ECE 637 Lab 7 Report

Image Restoration

Xihui Wang

Section1 – Minimum Mean Square Error (MMSE) Linear Filters

1.1 The four original images

Fig 1-1-1 The original *img14g.tif*

Fig 1-1-2 The blurred version *img14bl.tif*

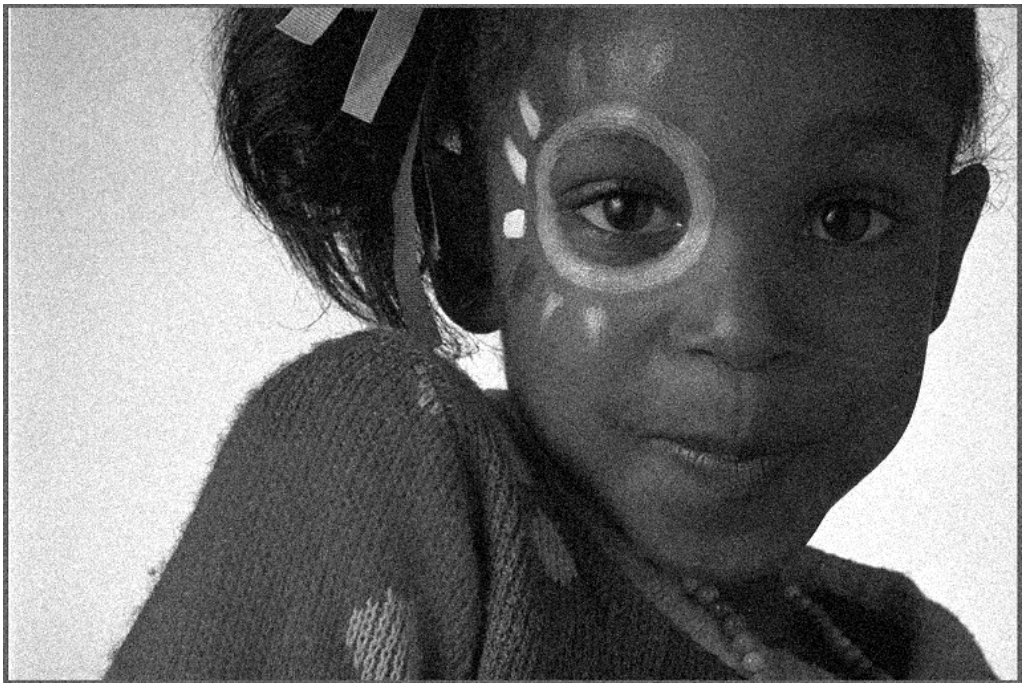

Fig 1-1-3 The noisy version *img14gn.tif*

Fig 1-1-4 The noisy version *img14sp.tif*

1.2 The output of the optimal filtering for the blurred image and the two noisy images

Fig 1-2-1 The output of the optimal filtering for blurred image *img14bl.tif*

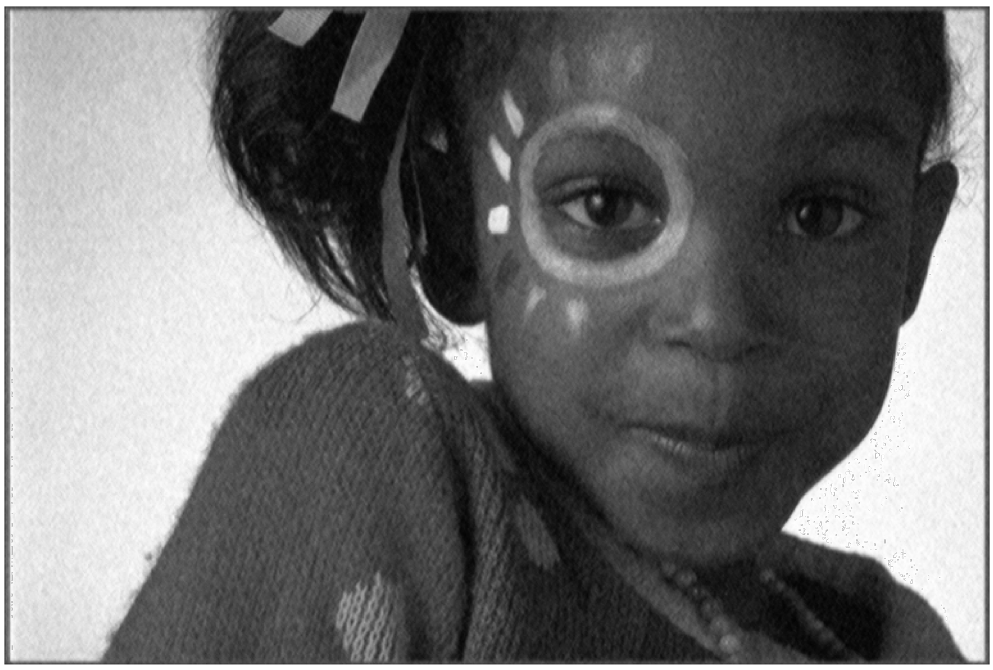

Fig 1-2-2 The output of the optimal filtering for noisy image *img14gn.tif*

Fig 1-2-3 The output of the optimal filtering for noisy image *img14sp.tif*

1.3 The MMSE filters that I computed from the blurred image and two noisy images

Table – 1 Filter coefficient θ^* calculated from *img14bl.tif*

$\theta^* =$

0.3720	0.2052	-0.9682	1.0572	0.1961	-1.0020	0.9254
-0.0431	0.4069	-1.2219	-0.0281	-0.6146	-1.3229	0.4024
-0.3541	-0.3242	-0.4810	0.3321	0.7580	-0.0871	-0.7923
