Lab 7 Report

Ruijie Song

Apr.2.2021

1 Minimum Mean Square Error (MMSE) Linear Filters

1.

Figure 1. img14bl

Figure 2. img14g

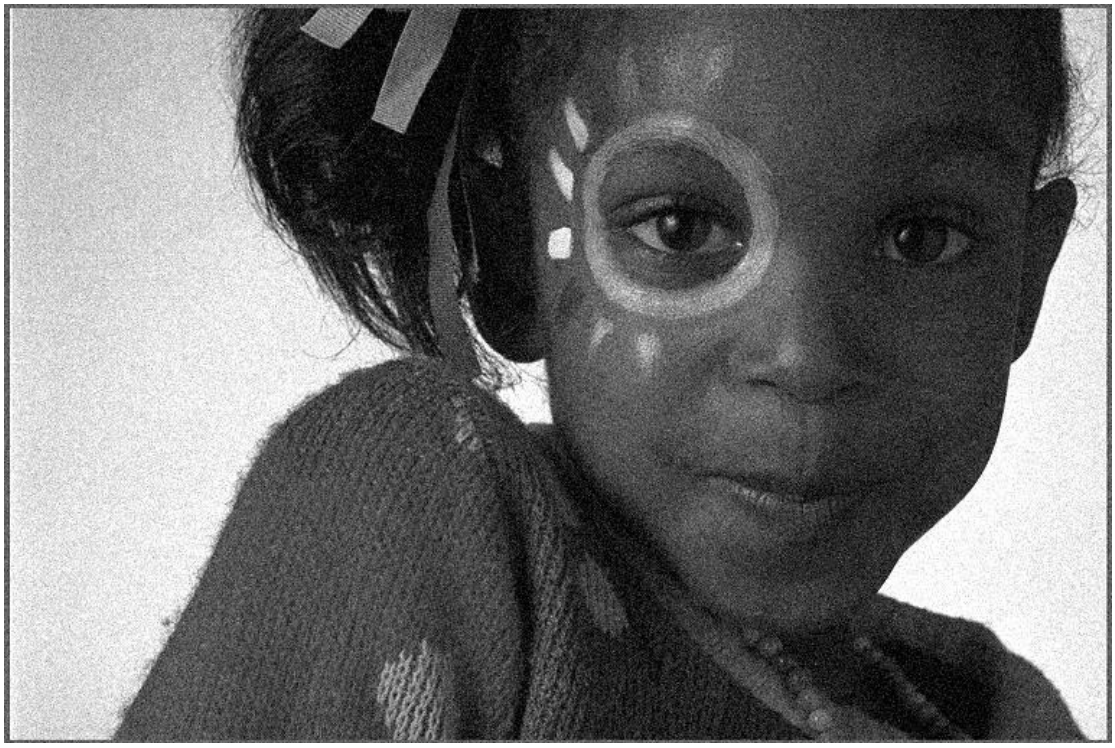

Figure 3. img14gn

Figure 4. img14sp

2.

Figure 5. output for the blurred image

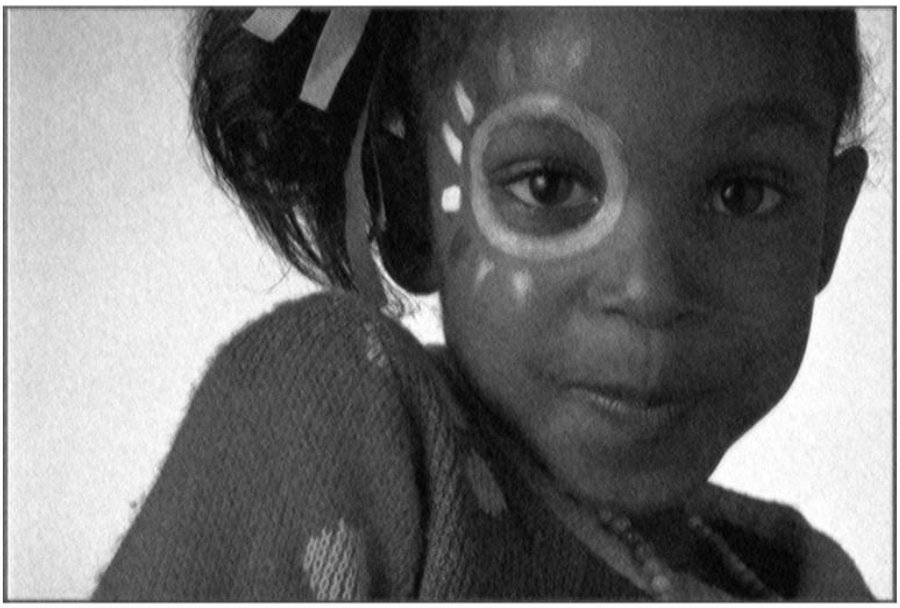

Figure 6. output from img14gn

Figure 7. output from img14sp

3.

