Sistemas de Visão e Percepção Industrial

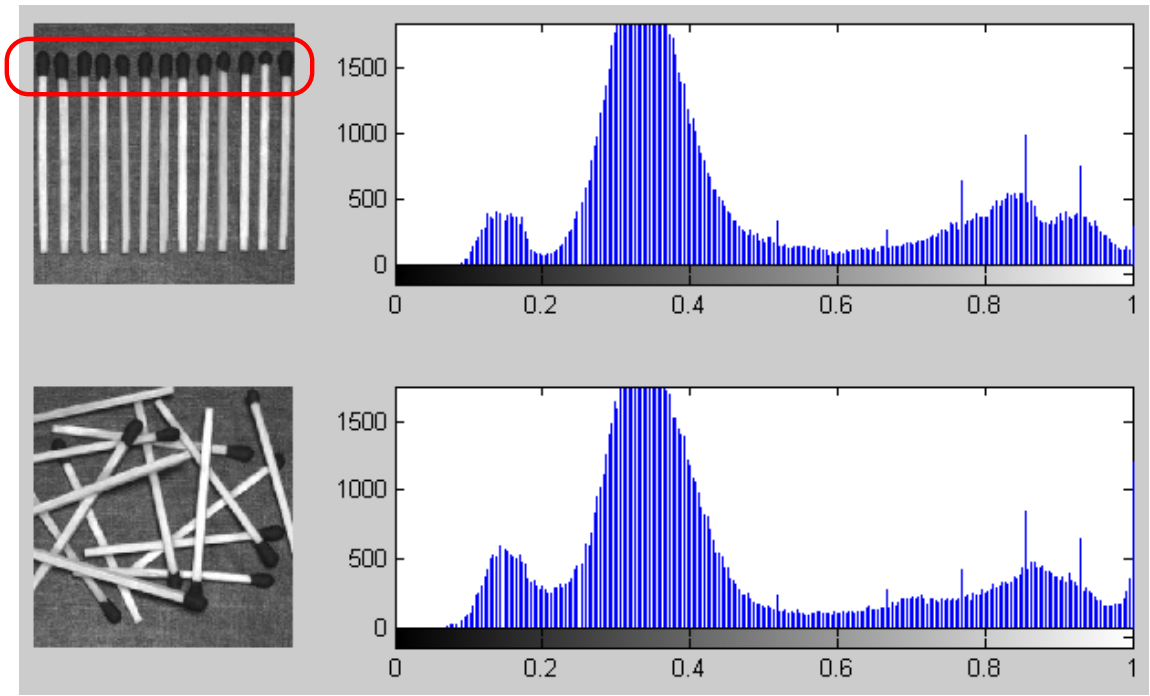
Aula Prática nº 5

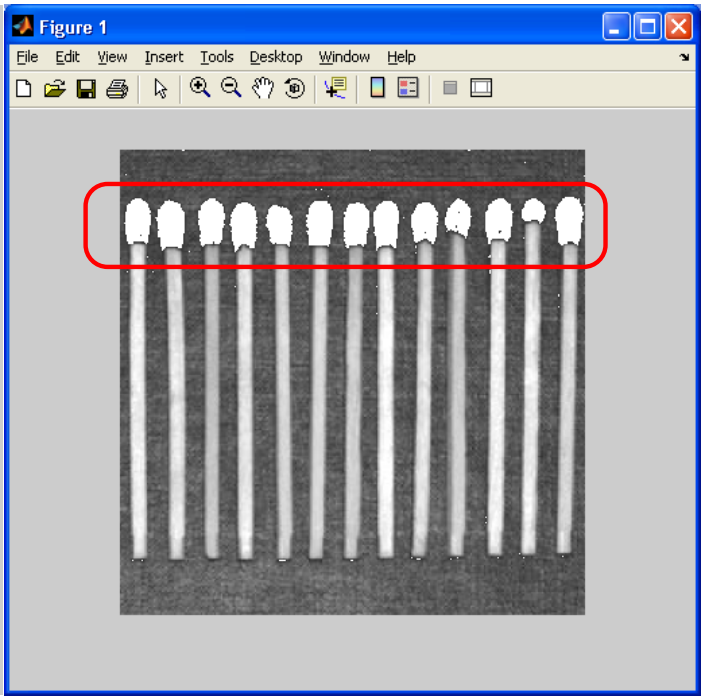
Histogramas. Binarização.

