PTC3492 – Princípios da formação e processamento de imagens médicas

Relatório do EP07 – Filtro adaptativo de Lee para ruído Speckle

Marcelo Monari Baccaro – 8989262

Objetivos

Implementar um filtro de Lee adaptativo 2D para ruído do tipo Speckle. Aplicar este filtro na imagem ruidosa (abaixo à direita), que é equivalente a imagem original (abaixo à esquerda) corrompida por ruído do tipo Speckle com coeficiente de variação (desvio padrão) igual a 0.3. Além disso, calcular as métricas NRMSE, E_{max} e SSIM entre a imagem filtrada e a imagem sem ruído abaixo. E fazer o mesmo entre a imagem ruidosa e a sem ruído. Por fim, comparar essas métricas para analisar melhoras.

Metodologia

Antes de implementar um filtro, é necessário ter um modelo matemático de como o processo estocástico do ruído se comporta. Para o ruído do tipo Speckle, um modelo utilizado é o de ruído multiplicativo:

$$g(x) = f(x) \cdot (1 + \gamma \cdot N(0,1))$$

O filtro adaptativo de Lee assume uma relação afim entre o estimador $\hat{f}(x)$ e a saída $g(x)$ para tentar estimar $f(x)$ a partir de $g(x)$:

$$\hat{f}(x) = a + b \cdot g(x)$$

Utilizando otimização não-linear sem restrição e usando como funcional o valor esperado do quadrado da diferença entre o estimador $\hat{f}(x)$ e o sinal $f(x)$, é possível chegar na expressão do estimador ótimo do filtro de Lee adaptativo para ruído do tipo Speckle:

$$\hat{f}(x) = \mu_g \cdot \alpha_s + (1 - \alpha_s) \cdot g(x)$$

Em que $\alpha_s = \frac{c_{g_H}}{c_g}$ para $c_{g_H} = \frac{\sigma_{g_H}^2}{E\left\{\left(g_H(x)\right)^2\right\}}$ e $c_g = \frac{\sigma_g^2}{E\left\{\left(g(x)\right)^2\right\}}$

A estratégia do filtro de Lee é calcular as estatística de g de maneira local, ou seja, para cada pixel, ele aplica uma janela móvel de tamanho fixo para calcular μ_g e $E\left\{\left(g(x)\right)^2\right\}$ para chegar em σ_g. Ele faz o mesmo para uma região em que f seria homogêneo para calcular as estatísticas de g_H, mas estas só precisam ser calculadas uma única vez. Assim, α_s é calculada para cada pixel. Nos resultados deste relatório, foram usados janelas de tamanho 3x3 e a região homogênea é delimitada por um ROI (Region of Interest) retangular.

