

Worksheet4_pdfversion

Creation date: 2024-12-12 18:08

Modification date: Thursday 12th December 2024 18:08:01

PDF Ref:

Morfologia Matemática

Código para as operações:

```
morfologia

1 Mat dilatation(Mat& image, Mat structuringElement) {
2     Mat imageT;
3     dilate(image, imageT, structuringElement);
4     return imageT;
5 }
6 Mat erosion(Mat& image, Mat structuringElement) {
7     Mat imageT;
8     erode(image, imageT, structuringElement);
9     return imageT;
10 }
11 Mat closing(Mat& image, Mat structuringElement) {
12     Mat imageT, imageT2;
13     dilate(image, imageT, structuringElement);
14     erode(imageT, imageT2, structuringElement);
15     return imageT2;
16 }
17 Mat opening(Mat& image, Mat structuringElement) {
18     Mat imageT, imageT2;
19     erode(image, imageT, structuringElement);
20     dilate(imageT, imageT2, structuringElement);
21     return imageT2;
22 }
```

1. Utilize os conceitos de morfologia matemática para obter a imagem "BinaryImg_processed.png" a partir da imagem "BinaryImg.png".

Código:

```
exercise1

1 void exercise1_ws4(const string& imagePath, const string& imageName) {
2
3     Mat image = imread(imagePath, IMREAD_GRAYSCALE);
4
5     // Define the size of the structuring element
6     Size ksizeRect(3, 3);
7     Mat structElementRect = getStructuringElement(MORPH_RECT, ksizeRect);
8     // Define the size of the structuring element
9     Size ksizeLine(3, 1); // to make a line
10    Mat structElementLine = getStructuringElement(MORPH_RECT, ksizeLine);
11
12    // Remove points from the image
13    Mat outputErosionImage = closing(image, structElementRect);
14    // Fill the line in the first rectangle
15    Mat outputDilationImage = opening(outputErosionImage, structElementRect);
16    // Fix the line between rectangles
17    Mat outputDilationImage2 = opening(outputDilationImage, structElementLine);
18
19    imshow(imageName, outputDilationImage2);
20    waitKey(0);
21 }
```

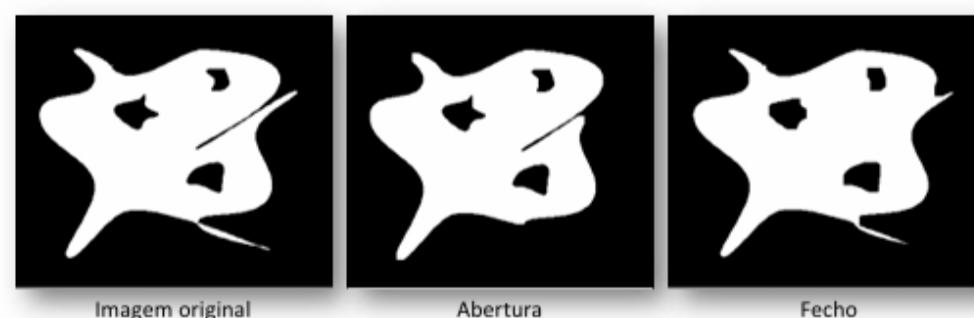
Referências:

Abertura:

- Erosão seguida de uma dilatação.
- Remover pequenas regiões.
- Calcula o valor máximo da imagem A, na região definida pelo elemento estruturante B, e atribui esse valor ao pixel central da região.

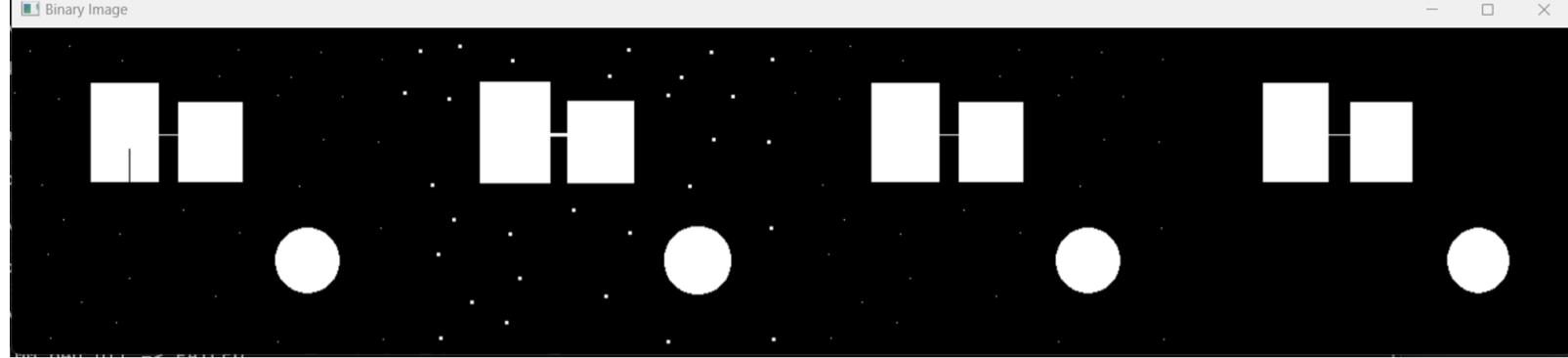
Fecho:

- Dilatação seguida de uma erosão
- Preenchimento de falhas
- Calcula o valor mínimo da imagem A, na região definida pelo elemento estruturante B, e atribui esse valor ao pixel central da região



Pasted image 20241212173009.png#center

Resultados:



Pasted image 20241212184624.png#center

Código:

- Leitura de uma imagem.
- Criação de elementos estruturantes do tipo retângulo (um deles estilo uma linha).
- Aplicação das funções de closing e opening consecutivamente até à imagem final
- Display da imagem.

Processo:

- Dois elementos estruturantes foram criados conforme os elementos da imagem original. Um com forma retangular 3x3 e outro com a mesma forma mas 3x1, que é abordagem possível para obter uma linha.
- Inicialmente realizou-se uma abertura para permitir preencher a falha dentro do retângulo mais à esquerda. No entanto, todos os elementos da imagem ficaram maiores devido ao efeito da dilatação.
- Para corrigir esse efeito sem perder a correção da falha dentro do retângulo, foi realizado um fecho com o mesmo elemento estruturante. Os pontos exteriores diminuíram, mas ainda estão presentes.
- Se for aplicado mais um fecho com o mesmo elemento 3x3, a linha entre os retângulos vai desaparecer. Para isso não acontecer, o elemento agora é uma linha de 3x1, que vai permitir eliminar apenas os pontos.