Universidade de Aveiro
2021-2022

Exercício 1)

- Visualizar e comparar os histogramas das imagens 'matches1.png' e 'matches2.png' – usar `imhist()`.
- Com base na observação visual dos histogramas, aplicar uma operação simples para forçar as cabeças dos fósforos da 1.ª imagem ao nível branco (pixels com valor 1)

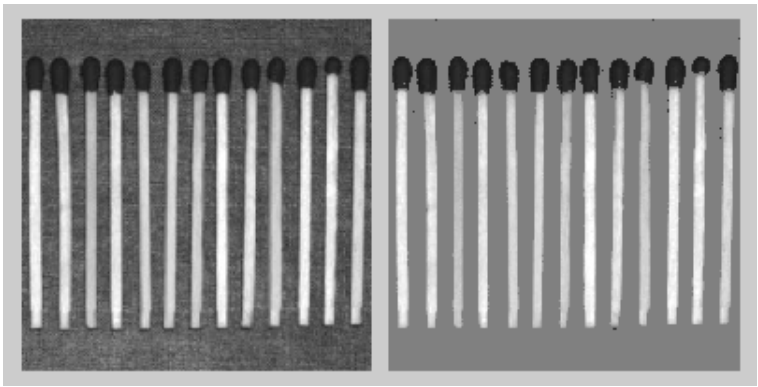


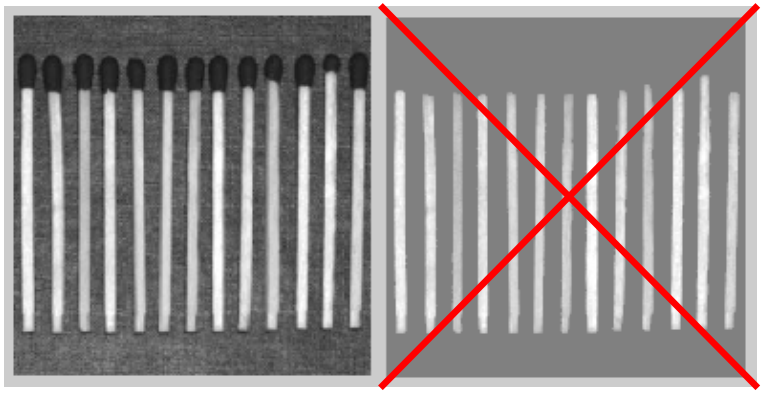


- O efeito não resulta tão bem na imagem 'matches2'. Porquê?

Exercício 2)

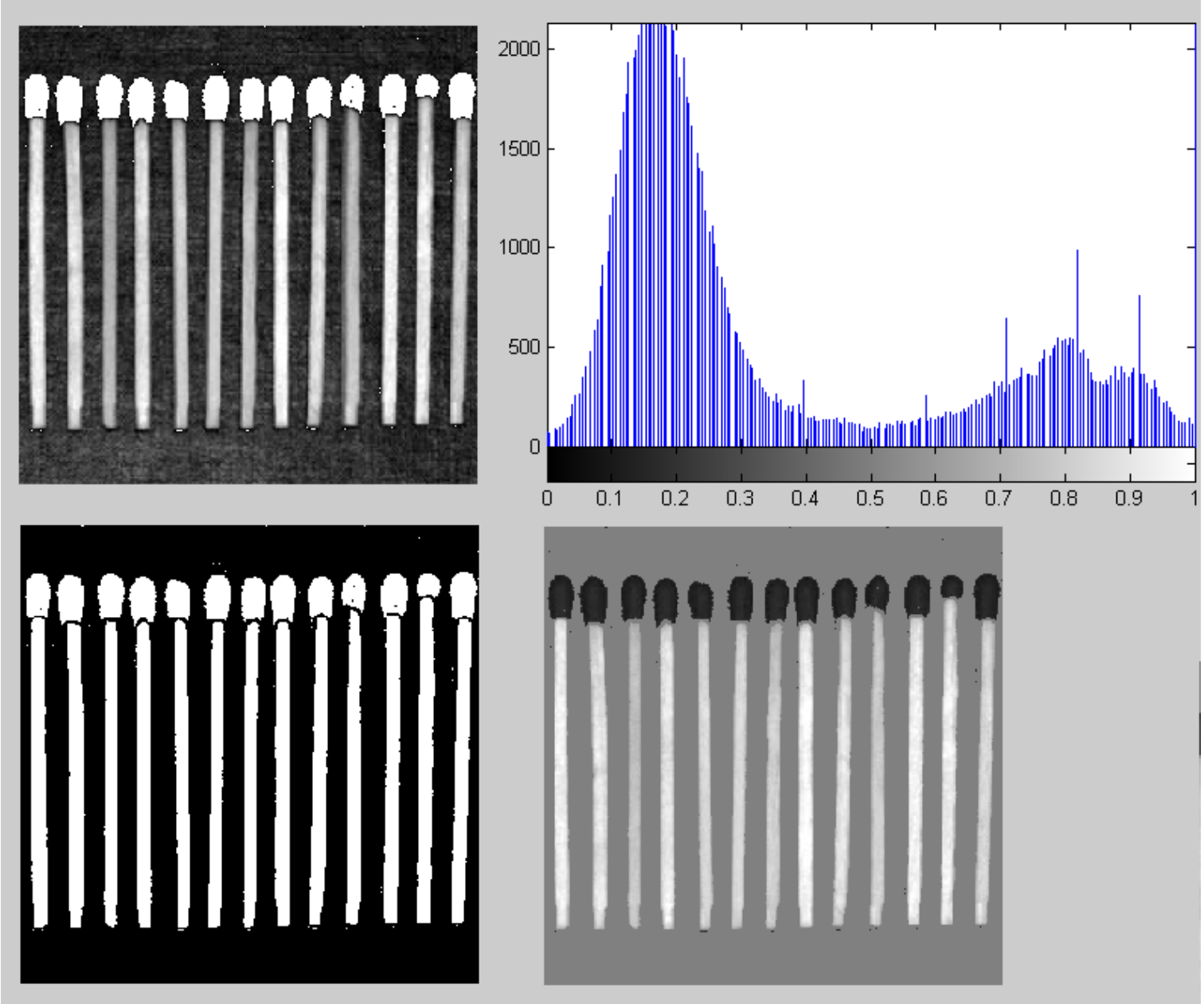
- Obter uma imagem onde se separam os fósforos do fundo, e o fundo é forçado a cinzento de 50%. (usar a imagem 'matches1.png')
  - NB. A imagem final deve incluir a cabeça dos fósforos
  - **Especificidade do exercício**: Os objetos têm componentes mais claras e mais escuras do que o fundo! (Histograma tri-modal)





