MO815 - Tópicos em Processamento de Imagens
Relatório TP1

Aluno: André Vital Saúde RA:961954
Professor: Neucimar J. Leite

24 de março de 2003

1 Filtragem da Média

Atividade: Implemente um programa que realiza a filtragem da media a partir da convolucao de uma imagem ruidosa com uma mascara de tamanho nxn (n, impar, eh um parametro de entrada).

Abaixo está listado o fonte da função que faz a filtragem pela média:

```
IMAGE avg(IMAGE x, int n)
{
int i,j,mi,mj;
    IMAGE y;
    unsigned int av;

    y = newimage(x->info->nr, x->info->nc);
for (i = n/2; i < x->info->nr-n/2; i++) {
        for (j = n/2; j < x->info->nc-n/2; j++) {
            av = 0;
            //Average calculation
            for (mi = i - n/2; mi <= i + n/2; mi++) {
                for (mj = j - n/2; mj <= j + n/2; mj++) {
                    av += x->data[mi][mj];
                }
            }
            y->data[i][j] = (unsigned char)(av / (n*n));
        }
    }

    return(y);
}
```

Esta filtragem foi utilizada na imagem Lenna-ruido.pgm obtendo os seguintes resultados:

2 Filtro de Suavização Gaussiano

Atividade: Implemente o filtro de suavizacao gaussiano (separavel), tal como abordado em sala de aula, considerando n=4*sigma + 1 (n eh o parametro de entrada do filtro).

Abaixo está listado o fonte que implementa o filtro de suavização gaussiano:

```
IMAGE gauss(IMAGE x, int n)
{
int i,j,mi,mj;
    double *umask, sigma, g;
    double **mask;
    IMAGE y;
```

Figura 1: Imagem com ruído

Figura 2: Imagem após filtragem da média

```
    //mask creation
    umask = (double *)malloc(sizeof(double)*n);
    mask = (double **)malloc(sizeof(double *)*n);
    sigma = ((double)n-1)/4;
    for (i = 0; i < n; i++) {
        umask[i] = (1 / (sqrt(2*PI) * sigma) *
                exp(-((i - ((double)n-1)/2)*(i - ((double)n-1)/2))/(2*sigma*sigma)));
    }

    //multiplication
    for (i = 0; i < n; i++) {
        mask[i] = (double *)malloc(sizeof(double)*n);
        for (j = 0; j < n; j++) {
            mask[i][j] = umask[i] * umask[j];
        }
    }

    y = newimage(x->info->nr, x->info->nc);
for (i = n/2; i < x->info->nr-n/2; i++) {
        for (j = n/2; j < x->info->nc-n/2; j++) {
            g = 0;
            //calculation
            for (mi = -n/2; mi <= n/2; mi++) {
                for (mj = -n/2; mj <= n/2; mj++) {
                    g += (double)(mask[mi + n/2][mj + n/2] * x->data[i + mi][j + mj]);
                }
            }
            y->data[i][j] = (unsigned char)g;
        }
    }

    return(y);
}
```

Veja os resultados:

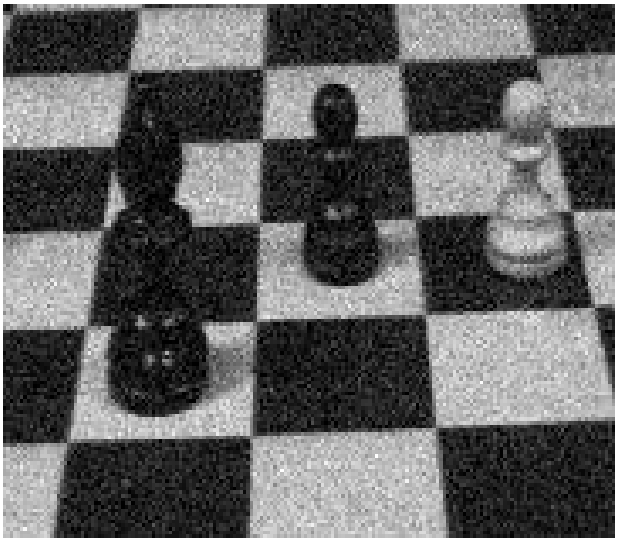

Figura 3: Imagem com ruído gaussiano de sigma=16, gerada usando o programa gnoise

