PTC3492 - Princípios de Formação e Processamento de Imagens Médicas

Projeto final - Restauração de imagem por filtro de Wiener

Guilherme Antonio Rodrigues - 9381256
Marcelo Monari Baccaro - 8989262
Renan Weege Achjian - 9344772

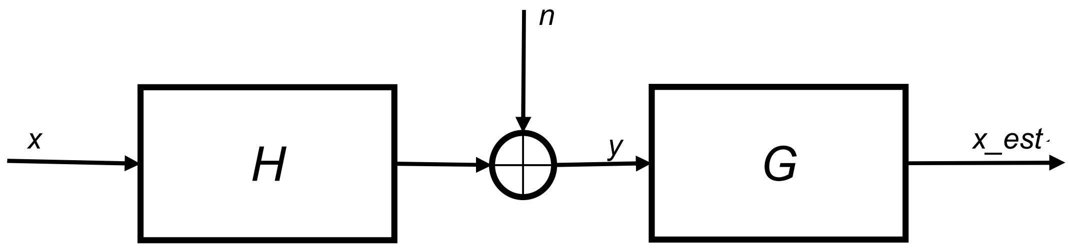
1. Objetivos

Este projeto tem por objetivo implementar um programa que, dada uma imagem de tomografia computadorizada (CT) degradada por um borramento por filtro gaussiano e também por ruído aditivo branco gaussiano, utiliza a técnica do filtro de Wiener para restaurá-la a um estado próximo ao original. Além disso, este filtro é comparado com duas técnicas de deconvolução cega para analisar o impacto do conhecimento sobre o borramento na estimação.

2. Motivação

Imagens médicas estão sujeitas a diversos tipos de ruído, que apresentam origem e características diferentes, dependendo do equipamento e condições do ambiente em que são adquiridas. No caso do CT, os principais efeitos degradantes são a incerteza quanto à intensidade de cada pixel (modelada como ruído aditivo branco gaussiano) e a influência do valor de um pixel nos vizinhos (*Point Spread Function*, erro inerente aos sensores utilizados). Este último é mais bem representado como uma convolução.

O sistema de restauração proposto tenta identificar e reverter estes dois efeitos, encontrando para isso a função $g(t)$ adequada.



Na figura acima, x é a imagem original, H é um sistema linear, n uma variável aleatória que representa o ruído, y representa o que é de fato disponibilizado pela máquina que adquire as imagens, G é o filtro projetado, e x_est é a imagem corrigida, idealmente idêntica a x.

Assim, mesmo que seja impossível desenvolver um sistema de aquisição de imagens livre de ruídos e incertezas, pode-se tentar remover os efeitos degradantes *a posteriori*, e chegar perto do que seria esta imagem "original".

3. Metodologia

O filtro de Wiener proposto atua no domínio da frequência, tentando minimizar o impacto do ruído em frequências com uma baixa SNR. Ou seja, este filtro é o resultado de um problema de otimização, em que o funcional é descrito por: [1]

$$e(u,v) = E\left\{\left|X(u,v) - \hat{X}(u,v)\right|^2\right\}$$

Sendo que,

$$\hat{X}(u,v) = G(u,v) \cdot Y(u,v) = G(u,v) \cdot \big(X(u,v) \cdot H(u,v) + N(u,v)\big)$$

Considerando que $\mathrm{n(i,j)}$ é um processo estocástico espacialmente invariante e que ele se comporta como um ruído branco com média nula e independente da entrada $x(i,j)$, temos que:

$$E\{X(u,v) \cdot N^*(u,v)\} = 0$$

Achar o $G(u,v)$ que minimiza esse funcional, significa resolver o problema de otimização sem restrição, em que deriva-se$e(u,v)$ em relação a $G(u,v)$ e iguala-se a zero:

$$\frac{\partial e(u,v)}{\partial G(u,v)} = 0 \rightarrow G(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + \frac{S_n(u,v)}{S_x(u,v)}}$$

Em que, $S_n(u,v)$ é o *PSD* (Power Spectral Density) de $n(i,j)$, enquanto que $S_x(u,v)$ é o PSD de $x(i,j)$. Eles definidos e calculados como segue: [2]

$$S_n(u,v) = E\{|N(u,v)|^2\} = \big|FFT\big(n(i,j)\big)\big|^2$$

$$S_x(u,v) = E\{|X(u,v)|^2\} = \big|FFT\big(x(i,j)\big)\big|^2$$

Caso não se tenho uma boa estimação de $S_n(u,v)$ e $S_x(u,v)$, pode-se por uma constante K relacionada a *SNR* (Signal to Noise Ratio), como segue a baixo:

$$SNR = 20 \cdot log_{10}\left(\frac{\sigma_x}{\sigma_n}\right) \rightarrow \frac{\sigma_n^2}{\sigma_x^2} = 10^{-\frac{SNR}{10}} = K$$

$$G(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + K}$$

Embora serão utilizados métodos de deconvolução cega neste relatório, como eles não serão implementados e apenas serão utilizadas funções nativas do Matlab que os implementam, então não é necessário uma discussão extensa a respeito de sua dedução. Mas é importante ressaltar que serão dois métodos: Richardson–Lucy deconvolution [3] e Generalized Blind Deconvolution [4]. Ambos são deduzidos a partir da Teoria da Estimação, de maneira mais específica, eles são baseados em estimação *MAP* (Maximum A Posteriori) [5], e utilizam um chute inicial do *PSF* (Point Spread Function), em que eles também chegam a uma estimativa deste, além da própria estimação da imagem de entrada.