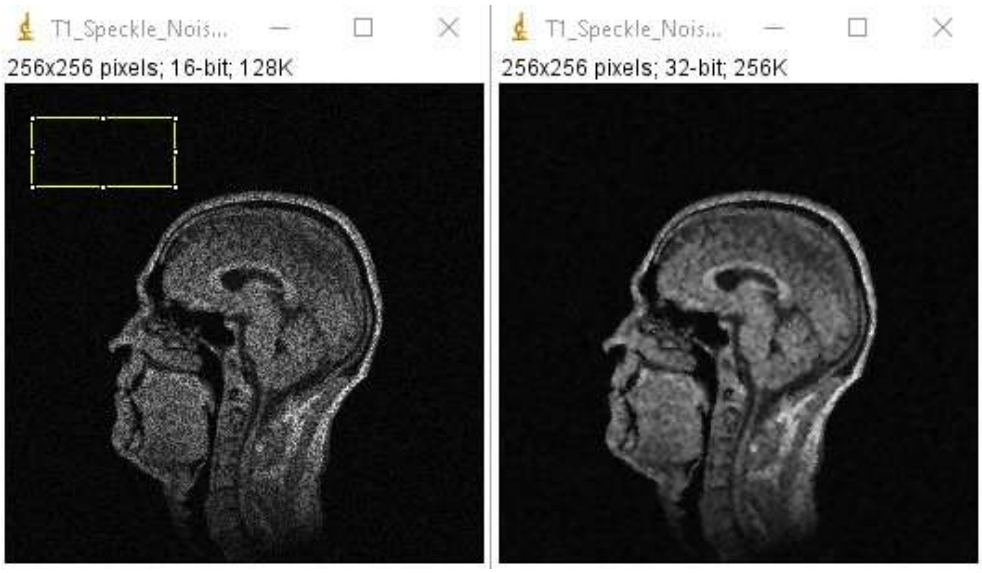
Abaixo segue a listagem do plugin para o ImageJ que implementa este filtro:

```
import java.awt.Rectangle;
import ij.*;
import ij.process.*;
import ij.plugin.filter.*;

public class EP07_Speckle_Lee_Filter implements PlugInFilter {
    ImagePlus imp;

    public int setup(String arg, ImagePlus imp) {
        this.imp = imp;
        return DOES_ALL;
    }

    public void run(ImageProcessor ip) {
        FloatProcessor ip_float = (FloatProcessor)ip.convertToFloat();

        int w = ip_float.getWidth();
        int h = ip_float.getHeight();



        // NOISE VARIANCE ESTIMATION -> ROI
        Rectangle roi_s = ip.getRoi();

        int r0 = roi_s.y;
        int c0 = roi_s.x;
        int hROI = roi_s.height;
        int wROI = roi_s.width;
        int HROI = hROI * wROI;

        double vr, sum_ROI = 0, sum_ROI_sq = 0;

        for (int y=r0; y<r0+hROI; y++)
            for (int x=c0; x<c0+wROI; x++) {
                vr = ip_float.getf(x,y);
                sum_ROI = sum_ROI + vr;
                sum_ROI_sq = sum_ROI_sq + vr * vr;
            }
```

```
        double mean_ROI = sum_ROI / HROI;
        double mean_ROI_sq = sum_ROI_sq / HROI;
        double var_ROI = mean_ROI_sq - mean_ROI * mean_ROI;



        // SPECKLE LEE FILTER
        int nc = 1; // dimensions of the square filter -> (2*nc+1) X (2*nc+1)
        int N_win = (2*nc+1) * (2*nc+1); // number of elements of the filter
        int xc, yc; // current x and y

        double c_ROI = var_ROI / mean_ROI_sq;
        double v, sum_win, sum_win_sq, mean_win, mean_win_sq, var_win, c_win, alpha, f_est;

        FloatProcessor ip_speckle_lee = new FloatProcessor (w, h);

        for (int y=0; y<h; y++)
              for (int x=0; x<w; x++) {
                    sum_win = 0;
                    sum_win_sq = 0;

                    for (int r=-nc; r<nc+1; r++) // row -> y
                          for (int c=-nc; c<nc+1; c++) { // column -> x

                                // Symmetric boundary
                                if (x+c < 0) xc = nc;
                                else if (!(x+c < w)) xc = w - nc;
                                else xc = x + c;

                                if (y+r < 0) yc = nc;
                                else if (!(y+r < h)) yc = h - nc;
                                else yc = y + r;

                                v = ip_float.getf(xc, yc);

                                sum_win = sum_win + v;
                                sum_win_sq = sum_win_sq + v * v;
                          }

                    mean_win = sum_win / N_win;
                    mean_win_sq = sum_win_sq / N_win;
                    var_win = mean_win_sq - mean_win * mean_win;
                    c_win = var_win / mean_win_sq;

                    alpha = c_ROI / c_win;
                    if (alpha > 1) alpha = 1;
                    f_est = (1 - alpha) * ip_float.getf(x,y) + alpha * mean_win;
                    ip_speckle_lee.setf(x, y, (float)f_est);
              }



        // OUTPUT
        ImagePlus imp_speckle_lee = new ImagePlus ("Lee filter for Speckle Noise", ip_speckle_lee);
        imp_speckle_lee.show();
    }
}
```

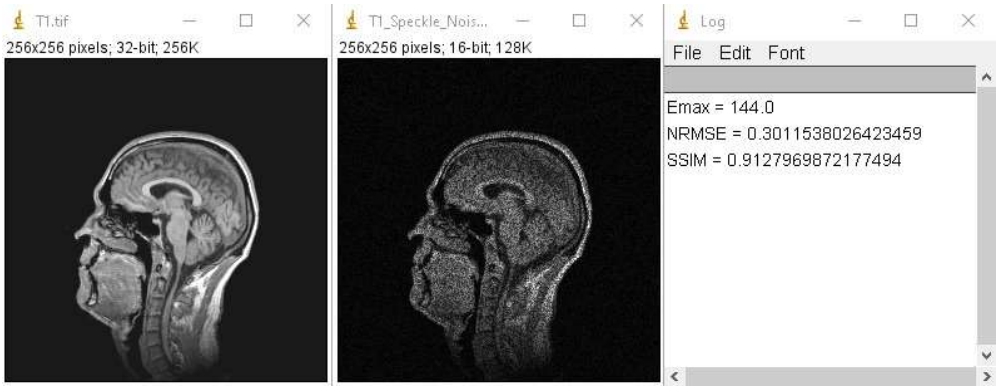
Resultados

1) Saída do filtro de Lee adaptativo para ruído do tipo Speckle:



2) Comparação das métricas NRMSE, E_{max} e SSIM (uso do plugin do EP02):

2.1) Imagem original com a ruidosa:



2.2) Imagem original com a filtrada por filtro de Lee para ruído Speckle: