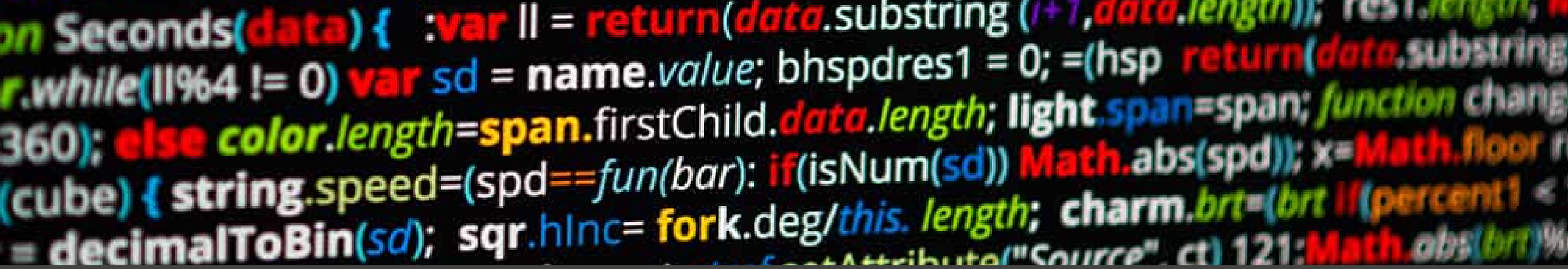


PROCESSAMENTO DE IMAGEM

LAB 03
FILTRAGEM ESPACIAL

Vitória Maria Bezerra
Erick Mendonça
Welainny Viana
Engenharia Biomédica

Problema??

Implementar alguns filtros para remoção de ruídos de imagens e avaliar o desempenho dos mesmos no tocante à comparação de métricas de desempenho.

IMAGENS:

- 1 imagem original
- 9 imagens com ruídos aleátorios

Resolução

- Programa em Java com o auxilio do ImageJ

FILTROS A SEREM IMPLEMENTADOS:

- Gaussian Blur(GB),
- Moving Average
- Median (Med),
- Wiener (Wien)/ Lee filter
- Interference based speckle filter (IBSF)

MÉTRICAS A SEREM ANALISADAS:

- Root mean squared error (RMSE),
- Structural Similarity Index (SSIM),
- Coeficiente de correlação (r)
- Relação sinal-ruído (SNR)
- Quantidade de pontos de junção (corners) - Harris detector

Algoritmo

1. Implementar cada filtro;
2. Gerar as métricas de cada filtro e fazer a comparação;
3. Analisar os dados, gerar e exportar tabela;
4. Gerar os gráficos boxplot ou vioplot.

Foram criadas 3 classes:

1. Classe principal (Filtros_Gerais)
2. Classe de Filtros (Filtros_Gerais)
3. Classe para obtenção das metrícas (Metricas_Labo3)

Algoritmo

```
public int[][] FiltroWiener(int raio, ImageProcessor ip1, int[][] img, int criar) {

    if (criar == 1) {
        int h = ip1.getHeight();
        int w = ip1.getWidth();
        imagem_AUX = NewImage.createByteImage("Filtro Wiener", w, h, 1, NewImage.FILL_WHITE);
        ip1 = imagem_AUX.getProcessor();
        imagem_AUX.show();

    }
```

Algoritmo

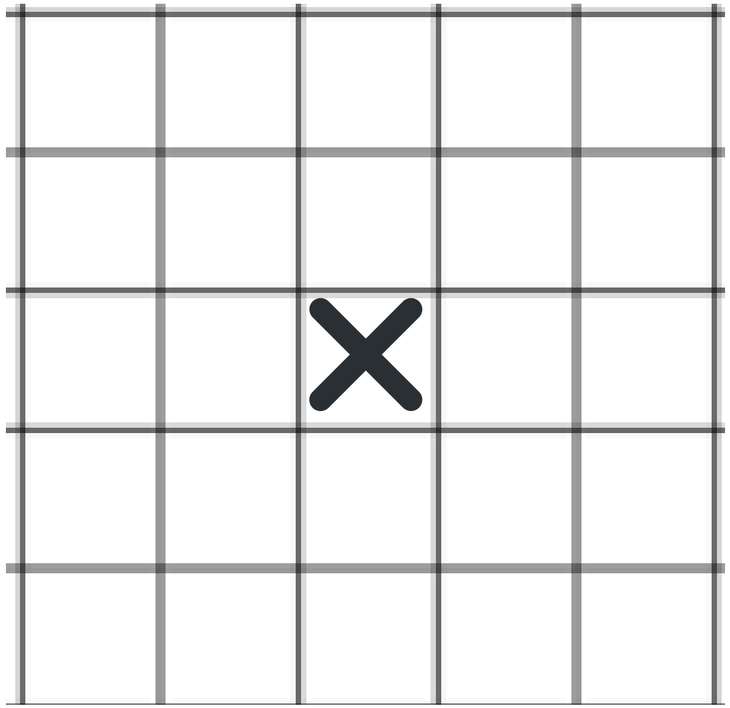
```
for (int x = 0; x < w; x++) {
    for (int y = 0; y < h; y++) {
        valorRef = img[x][y];
        valorFil = img_filtrada[x][y];
        quad = valorRef * valorRef;
        soma1 = soma1 + quad;

        dif = (valorRef - valorFil);
        quad2 = dif * dif;
        soma2 = soma2 + quad2;

    }
}
```

Algoritmo

```
for (int x = 0; x < w; x++) {
    for (int y = 0; y < h; y++) {
        valorRef = img[x][y];
        valorFil = img_filtrada[x][y];
        quad = valorRef * valorRef;
        soma1 = soma1 + quad;

        dif = (valorRef - valorFil);
        quad2 = dif * dif;
        soma2 = soma2 + quad2;

    }
}
```

Filtro Média

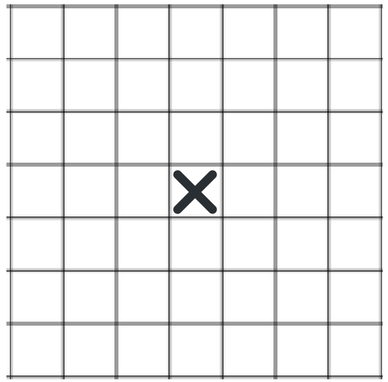
Matriz: 5x5



1. Varrer os vizinhos;
2. Soma a intensidade de vizinhos numa lista;
3. Divide pela quantidade de vizinhos analisados;
4. Pinta o (x,y) com a média;
5. Retorna o ip da imagem filtrada