4. Resultados

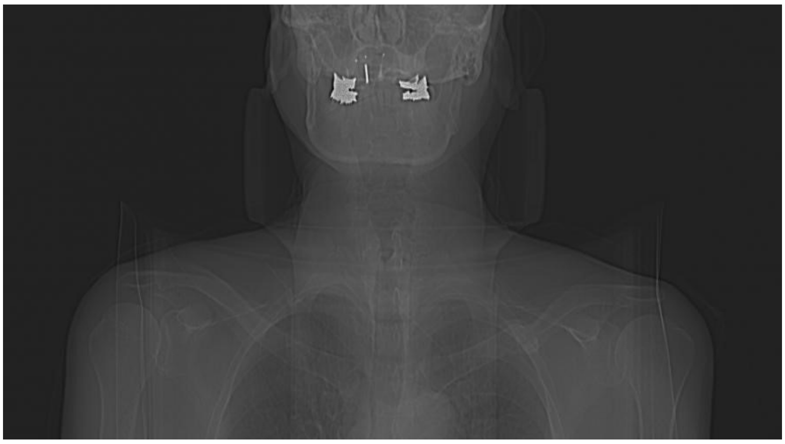
Os resultados são apresentados em forma de tabela e também com as imagens da estimação da entrada pelos filtros. Na tabela, consta três métricas de desempenho:

NRMSE, Emax e SSIM. Testadas para três valores de SNR: 10dB, 6dB e 3dB. Por fim, são analisados cinco filtros (2 Wiener e 3 Blind):

1) Wiener sabendo $S_n(u,v)$ e $S_x(u,v)$ =>*wiener_filter_Sn_Sf*
2) Wiener sem saber $S_n(u,v)$ e $S_x(u,v)$ =>*wiener_filter_SNR*
3) Richardson–Lucy Deconvolution com PSF inicial pouco ruidoso =>*deconvlucy* [6]
4) Generalized Blind Deconvolution com PSF inicial pouco ruidoso=>*deconvblind* [7]
5) Generalized Blind Deconvolution com PSF inicial muito ruidoso=>*deconvblind* [7]

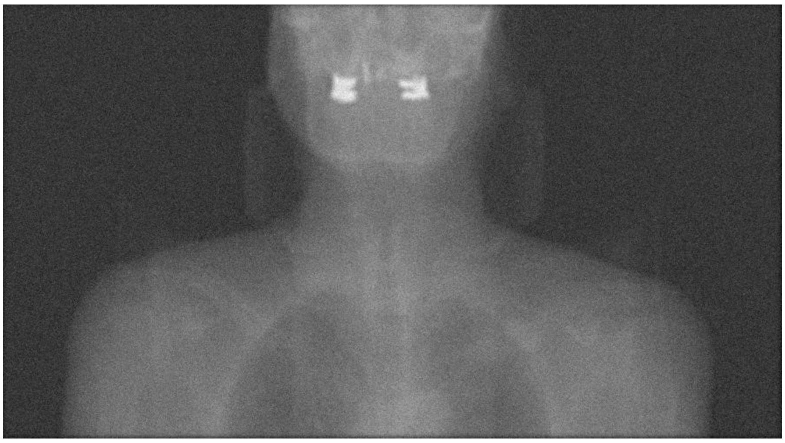
FILTERS	NRMSE	Emax	SSIM	Signal to Noise Ratio
Blurred Noisy Image	0.14330018	130.7070384	0.927036533	SNR = 10 dB
	0.207705074	137.4318568	0.863199505	SNR = 6 dB
	0.283419546	176.3554776	0.776203119	SNR = 3 dB
Wiener Sn Sf	0.134551269	132.4727155	0.943853749	SNR = 10 dB
	0.115413567	128.8240357	0.956499243	SNR = 6 dB
	0.119902172	127.844215	0.953256318	SNR = 3 dB
Wiener SNR	0.125520731	138.4993812	0.878011008	SNR = 10 dB
	0.221029189	155.2461867	0.752440674	SNR = 6 dB
	0.347183433	175.7616789	0.590648892	SNR = 3 dB
Lucy	0.218874008	123.5802402	0.855282985	SNR = 10 dB
	0.350064632	187.2769449	0.706223508	SNR = 6 dB
	0.459999708	308.2437384	0.561128419	SNR = 3 dB
Blind 1	0.201638003	110.5188314	0.873238352	SNR = 10 dB
	0.320541368	135.4558568	0.736698121	SNR = 6 dB
	0.486479107	264.8564707	0.550231068	SNR = 3 dB
Blind 2	0.277902324	191.2376945	0.790789274	SNR = 10 dB
	0.425837465	257.3079422	0.619890351	SNR = 6 dB
	0.630559803	459.3365304	0.428844739	SNR = 3 dB

