Digital Image Processing Laboratory

Image Restoration

Alexandre Olive Pellicer

1. Minimum Mean Square Error (MMSE) Linear Filters
    1.1. Hand in the four original images *img14g.tif, img14bl.tif, img14gn.tif and img14sp.tif*

Fig 1: img14g.tif

Fig 2: img14bl.tif

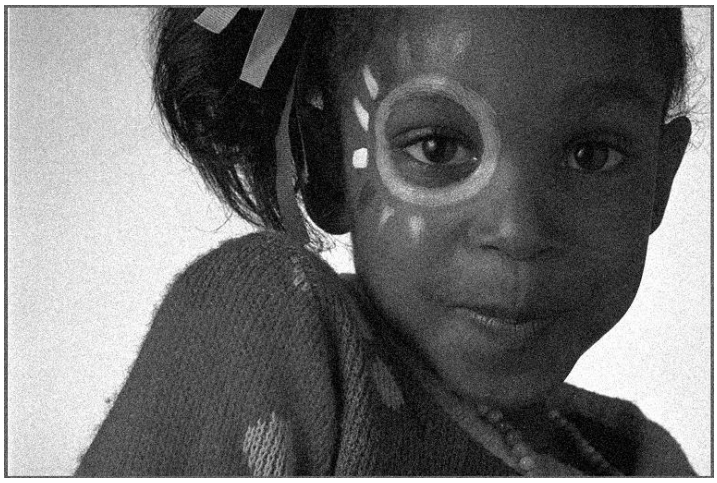


Fig 3: img14gn.tif

Fig 4: img14sp.tif

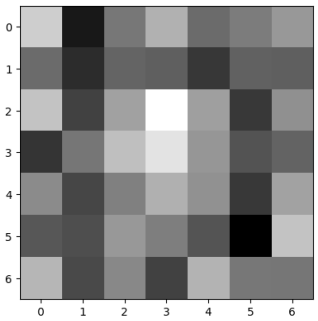
1.2. Hand in the output of the optimal filtering for the blurred image and the two noisy images

Fig 5: Blurred version image after optimal filtering

Fig 6: GN noise image after optimal filtering

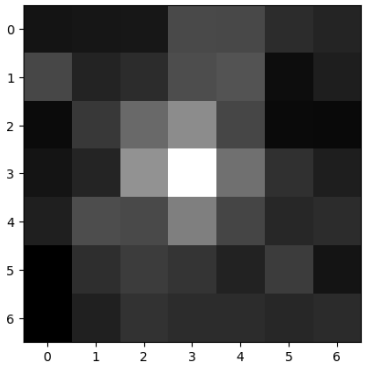
Fig 7: SP noise image after optimal filtering

1.3. Hand the MMSE filters that you computed for the blurred image and the two noisy images. (Each filter is specified by the optimum value of θ^* that you calculated.) Orient each filter into a 7x7 array to make the spatial orientation clear. For each filter, clearly state which corrupted image was used to compute the filter coefficients, θ^*



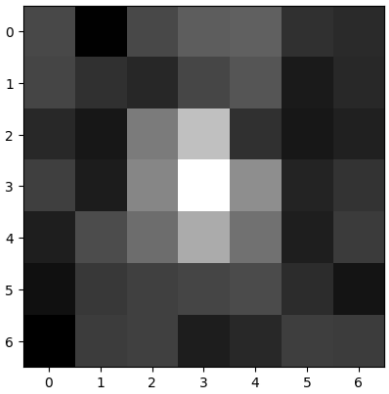
```
[[ 1.5208 -1.6833 -0.0147  0.9993 -0.2206  0.0671  0.5672]
 [-0.2197 -1.3117 -0.3552 -0.4407 -1.1395 -0.4067 -0.4302]
 [ 1.3415 -0.97    0.7195  2.3861  0.6888 -1.1164  0.4174]
 [-1.1754 -0.0361  1.2484  1.8966  0.5231 -0.6413 -0.3591]
 [ 0.3286 -0.8734  0.1385  0.9866  0.4403 -1.1208  0.733 ]
 [-0.578  -0.7402  0.5593  0.1056 -0.636  -2.1178  1.3349]
 [ 1.0864 -0.8139  0.2833 -0.9683  1.0474 -0.0205 -0.0308]]
```