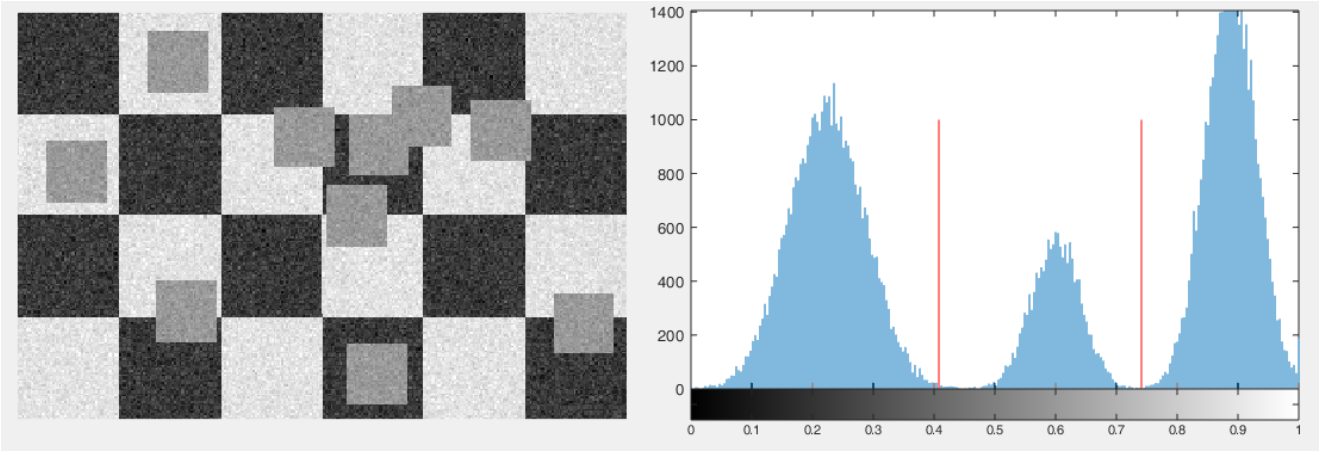
- Passos do exercício
  - Partir da imagem do exercício anterior onde os fósforos têm cabeças brancas.
  - Expandir o histograma de forma a aproveitar a gama total de cinzentos
  - Criar uma máscara para os fósforos por binarização (Otsu)
  - Usar a máscara anterior (ou a sua negação) para forçar o fundo a cinzento de 50% (explorar as capacidades de indexação do Matlab)

Exercício 2) – Ilustração dos passos



Exercício 3) – Histogramas trimodais

- Para a imagem `trimodalchess.png`, e sabendo que tem um histograma trimodal, determinar automaticamente os dois limiares pela técnica de Otsu (função `multithresh()`).