Filtro Mediana

Matriz: 7x7

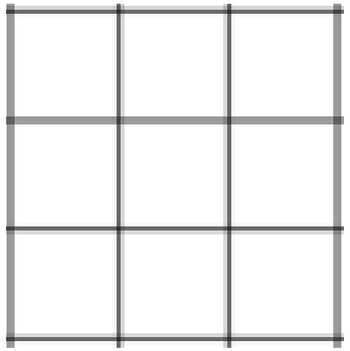


1. Varrer os vizinhos;
2. Adiciona as intensidade dos vizinhos numa lista;
3. Ordena a lista e acha o valor da mediana da lista;
4. Pinta o (x,y) com a mediana;
5. Retorna o ip da imagem filtrada;

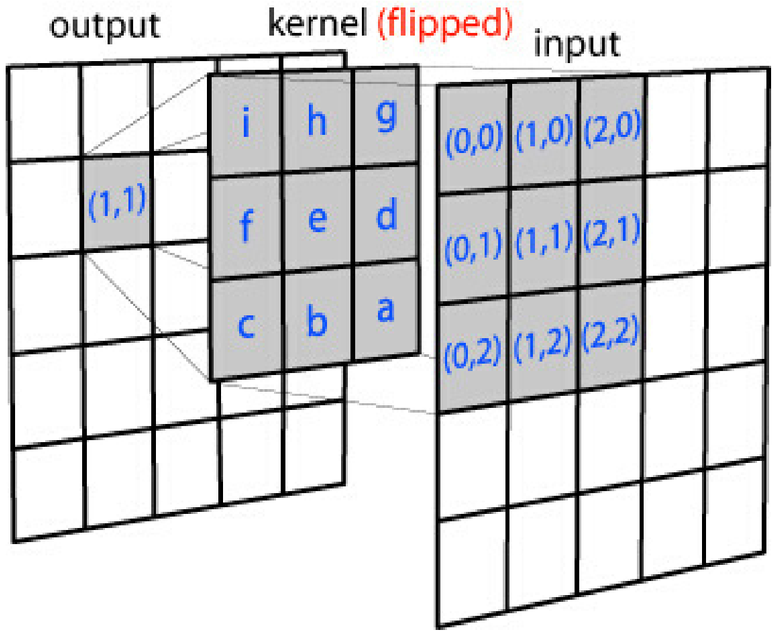
```
if ((size % 2) == 0) {
    int div = size / 2;
    Collections.sort(lista_aux);
    int xx = lista_aux.get(div);
    int yy = lista_aux.get(div - 1);
    int mediana = (xx + yy) / 2;

    ip1.set(x, y, mediana);
}

else {
    int div = size / 2;
    Collections.sort(lista_aux);
    int mediana = lista_aux.get(div);
    ip1.set(x, y, mediana);
}

lista_aux = new ArrayList<Integer>();
```

Filtro Gaussian Blur (GB)

Matriz: 3x3

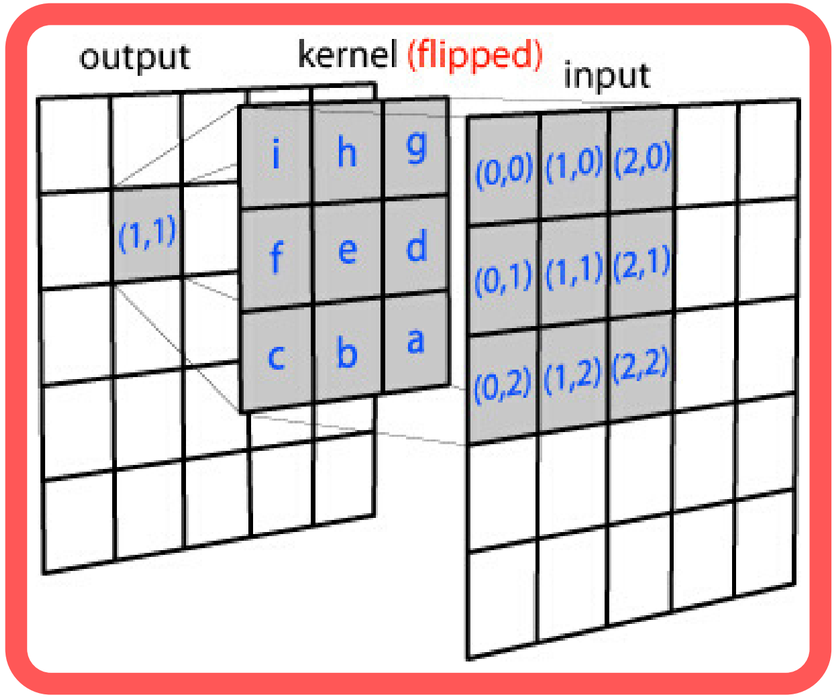


1. Varre os vizinhos;
2. Mulitplica pixel a pixel pela matriz do kernel
3. Soma todos os pontos da matrix
4. Divide pela soma de ponrtos (até 9)
5. Pinta o (x,y) ;
6. Retorna o ip da imagem filtrada;



Filtro Gaussian Blur (GB)

```
--- ---;
double kernel[] = { 1, 2, 1, 2, 4, 2, 1, 2, 1 };
int cont = 0;
for (int x = 0; x < w; x++) {// Varrendo as linhas atÃ© w-1
    for (int y = 0; y < h; y++) {
        soma = 0;
        soma2 = 0;
        cont = 0;
        for (int x1 = x - raio; x1 <= x + raio; x1++) {
            for (int y1 = y - raio; y1 <= y + raio; y1++) {

                if (x1 >= 0 && x1 < w && y1 >= 0 && y1 < h) {

                    soma = soma + (img[x1][y1] * kernel[cont]);
                    soma2 = soma2 + kernel[cont];
                }
                cont++;

            }
        }
        div = soma / soma2;
        cor = (int) Math.round(div);
        ip1.set(x, y, cor);
```



Filtro Wiener/Lee

$$g(x,y) = \alpha \cdot f(x,y) + (1-\alpha)\bar{f}(x,y)$$

$$\bar{f}(x,y) = \frac{1}{n(W(x,y))} \sum_{(i,j)\in W(x,y)} f(x,y)$$

$$\alpha = 1 - \frac{q_H^2}{q(x,y)^2}$$

$$q_H = \frac{\sigma_H}{\mu_H}$$

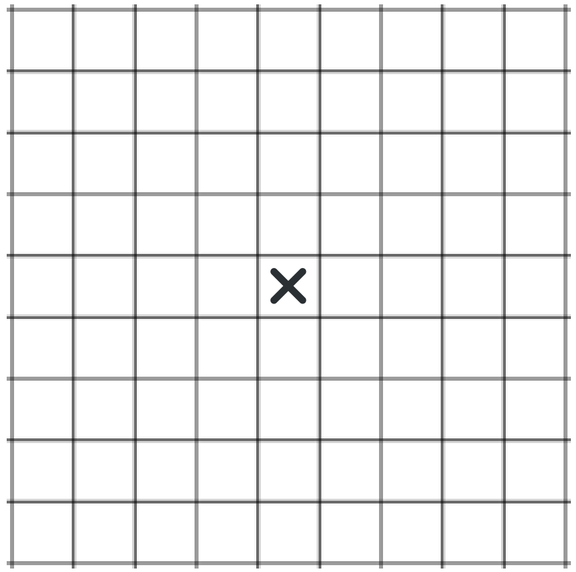
$$q(x,y) = \frac{\sigma(x,y)}{\mu(x,y)}$$

1. Definição da região homogênea;
2. Acha os valores de média e desvio padrão;
3. define a constante qh.

```
double valor;
double soma = 0;
double media;
double cont = 0;

for (int x = 31; x <= 163; x++) {
    for (int y = 272; y <= 329; y++) {
        valor = img[x][y];
        soma = soma + valor;
        cont++;

    }
}
media = soma / cont;
```

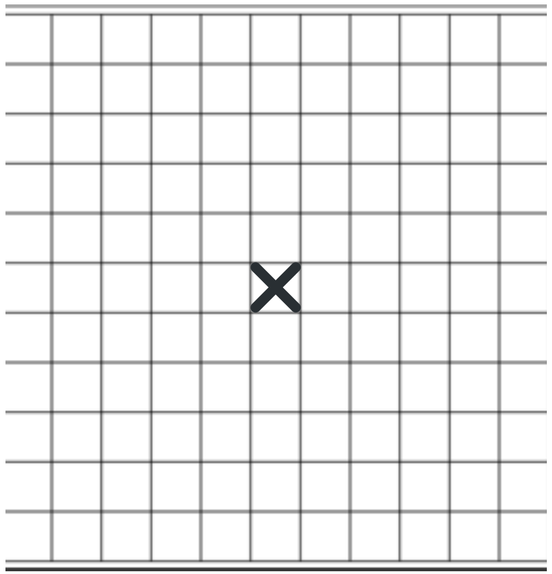
Filtro Wiener

Matriz: 9x9



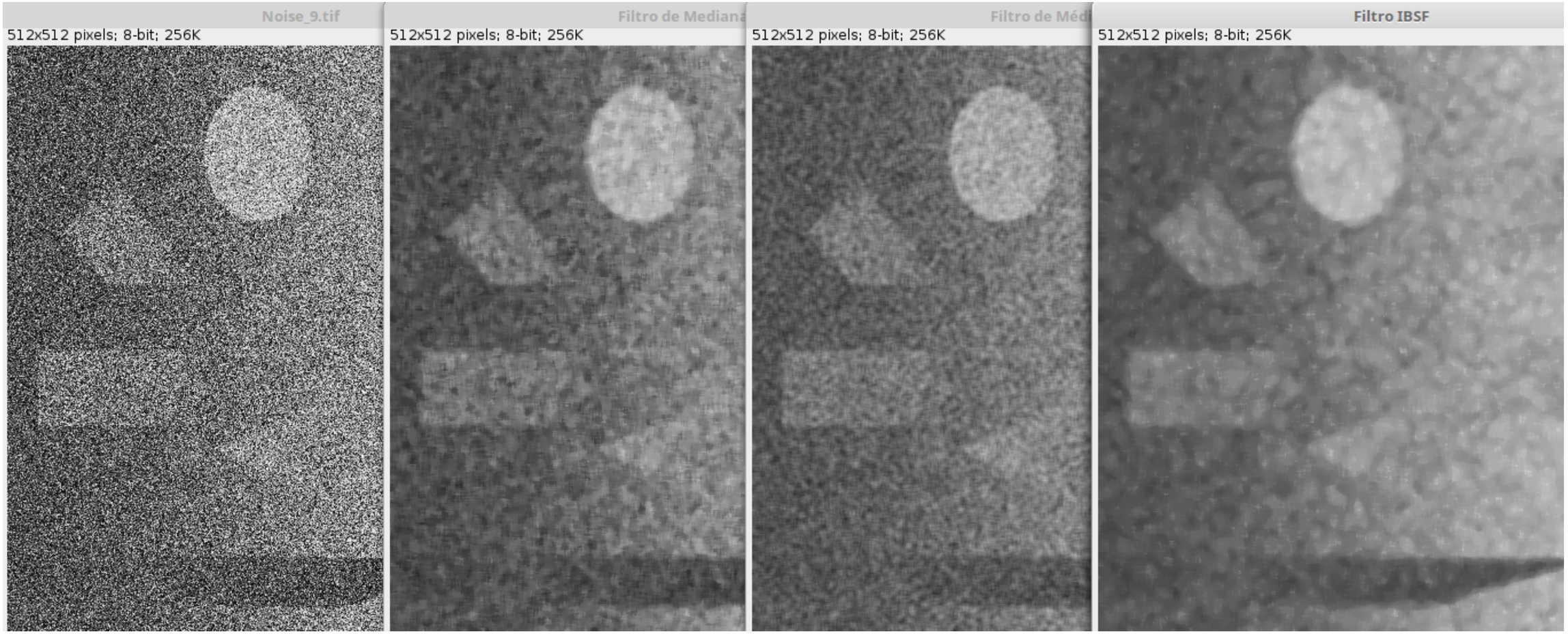
1. Acha os valores de media de desvio todos padrão numa janela;
2. Acha o valor de alfa pra cada pixel
3. Pinta a posição (x,y)
4. Retorna o ip

