Lab 7: Image Restoration

Course Title: Image Processing I (Spring 2022)

Course Number: ECE 63700

Instructor: Prof. Charles A. Bouman

Author: **Zhankun Luo**

Lab 7: Image Restoration

- 1. Mean Square Error (MMSE) Linear Filters
  - 1.1. the four original images `img14g.tif`, `img14bl.tif`, `img14gn.tif` and `img14sp.tif`
  - 1.2. the output of the optimal filtering for the blurred image and the two noisy images
  - 1.3. the MMSE filters $\theta^{\star}$ for the blurred image and the two noisy images
- 2. Weighted Median Filtering
  - 2.1. results of median filtering for noisy images `img14gn.tif` and `img14sp.tif`
  - 2.2. listing of C codes
- Appendix
  - Python code for MMSE linear filtering: `utils.py`
  - Python code for solution
    - solution to section 1: `soln_1.py`
  - C codes for weighted median filtering : `restore.h`, `restore.c`
  - C codes for solutions
    - solution to section 2: `soln_2.c`

1. Mean Square Error (MMSE) Linear Filters

1.1. the four original images `img14g.tif`, `img14bl.tif`, `img14gn.tif` and `img14sp.tif`

solution

Original Grayscale Image img14g.tif (left), and Blured Image imag14bl.tif (right)

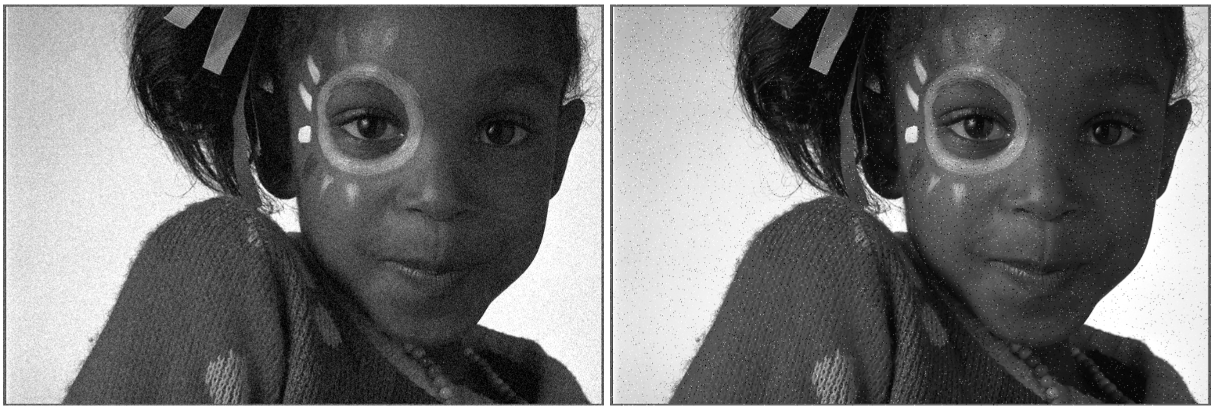


Image with Gaussian Noise img14gn.tif (left), and Image with Salt and Pepper Noise img14sp.tif (right)

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

solution

Filtered Result for the Blured Image img14bl.tif

Filtered Result for the Noisy Image img14gn.tif

Filtered Result for the Noisy Image img14sp.tif

1.3. the MMSE filters $\theta^\star$ for the blurred image and the two noisy images

solution

the MMSE filters $\theta^\star$ for the blurred image `img14bl.tif`

$$\theta^\star = \begin{bmatrix} 1.7122 & 0.7401 & 0.928 & 0.8153 & -0.9378 & -1.8137 & 1.8075 \\ -1.4864 & -1.8092 & -0.9006 & -0.5821 & -2.9833 & 0.5828 & 1.3537 \\ -0.9434 & -2.7623 & 0.3063 & 2.9781 & 0.7147 & -2.7348 & -0.8187 \\ 2.0282 & -0.5532 & 3.635 & 3.4485 & 3.2368 & -3.1583 & 0.6775 \\ 1.6263 & -3.0881 & -0.4136 & 5.0773 & -0.1598 & -1.4265 & 0.7996 \\ -0.3035 & -1.874 & -2.094 & -2.2209 & -0.0368 & -1.504 & 0.925 \\ 1.2638 & -0.7317 & 1.2382 & 1.8732 & -1.1491 & -1.2882 & 1.0076 \end{bmatrix}$$

the MMSE filters $\theta^\star$ for the noisy image `img14gn.tif`

$$\theta^\star = \begin{bmatrix} 0.01 & -0.0289 & 0.0224 & 0.0446 & -0.0404 & -0.0059 & 0.0049 \\ 0.0132 & -0.0157 & 0.008 & 0.0326 & -0.0111 & -0.0079 & 0.0098 \\ -0.0307 & -0.0295 & 0.033 & 0.1277 & 0.0225 & -0.032 & -0.0389 \\ 0.0341 & 0.0474 & 0.1363 & 0.2891 & 0.0849 & 0.0021 & 0.0244 \\ -0.0162 & -0.0004 & 0.0903 & 0.1354 & 0.0414 & -0.0036 & 0.0128 \\ -0.0148 & 0.0197 & -0.0215 & 0.0679 & -0.0216 & -0.0029 & 0.0278 \\ 0.019 & 0.0104 & -0.0148 & 0.0547 & -0.04 & -0.038 & -0.0049 \end{bmatrix}$$

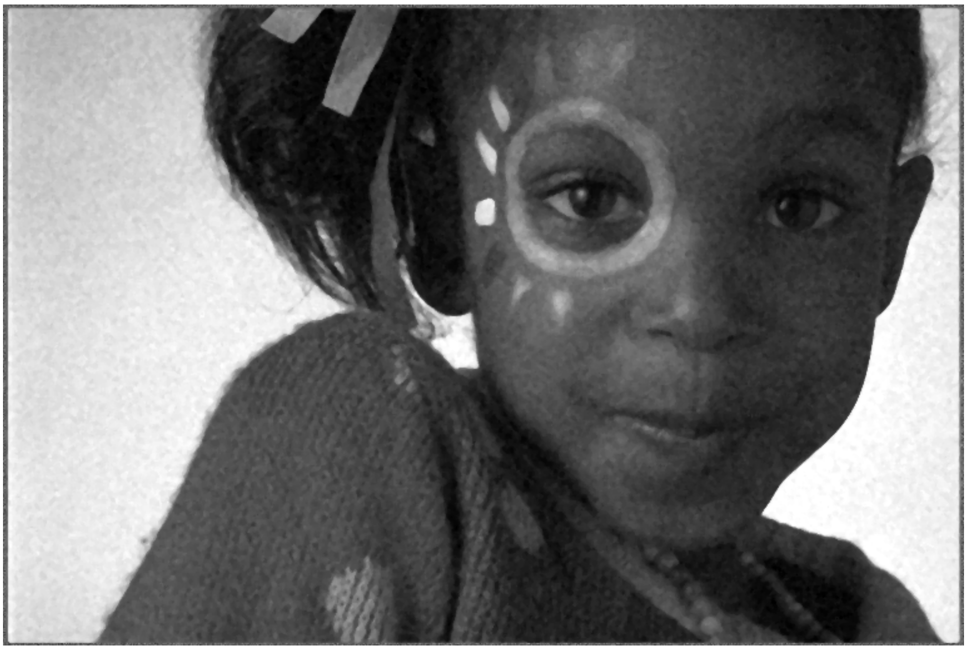
the MMSE filters $\theta^\star$ for the noisy image `img14sp.tif`

$$\theta^\star = \begin{bmatrix} 0.0157 & 0.0107 & -0.0217 & 0.0201 & -0.0561 & -0.0011 & -0.0152 \\ -0.0133 & -0.0348 & 0.0594 & 0.0531 & -0.0266 & 0.0588 & 0.0191 \\ -0.0342 & -0.0027 & 0.0424 & 0.1107 & -0.0124 & -0.0368 & -0.0545 \\ 0.0299 & 0.0158 & 0.0955 & 0.3146 & 0.0917 & -0.0118 & 0.0087 \\ -0.0006 & 0.0084 & 0.1002 & 0.1585 & 0.0433 & 0.0073 & 0.0017 \\ 0.001 & -0.0286 & -0.0067 & 0.063 & -0.0171 & 0.0054 & 0.0593 \\ 0.0336 & -0.0017 & -0.0011 & 0.0466 & -0.0435 & -0.0392 & -0.0095 \end{bmatrix}$$

2. Weighted Median Filtering

2.1. results of median filtering for noisy images `img14gn.tif` and `img14sp.tif`

solution



Result of Weighted Median Filtering for the Noisy Image img14gn.tif

Result of Weighted Median Filtering for the Noisy Image img14sp.tif

2.2. listing of C codes

solution

Code Snippet for Weighted Median Filtering in **restore.c**

```
#include "../include/restore.h"
#define WEIGHT_INIT\
    {   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   }
#define SIZE_KERNEL 25
#define SUM_HALF 17 // ceil( sum(WEIGHT_INIT) / 2.0 )
#define CHUNK(a, i, j)\
    {ROW(a, i-2, j), ROW(a, i-1, j), ROW(a, i, j), \
    ROW(a, i+1, j), ROW(a, i+2, j)}
#define ROW(a, i, j)\
    a[i][j-2], a[i][j-1], a[i][j], a[i][j+1], a[i][j+2]
static void exch(unsigned char* px, unsigned char* pw,
                 int16_t i, int16_t j) {
    unsigned char s =px[i], t = pw[i];
    px[i] = px[j]; pw[i] = pw[j];
    px[j] = s; pw[j] = t;
}
static int16_t partition(unsigned char* px, unsigned char* pw,
                         int16_t lo, int16_t hi) {
    int16_t i = lo + 1;
    int16_t j = hi;
    unsigned char v = px[lo];
    while (1) {
        while (px[i] >= v) {
            if (i == hi) break;
            i++;
        }
        while (v >= px[j]) {
            if (j == lo) break;
            j--;
        }
        if (i >= j) break;
```

```
        exch(px, pw, i, j);
    }
    exch(px, pw, lo, j);
    return j;
}
static void sort(unsigned char* px, unsigned char* pw,
                 int16_t lo, int16_t hi) {
    if (hi <= lo) return;
    int16_t j = partition(px, pw, lo, hi);
    sort(px, pw, lo, j-1);
    sort(px, pw, j+1, hi);
}
static unsigned char point_median_weighted(
    unsigned char **array, int16_t i, int16_t j) {
    unsigned char x[SIZE_KERNEL] = CHUNK(array, i, j);
    unsigned char weight[SIZE_KERNEL] = WEIGHT_INIT;
    unsigned char sum_half = SUM_HALF, sum_part = 0;
    sort(x, weight, 0, SIZE_KERNEL-1);
    for (int16_t i=0; i<SIZE_KERNEL; i++) {
        sum_part += weight[i];
        if (sum_part >= sum_half) { return x[i]; }
    }
}
void filter_median_weighted(
    unsigned char **a, unsigned char **a_t, int16_t W, int16_t H) {
    // assert((j-2 >= 0)&&(j+2 <= W-1)&&(i-2 >= 0)&&(i+2 <= H-1));
    for (int16_t i=2; i<H-2; i++) {
        for (int16_t j=2; j<W-2; j++) {
            a_t[i][j] = point_median_weighted(a, i, j);
        }
    }
    /* fill the borders of output image */
    for (int16_t i=0; i<2; i++) {
        for (int16_t j = 0; j < W; j++) { a_t[i][j]  =a[i][j];  }
    }
    for (int16_t i=H-2; i<H; i++) {
        for (int16_t j = 0; j < W; j++) { a_t[i][j]  =a[i][j];}
    }
    for (int16_t i=2; i<H-2; i++) {
        a_t[i][0]   = a[i][0];     a_t[i][1]   = a[i][1];
        a_t[i][W-2] = a[i][W-2];   a_t[i][W-1] = a[i][W-1];
    }
}
```

Appendix

Python code for MMSE linear filtering: `utils.py`

```python
from numpy import ndarray, size, zeros, reshape
from numpy.linalg import solve

def filter_FIR(x: ndarray, kernel: ndarray) -> ndarray:
    height, width = x.shape
    ky, kx = kernel.shape[0], kernel.shape[-1]
    dy, dx = ky//2, kx//2
    x_out = zeros((height, width))
    x_out[:dy, :]   = x[:dx, :]
    x_out[-dy:, :]  = x[-dy:, :]
    x_out[:, :dx]   = x[:, :dx]
    x_out[:, :-dx]  = x[:, :-dx]
    for i in range(dy, height-dy):
        for j in range(dx, width-dx):
            x_out[i][j] = (x[i-dy:i-dy+ky, j-dx:j-dx+kx]
                           * kernel).sum()
    return x_out

def estimate_kernel(x: ndarray, y: ndarray,
    size_kernel: int=7, rate_sample: int=20) -> ndarray:
    dy, dx = size_kernel//2, size_kernel//2
    height, width = x.shape
    h_sample = (height - size_kernel)//rate_sample + 1
    w_sample = (width  - size_kernel)//rate_sample + 1
    z = zeros((h_sample, w_sample, size_kernel, size_kernel))
    for iz, i in enumerate(range(0, height-(size_kernel-1),
                                 rate_sample)):
        for jz, j in enumerate(range(0, width-(size_kernel-1),
                                     rate_sample)):
            z[iz][jz] = x[i:i+size_kernel, j:j+size_kernel]
    y_s = y[dy: height+dy-(size_kernel-1): rate_sample, dx:
            width+dx-(size_kernel-1): rate_sample].flatten()
    z = reshape(z, (-1, size_kernel*size_kernel))
    R_zz = z.T @ z / (h_sample*w_sample)
    r_zy = y_s @ z / (h_sample*w_sample)
    return solve(R_zz, r_zy).reshape(size_kernel, size_kernel)
```