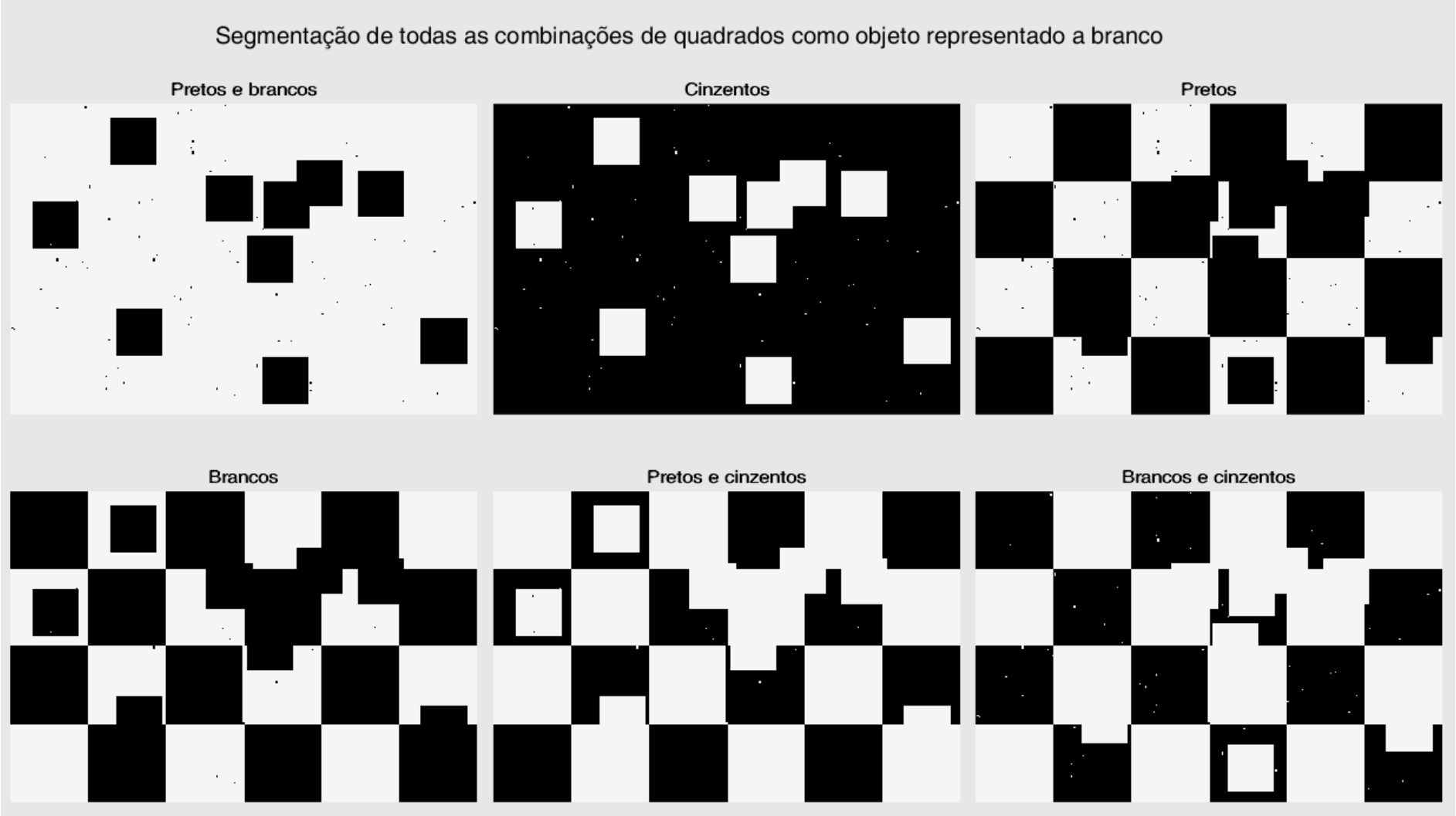


- Usando binarização com os limiares anteriores, obter as binarizações de todas as combinações possíveis do que poderiam ser objetos na imagem (ficam a branco na imagem binarizada):

- Os quadrados pretos e os brancos
- Os quadrados cinzentos
- Os quadrados pretos
- Os quadrados brancos
- Os quadrados pretos e os cinzentos
- Os quadrados brancos e os cinzentos

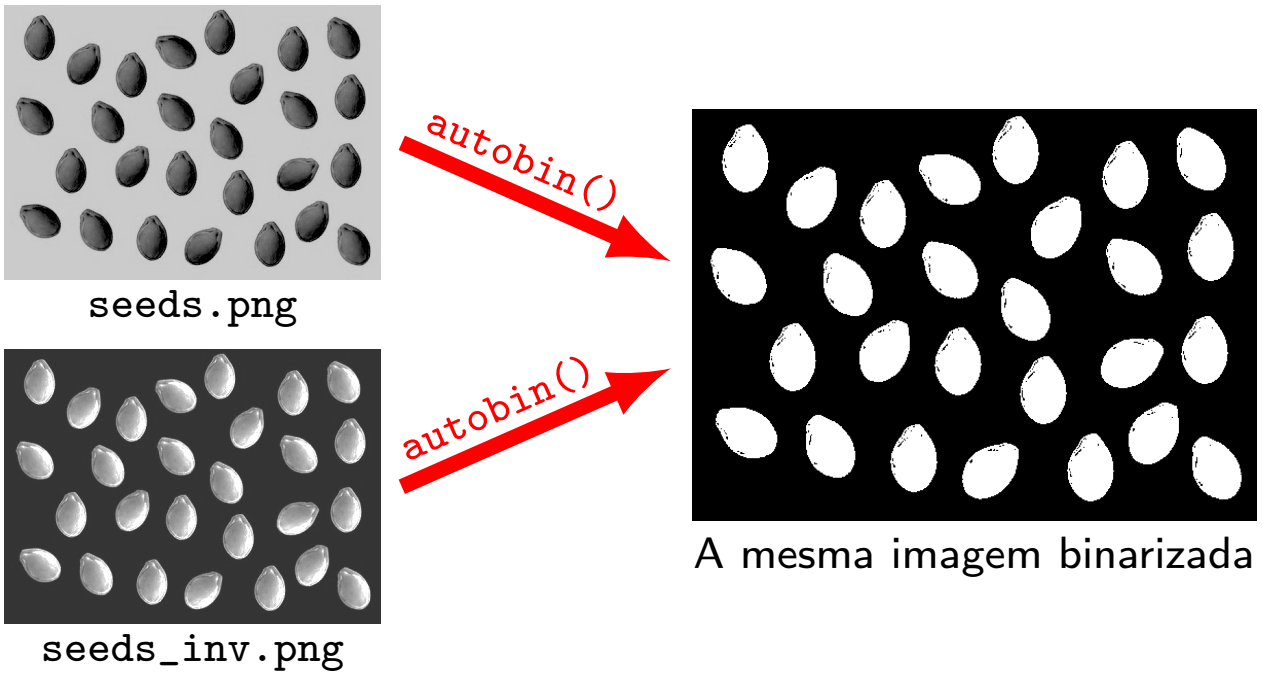
Exercício 3) – Ilustração do resultado

- Illustração da resposta a obter neste exercício



Exercício 4) – Criar a função `autobin.m`

- Usando `imbinarize()`, criar a função `autobin.m` que aceita uma imagem e devolva uma imagem binarizada por Otsu, mas que faça a inversão automática da imagem a devolver caso o número de pixels brancos seja maior do que o de pixels pretos, ou seja, no caso de 'seeds.png' e 'seeds_inv.png' o resultado da binarização deve ser sempre uma imagem com fundo preto, como ilustrado.



`seeds.png`

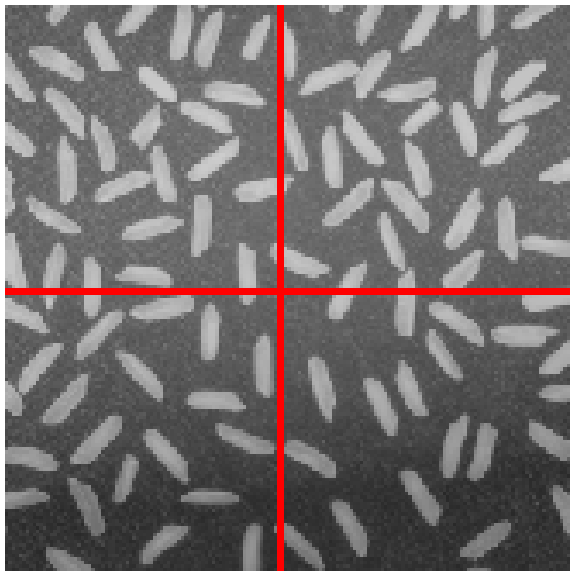
`seeds_inv.png`

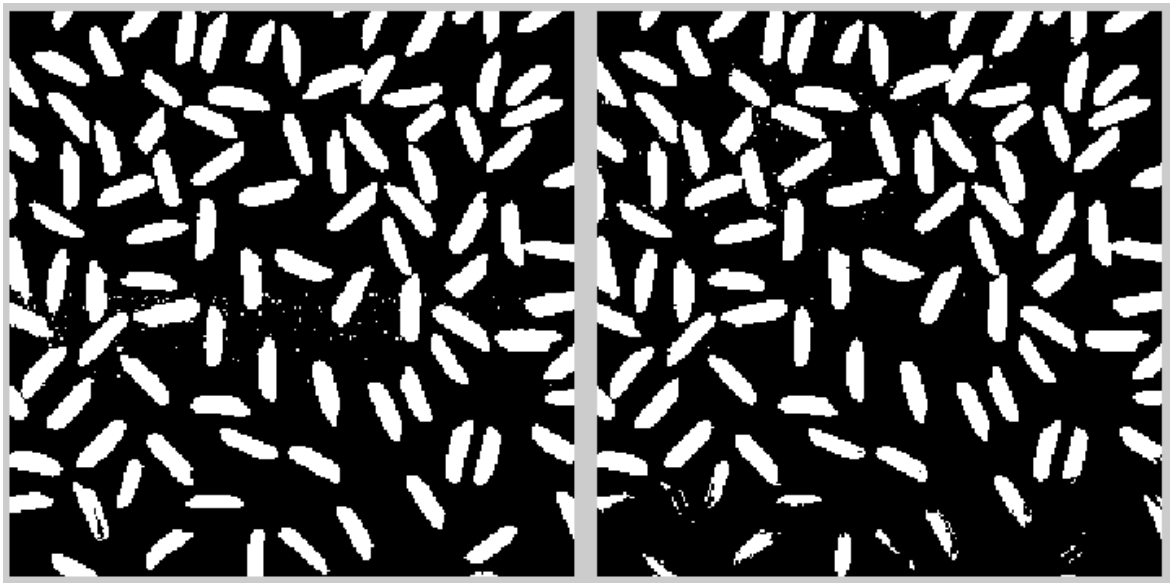
A mesma imagem binarizada

- Esta função `autobin()` funcionará sempre bem? Quando falha?

Exercício 5) – Binarização por regiões

- Dividir a imagem 'rice.png' em quatro partes, binarizar cada uma delas separadamente e juntar os resultados numa imagem completa.

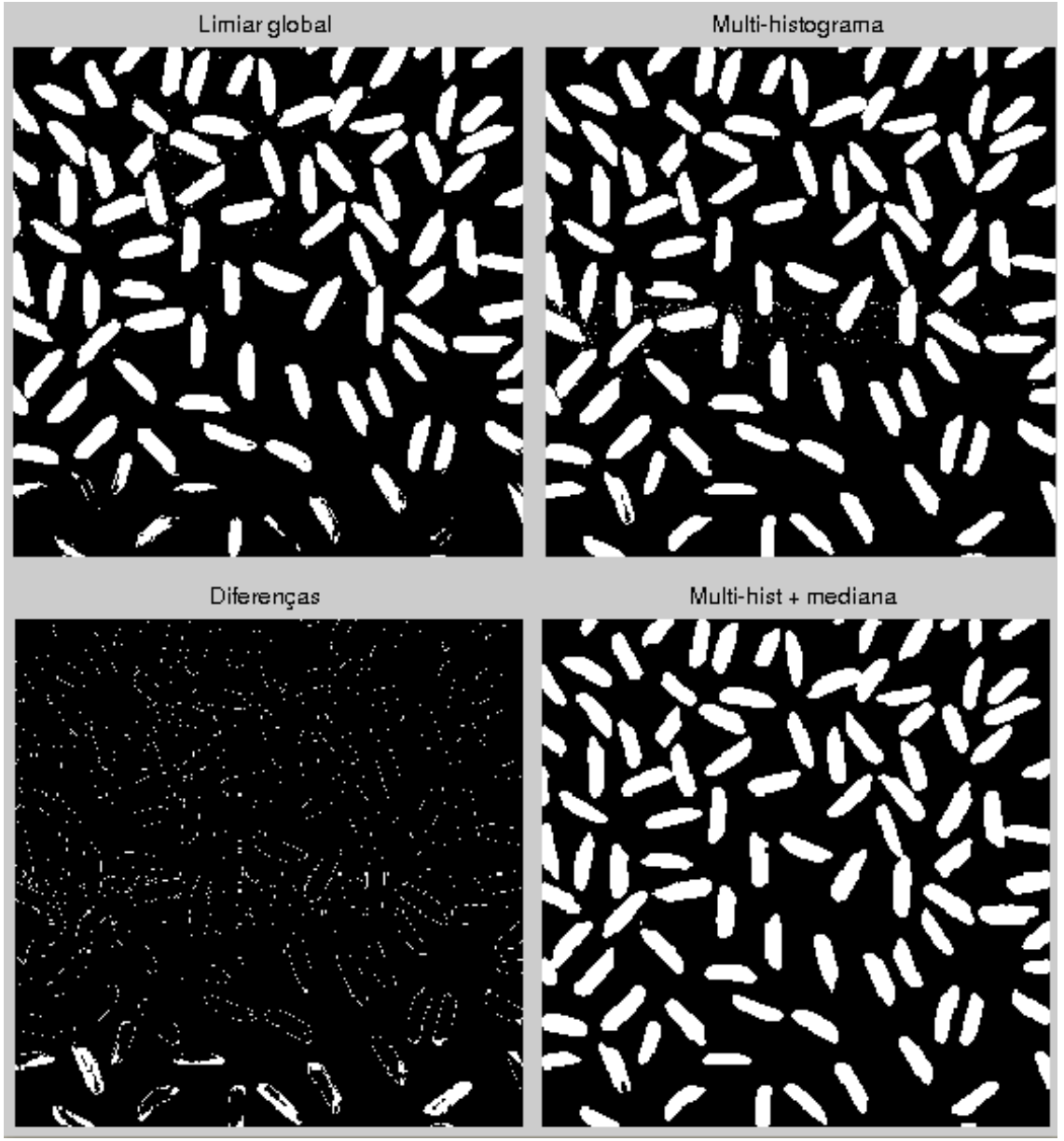




- Legenda da esquerda para a a direita:
  - 4 regiões
  - Justaposição das 4 regiões binarizadas separadamente (4 histogramas)
  - Binarização da imagem global (usando Otsu e um histograma único)

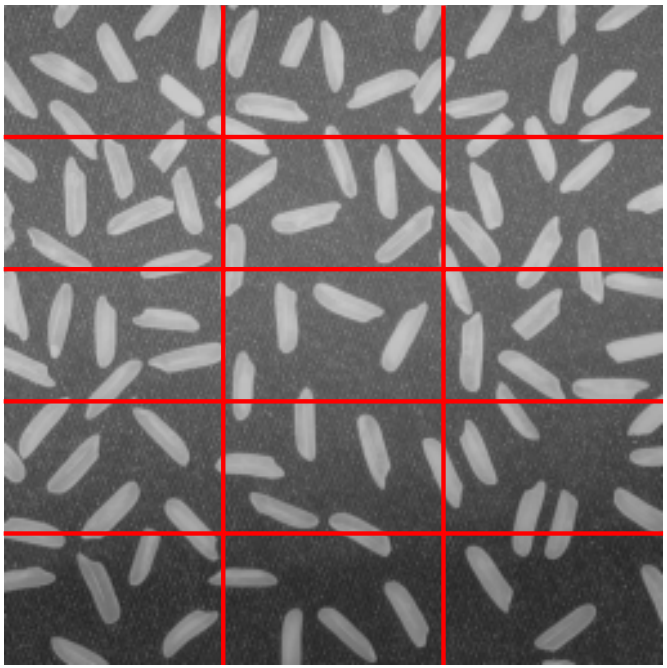
Exercício 5) – Comparar resultados

- Comparar os resultados do exercício anterior ilustrando os pixels diferentes nas imagens que resultam dos dois processos. Sugestão: usar o operador `xor()`.
- Proceder a uma operação de 'limpeza' da imagem resultante de múltiplos histogramas para eliminar alguns pixels menos relevantes.



