ECE 637 Lab 7 - Image Restoration

Yifan Fei

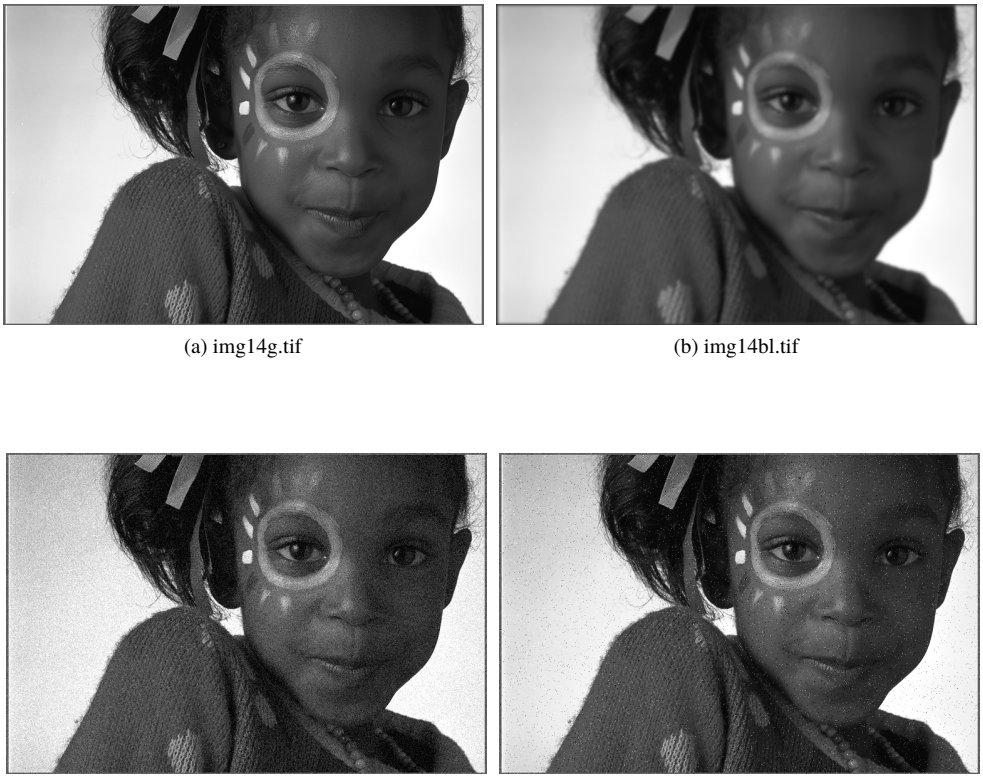
Electrical Engineering
Purdue University

Instructor: Prof. Charles A. Bouman

Spring 2018

Section 1: Minimum Mean Square Error (MMSE) Linear Fil- ters

1: four original images



(a) img14g.tif (b) img14bl.tif

(c) img14gn.tif (d) img14sp.tif

Figure 1: four original images

2: output of the optimal filtering for the blurred image and the two noisy images

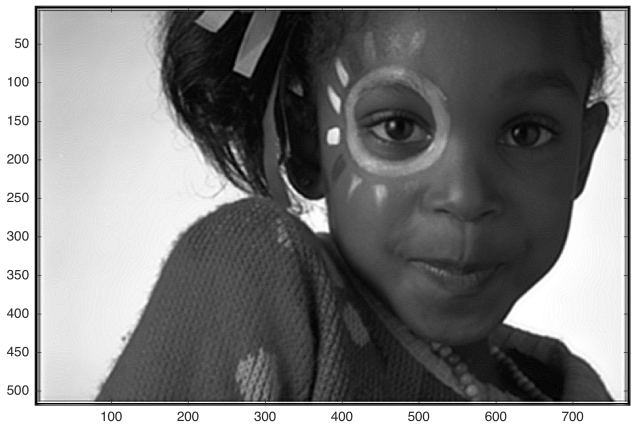


Figure 2: Filtered Blurred image

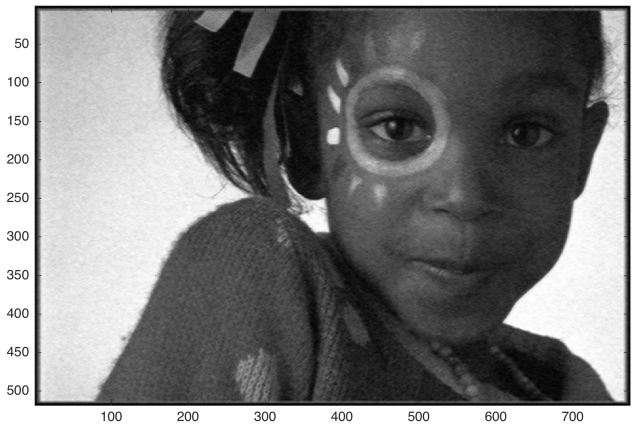


Figure 3: Filtered noisy image

Figure 4: Filtered noisy image with spot

3: MMSE filters

For the blurred image:

$$\theta^\star = \begin{bmatrix} 1.7122 & 0.7401 & 0.9280 & 0.8153 & -0.9378 & -1.8137 & 1.8075 \\ -1.4864 & -1.8092 & -0.9006 & -0.5821 & -2.9833 & 0.5828 & 1.3537 \\ -0.9434 & -2.7623 & 0.3063 & 2.9781 & 0.7147 & -2.7348 & -0.8187 \\ 2.0282 & -0.5532 & 3.6350 & 3.4485 & 3.2368 & -3.1583 & 0.6775 \\ 1.6263 & -3.0881 & -0.4136 & 5.0773 & -0.1598 & -1.4265 & 0.7996 \\ -0.3035 & 0.3035 & 0.3035 & 0.3035 & 0.3035 & 1.5040 & 0.9250 \\ 1.2638 & 1.2638 & 1.2382 & 1.8732 & 1.1491 & 1.2882 & 1.0076 \end{bmatrix}$$

For the noisy image:

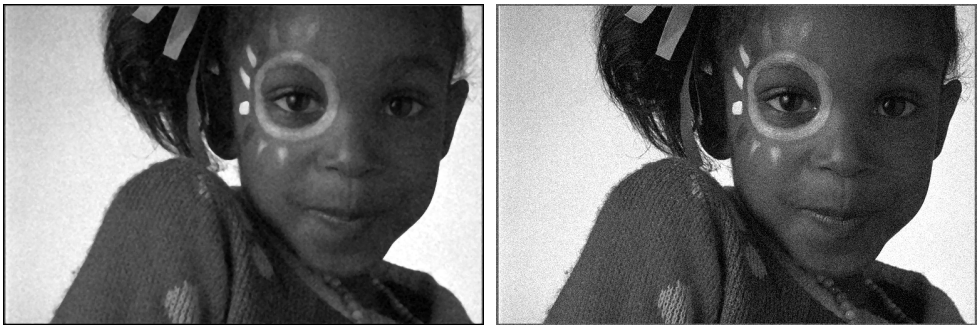
$$\theta^\star = \begin{bmatrix} 0.0165 & -0.0055 & -0.0105 & -0.0091 & -0.0050 & -0.0044 & -0.0053 \\ 0.0259 & 0.0053 & -0.0125 & -0.0153 & -0.0222 & 0.0079 & -0.0043 \\ 0.0044 & 0.0355 & 0.0674 & 0.0476 & 0.0423 & 0.0307 & 0.0154 \\ 0.0050 & 0.0205 & 0.0731 & 0.2306 & 0.1117 & 0.0268 & 0.0127 \\ -0.0080 & 0.0464 & 0.0470 & 0.0891 & 0.0650 & 0.0088 & 0.0140 \\ 0.0302 & 0.0091 & 0.0290 & -0.0175 & -0.0118 & -0.0063 & 0.0183 \\ -0.0259 & 0.0066 & -0.0030 & 0.0011 & 0.0069 & 0.0192 & 0.0054 \end{bmatrix}$$

For the noisy image with spot:

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

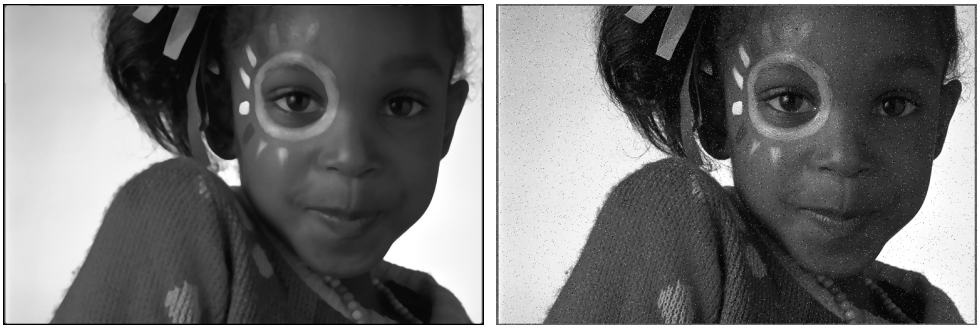
Section2:

1:Results of Median Filtering



(a) Median Filtered noisy image (b) Original noisy image

Figure 5: Comparison between original noisy image and Median Filtered noisy image



(a) Median Filtered noisy image with spot (b) Orignal noisy image with spot

Figure 6: Comparison between original noisy image with spot and Median Filtered noisy image with spot

By comparison, we can find that the corrupted images on the right can be restored to some degree by median filtering.