Fig 8: Optimal filter for the blurred version image



```
[[-0.0106 -0.0088 -0.0079  0.0365  0.0359  0.0115  0.0035]
 [ 0.0354  0.0035  0.0111  0.0405  0.0461 -0.0169 -0.0016]
 [-0.0184  0.0214  0.0657  0.0963  0.0337 -0.0189 -0.0201]
 [-0.0106  0.0043  0.1015  0.1991  0.0717  0.0155 -0.0015]
 [-0.0004  0.0407  0.0371  0.0851  0.0333  0.007   0.0119]
 [-0.0285  0.0128  0.0258  0.0179  0.0019  0.025  -0.0105]
 [-0.0284  0.0007  0.0164  0.012   0.0117  0.0066  0.0102]]
```

Fig 9: Optimal filter for the GN noise image



```
[[ 0.026 -0.035  0.025  0.043  0.046  0.005 -0.   ]
 [ 0.023  0.005 -0.003  0.024  0.037 -0.014 -0.002]
 [-0.001 -0.017  0.069  0.129  0.006 -0.017 -0.008]
 [ 0.018 -0.012  0.079  0.183  0.086 -0.006  0.007]
 [-0.011  0.029  0.057  0.111  0.061 -0.011  0.014]
 [-0.023  0.013  0.018  0.023  0.028  0.001 -0.02 ]
 [-0.037  0.016  0.018 -0.012 -0.002  0.017  0.016]]
```

Fig 10: Optimal filter for the SP noise image

2. Weighted Median Filtering

2.1. Hand in your results of median filtering

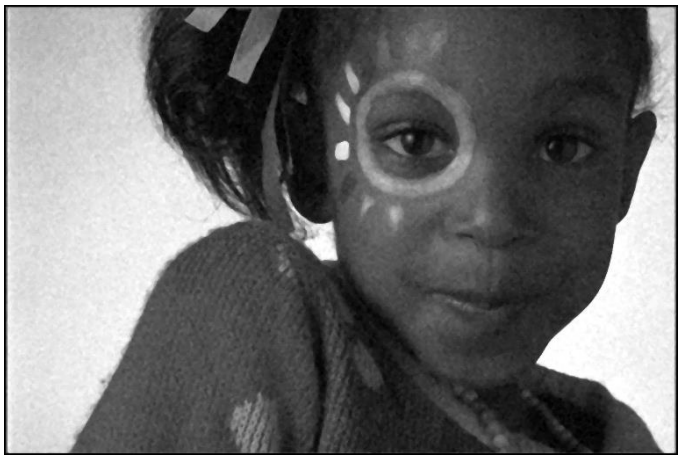


Fig 11: GN noise image after median filtering

Fig 12: SP noise image after median filtering

2.2. Hand in your C code

Find it in the next page

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

void error(char *name);

int main (int argc, char **argv) {
    FILE *fp;
    struct TIFF_img input, output;
    int32_t i,j;

    int aux_pix[5*5];
    int weight_arr[5*5] = {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};

    int a, b, c, d, e, f, sum, limit;

    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 ) ) {
        fprintf ( stderr, "error reading file %s\n", argv[1] );
        exit ( 1 );
    }

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

    /* check the type of image data */
    if ( input.TIFF_type != 'g' ) {
        fprintf ( stderr, "error:  image not grey\n" );
        exit ( 1 );
    }

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

    for (int i = 0; i < 5*5; i++) {
        sum += weight_arr[i];
    }
    limit = sum / 2;

    /* Filter image along horizontal direction */
    for ( i = 2; i < input.height-2; i++ ){
        for ( j = 2; j < input.width-2; j++ ){
            c = 0;
            for (a=i-2; a<=i+2; a++){
                for (b=j-2; b<=j+2; b++)
                {
                    aux_pix[c] = input.mono[a][b];
                    c = c + 1;
                }
            }

            for(d = 0; d < 5*5 - 1; d ++){
                for(e = d+1; e < 5*5; e ++){
                    if(aux_pix[d] < aux_pix[e]){
                        f = aux_pix[d];
                        aux_pix[d] = aux_pix[e];
                        aux_pix[e] = f;
```

```
                    f = weight_arr[d];
                    weight_arr[d] = weight_arr[e];
                    weight_arr[e] = f;
                }
            }
        }

        int aux = 0;
        for (a=0; a<5*5; a++){
            aux = aux + weight_arr[a];
            if (aux >= limit){
                break;
            }
        }
        output.mono[i][j] = aux_pix[a];
    }
}

/* open output image file */
if ( ( fp = fopen ( "filtered.tif", "wb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file filtered.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 );

/* de-allocate space which was used for the images */
free_TIFF ( &(input) );
free_TIFF ( &(output) );

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);
}
```