-0.2238	-0.0481	-0.8803	1.5824	-0.0223	-1.0231	0.7140
1.4622	1.3026	-1.6030	0.4277	-0.9096	-1.2325	0.1544
-1.5240	-0.3205	-1.8417	0.3442	0.6119	-1.2780	0.7424
0.9781	-2.4388	2.3202	4.2414	2.3817	-0.8995	0.2587
0.7760	-0.6760	-1.3455	1.4239	1.4107	-0.2431	0.0724
-1.5503	0.4792	-0.6271	0.4669	-2.3104	-2.4858	-0.2879
1.1180	1.0782	-2.3796	1.3472	0.2899	-0.0413	1.2129

Figure 8. theta from bl

0.0288	0.0344	0.0228	0.0189	-0.0380	0.0359	-0.0327
-0.0169	0.0142	0.0261	0.0454	0.0406	0.0049	0.0220
-0.0146	-0.0142	0.0648	0.0683	0.0274	0.0242	-0.0071
-0.0188	-0.0453	0.0487	0.2368	0.0925	-0.0036	0.0124
0.0087	-0.0439	0.0214	0.1212	0.0566	-0.0114	-0.0048
0.0285	9.2028e-04	0.0362	0.0305	9.3343e-04	-0.0072	0.0108
-0.0085	3.7111e-04	0.0160	0.0124	0.0151	0.0185	0.0278

Figure 9. theta from gn

0.0061	0.0082	0.0163	0.0085	0.0391	-0.0206	-0.0314
0.0074	-0.0077	0.0220	0.0440	-8.0247e-...	0.0037	0.0223
-0.0102	0.0076	0.0443	0.1101	0.0384	-0.0161	0.0338
6.8148e-04	-0.0221	0.0288	0.3299	0.1336	-0.0215	0.0131
0.0236	-0.0375	0.0368	0.0871	-0.0032	-0.0387	0.0248
-0.0108	-0.0179	0.0209	0.0583	0.0264	0.0030	0.0013
2.5705e-04	0.0272	-0.0022	0.0106	-2.8555e-...	0.0129	-0.0073