Filtro IBSF

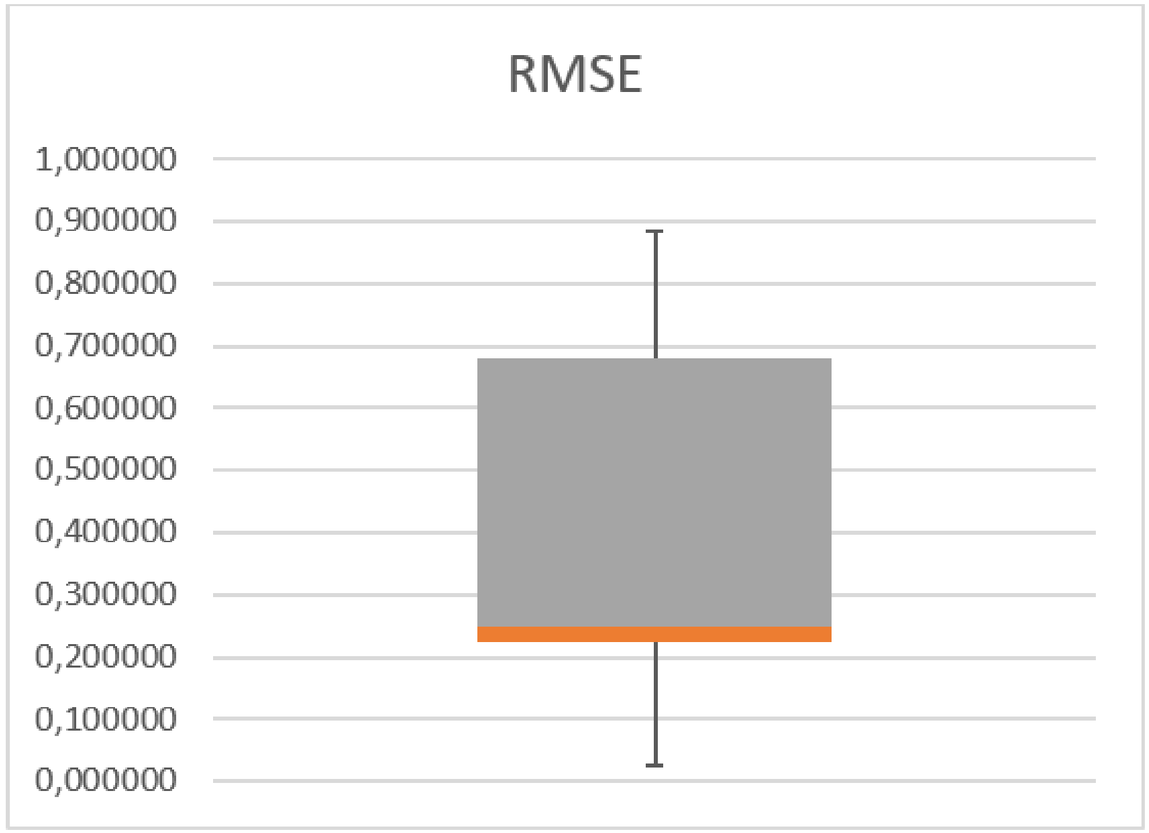


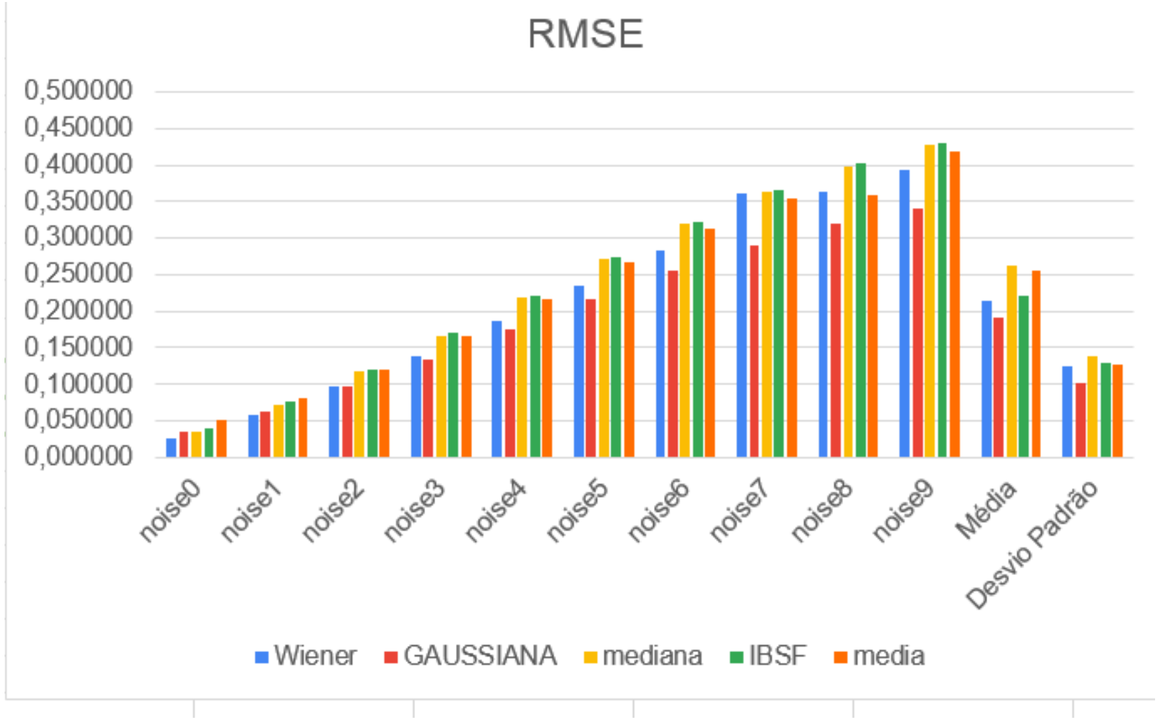
1. Aplica o filtro mediana janela de raio5
2. Compara pixel a pixel da original com a do filtro mediana e retorna a com maior intensidade, ou seja, a mais clara
3. Aplica novamente um filtro mediana, desta vez 3x3
4. pinta os pixels ;
5. Retorna o ip.