Ao passar pela linha:

→ todos elementos a 1, mas elimina o central
Valor máximo da imagem dentro do elemento

Ao passar pelos pontos:

→ pelo menos 1 elemento a 0.
Elimina o central
Valor mínimo da imagem dentro do elemento

SmartSelect_20241212_190338_Samsung Notes 1.png#center

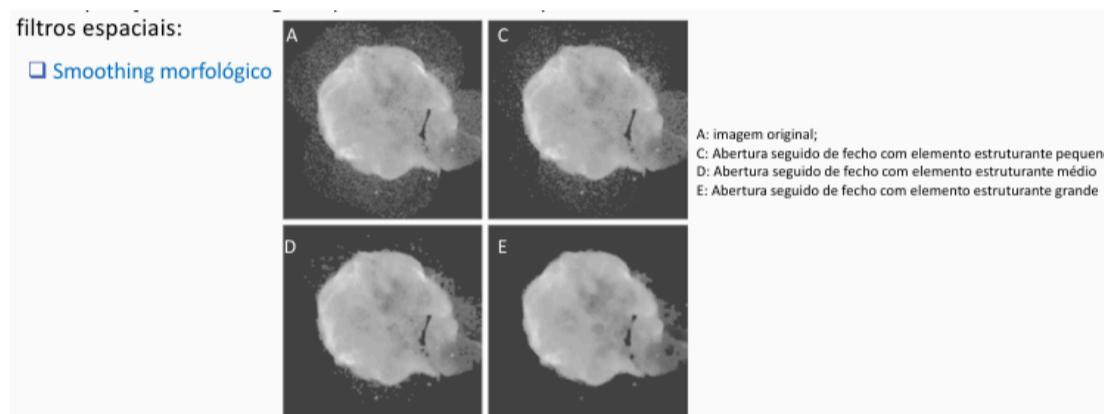
2. Suavize a imagem "head_gaussian_noise.png" utilizando a morfologia matemática. Experimente diferentes tamanhos de kernel para um elemento de estruturação de disco. Analise os resultados.

Código:

Smoothing

```
1 void exercise2_ws4(const string& imagePath, const string& imageName, Mat structuringElement) {  
2  
3     Mat image = imread(imagePath, IMREAD_GRAYSCALE);  
4  
5     // Electronic smoothing  
6     Mat outputOpeningImage = opening(image, structuringElement);  
7     Mat outputClosingImage = closing(outputOpeningImage, structuringElement);  
8  
9     Mat outputImage1, outputImage2, outputImage3;  
10    hconcat(image, outputClosingImage, outputImage2);  
11  
12    imshow(imageName, outputImage2);  
13    waitKey(0);  
14 }  
15
```

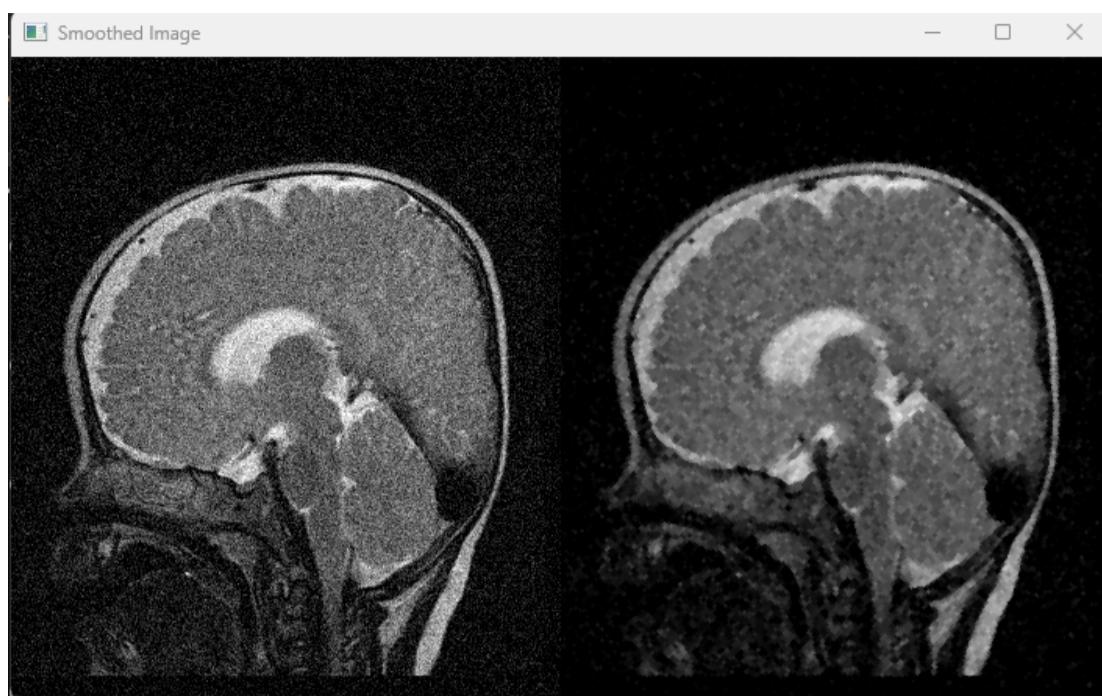
Referências:



Pasted image 20241212173156.png#center

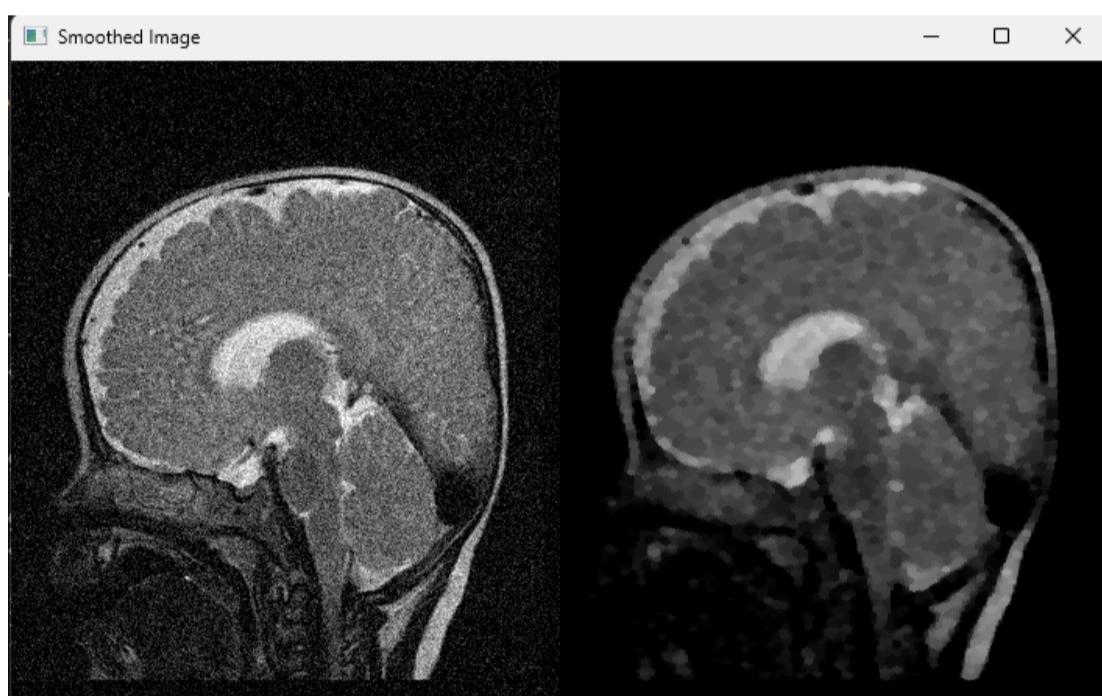
Resultados:

- Elemento estruturante 3x3 (pequeno):



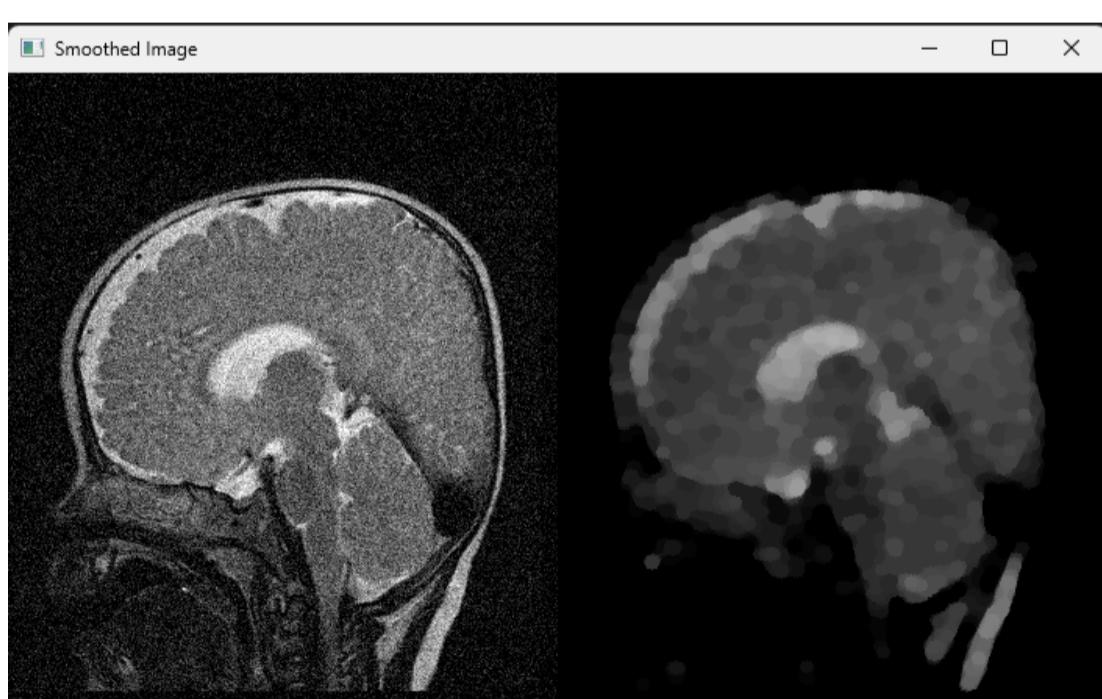
Pasted image 20241214131709.png#center

- Elemento estruturante 5x5 (médio):



Pasted image 20241214131839.png#center

- Elemento estruturante 9x9 (grande):



Pasted image 20241214131918.png#center

Código:

- Leitura da imagem em greyscale.
- Aplicação de morfologia matemática através das funções de opening e closing, com um elemento estruturante definido pelo utilizador.
- Display da imagem

Smoothing:

- O efeito de smoothing da imagem trata-se da aplicação de uma abertura seguida de um fecho, com um elemento estruturante de tamanho e tipo arbitrários (neste caso o tipo foi sempre uma elipse/disco).
- Quanto maior o elemento estruturante, maior a suavização da imagem e eliminação do ruído.
- No primeiro teste, os contornos e os detalhes da imagem original ainda são preservados, mas há uma leve redução no ruído.
- No entanto, ao usar um elemento estruturante cada vez maior, os maiores elementos da imagem vão ser afetados, e a imagem fica demasiado suavizada e perde os seus detalhes, ficando praticamente imperceptível.

3. Usando o conceito de filtragem e os filtros de Sobel, detete os contornos das imagens obtidas na questão 2. Analise os resultados.

Código:

SobelFiltering

```
1 void exercise5_ws3(const string& imagePath, const string& imageName) {
2
3     Mat image = imread(imagePath, IMREAD_GRAYSCALE);
4     Mat paddedImage;
5     int pad = 3 / 2; // Kernel size / 2
6     copyMakeBorder(image, paddedImage, pad, pad, pad, pad, BORDER_CONSTANT);
7
8     // Intermediate matrices for Sobel filter results
9     Mat filterImageHorizontal(image.rows, image.cols, CV_32F, Scalar(0));
10    Mat filterImageVertical(image.rows, image.cols, CV_32F, Scalar(0));
11    Mat filterImageMagnitude(image.rows, image.cols, CV_32F, Scalar(0));
12
13    // Sobel kernels
14    float kernelx[3][3] = { {1, 0, -1},
15                           {2, 0, -2},
16                           {1, 0, -1} };
17
18    float kernely[3][3] = { {1, 2, 1},
19                           {0, 0, 0},
20                           {-1, -2, -1} };
21
22    for (int r = pad; r < paddedImage.rows - pad; r++) {
23        for (int c = pad; c < paddedImage.cols - pad; c++) {
24            float sum_x = 0.0, sum_y = 0.0; // Sums for both axes
25
26            for (int kx = -pad; kx <= pad; kx++) { // Kernel x
27                for (int ky = -pad; ky <= pad; ky++) { // Kernel y
28                    float pixel_value = static_cast<float>(paddedImage.at<uchar>(r + kx, c +
ky));
29                    sum_x += pixel_value * kernelx[kx + pad][ky + pad]; // Convolution with
Sobel X
30                    sum_y += pixel_value * kernely[kx + pad][ky + pad]; // Convolution with
Sobel Y
31                }
32            }
33
34            // Sobel results
35            filterImageHorizontal.at<float>(r - pad, c - pad) = sum_x;
36            filterImageVertical.at<float>(r - pad, c - pad) = sum_y;
37
38            // Magnitude (combined Sobel X and Y results)
39            float magnitude = sqrt(sum_x * sum_x + sum_y * sum_y);
40            filterImageMagnitude.at<float>(r - pad, c - pad) = magnitude;
41        }
42    }
43
44    // Low-pass filter results
45    Mat lowPassSobelHorizontal = lowpassFilter(image, filterImageHorizontal, pad);
46    Mat lowPassSobelVertical = lowpassFilter(image, filterImageVertical, pad);
47    Mat lowPassSobelMagnitude = lowpassFilter(image, filterImageMagnitude, pad);
48
49    // Normalize low-pass results for visualization
50    Mat lowPassHorizontalDisplay, lowPassVerticalDisplay, lowPassMagnitudeDisplay;
51    normalize(lowPassSobelHorizontal, lowPassHorizontalDisplay, 0, 255, NORM_MINMAX, CV_8UC1);
52    normalize(lowPassSobelVertical, lowPassVerticalDisplay, 0, 255, NORM_MINMAX, CV_8UC1);
53    normalize(lowPassSobelMagnitude, lowPassMagnitudeDisplay, 0, 255, NORM_MINMAX, CV_8UC1);
54
55    // Combine images for visualization
56    Mat combinedImage, combinedImage2, combinedImage3;
57    hconcat(image, lowPassHorizontalDisplay, combinedImage);
58    hconcat(combinedImage, lowPassVerticalDisplay, combinedImage2);
59    hconcat(combinedImage2, lowPassMagnitudeDisplay, combinedImage3);
```

```
59
60     // Add labels to identify each section
61     putText(combinedImage3, "Original", Point(50, 50), FONT_HERSHEY_SIMPLEX, 1, Scalar(255, 255,
62     255), 2);
62     putText(combinedImage3, "Horizontal", Point(450, 50), FONT_HERSHEY_SIMPLEX, 1, Scalar(255,
63     255, 255), 2);
63     putText(combinedImage3, "Vertical", Point(800, 50), FONT_HERSHEY_SIMPLEX, 1, Scalar(255, 255,
64     255), 2);
64     putText(combinedImage3, "Magnitude", Point(1100, 50), FONT_HERSHEY_SIMPLEX, 1, Scalar(255,
65     255, 255), 2);
65
66     // Display the final combined image
67     imshow(imageName, combinedImage3);
68     waitKey(0);
69 }
```