Python code for solution

solution to section 1: `soln_1.py`

```
from PIL import Image
import numpy as np
from numpy import array
from utils import filter_FIR, estimate_kernel
from os.path import join
from sympy import Matrix, latex

if __name__ == "__main__":
    root = 'bin'
    y = array(Image.open(join(root, 'img14g.tif')))
    list_img = ['img14bl.tif', 'img14gn.tif', 'img14sp.tif']
    for img, char in list(zip(list_img, ['a', 'b', 'c'])):
        x = array(Image.open(join(root, img)))
        kernel = estimate_kernel(x, y, size_kernel=7, rate_sample=20)
        # print(kernel.round(decimals=4))
        print(latex((Matrix(kernel.round(decimals=4)))),
              '\n') # keep 4 digits
        y_hat = filter_FIR(x, kernel)
        img_filtered
            = Image.fromarray(y_hat.clip(0, 255).astype(np.uint8))
        img_filtered.save(join('result', 'fig_1_2' + char +'.png'))
```

C codes for weighted median filtering : `restore.h`, `restore.c`

restore.h

```c
#ifndef _RESTORE_H_
#define _RESTORE_H_

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <math.h>
#include "allocate.h"
#include "typeutil.h"
#include "tiff.h"

void filter_median_weighted(unsigned char **a,
                            unsigned char **a_t, int16_t W, int16_t
H);
void assign_img2arr(struct TIFF_img *img, unsigned char **array);
void assign_arr2img(unsigned char **array, struct TIFF_img *img);
#endif /* _RESTORE_H_ */
```

restore.c

```c
#include "../include/restore.h"
#define WEIGHT_INIT\
    {   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   }
#define SIZE_KERNEL 25
#define SUM_HALF 17 // ceil( sum(WEIGHT_INIT) / 2.0 )
#define CHUNK(a, i, j)\
    {ROW(a, i-2, j), ROW(a, i-1, j), ROW(a, i, j), \
    ROW(a, i+1, j), ROW(a, i+2, j)}
#define ROW(a, i, j)\
    a[i][j-2], a[i][j-1], a[i][j], a[i][j+1], a[i][j+2]
static void exch(unsigned char* px, unsigned char* pw,
```

```
                 int16_t i, int16_t j) {
    unsigned char s =px[i], t = pw[i];
    px[i] = px[j]; pw[i] = pw[j];
    px[j] = s; pw[j] = t;
}
static int16_t partition(unsigned char* px, unsigned char* pw,
                         int16_t lo, int16_t hi) {
    int16_t i = lo + 1;
    int16_t j = hi;
    unsigned char v = px[lo];
    while (1) {
        while (px[i] >= v) {
            if (i == hi) break;
            i++;
        }
        while (v >= px[j]) {
            if (j == lo) break;
            j--;
        }
        if (i >= j) break;
        exch(px, pw, i, j);
    }
    exch(px, pw, lo, j);
    return j;
}
static void sort(unsigned char* px, unsigned char* pw,
                 int16_t lo, int16_t hi) {
    if (hi <= lo) return;
    int16_t j = partition(px, pw, lo, hi);
    sort(px, pw, lo, j-1);
    sort(px, pw, j+1, hi);
}
static unsigned char point_median_weighted(
    unsigned char **array, int16_t i, int16_t j) {
    unsigned char x[SIZE_KERNEL] = CHUNK(array, i, j);
    unsigned char weight[SIZE_KERNEL] = WEIGHT_INIT;
    unsigned char sum_half = SUM_HALF, sum_part = 0;
    sort(x, weight, 0, SIZE_KERNEL-1);
    for (int16_t i=0; i<SIZE_KERNEL; i++) {
        sum_part += weight[i];
        if (sum_part >= sum_half) { return x[i]; }
    }
}
```

```c
void filter_median_weighted(
    unsigned char **a, unsigned char **a_t, int16_t W, int16_t H) {
    // assert((j-2 >= 0)&&(j+2 <= W-1)&&(i-2 >= 0)&&(i+2 <= H-1));
    for (int16_t i=2; i<H-2; i++) {
        for (int16_t j=2; j<W-2; j++) {
            a_t[i][j] = point_median_weighted(a, i, j);
        }
    }
    /* fill the borders of output image */
    for (int16_t i=0; i<2; i++) {
        for (int16_t j = 0; j < W; j++) { a_t[i][j]  =a[i][j];  }
    }
    for (int16_t i=H-2; i<H; i++) {
        for (int16_t j = 0; j < W; j++) { a_t[i][j]  =a[i][j];}
    }
    for (int16_t i=2; i<H-2; i++) {
        a_t[i][0]   = a[i][0];     a_t[i][1]   = a[i][1];
        a_t[i][W-2] = a[i][W-2];   a_t[i][W-1] = a[i][W-1];
    }
}
void assign_img2arr(struct TIFF_img *img, unsigned char **array) {
    int16_t W, H;
    W = img->width; H = img->height;
    for (int16_t i = 0; i < H; i++ ) {
        for (int16_t j = 0; j < W; j++ ) {
            array[i][j] = img->mono[i][j];
        }
    }
}
void assign_arr2img(unsigned char **array, struct TIFF_img *img) {
    int16_t W, H;
    W = img->width; H = img->height;
    for (int16_t i = 0; i < H; i++ ) {
        for (int16_t j = 0; j < W; j++ ) {
            img->mono[i][j] = array[i][j];
        }
    }
}
```