Filtros

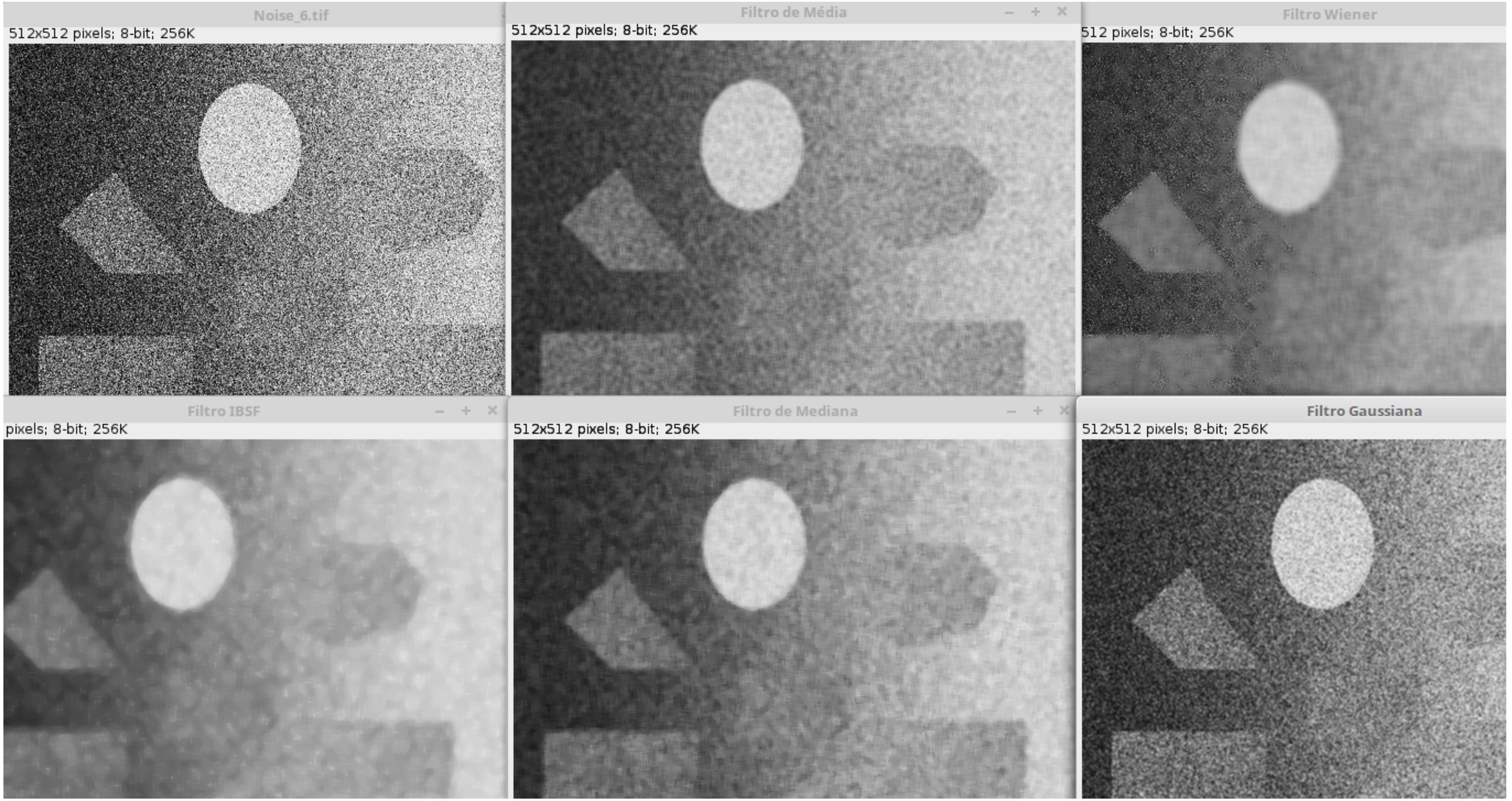

Noise_9.tif
512x512 pixels; 8-bit; 256K
Filtro de Mediana
512x512 pixels; 8-bit; 256K
Filtro de Méd
512x512 pixels; 8-bit; 256K
Filtro IBSF
512x512 pixels; 8-bit; 256K

Root Mean Squared Error (RMSE)

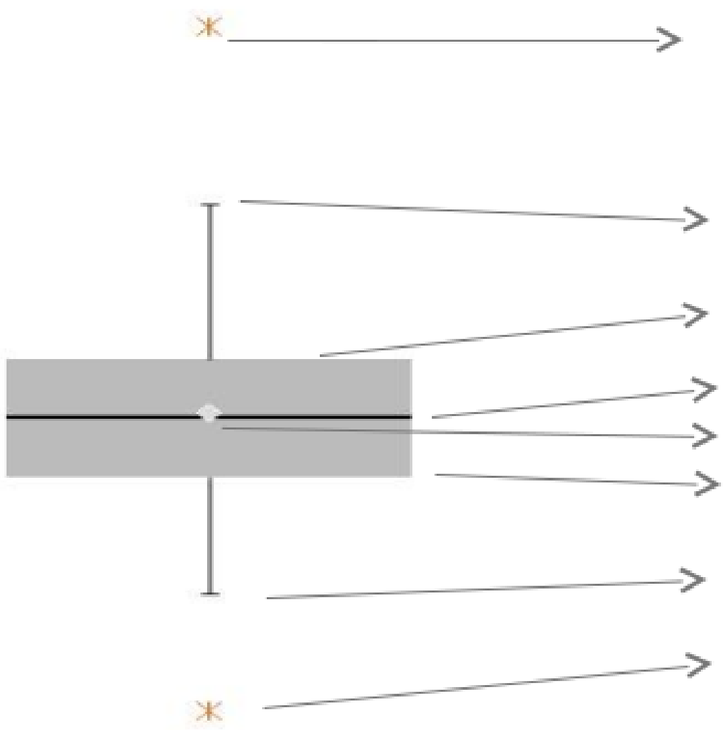




Filtros


Noise_6.tif
512x512 pixels; 8-bit; 256K
Filtro de Média
512x512 pixels; 8-bit; 256K
Filtro Wiener
512 pixels; 8-bit; 256K
Filtro IBSF
pixels; 8-bit; 256K
Filtro de Mediana
512x512 pixels; 8-bit; 256K
Filtro Gaussiana
512x512 pixels; 8-bit; 256K

BOXPLOT



Outliers superiores

Máximo entre os valores até Q3+1,5*(Q3-Q1)