Referências:

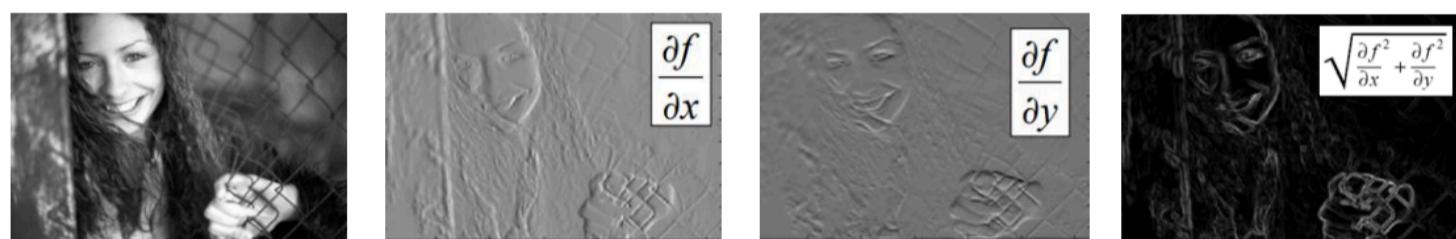
- ❑ Calcula o gradiente da imagem na direção X ou Y
 - Primeira derivada na respetiva direção
- ❑ Deteção de contornos
- ❑ Muitas vezes precedido por um filtro passa-baixo para eliminar ruído

Pasted image 20241202142414.png#center

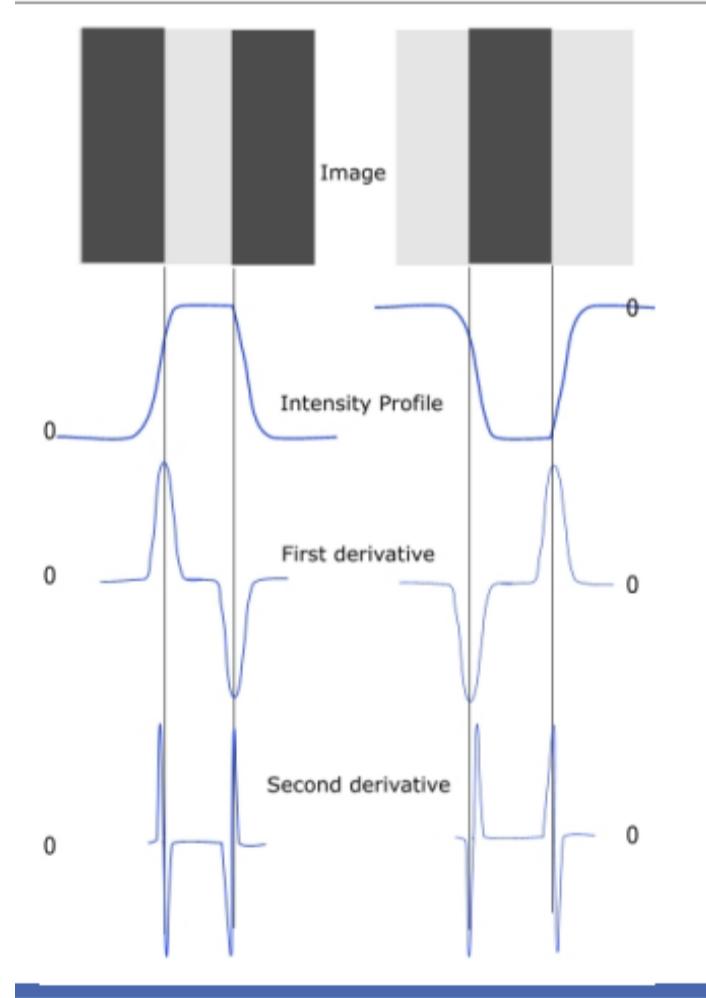


Pasted image 20241202115858.png#center

GRADIENTE



Pasted image 20241203152114.png#center

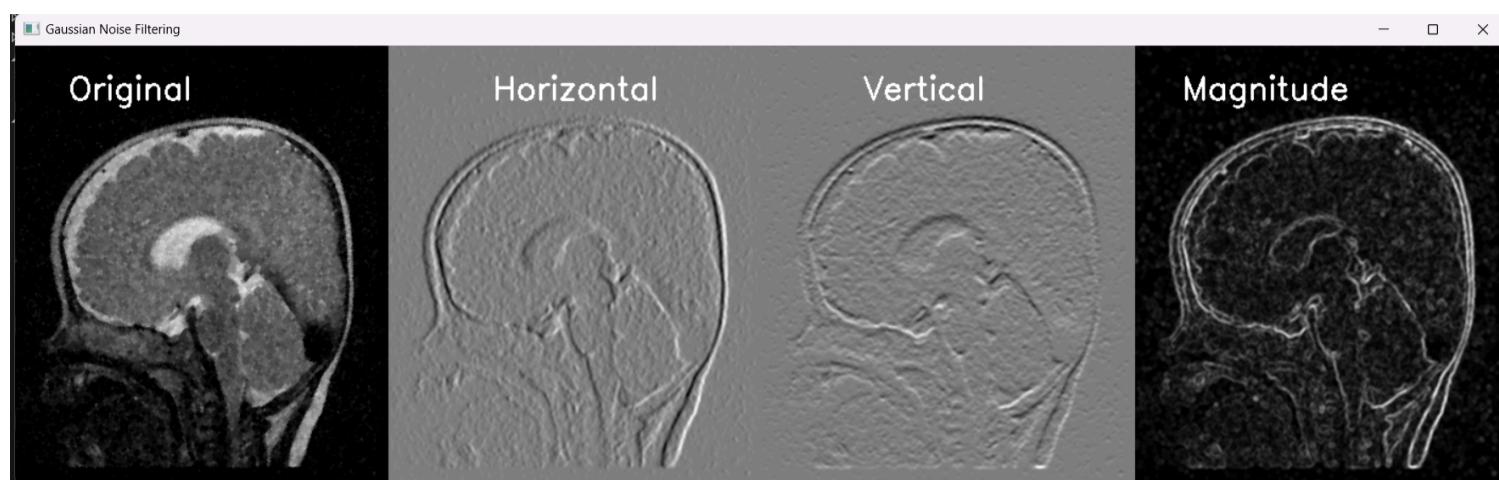


Pasted image 20241204145735.png#center

Resultados:

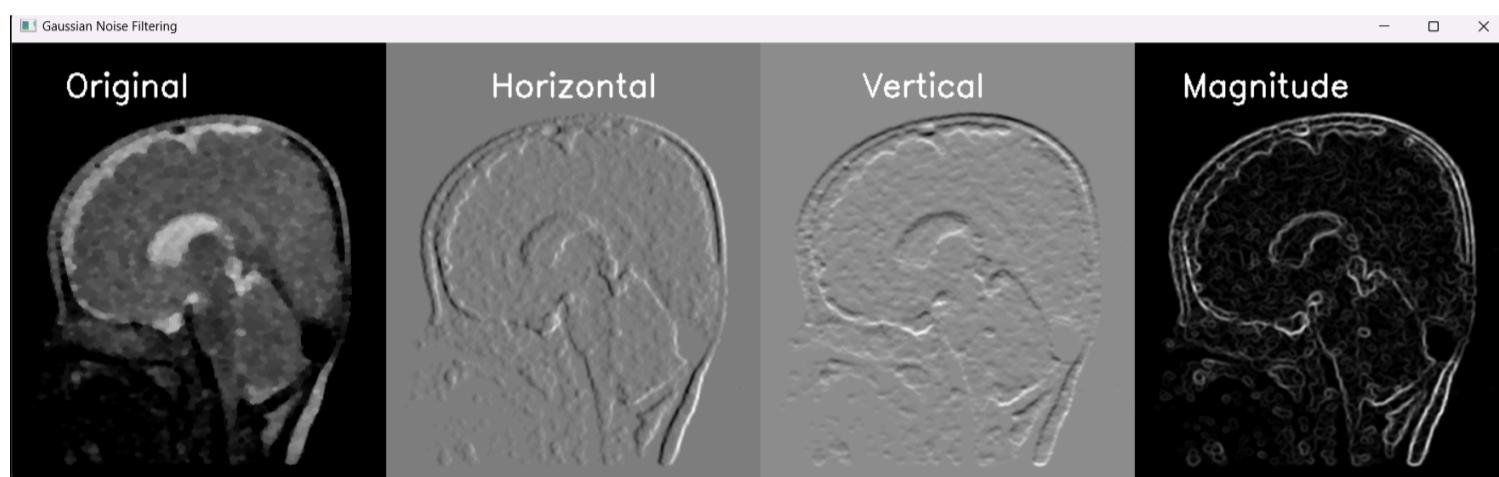
Com filtro passa baixo e normalização:

- Elemento Estruturante 3x3:



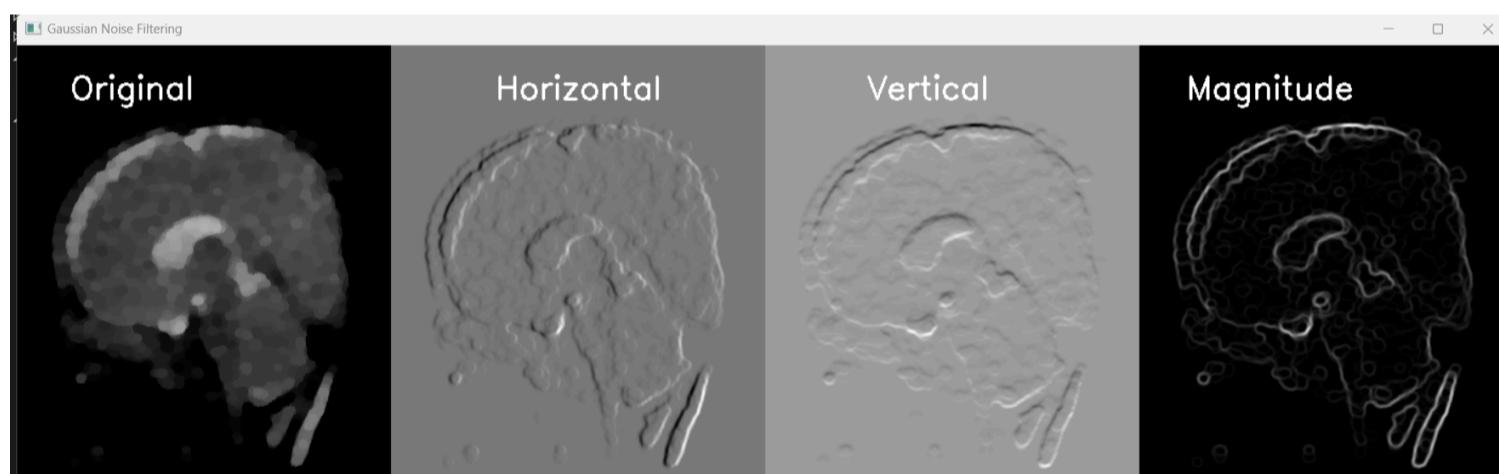
Pasted image 20241214163004.png#center

- Elemento Estruturante 5x5:



Pasted image 20241214163103.png#center

- Elemento Estruturante 9x9:



Pasted image 20241214163132.png#center

Código:

- Realiza-se uma leitura de imagem original, e de seguida é criada uma cópia com padding.
- Criou-se também três matrizes tipo float (porque tratam com valores positivos e negativos) que vão receber os resultados das direções e da magnitude.
- Os kernels são os recomendados pelos slides, conforme é possível ver nas referências.
- De seguida é realizada uma convolução entre a imagem e os kernels, para x e y.
- E no final da operação a cada pixel, é realizada o cálculo da magnitude dos valores.
- Os respetivos valores de x, y e magnitude são copiados para as 3 matrizes criadas.
- Depois as imagens são filtradas para eliminar o ruído (por causa da recomendação nos slides), através de uma função criada para o propósito, que recebe como parâmetros a imagem original, a imagem a aplicar o filtro e o valor do pad (metade do kernel).
- Os valores são todos normalizados (para estarem num intervalo visualizável através do openCV).

Sobel:

- Como é possível verificar na imagem de output, os contornos são detetados com sucesso, em ambas as direções, e na magnitude dos valores.
- Todas as zonas da imagem original onde existe uma transição rápida dos valores de intensidade, são "detetados" como contornos, em ambas as direções.
- **(No filtro de Sobel, a primeira derivada é calculada tanto na direção horizontal quanto vertical, identificando transições no brilho da imagem original.)**
- As zonas brancas nos contornos do crânio, são zonas onde existem essas mudanças abruptas de intensidade.
- Todas as imagens passaram por um filtro passa-baixo.