1.1089	-2.4308	1.9317	3.7782	1.5691	-0.0701	0.0615
0.3791	-0.4590	-1.1045	1.2263	0.8358	-1.4710	0.3905
-1.0990	-0.1802	-0.2944	1.0624	-1.8928	-1.9628	0.8126
1.1560	0.4776	-1.7439	0.6483	0.2948	0.2604	0.3042

Table – 2 Filter coefficient θ^* calculated from *img14gn.tif*

$\theta^* =$

0.0165	0.0259	0.0044	0.0050	-0.0080	0.0302	-0.0259
-0.0055	0.0053	0.0355	0.0205	0.0464	0.0091	0.0066
-0.0105	-0.0125	0.0674	0.0731	0.0470	0.0290	-0.0030
-0.0091	-0.0153	0.0476	0.2306	0.0891	-0.0175	0.0011
-0.0050	-0.0222	0.0423	0.1117	0.0650	-0.0118	0.0069
-0.0044	0.0079	0.0307	0.0268	0.0088	-0.0063	0.0192
-0.0053	-0.0043	0.0154	0.0127	0.0140	0.0183	0.0054

Table – 3 Filter coefficient θ^* calculated from *img14sp.tif*

$\theta^* =$

0.0080	0.0048	-0.0016	-0.0050	0.0257	-0.0209	-0.0185
0.0017	-0.0016	0.0558	0.0267	0.0435	0.0214	0.0196
-0.0010	0.0042	0.0413	0.0968	0.0212	-0.0196	0.0199
-0.0014	-0.0203	0.0350	0.2652	0.1492	-0.0287	0.0083
0.0252	0.0023	0.0612	0.0965	0.0154	-0.0412	0.0233
-0.0099	-0.0006	0.0313	0.0497	0.0143	0.0038	0.0131
-0.0407	0.0162	-0.0068	0.0100	0.0079	0.0129	-0.0110