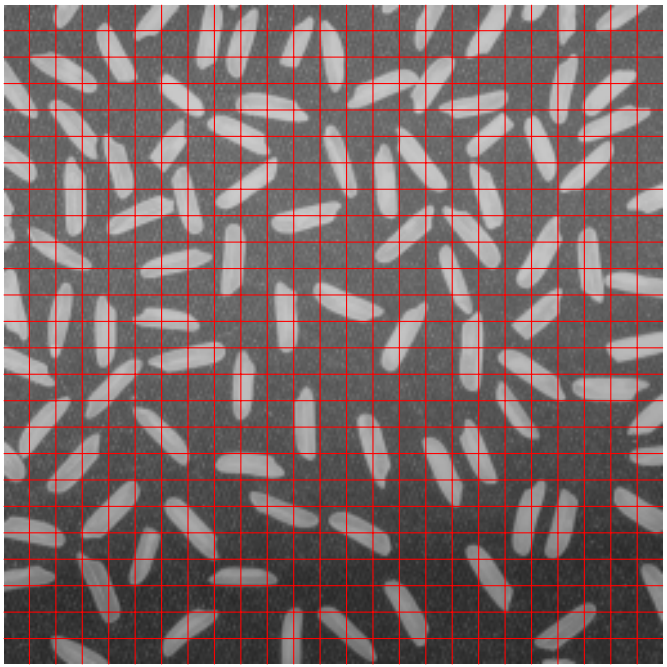
Exercício 6) – Opcional

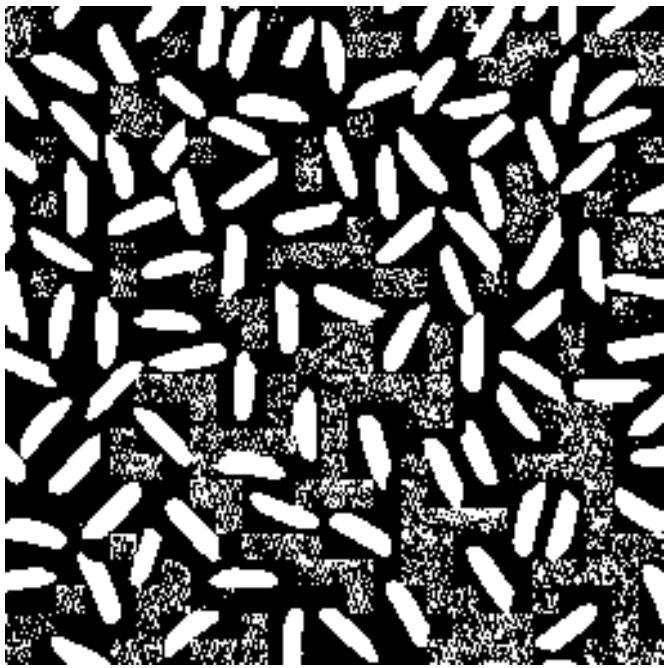
- Generalizar o caso anterior e criar uma função para binarizar uma imagem em NxM regiões similares: `B=MultiRegionBin(A,N,M)`:
  - A $\rightarrow$ imagem original a binarizar
  - N $\rightarrow$ número de linhas de regiões
  - M $\rightarrow$ número de colunas de regiões
- Exemplo com N=5 e M=3:



Exercício 6) – Notas adicionais

- Observa-se que o aumento do número de regiões de binarização melhorou o processo de binarização.
- Porém, pode perder eficácia para um elevado número de regiões.
- Se se optasse por um valor exagerado como, por exemplo 25x25 regiões, o resultado agrava-se como se pode verificar.

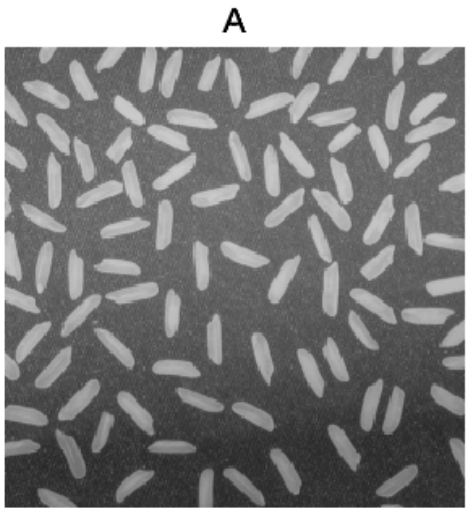


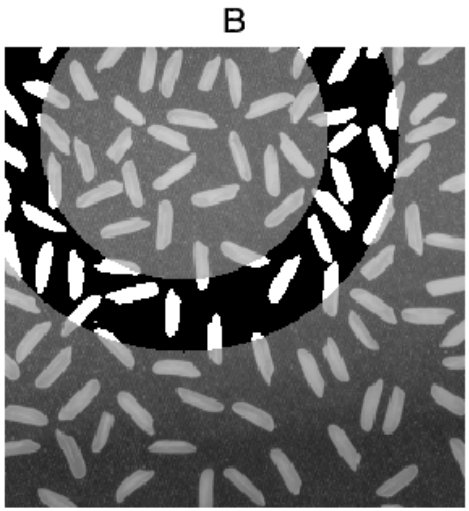


- Em geral, o processo perde eficácia quando a região não cobre partes suficientes de objeto e de fundo; nem só de um nem só de outro!