Análise das imagens:

- Facilmente verifica-se o efeito do ruído nas imagens pós aquisição dos contornos, onde é mais evidente na imagem de smoothing com elemento estruturante mais baixo (3x3).
- A imagem de 5x5 é a melhor transformação, onde todo o ruído foi eliminado fora do crânio ao obter os contornos. No entanto, dentro da zona de interesse ainda é possível encontrar algum ruído.
- No maior kernel, tal como visto anteriormente, o efeito é muito intenso e as zonas de interesse da imagem são afetadas pela suavização excessiva.

4. Usando o conceito de morfologia matemática, detete os contornos das imagens obtidas na questão 2. Analise os resultados.

Código:

```
OutlineDetection

1 void exercise4_ws4(const string& imagePath, const string& imageName, Mat structuringElement, Mat
2 structuringElement2) {
3
4     Mat image = imread(imagePath, IMREAD_GRAYSCALE);
5
6     // Dilation
7     Mat imageDilation = dilatation(imageClosing, structuringElement2);
8     // Erosion
9     Mat imageErosion = erosion(imageClosing, structuringElement2);
10
11    // new empty image
12    Mat result = Mat::zeros(imageErosion.size(), imageErosion.type());
13
14    for (int r = 0; r < imageErosion.rows; r++) {
15        for (int c = 0; c < imageErosion.cols; c++) {
16            result.at<uchar>(r, c) = static_cast<int>(imageDilation.at<uchar>(r, c)) -
17            imageErosion.at<uchar>(r, c);
18        }
19    }
20
21    Mat outputImage1, outputImage2, outputImage3, outputImage4;
22    hconcat(image, result, outputImage1);
23    //hconcat(outputImage1, imageDilation, outputImage2);
24    //hconcat(outputImage2, imageErosion, outputImage3);
25    //hconcat(outputImage3, result, outputImage4);
26
27    imshow(imageName, outputImage1);
28    waitKey(0);
29 }
```

Referências:

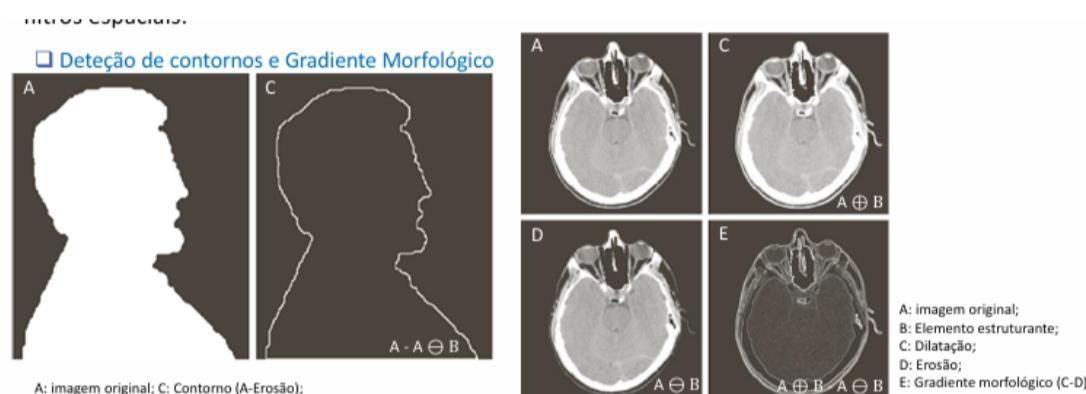
[Detecção de Bordas e Transformações Morfológicas em Imagens com OpenCV | by Luísa Mendes Heise | Turing Talks | Medium](#)



Pasted image 20241214185356.png#center

Para ter o contorno:

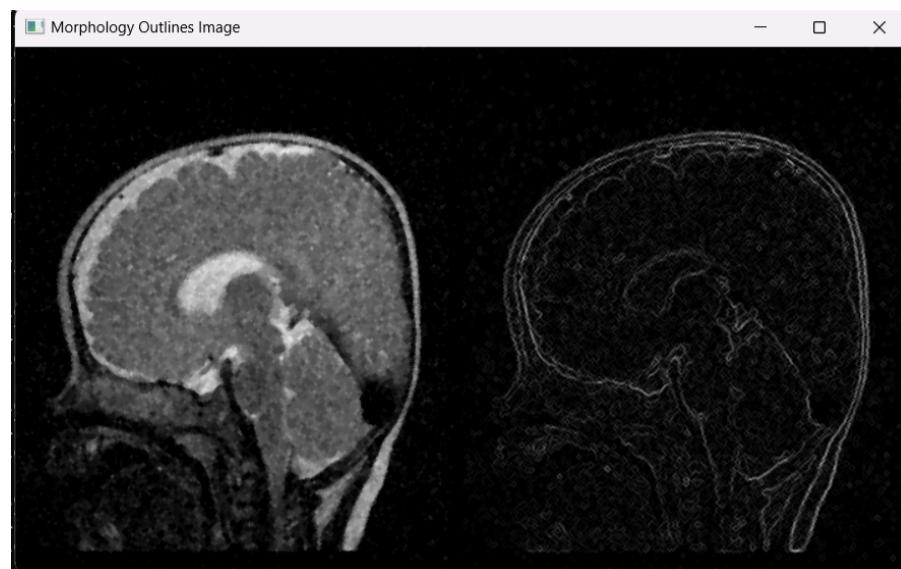
- Fazer uma dilatação na imagem original e depois uma erosão também, e depois subtrair as duas.
- É possível também erodir e subtrair a original com a erodida, para ter o contorno interno.



Pasted image 20241212173126.png#center

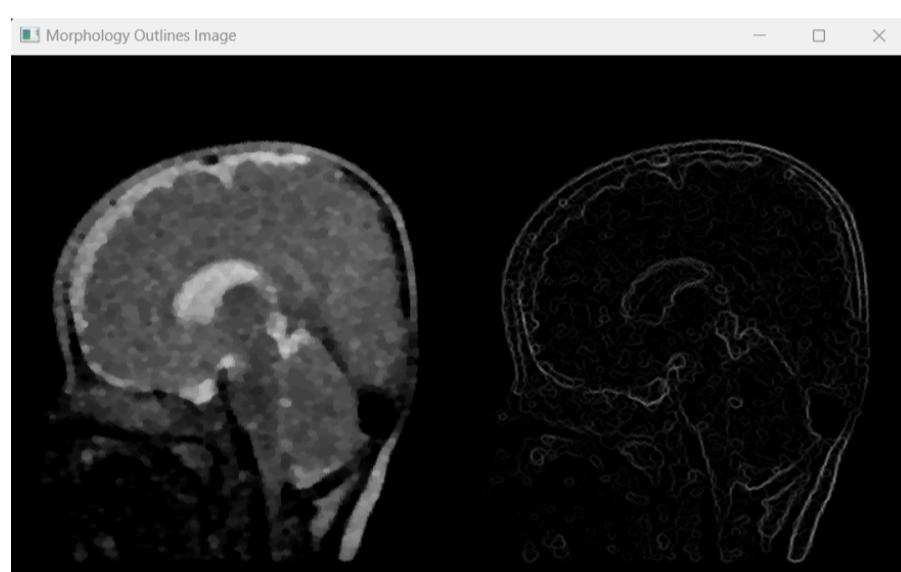
Resultados:

- Imagem smooth 3x3 com elemento estruturante 3x3 tipo elipse:



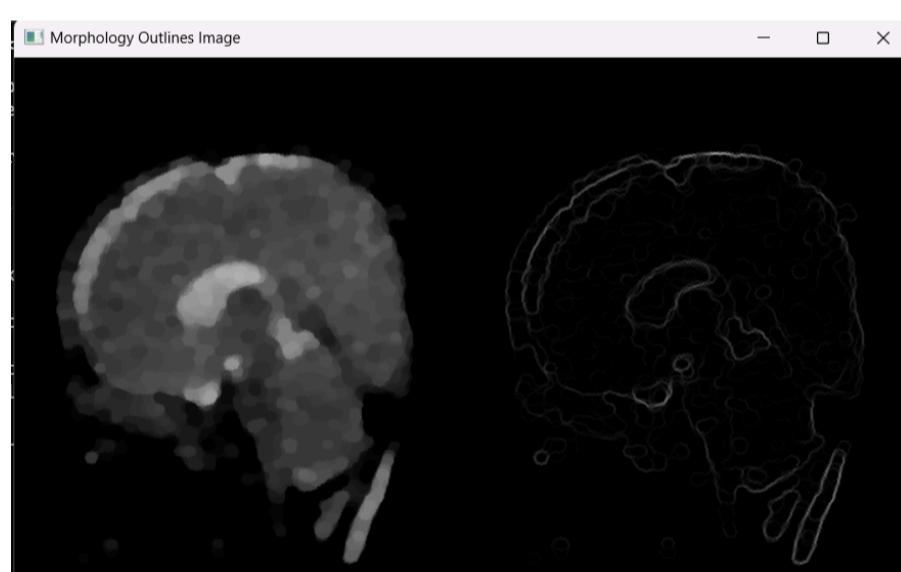
Pasted image 20241214165516.png#center

- Imagem smooth 5x5 com elemento estruturante 3x3 tipo elipse:



Pasted image 20241214174537.png#center

- Imagem smooth 9x9 com elemento estruturante 3x3 tipo elipse:



Pasted image 20241214174617.png#center

Código:

- Operações de dilatação da imagem seguida de uma erosão com o mesmo elemento estruturante de 3x3 (tipo elipse), que são aplicadas à imagem original.
- For loops que percorrem a imagem pixel a pixel e "transferem" todos os valores da intensidade para a uma imagem de zeros. O valor da intensidade é o resultado da diferença entre as operações de dilatação e erosão aplicadas à imagem original.
- Display dos resultados.

Contornos:

- O processo de deteção de contornos em morfologia matemática não segue o mesmo princípio que a filtragem, que utiliza kernels específicos (por exemplo o Sobel).
- Na morfologia, a técnica usada é aplicar uma dilatação e uma erosão (processos distintos) à imagem que vai ser analisada, e posteriormente fazer a diferença entre as duas imagens.
- A dilatação vai "expandir" essas bordas e a erosão vai "eliminar" essas bordas. Logicamente, a diferença entre o processo de dilatação e o de erosão, resulta nas bordas da imagem, mais propriamente os contornos.
- Comparando os outputs das duas implementações (filtro Sobel vs Gradiente Morfológico), a filtragem Sobel parece ser a melhor opção e onde os contornos ficam mais destacados, visto que se trata de uma método mais sensível a mudanças suaves na intensidade dos pixels.
- No entanto é mais suscetível ao ruído que a operação morfológica (com o mesmo kernel/elemento estruturante).

5. Construa uma pipeline para extrair os rins da imagem "kidney.png". Deve começar por binarizar a imagem usando um threshold adequado. Posteriormente, deve aplicar pós-processamento por morfologia matemática para obter uma máscara semelhante à imagem "kidney_segmented.png".

Código:

```
pipelineKidney

1 void exercise5_ws4(const string& imagePath, const string& imageName) {
2
3     Mat image = imread(imagePath, IMREAD_GRAYSCALE);
4
5     Mat binaryImage = Mat::zeros(image.size(), image.type()); // Initialize with zeros (all
6     black)
7
8     for (int r = 0; r < image.rows; r++) {
9         for (int c = 0; c < image.cols; c++) {
10             uchar pixelValue = image.at<uchar>(r, c);
11             if (pixelValue > 200) {
12                 binaryImage.at<uchar>(r, c) = 255; // Set to white
13             }
14         }
15     }
16
17     Mat strucElement1 = structuringElement(25,1);
18     Mat strucElement2 = structuringElement(47,1);
19     Mat strucElement3 = structuringElement(40,1);
20
21     Mat imageClosing = closing(binaryImage, strucElement2);
22     Mat imageOpening = opening(imageClosing, strucElement1);
23     //Mat imageDilation = dilatation(imageOpening, strucElement1);
24     Mat imageOpening2 = opening(imageOpening, strucElement3);
25
26     Mat resizedImage, resizedBinaryImage, resizedImageOpening, resizedImageDilation,
27     resizedImageClosing;
28     Mat resizedImageOpening2;
29
30     // Resize images to make them smaller
31     resize(image, resizedImage, Size(), 0.5, 0.5);
32     resize(binaryImage, resizedBinaryImage, Size(), 0.5, 0.5);
33     resize(imageOpening, resizedImageOpening, Size(), 0.5, 0.5);
34     resize(imageClosing, resizedImageClosing, Size(), 0.5, 0.5);
35     resize(imageOpening2, resizedImageOpening2, Size(), 0.5, 0.5);
36
37     // Concatenate horizontally
38     Mat outputImage1, outputImage2, outputImage3, outputImage4;
39     //hconcat(resizedImage, resizedBinaryImage, outputImage1);
40     //hconcat(outputImage1, resizedImageClosing, outputImage2);
41     //hconcat(outputImage2, resizedImageOpening, outputImage3);
42
43     // Display the result
44     imshow(imageName, resizedImageOpening2); // only showing the final output
        waitKey(0);
}
```

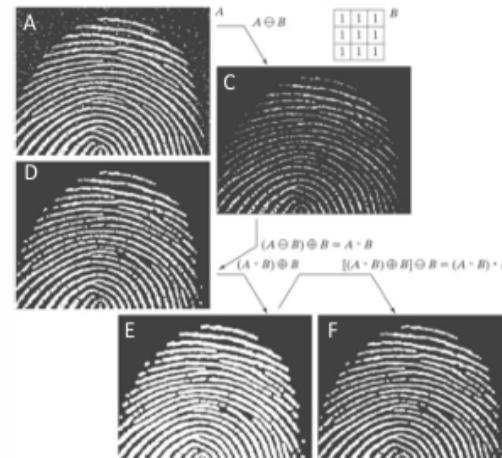
Referências:

Binarizar uma imagem usando um **threshold** consiste em converter a imagem em tons de cinza para apenas dois valores: preto (0) e branco (255). O valor do threshold serve como limite: pixels com intensidade acima do limite serão convertidos para branco, e os abaixo, para preto.

Pasted image 20241215194846.png#center

filtros espaciais:

Remoção de ruído



A: imagem original;
B: Elemento estruturante;
C: Erosão;
D: Abertura;
E: Dilatação de D
F: Fecho de E

Pasted image 20241212173148.png#center

Abertura:

- Erosão seguida de uma dilatação.
- Remover pequenas regiões.

Fecho:

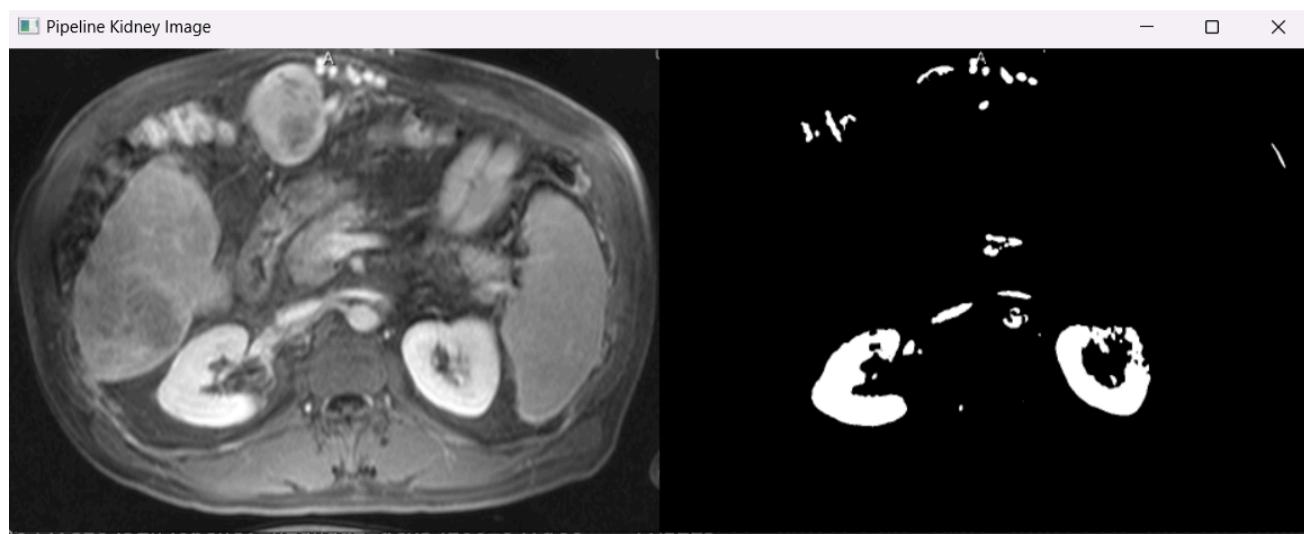
- Dilatação seguida de uma erosão.
- Preenchimento de falhas.



kidney_segmented.png#center

Resultados:

- Binarização:



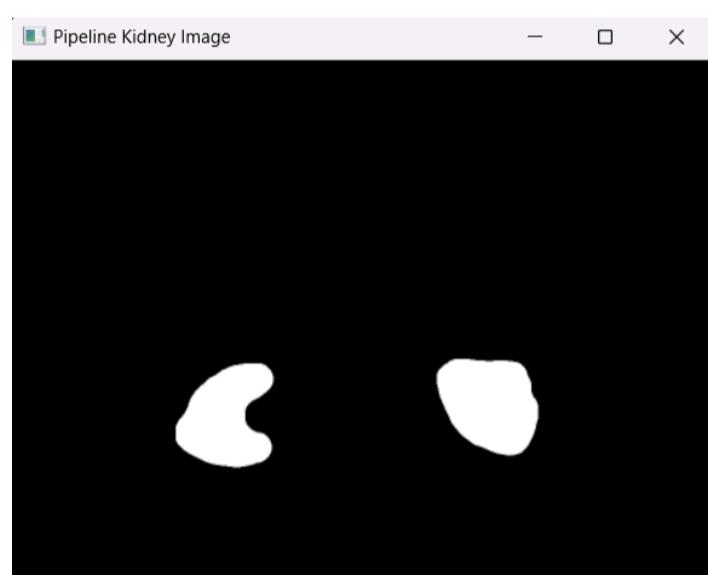
Pasted image 20241215200208.png#center

- Closing 47x47 → Opening 25x25 → Opening2 40x40 (todos elipse):



Pasted image 20241215200504.png#center

- Final:



Pasted image 20241215195132.png#center

Código:

- Leitura de uma imagem em greyscale.
- Criação de uma imagem zeros para "transferir" as intensidades dos pixels dentro do threshold definido (200).
- For loops que percorrem a imagem pixel a pixel e transferem os valores das intensidades para a imagem binária.
- Aplicação de 3 operações morfológicas diferentes com 3 elementos estruturantes diferentes.
- Resize das imagens para caber na concatenação.
- Display de todas as transformações.

Pipeline:

- A imagem é lida através do imread() em greyscale, para poder ser binarizada através de dois for loops que analisam pixel a pixel, todos os valores de intensidade.
- Todos os valores superiores ao threshold (200), são "convertidos e transferidos" para o valor 255 (branco) na imagem de zeros criada.
- Depois através de tentativa e erro, criou-se elementos estruturantes de um certo tamanho para aplicar as operações morfológicas e isolar os rins na imagem.
- Inicialmente aplicou-se um closing com o intuito de preencher falhas, mais propriamente, preencher o espaço preto dentro do objeto/rim da direita. E também as pequenas falhas no rim da esquerda. (e.e → 25x25)
- Após a primeira operação, realizou-se um opening para eliminar pequenas regiões correspondentes a tudo menos a zona de interesse. Zonas onde a intensidade dos pixels estão dentro do threshold, mas não são os rins. (e.e → 47x47)
- Para finalizar, aplicou-se novamente um opening para remover as áreas e isolar as duas zonas de interesse. (e.e → 40x40)
- O elemento estruturante para o closing não é tão grande quanto os outros, devido à facilidade de preenchimento das zonas (pequenas falhas)
- Os outros dois foram quase o dobro, para tentar remover os blobs sem interesse, que eram relativamente grandes.

Resultado vs Expectado:

- A imagem final está bastante próxima da imagem segmentada de referência, no entanto não acredito que fosse possível chegar ao resultado apenas com aplicação das operações morfológicas.
- O threshold também tem bastante influência no pipeline, porque é esse limiar que vai ditar as intensidades que vão ser "copiadas" para branco na nova imagem. Quanto maior o valor, algumas zonas do rim perdiam-se, porque a intensidade não era uniforme. Quanto menor o valor, mais zonas sem interesse apareciam, o que ia obrigar a realizar ainda mais operações para segmentar a imagem.
- Acredito que a melhor solução seria um closing inicial para permitir preencher as falhas nos rins, com um elemento estruturante menor. E de seguida implementar um sistema de transformação dos blobs em objetos e proceder à aquisição de todas as suas áreas. Essas áreas seriam posteriormente avaliadas num loop condicional, e todos os objetos inferiores a uma certa área (inferior à área do rim menor) eram eliminados da imagem.