Section2 – Weighted Median Filtering

2.1 The results of median filtering

Fig 2-1-1 The output of the optimal filtering for blurred image *img14bl.tif*

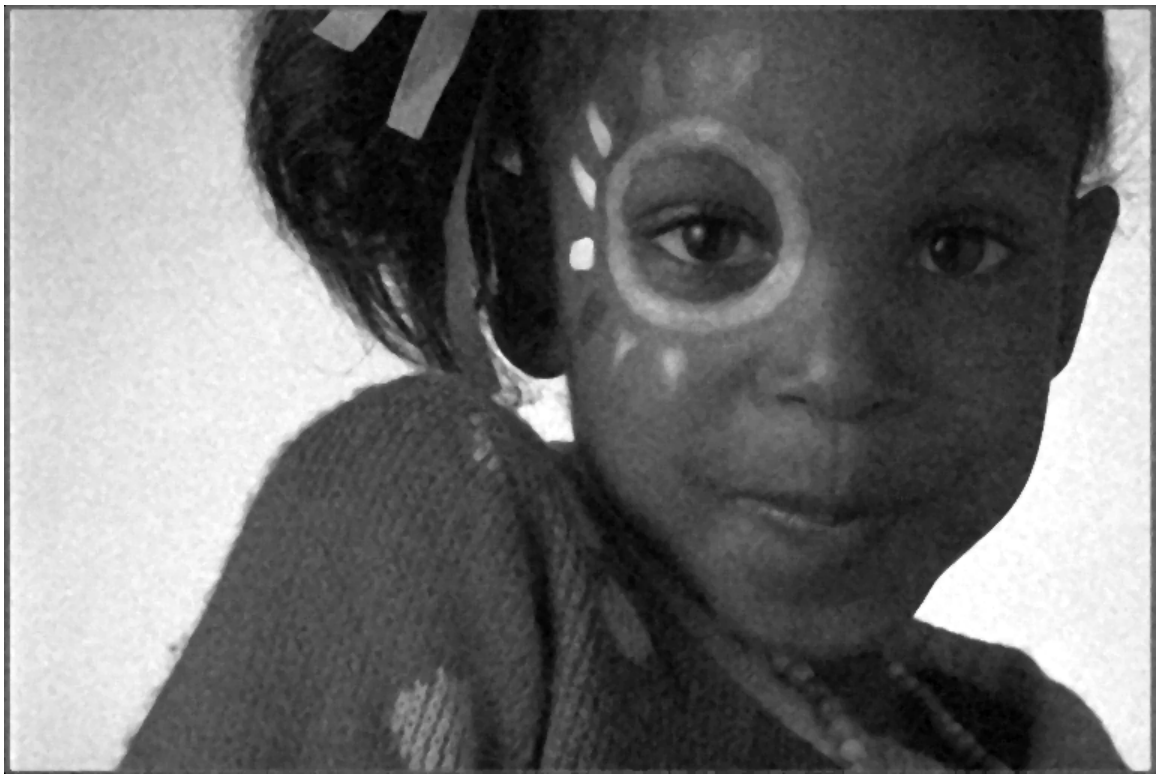

Fig 2-1-2 The output of the optimal filtering for noisy image *img14gn.tif*

Fig 2-1-3 The output of the optimal filtering for noisy image *img14sp.tif*

2.2 The C code

```
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

struct data
{
    int weight;
    double value;
};
```

```
double wmfilter(double **img, int i, int j);

int main(int argc, char const *argv[])
{
    FILE *fp;
    struct TIFF_img input_img, output;

    /* open image file */
    if ( ( fp = fopen ( "img14gn.tif", "rb" ) ) == NULL ) {
       fprintf ( stderr, "cannot open file %s\n", argv[1] );
       exit ( 1 );
    }

    /* read image */
    if ( read_TIFF ( fp, &input_img ) ) {
       fprintf ( stderr, "error reading file %s\n", argv[1] );
       exit ( 1 );
    }

    /* close image file */
    fclose ( fp );

    double **img;
    img = (double
**)get_img(input_img.width+4,input_img.height+4,sizeof(double));
    int i, j;

    for(i=0; i<input_img.height+4; i++)
    {
        for(j=0; j<input_img.width+4; j++)
```

```
    {
        if((i>1)&&(j>1))
        {
            img[i][j] = input_img.mono[i-2][j-2];
        }
        else
        {
            img[i][j] = 0;
        }
    }
}

get_TIFF ( &output, input_img.height, input_img.width, 'g' );

for(i=2; i<input_img.height+2; i++)
{
    for(j=2; j<input_img.width+2; j++)
    {
        output.mono[i-2][j-2] = wmfilter(img, i, j);
    }
}

/* open output image file */
if ( ( fp = fopen ( "output.tif", "wb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file output.tif\n");
    exit ( 1 );
}

/* write output image */
if ( write_TIFF ( fp, &output ) ) {
```

```
        fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
        exit ( 1 );
    }

    /* close output image file */
    fclose ( fp );
    free_img( (void**)img );
    free_TIFF ( &(input_img) );
    free_TIFF ( &(output) );
    return 0;
}



double wmfilter(double **img, int i, int j)
{
    int k,l,count;
    struct data temp[25];
    struct data t;
    count = 0;

    // get the pixel value and its weight
    for(k=0; k<5; k++)
    {
        for(l=0; l<5; l++)
        {
            if ((k!=0)&&(k!=4)&&(l!=0)&&(l!=4))
            {
                temp[count].weight = 2;
            }
```

```
            else
            {
                temp[count].weight = 1;
            }
            temp[count++].value = img[i+k-2][j+l-2];
        }
    }

    // sort
    for(count=0; count<24; count++)
    {
        for(k=count+1; k<25; k++)
        {
            if(temp[count].value < temp[k].value)
            {
                t = temp[count];
                temp[count] = temp[k];
                temp[k] = t;
            }
        }
    }
    // find the weighted median
    int sum_h = 0;
    int sum_t = 0;
    for(count=0; count<25; count++)
    {
        sum_h += temp[count].weight;
        sum_t = 34 - sum_h;
        if(sum_h >= sum_t)
        {
```

```
                return temp[count].value;
            }
        }
        return 0.0;
    }
```