Exercício 7) – Opcional**

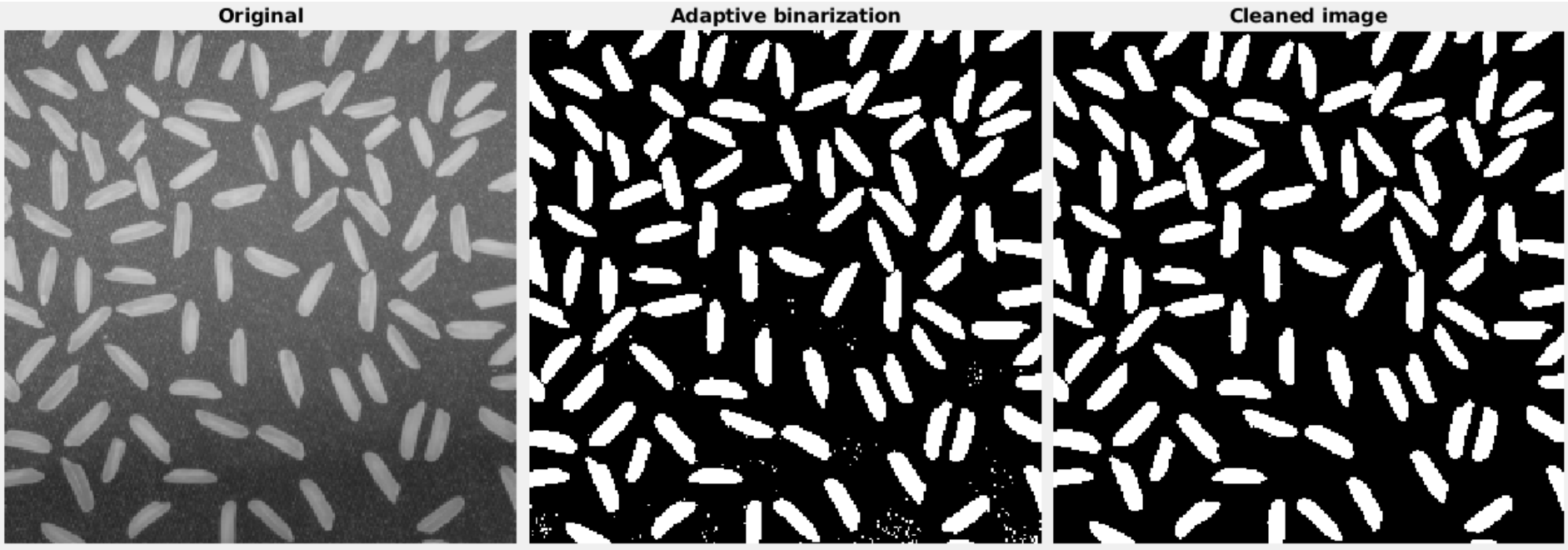
- Criar a função `B=autobinwithmask(A,M)`
  - A=Imagem a binarizar. M=máscara binária das mesmas dimensões. B=imagem binarizada apenas na zona equivalente da máscara M.
  - O limiar de binarização deve ser do tipo de Otsu e calculado apenas pelos pixels cobertos pela máscara.
- Criar a função `M=circularROI(x0,y0,ri,re,A)`
  - M=máscara binária de dimensões iguais às de A e em forma de coroa circular centrada no ponto `(x0,y0)` e com raio interno `ri` e externo `re`.
- Binarizar a imagem 'rice.png' com uma ROI em forma de coroa circular centrada em `(100,50)` com `ri`=80 e `re`=120.





Ex. 8) – Binarização adaptativa simples

- Usar binarização adaptativa com os parâmetros de defeito para obter uma versão da imagem "rice.png" sem as limitações que a binarização global apresenta.



- Como se poderia "limpar" a imagem resultante para ficarem apenas os objetos de grãos de arroz como ilustrado à direita?

Ex. 9) – Binarização adaptativa parametrizada

- Fazer uma binarização adaptativa para obter a máscara da direita a partir da imagem samples2.png à esquerda.
- Usar a função **adaptthresh()** com a sensibilidade adequada (limiar entre 0 e 1 para definir um pixel como objeto) e com a vizinhança mais adequada.

- NB. A função **adaptthresh()** devolve uma matriz T de limiares de binarização para cada pixel que será depois usada pela operação **imbinarize()**.
- Recomenda-se o uso da função **bwareaopen()** para eliminar "pequenos" objetos. Na ilustração foram eliminados os objetos com menos de 100 pixels!