Figure 10. theta from sp

2 Weighted Median Filtering

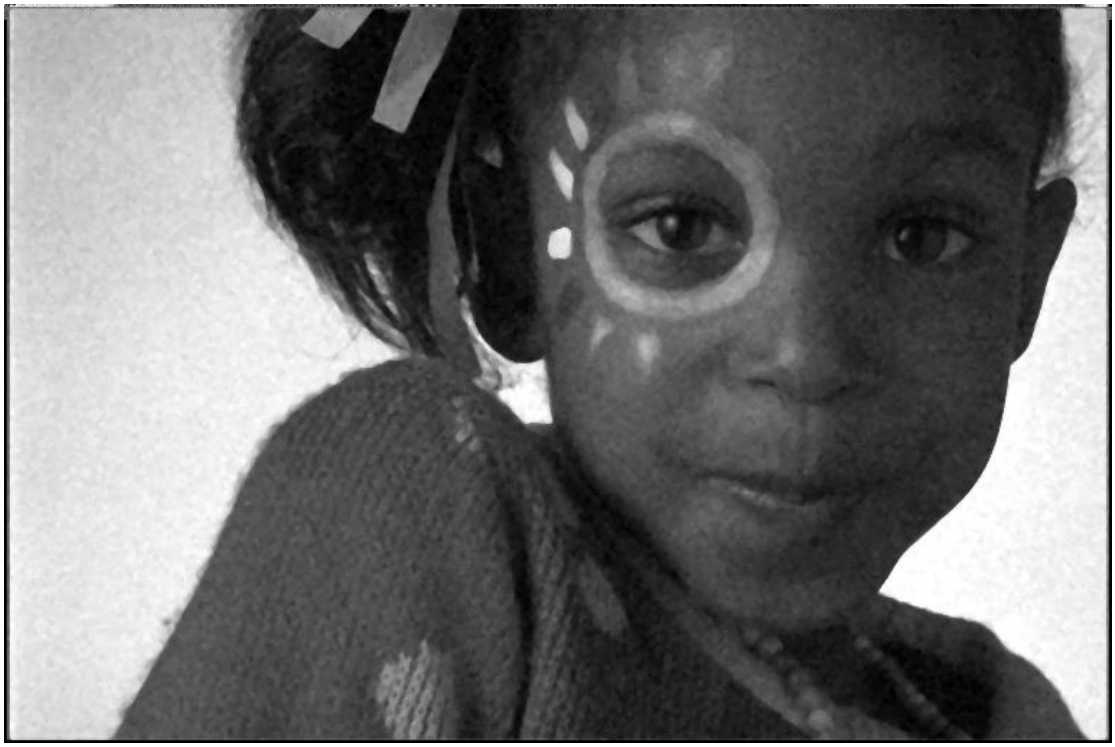

Figure 11. results from gn

Figure 12. results from sp

```c

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

void error(char *name);

// sort array
void sort(int *pixel, int *weight, int length)
{
    int i, j;
    int v;
    for(i = 0; i < length - 1; i ++)
        for(j = i+1; j < length; j ++)
        {
            if(pixel[i] < pixel[j])
            {
                v = pixel[i];
                pixel[i] = pixel[j];
                pixel[j] = v;

                v = weight[i];
                weight[i] = weight[j];
                weight[j] = v;
            }
        }
}

// input a pixel in a image, output the weighted median
int WMfilter(struct TIFF_img input_img, int x, int y)
{
    int pixel[25]; // pixels by 5*5 filter
    int weight[25] = {1,1,1,1,1,1,2,2,2,1,1,2,2,2,1,1,2,2,2,1,1,1,1,1,1}; //
weighted factors
    int i, j, k=0;
    // int istar; // output
    // get 5*5 window from image
    for (i=x-2; i<=x+2; i++)
    {
        for (j=y-2; j<=y+2; j++)
        {
            pixel[k] = input_img.mono[i][j];
            k = k + 1;
        }
    }
    // sort the pixel and weight
    sort(pixel,weight,25);
    // find i*
    int temp = 0;
    for (i=0; i<25; i++)
    {
        temp = temp + weight[i];
        if (temp >= 17)
        {
            break;
        }
    }
    return pixel[i];
```

```
}



int main (int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, output_img;
    // double **img1,**img2;
    int32_t i,j;

    printf("processstart");

    if ( argc != 2 ) error( argv[0] );

    /* open image file */
    if ( ( fp = fopen ( argv[1], "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 );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'g' ) {
    fprintf ( stderr, "error:  image must be grey scale\n" );
    exit ( 1 );
    }

    /* Allocate image of double precision floats */
    // img1 = (double **)get_img(input_img.width,input_img.height,sizeof(double));
    // img2 = (double **)get_img(input_img.width,input_img.height,sizeof(double));

    /* set up structure for output achromatic image */
    /* to allocate a full color image use type 'c' */
    get_TIFF ( &output_img, input_img.height, input_img.width, 'g' );

    /*
    // copy image component to double array
    for ( i = 0; i < input_img.height; i++ )
    for ( j = 0; j < input_img.width; j++ ) {
    img1[i][j] = input_img.mono[i][j];
    }
    */

    /* Filter image */
    for ( i = 2; i < input_img.height-2; i++ )
    for ( j = 2; j < input_img.width-2; j++ ) {
    output_img.mono[i][j] = WMfilter(input_img,i,j);
    }

    /*fill in boundary pixels*/
    for ( i = 0; i < 2; i++ )
```

```
    for ( j = 2; j < 2; j++ ) {
    output_img.mono[i][j] = 0;
    }
    for ( i = input_img.height-2; i < input_img.height; i++ )
    for ( j = input_img.width-2; j < input_img.width; j++ ) {
    output_img.mono[i][j] = 0;
    }
    for ( i = 0; i < 2; i++ )
    for ( j = input_img.width-2; j < input_img.width; j++ ) {
    output_img.mono[i][j] = 0;
    }
    for ( i = input_img.height-2; i < input_img.height; i++ )
    for ( j = 2; j < 2; j++ ) {
    output_img.mono[i][j] = 0;
    }

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

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

    /* de-allocate space which was used for the images */
    free_TIFF ( &(input_img) );
    free_TIFF ( &(output_img) );
    //free_TIFF ( &(color_img) );

    //free_img( (void**)img1 );
    //free_img( (void**)img2 );

    printf("processsuccess");

    return(0);
}

void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds noise,\n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
}
```