3o. Quartil (Q3) - 75% dos dados estão abaixo deste valor

Mediana ou 2o. Quartil (Q2) - 50% dos dados estão abaixo deste valor

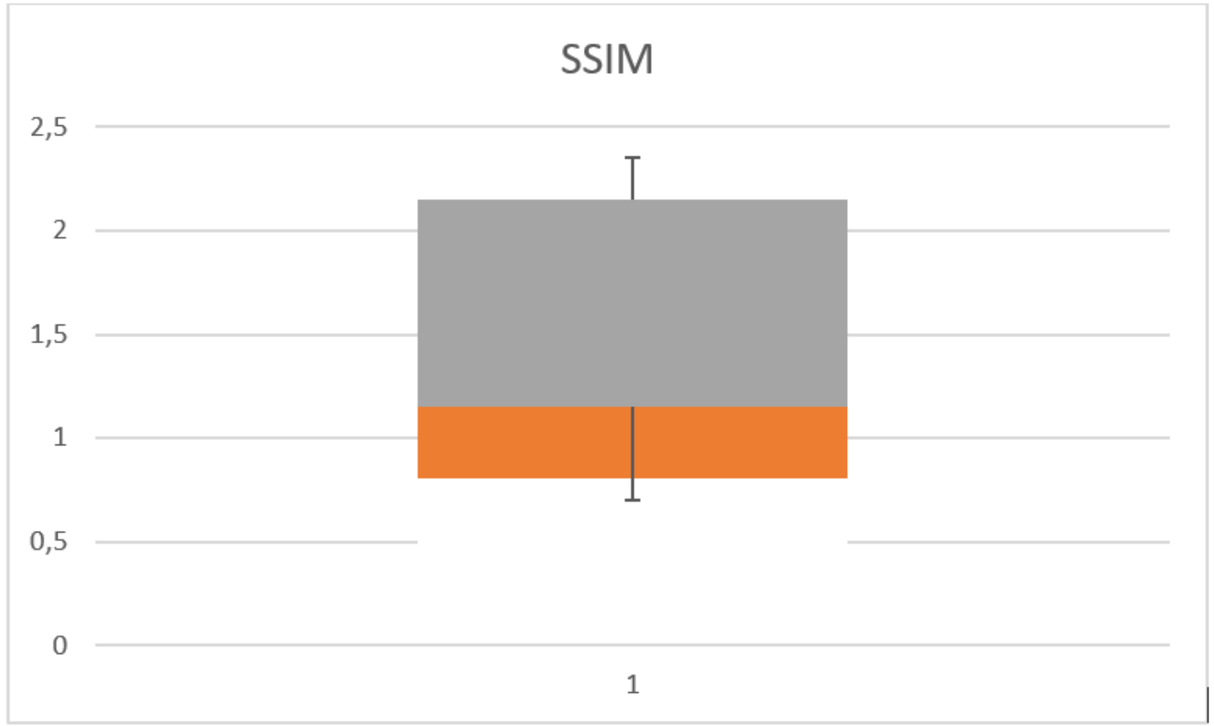
Média (símbolo de diamante)

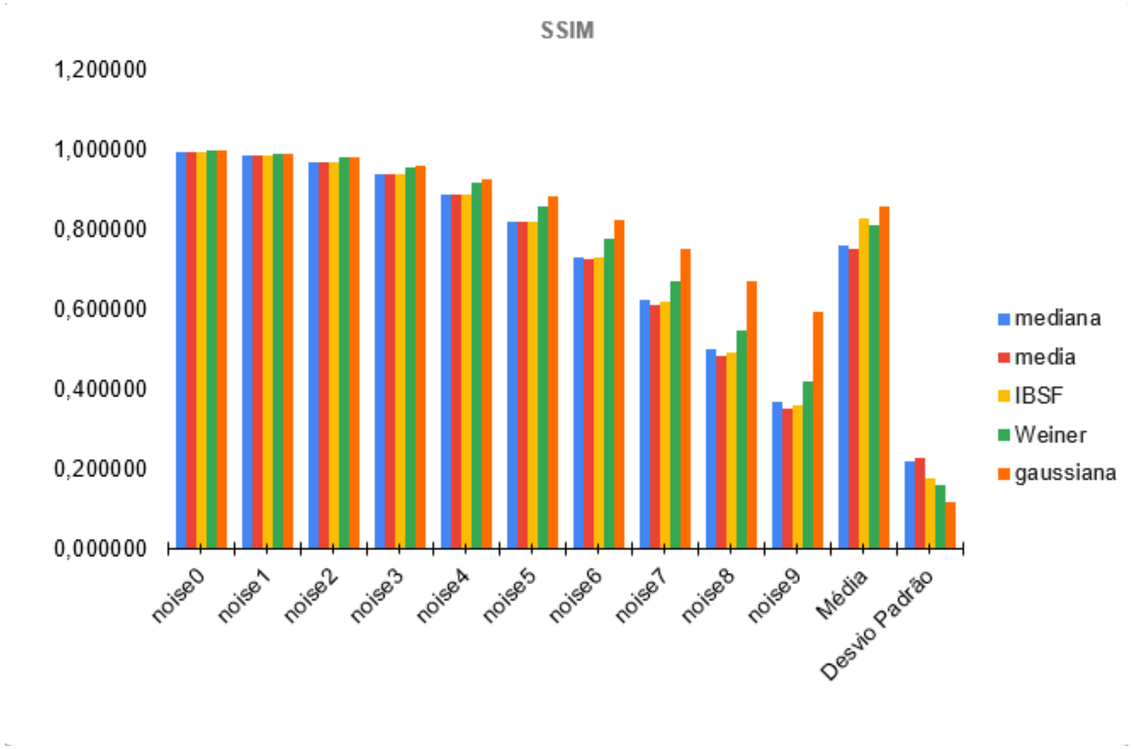
1o. Quartil (Q1) - 25% dos dados estão abaixo deste valor

Mínimo entre os valores até Q1-1,5*(Q3-Q1)

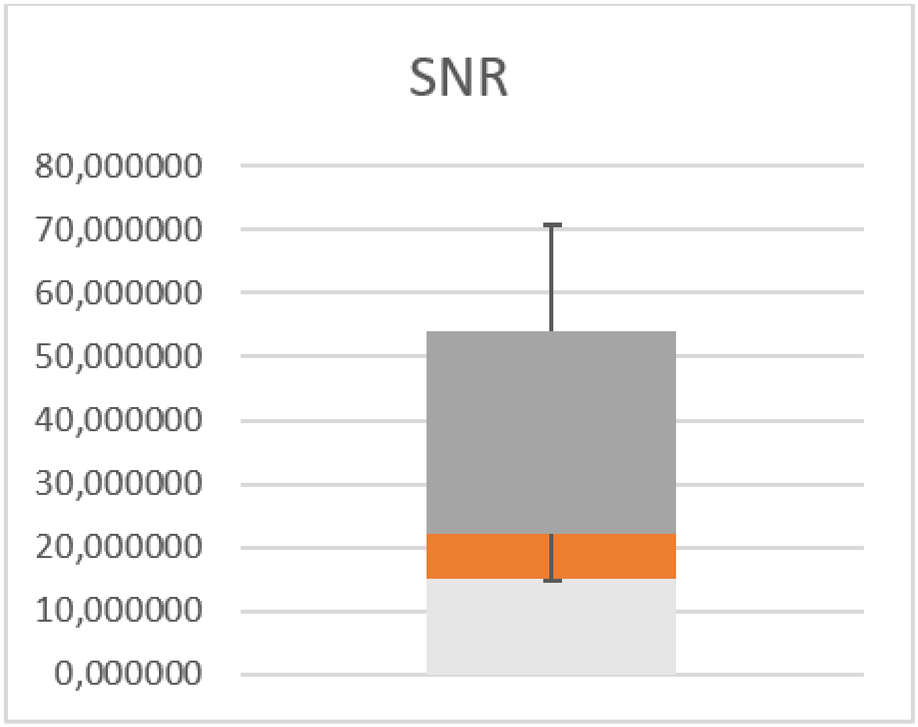
Outliers inferiores

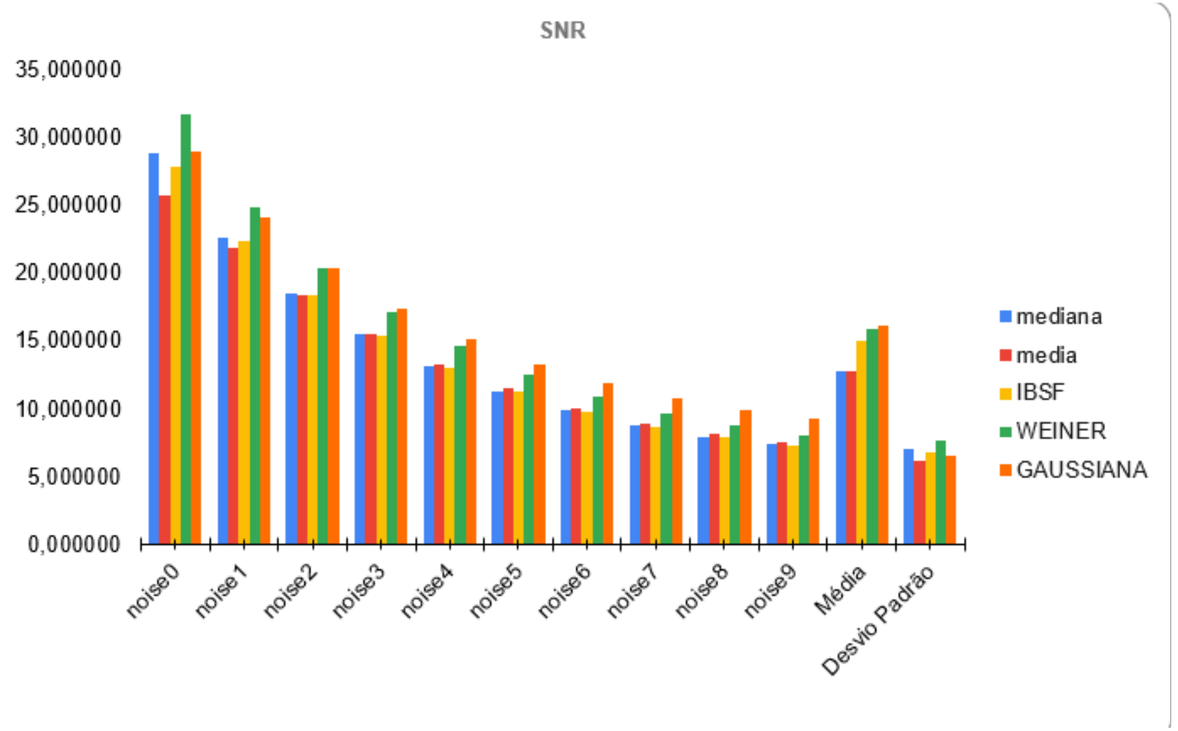
Structural Similarity Index (SSIM)