C codes for solutions

solution to section 2: `soln_2.c`

```
/* ECE 637 Image Processing I, Spring 2022
 * @author: Zhankun Luo, luo333@purdue.edu
 * lab 7: Image Restoration
 * solution to section 2
 * run it with: ./soln_2 img14gn.tif
 * or:          ./soln_2 img14sp.tif
 **/

#include "../include/tiff.h"
#include "../include/allocate.h"
#include "../include/typeutil.h"
#include "../include/restore.h"

void error(char *name) {
    printf("usage:  %s  image.tif \n\n",name);
    exit(1);
}

int main(int argc, char **argv) {
    if ( argc != 2 ) error( argv[0] );
    FILE *fp;
    struct TIFF_img img, img_out;
    int16_t W, H;
    /* 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, &img ) ) {
        fprintf( stderr, "error reading file %s\n", argv[1] );
        exit( 1 );
    }
    /* close image file */
    fclose( fp );
    /* check the type of image data: grayscale */
```

```c
    if ( img.TIFF_type != 'g' ) {
        fprintf( stderr, "error:  image must be grayscale image\n" );
        exit( 1 );
    }
    W = img.width; H = img.height;
    get_TIFF( &img_out, H, W, 'g' );
    /* image restoration */
    unsigned char **arr = \
        (unsigned char **)get_img(W, H, sizeof(unsigned char));
    unsigned int **arr_out = \
        (unsigned int **)get_img(W, H, sizeof(unsigned int));
    assign_img2arr(&img, arr);
    filter_median_weighted(arr, arr_out, W, H);
    assign_arr2img(arr_out, &img_out);
    /* open output image */
    char path_out[50], index;
    char *dot = strrchr(argv[1], '.');
    if (dot && !strcmp(dot-2, "gn.tif")) {
        index = 'a';
    } else if (!strcmp(dot-2, "sp.tif")) {
        index = 'b';
    } else { exit(1); }
    sprintf(path_out, "../result/fig_2_1%c.tif", index);
    if ( ( fp = fopen ( path_out, "wb" ) ) == NULL ) {
        fprintf( stderr, "cannot open TIFF file\n");
        exit( 1 );
    }
    /* write output image */
    if ( write_TIFF( fp, &img_out ) ) {
        fprintf( stderr, "error writing TIFF file\n");
        exit( 1 );
    }
    /* close output image file */
    fclose( fp );
    /* de-allocate memory */
    free_TIFF( &(img) );
    free_TIFF( &(img_out) );
    return(0);
}
```