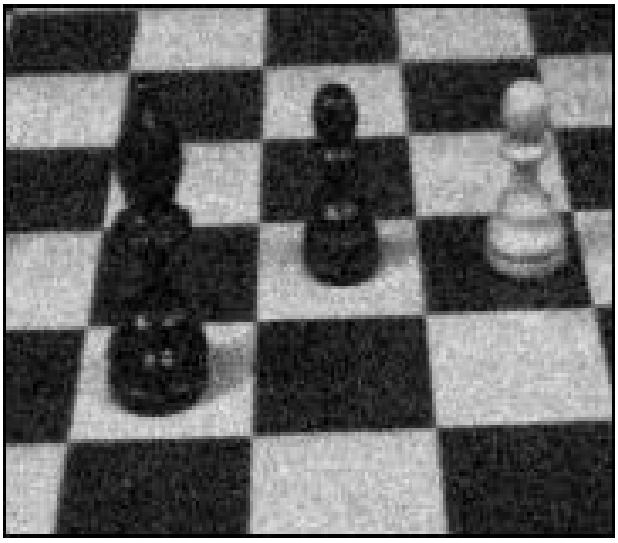

Figura 4: Imagem após o uso do filtro de suavização gaussiano

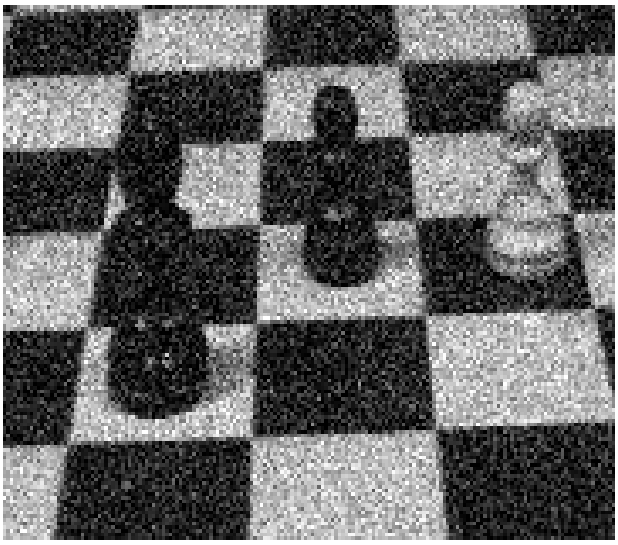

Figura 5: Imagem com ruído gaussiano de sigma=30, gerada usando o programa gnoise

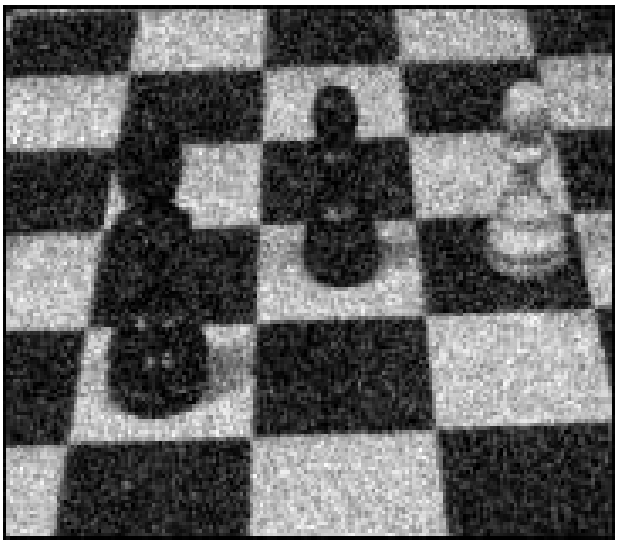

Figura 6: Imagem após o uso do filtro de suavização gaussiano

3 Tempo de Execução

Atividade: Compare o tempo de execucao dos dois procedimentos de filtragem.

O tempo de execução de ambos os filtros depende somente das dimensões da imagem. O tipo e o nível do ruído não influem. Rodamos ambos os filtros na imagem Lenna-ruido.pgm e obtivemos os seguintes resultados:

4 Filtro da Mediana

Atividade: Aplique o filtro da mediana (diferentes tamanhos de janela NxN) as imagens com ruido impulsivo e gaussiano e analise os resultados.

Abaixo está listado o fonte que implementa o filtro de suavização gaussiano:

```
IMAGE mediana(IMAGE x, int n)
{
int i,j,mi,mj;
    IMAGE y;
    unsigned char med, *ord;

    y = newimage(x->info->nr, x->info->nc);
    ord = (unsigned char *)malloc(n*n);

for (i = n/2; i < x->info->nr-n/2; i++) {
        for (j = n/2; j < x->info->nc-n/2; j++) {
            //calculation
            for (mi = -n/2; mi <= n/2; mi++) {
                for (mj = -n/2; mj <= n/2; mj++) {
                    ord[(mi + n/2) * n + (mj + n/2)] = x->data[i + mi][j + mj];
                }
            }
            sort(ord, n*n);
            y->data[i][j] = ord[n];
        }
    }

    return(y);
}
```

Abaixo está a aplicação do filtro da mediana sobre a imagem com ruído impulsivo Lenna-ruido.pgm:

Abaixo está a aplicação do filtro da mediana sobre a imagem com ruído gaussiando sigma=30:

5 Erro Médio Quadrático e Relação Sinal-Ruído

Atividade: Compare objetivamente os resultados acima, considerando o erro medio-quadratico e a relacao sinal-ruido das imagens.

Ruído impulsivo Lenna-ruido.pgm:

- Filtro da média: ERMS = nan; SNR = -0.996750
- Filtro gaussiano: ERMS = nan; SNR = -1.005019
- Filtro da mediana: ERMS = nan; SNR = -0.863484

Ruído gaussiano chess.pgm com sigma=30:

- Filtro da média: ERMS = nan; SNR = -0.987488
- Filtro gaussiano: ERMS = nan; SNR = -1.009445

Figura 7: Imagem com ruído impulsivo

Figura 8: Imagem após o uso do filtro da mediana

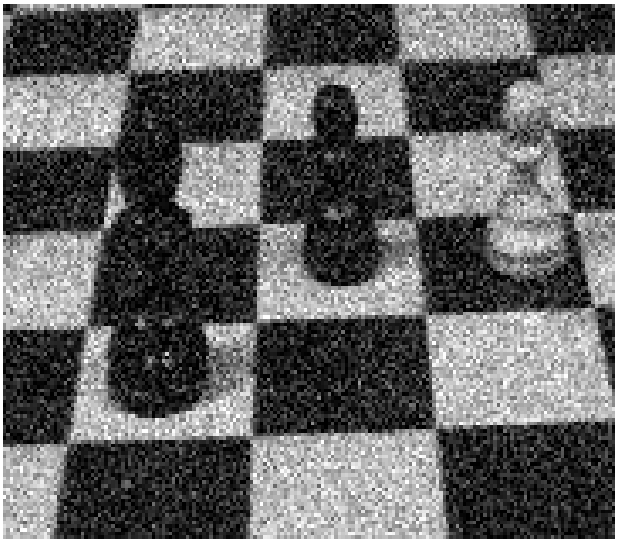

Figura 9: Imagem com ruído gaussiano de sigma=30, gerada usando o programa gnoise

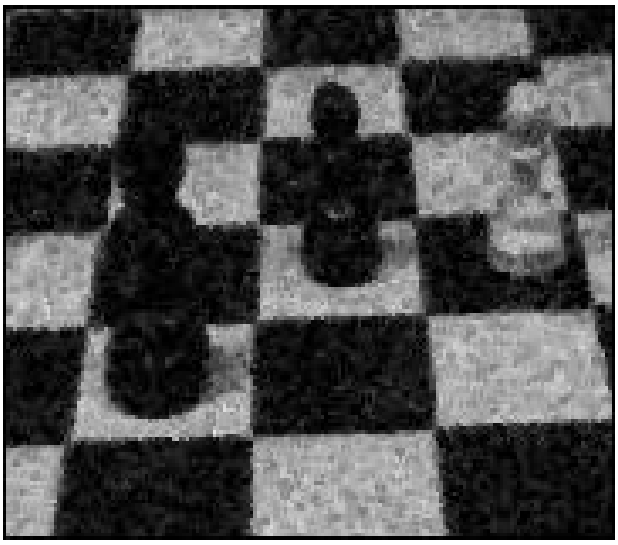

Figura 10: Imagem após o uso do filtro da mediana

- Filtro da mediana: ERMS = nan; SNR = -0.861107

Não pude compreender o erro no meu algoritmo do ERMS para que tenha dado sempre NAN. Abaixo estão listados o código de ambas as funções:

```
double erms(IMAGE x, IMAGE y)
{
int i,j;
    double e=0;

    for (i=0; i<x->info->nr; i++) {
        for (j = 0; j < x->info->nc; j++) {
            e += (double)SQ((double)x->data[i][j]-(double)y->data[i][j]);
        }
    }
    e = sqrt(e / (double)(x->info->nr * x->info->nc));

    return(e);
}

double snr(IMAGE x, IMAGE y)
{
int i,j;
    double e=0,s=0;

    for (i=0; i<x->info->nr; i++) {
        for (j = 0; j < x->info->nc; j++) {
            e += (double)SQ((double)x->data[i][j]-(double)y->data[i][j]);
            s += (double)SQ((double)x->data[i][j]);
        }
    }
    s /= e;

    return(s);
}
```