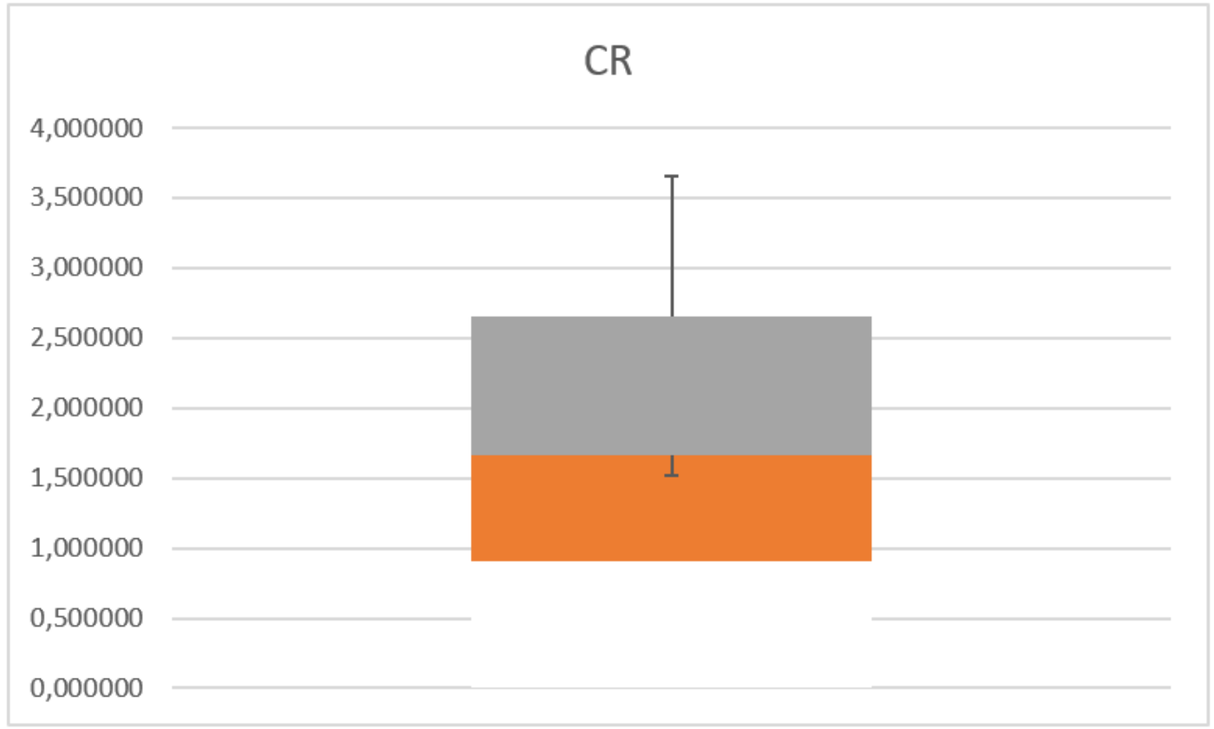


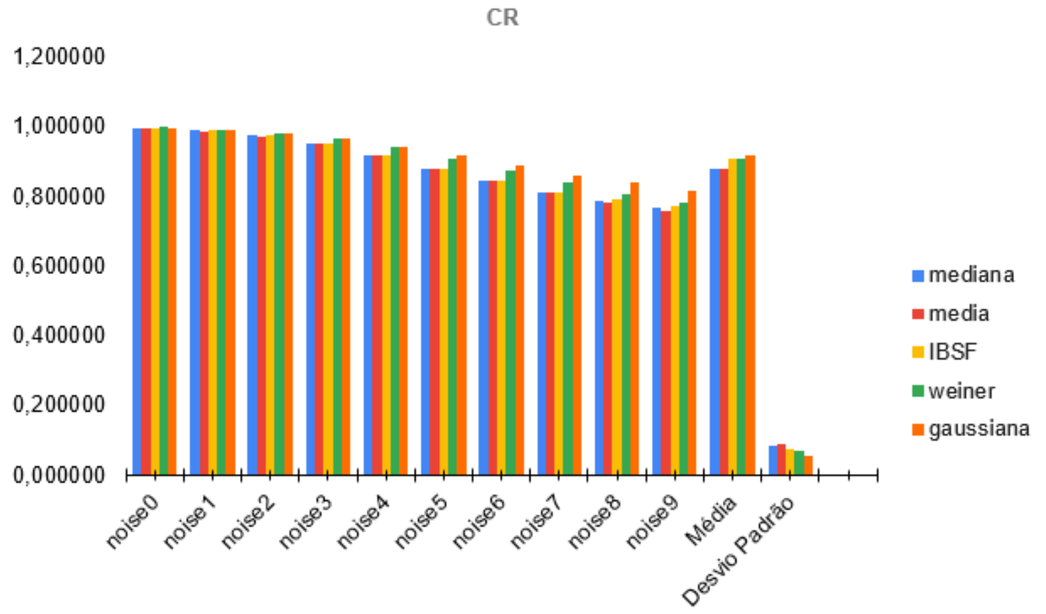
Relação Sinal - ruído (SNR)



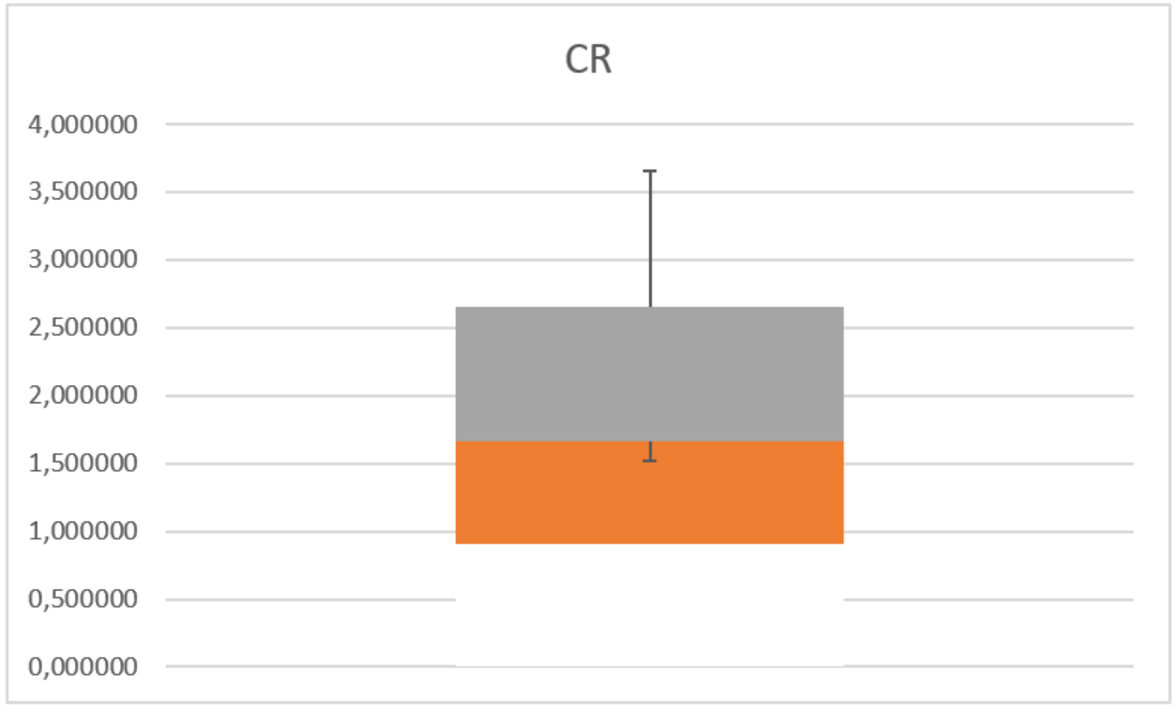


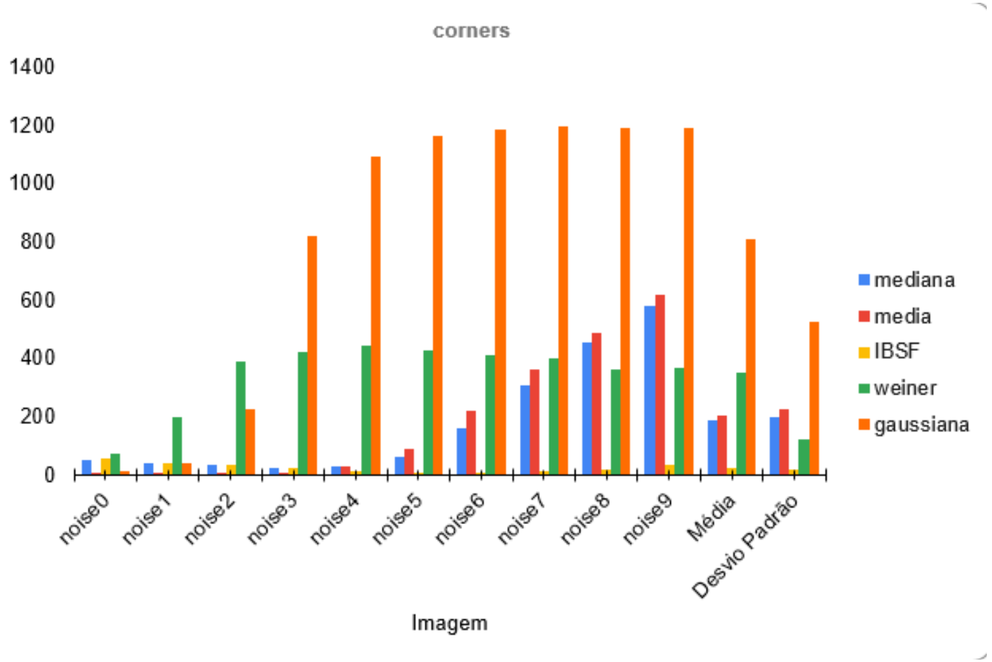
Coeficiente de Correlação (CR)





Harris detector (corners)





Obrigado!

Erick Mendonça
Vitória Bezerrra
Welainny Viana

Nunca desista!