4.1) Imagem de entrada original:

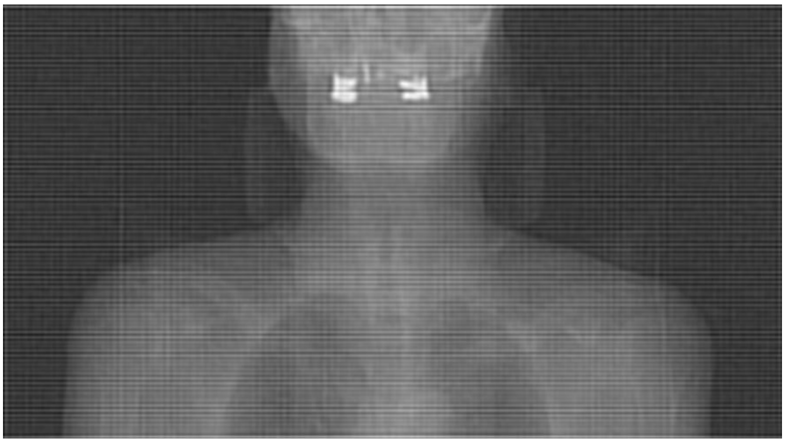


4.2) SNR = 10dB

4.2.1) Saída borrada e ruidosa:

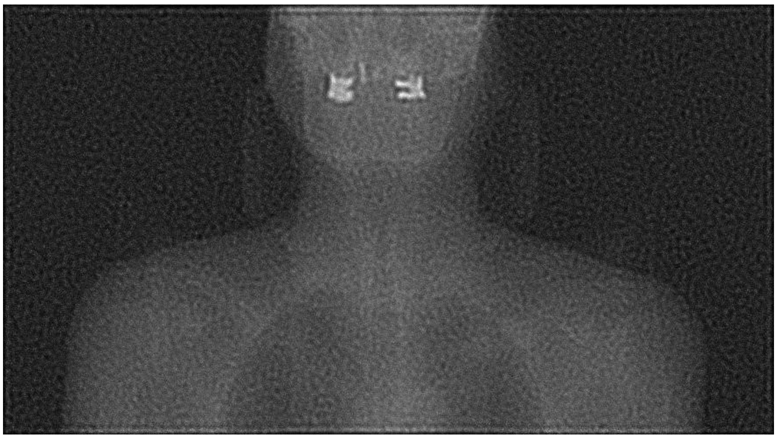


4.2.2) Wiener Sn Sf:

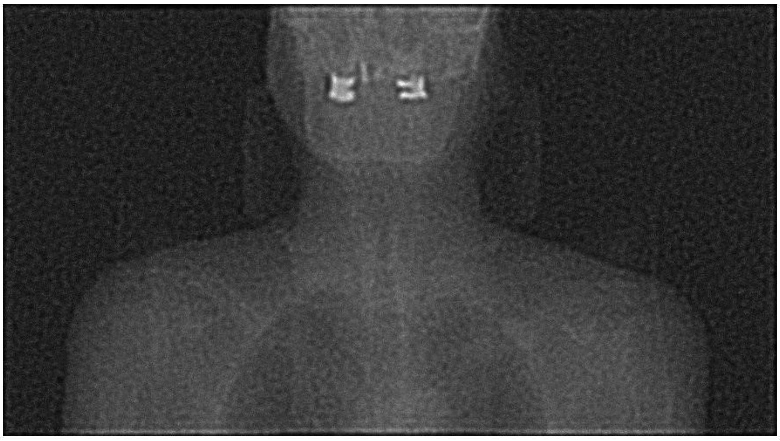


4.2.3) Wiener SNR:

4.2.4) Lucy:



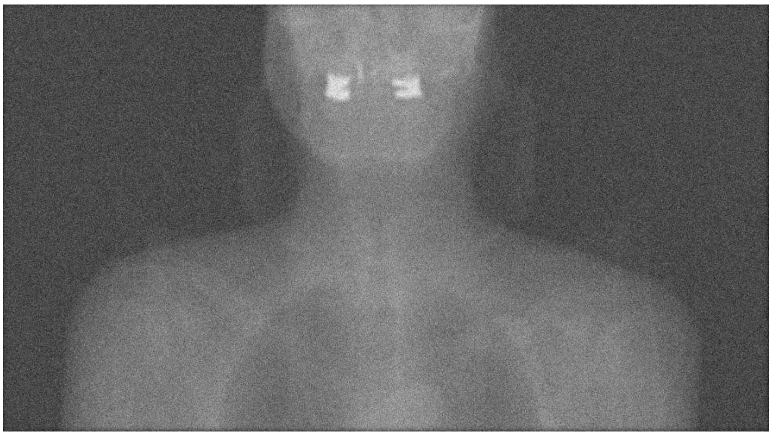
4.2.5) GBD1:



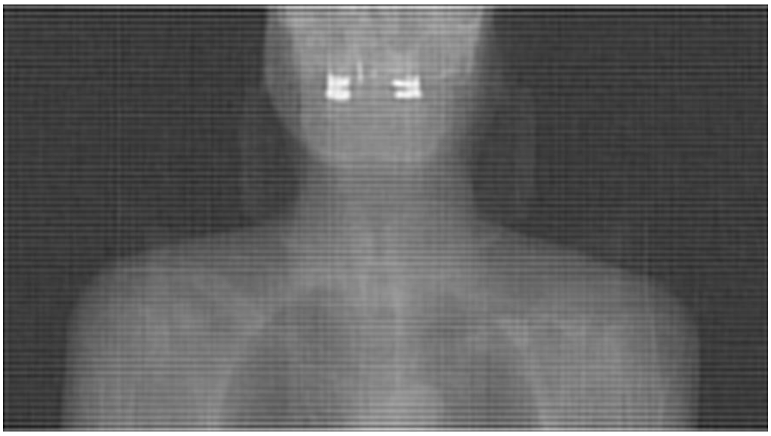
4.2.6) GBD2:

4.3) SNR = 6dB

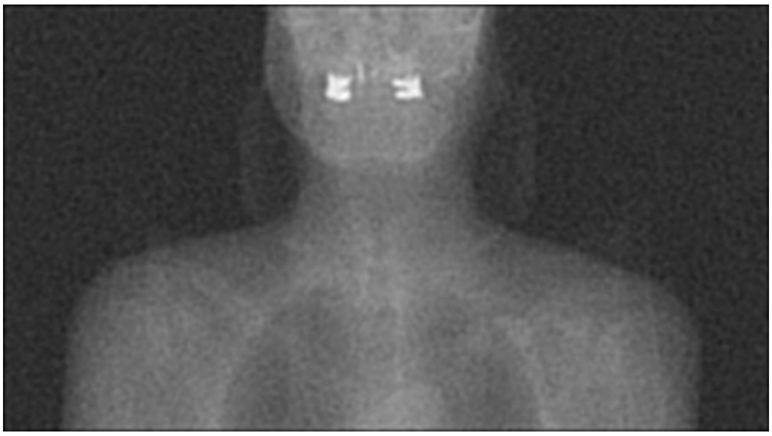
4.3.1) Saída borrada e ruidosa:



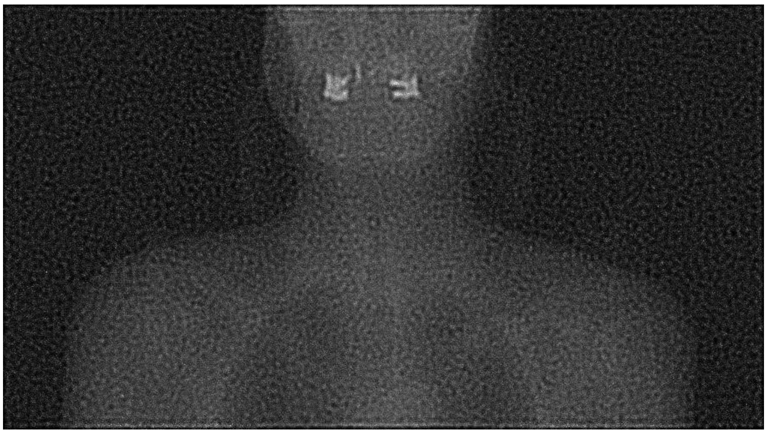
4.3.2) Wiener Sn Sf:



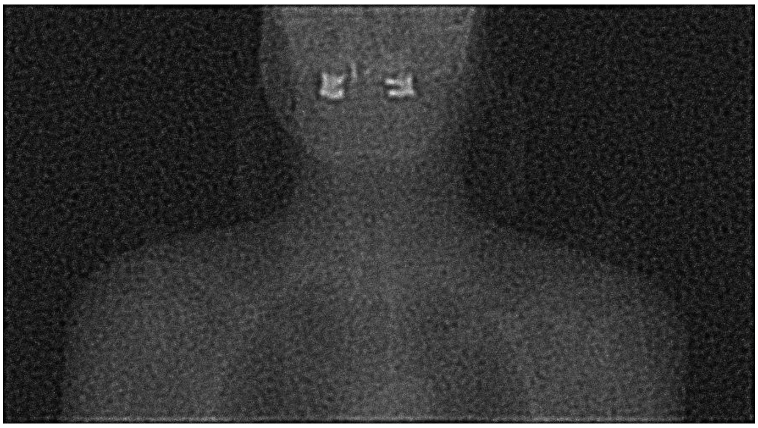
4.3.3) Wiener SNR:



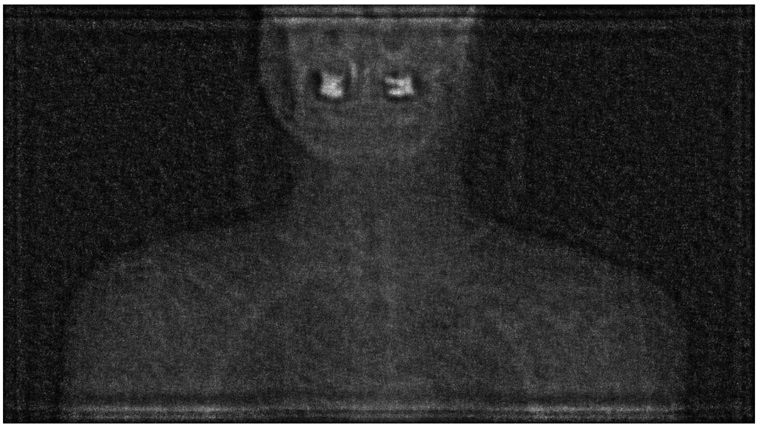
4.3.4) Lucy:



4.3.5) GBD1:

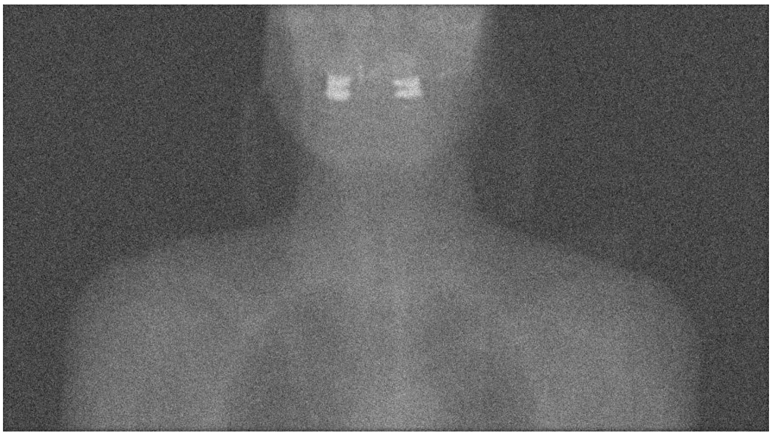


4.3.6) GBD2:

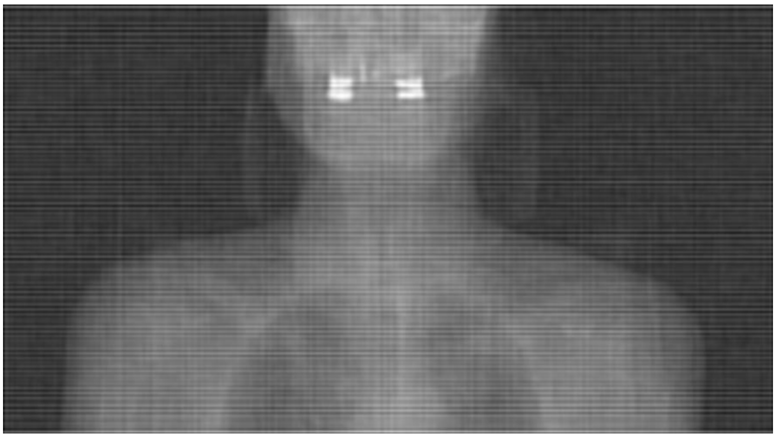


4.4) SNR = 3dB

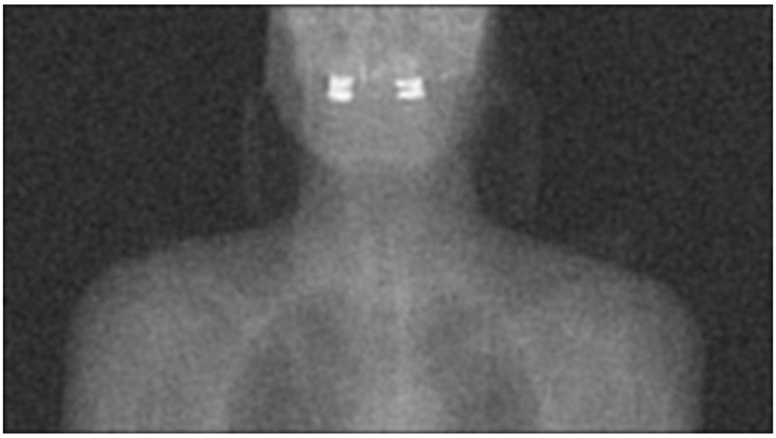
4.4.1) Saída borrada e ruidosa:



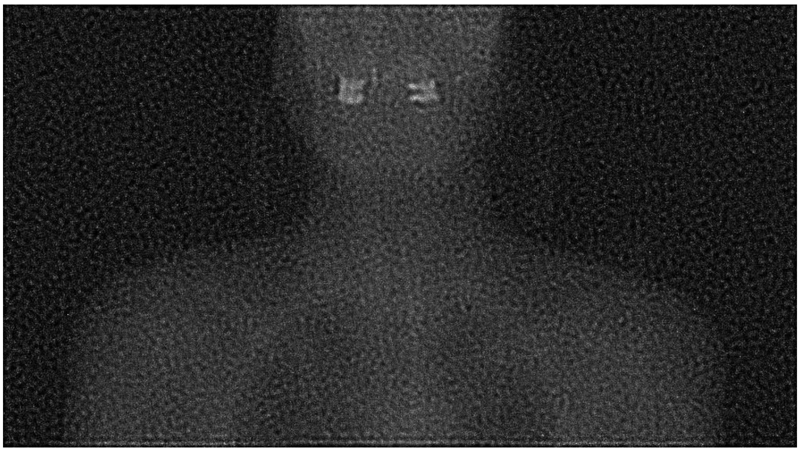
4.4.2) Wiener Sn Sf:



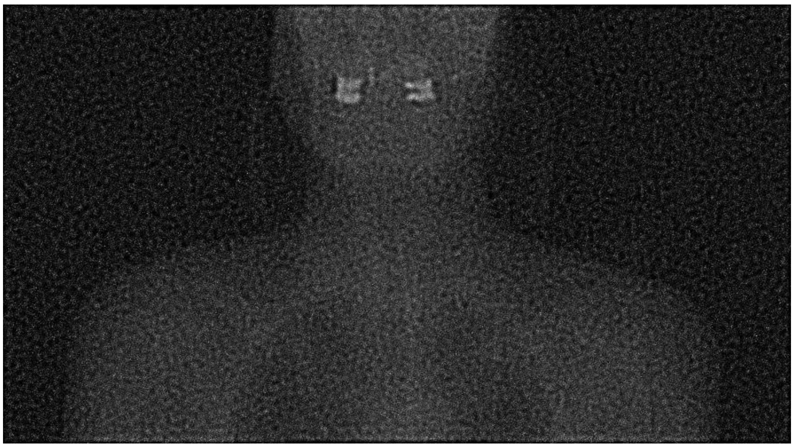
4.4.3) Wiener SNR:



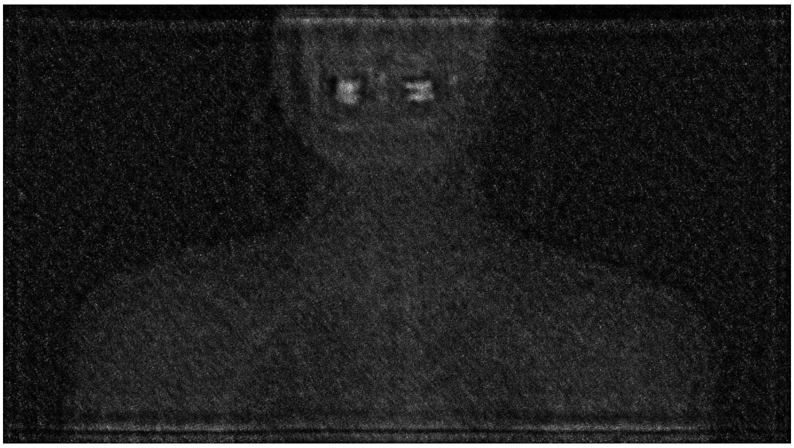
4.4.4) Lucy:



4.4.5) GBD1:



4.4.6) GBD2:



5. Avaliação

As métricas apresentadas na seção anterior mostram o quão próximo da imagem original estão as imagens, de tal forma que, para NRMSE e Emax, quanto menor os seus valores maior a proximidade, enquanto, para SSIM, isto ocorre quanto mais próximo seu valor estiver de 1. Desta forma, vemos que, em geral, quanto menor o SNR, piores ficam as métricas. A exceção a este padrão está no filtro *wiener_filter_Sn_Sf*, por motivos que serão discutidos na próxima seção. Vemos também como é impactante o conhecimento do PSF (Point Spread Function), ou pelo menos o conhecimento de SNR (Signal to Noise Ratio), na deconvolução, fato que é constado ao comparar os desempenhos dos filtros baseados no filtro de Wiener em relação aos filtros baseados em convolução cega.

Em relação as métricas estatísticas utilizadas, conforme SNR diminui, a estimação da imagem de entrada fica pior do que a própria imagem borrada e ruidosa, mesmo que a estimação apresente um formato mais parecido com a imagem original. Isto se deve ao tipo de funcional que estes métodos buscam minimizar, que não necessariamente também minimizam essas métricas estatísticas.

6. Discussão

Ao observar de perto os resultados do filtro *wiener_filter_Sn_Sf*, nota-se que o filtro introduz uma interferência senoidal na estimação da entrada. Isto é consequência do *spectral leakage*, pois foi utilizado a FFT na sua implementação, como este considera a imagem de entrada como periódica, descontinuidades surgem nas bordas (e, portanto, componentes de altas frequências). Uma boa solução para isto seria utilizar uma janela (Hanning, por exemplo) para atenuar o sinal nas extremidades, e garantir uma transição suave entre a imagem e suas repetições. Ou simplesmente aplicar um filtro Notch para remover estas frequências parasitas, caso for fácil a sua visualização no domínio da frequência.

É importante notar que foi utilizado a técnica de *zero padding* em todos os cálculos de FFT, em que esta sempre retornava uma matriz quadrada com dimensão igual a um exponencial de dois, o que acelera o algoritmo e reduz erros.