2: Code Listing

medfilter.h

```
#ifndef __MEDFILTER_H_
#define __MEDFILTER_H_

void error(char *name);
unsigned int filter_med(struct TIFF_img in, int i, int j);
unsigned int** filter(struct TIFF_img in);

#endif
```

medfilter.c

```c
/* gcc -o medfilter medfilter.c allocate.c randlib.c solve.c tiff.c */
/* ./medfilter img14gn.tif gn.tif */
/* ./medfilter img14sp.tif sp.tif */

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

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img;
  unsigned int **filtered_img;
  int32_t i,j,pixel;

  if ( argc != 3 ) 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 output */
  if ( input_img.TIFF_type != 'g' ) {
    fprintf ( stderr, "error:  image must be grayscale\n" );
    exit ( 1 );
  }


  // call function median filter
  filtered_img = filter(input_img);

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

  /* Free segment map */
```

```
  free_img((void *)filtered_img);

  /* open image file */
  if ( ( fp = fopen ( argv[2], "wb" ) ) == NULL ) {
      fprintf ( stderr, "cannot open file %s\n", argv[2]);
      exit ( 1 );
  }

  /* write color image */
  if ( write_TIFF ( fp, &input_img ) ) {
      fprintf ( stderr, "error writing TIFF file %s\n", argv[2]);
      exit ( 1 );
  }

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

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

  return(0);
}


unsigned int filter_med(struct TIFF_img input, int i, int j) {
  int m, n, k = 0;
  unsigned int sum1, sum2;
  unsigned int temp1, temp2;
  unsigned int output[25];
  unsigned int weights[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};

  for (m = i - 2; m < i + 3; m++) {
    for (n = j - 2; n < j + 3; n++) {
      output[k] = input.mono[m][n];
      k++;
    }
  }

  // sorting the pixels in the window in descending order
  for (m = 0; m < 25; m++) {
    for (n = m + 1; n < 25; n++) {
      if (output[m] < output[n]) {
        temp1 = output[m];
        output[m] = output[n];
        output[n] = temp1;

        // corresponding pixel weights are placed
        // in the same order as the sorted pixels
        temp2 = weights[m];
        weights[m] = weights[n];
```

```
        weights[n] = temp2;
      }
    }
  }

  sum1 = weights[0];
  sum2 = 34 - sum1;
  for (m = 0; m < 25; m++) {
    if (sum1 > sum2) {
      return output[m];
    }

    sum1 += weights[m+1];
    sum2 -= weights[m+1];
  }

  return output[m];
}


unsigned int** filter(struct TIFF_img input) {
  int i, j;
  unsigned int** filtered = (unsigned int**) get_img(input.width,input.height,sizeof(unsigned int));

  for (i = 2; i < input.height - 2; i++) {
    for (j = 2; j < input.width - 2; j++) {
      filtered[i][j] = filter_med(input, i, j);
    }
  }

  return filtered;
}

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);
}
```

Attachments: Matlab code

Section1.m

```
% Section 1 %

clear all;
```

```
clc;

%%
% X = imread('images/img14bl.tif');
% X = imread('images/img14gn.tif');
X = imread('images/img14sp.tif');
X = double(X);

Y = imread('images/img14g.tif');
Y = double(Y);

[r, c] = size(Y);
m = floor(r/20);
n = floor(c/20);
N = m * n;
Z = zeros(N, 49);
Y_col = zeros(N,1);

i = 1;
for j = 1:m
   for k = 1:n
       Z(i,:) = reshape(X(20*j - 3:20*j + 3, 20*k - 3:20*k + 3)', [1, 49]);
       Y_col(i) = Y(20 * j, 20 * k);
       i = i + 1;
    end
end

R_zz = (Z' * Z) ./ N;
r_zy = (Z' * Y_col) ./ N;
theta_star = R_zz \ r_zy;
theta = reshape(theta_star, [7, 7])
imgout = conv2(X, theta);
figure(1);
image(uint8(imgout));
axis('image');
colormap([0:255;0:255;0:255]'/255);
% print('-dpng', '-r300', 'report/Filtered_Blurred.png');
% print('-dpng', '-r300', 'report/Filtered_Noisy.png');
print('-dpng', '-r300', 'report/Filtered_Noisy_Spots.png');
```