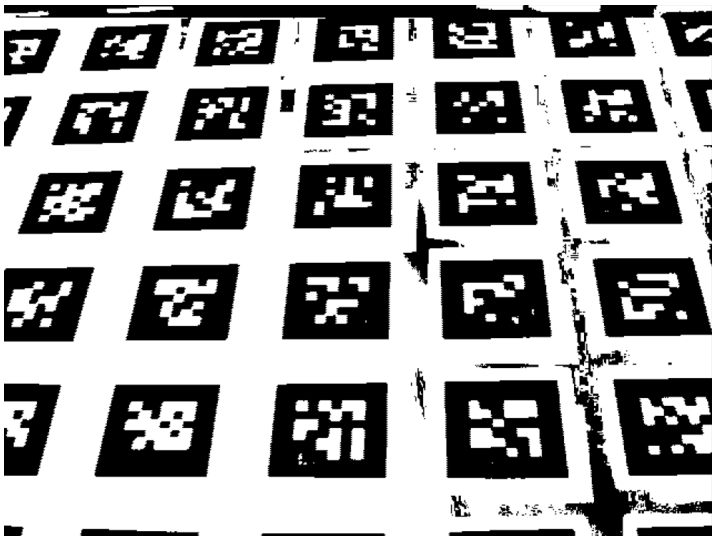

Figura 15: Imagem binária obtida a partir do método do Contraste

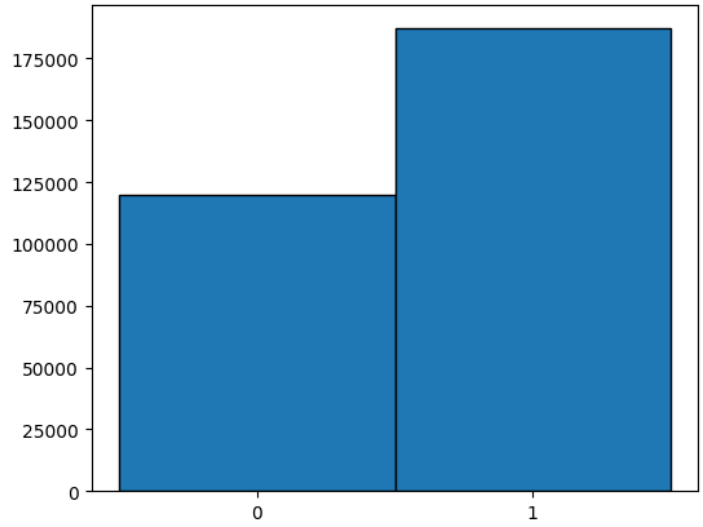

Figura 16: Histograma da imagem binária a partir do método do Contraste

3.8 Método da Média

O método da média, método local, consiste no cálculo da média local μ e calcular o limiar da forma $T = \mu - C$, sendo $C = 5$ uma constante de ajuste do limiar. Assim, o método foi implementado no programa *Media.py* com uma vizinhança de $n = 29$, novamente com detrimento do tempo de execução em função da qualidade da imagem resultante.

Assim, a imagem binária resultante pode ser vista na figura 17 junto com seu histograma, figura 18. Sua fração de pixels pretos em função da totalidade de pixels da imagem foi de 0.34. A partir disso, pode ser visto que o resultado pode ser considerado bom em realação aos outros métodos, apesar de uma quantidade peqeuan de ruído no fundo e algumas falhas no preenchimento dos quadrados.