7. Conclusões

Não é por trivialidade que se utiliza o filtro de Wiener como padrão de comparação de desempenho de novos filtros. O seu desempenho é muito bom em filtragem linear com ruídos aditivos, mesmo com problemas numéricos de implementação, mas desde que o SNR não seja muito baixo. Entretanto, os métodos baseados neste filtro exigem o conhecimento de informações que não são fáceis de estimar na prática, o que tornar os métodos baseados em convolução cega atraentes.Embora os filtros por convolução cega não precisem do conhecimento da PSF, a estimação inicial deste influencia profundamente a convergência dos algoritmos e também no valor de SNR, que também não pode ser muito baixo. Além disso, eles têm maior complexidade, o que implica em maior tempo de processamento. Assim, os filtros por deconvolução cega só têm vantagem em relação aos filtros de Wiener se pouco for conhecido a respeito da imagem.

8. Referências

[1] – https://en.wikipedia.org/wiki/Wiener_deconvolution

[2] - https://dsp.stackexchange.com/questions/736/how-do-i-implement-cross-correlation-to-prove-two-audio-files-are-similar
[3] - https://en.wikipedia.org/wiki/Richardson%E2%80%93Lucy_deconvolution
[4] - https://en.wikipedia.org/wiki/Blind_deconvolution
[5] - https://en.wikipedia.org/wiki/Maximum_a_posteriori_estimation
[6] – https://www.mathworks.com/help/images/ref/deconvlucy.html
[7] - https://www.mathworks.com/help/images/ref/deconvblind.html?s_tid=doc_ta

10. Listagem dos programas

10.1) MAIN.m

```
close; clear; clc;
%%% Input image
f = 255 * mat2gray(imread('CT.tif'));


%%% Linear dynamic system transfer function
dim_h = 32; var_h = 3;
h = fspecial('gaussian', [dim_h, dim_h], var_h); % Gaussian blur


for k = 1:3
%%% Noise
if (k == 1)
        SNR = 10;
elseif (k == 2)
        SNR = 6;
else
        SNR = 3;
end
    std_n = std2(f) * 10 ^ (-SNR / 20);
    n = std_n * randn(size(f)); % Adictive White Gaussian noise


%%% Output image
    g = image_output(f, n, h);


%%% FILTERS
    fe_wiener_Sn_Sf = wiener_filter_Sn_Sf(f, h, n, g);
    fe_wiener_SNR = wiener_filter_SNR(h, g, SNR);
    fe_lucy = deconvlucy(g, h + 1e-3 * randn(size(h)));
    [fe_blind1, ~] = deconvblind(g, h + 1e-3 * randn(size(h)));
    [fe_blind2, ~] = deconvblind(g, h + 1e-2 * randn(size(h)));


%%% METRICS
    Metrics_g = metrics_NRMSE_Emax_SSIM(f, g);
    Metrics_fe_wSnSf = metrics_NRMSE_Emax_SSIM(f, fe_wiener_Sn_Sf);
    Metrics_fe_wSNR = metrics_NRMSE_Emax_SSIM(f, fe_wiener_SNR);
    Metrics_fe_lucy = metrics_NRMSE_Emax_SSIM(f, fe_lucy);
    Metrics_fe_b1 = metrics_NRMSE_Emax_SSIM(f, fe_blind1);
    Metrics_fe_b2 = metrics_NRMSE_Emax_SSIM(f, fe_blind2);
```

```
%%% EXCEL
    xlswrite('Metrics.xlsx', Metrics_g, 'Sheet1', ['B' num2str(k+1) ':D'
num2str(k+1)]);
    xlswrite('Metrics.xlsx', Metrics_fe_wSnSf, 'Sheet1', ['B' num2str(k+4)
':D' num2str(k+4)]);
    xlswrite('Metrics.xlsx', Metrics_fe_wSNR, 'Sheet1', ['B' num2str(k+7)
':D' num2str(k+7)]);
    xlswrite('Metrics.xlsx', Metrics_fe_lucy, 'Sheet1', ['B' num2str(k+10)
':D' num2str(k+10)]);
    xlswrite('Metrics.xlsx', Metrics_fe_b1, 'Sheet1', ['B' num2str(k+13)
':D' num2str(k+13)]);
    xlswrite('Metrics.xlsx', Metrics_fe_b2, 'Sheet1', ['B' num2str(k+16)
':D' num2str(k+16)]);


%%% OUTPUT
    imwrite(uint8(255 * mat2gray(g)), ['CT_Blurred_Noisy_' num2str(SNR)
'.jpg']);
    imwrite(uint8(255 * mat2gray(fe_wiener_Sn_Sf)), ['CT_Est_WSnSf_'
num2str(SNR) '.jpg']);
    imwrite(uint8(255 * mat2gray(fe_wiener_SNR)), ['CT_Est_WSNR_'
num2str(SNR) '.jpg']);
    imwrite(uint8(255 * mat2gray(fe_lucy)), ['CT_Est_Lucy_' num2str(SNR)
'.jpg']);
    imwrite(uint8(255 * mat2gray(fe_blind1)), ['CT_Est_B1_' num2str(SNR)
'.jpg']);
    imwrite(uint8(255 * mat2gray(fe_blind2)), ['CT_Est_B2_' num2str(SNR)
'.jpg']);
end
```

10.2) image_output.m

```
function g = image_output (f, n, h)
%%% Input image
[a, b] = size(f);
d = 2 ^ nextpow2(max(a,b));
a1 = floor((d - a) / 2) + 1;
a2 = a1 + a - 1;
b1 = floor((d - b) / 2) + 1;
b2 = b1 + b - 1;

fzp = zeros(d, d);
fzp(a1:a2, b1:b2) = f;
fzp = fftshift(fzp);
F = fft2(fzp);



%%% Noise
nzp = zeros(d, d);
nzp(a1:a2, b1:b2) = n;
nzp = fftshift(nzp);
N = fft2(nzp);



%%% Linear dynamic system transfer function
[dim_h, ~] = size(h);
c1 = round((d - dim_h) / 2) + 1;
c2 = c1 + dim_h - 1;
```

```
hzp = zeros(d, d);
hzp(c1:c2, c1:c2) = h;
hzp = fftshift(hzp);
H = fft2(hzp);


%%% Output image
G = F .* H + N;
g = ifft2(G);
g = ifftshift(g);
g = g(a1:a2, b1:b2);
```

10.3) wiener_filter_Sn_Sf.m

```
function fe = wiener_filter_Sn_Sf (f, h, n, g)
%%% Input image
[a, b] = size(f);
d = 2 ^ nextpow2(max(a,b));
a1 = floor((d - a) / 2) + 1;
a2 = a1 + a - 1;
b1 = floor((d - b) / 2) + 1;
b2 = b1 + b - 1;

fzp = zeros(d, d);
fzp(a1:a2, b1:b2) = f;
fzp = fftshift(fzp);
F = fft2(fzp);
Sf = F .* conj(F); % Sf = abs(fft2(f, d, d)) .^ 2;


%%% Noise
nzp = zeros(d, d);
nzp(a1:a2, b1:b2) = n;
nzp = fftshift(nzp);
N = fft2(nzp);
Sn = N .* conj(N); % Sn = abs(fft2(n, d, d)) .^ 2;


%%% Linear dynamic system transfer function
[h_dim, ~] = size(h);
c1 = round((d - h_dim) / 2) + 1;
c2 = c1 + h_dim - 1;

hzp = zeros(d, d);
hzp(c1:c2, c1:c2) = h;
hzp = fftshift(hzp);
H = fft2(hzp);
H2 = abs(H) .^ 2;


%%% Output image
gzp = zeros(d, d);
gzp(a1:a2, b1:b2) = g;
gzp = fftshift(gzp);
G = fft2(gzp);
```

```
%%% Estimated input
gamma = 1;
Fe = conj(H) .* G ./ (H2 + gamma * (Sn ./ Sf));
fe = ifft2(Fe);
fe = ifftshift(fe);
fe = fe(a1:a2, b1:b2);
```

10.4) wiener_filter_SNR.m

```
function fe = wiener_filter_SNR (h, g, SNR)
[a, b] = size(g);
d = 2 ^ nextpow2(max(a,b));
a1 = floor((d - a) / 2) + 1;
a2 = a1 + a - 1;
b1 = floor((d - b) / 2) + 1;
b2 = b1 + b - 1;


%%% Linear dynamic system transfer function
[h_dim, ~] = size(h);
c1 = round((d - h_dim) / 2) + 1;
c2 = c1 + h_dim - 1;

hzp = zeros(d, d);
hzp(c1:c2, c1:c2) = h;
hzp = fftshift(hzp);
H = fft2(hzp);
H2 = abs(H) .^ 2;


%%% Output image
gzp = zeros(d, d);
gzp(a1:a2, b1:b2) = g;
gzp = fftshift(gzp);
G = fft2(gzp);


%%% Estimated input
K = 10 ^ (-SNR/10);
gamma = 1;
Fe = conj(H) .* G ./ (H2 + gamma * K);
fe = ifft2(Fe);
fe = ifftshift(fe);
fe = fe(a1:a2, b1:b2);
```

10.5) metrics_NRMSE_Emax_SSIM.m

```
function Metrics = metrics_NRMSE_Emax_SSIM (I1, I2)
%%% Emax
I_diff = I1 - I2;
I_diff_abs = abs(I_diff);
Emax = max(max(I_diff_abs));


%%% NRMSE
I_diff_sum2 = sumsqr(I_diff);
```

```
I1_sum2 = sumsqr(I1);
NRMSE = sqrt(I_diff_sum2 / I1_sum2);


% SSIM
I1_mean = mean2(I1);
I2_mean = mean2(I2);
I1_var = std2(I1) ^ 2;
I2_var = std2(I2) ^ 2;
I12_cov = mean2(I1 .* I2) - I1_mean * I2_mean;
c = ( (max(max(I1)) - min(min(I1))) / 1e4 ) ^ 2;
SSIM = (2 * I1_mean * I2_mean + c) * (2 * I12_cov + c) /...
    ((I1_mean^2 + I1_mean^2 + c) * (I1_var + I2_var + c));

Metrics = [NRMSE, Emax, SSIM];
```