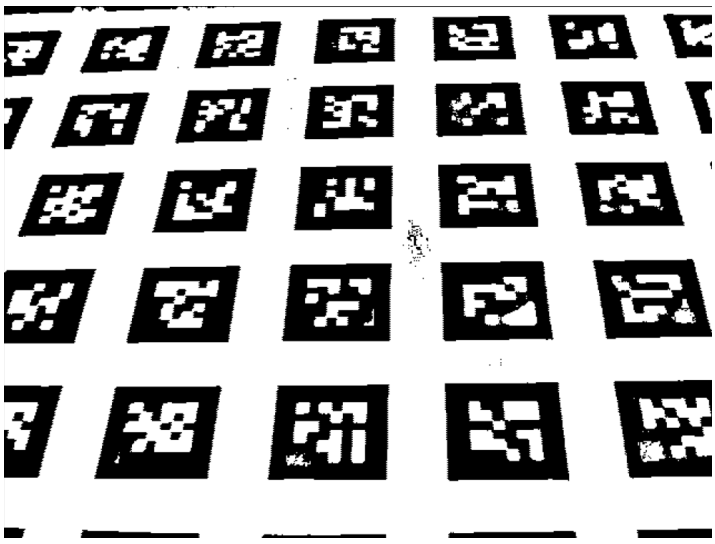

Figura 17: Imagem binária obtida a partir do método da Média

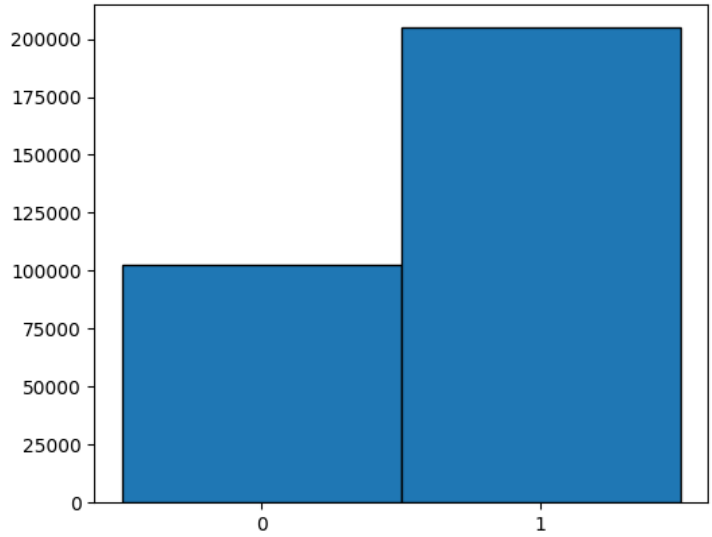

Figura 18: Histograma da imagem binária a partir do método da Média

3.9 Método da Mediana

O método da mediana, também local, consiste em encontrar a mediana dos valores dentro da vizinhança nxn de cada pixel. Caso o valor seja maior que a mediana, ele é determinado como 1, caso contrário, é determinado como 0. Esse método foi implementado com o programa `Mediana.py`, com uma vizinhança de tamanho $n = 15$.

O resultado da aplicação desse método pode ser visto nas figuras 19 e 20, a primeira mostrando a imagem binária resultante e a segunda mostrando o seu histograma correspondente. Além disso, a fração de pixels pretos pelos pixels da imagem foi de 0.29. A partir da análise desses resultados, é possível ver tanto os objetos determinados quanto a parte sombreada da imagem sofreram com a grande presença de ruído, apesar de uma certa definição das bordas dos objetos.

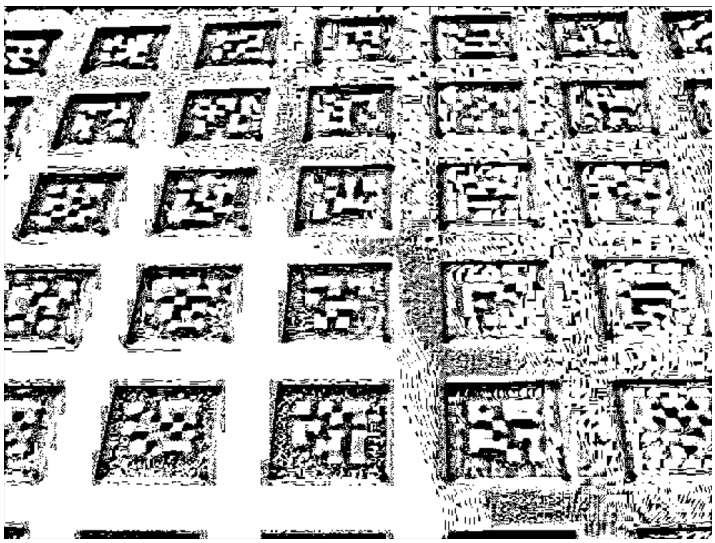

Figura 19: Imagem binária obtida a partir do método da mediana

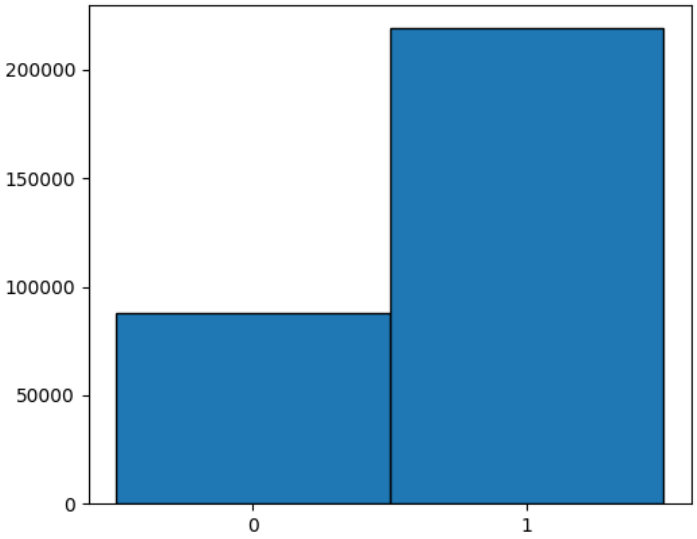

Figura 20: Histograma da imagem binária a partir do método da mediana