Digital Image Processing Laboratory 7

Image Restoration
Praneet Singh

100/100

April 3, 2020

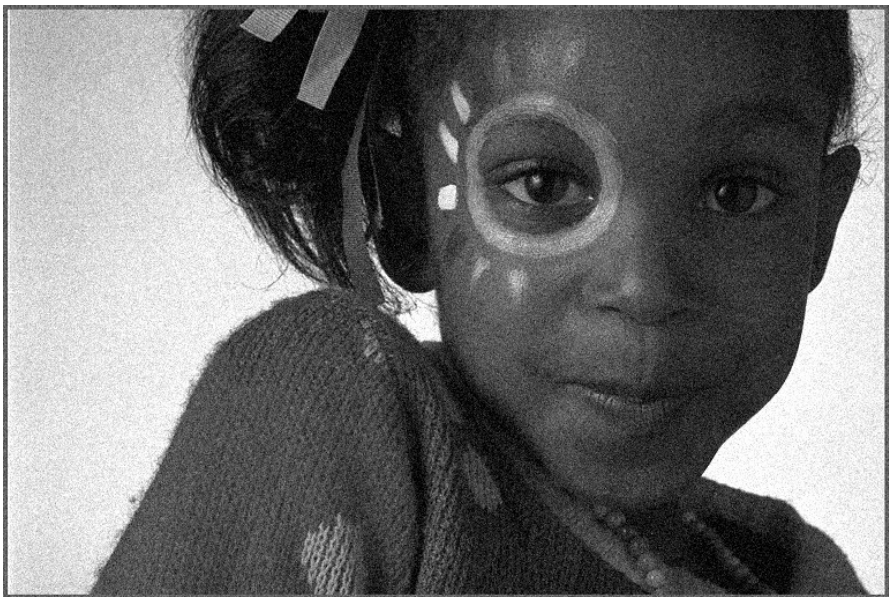
1 Minimum Mean Square Error (MMSE) Linear Filters

1.1 Original Images

1.1.1 Image *img14g*

1.1.2 Image *img14bl*

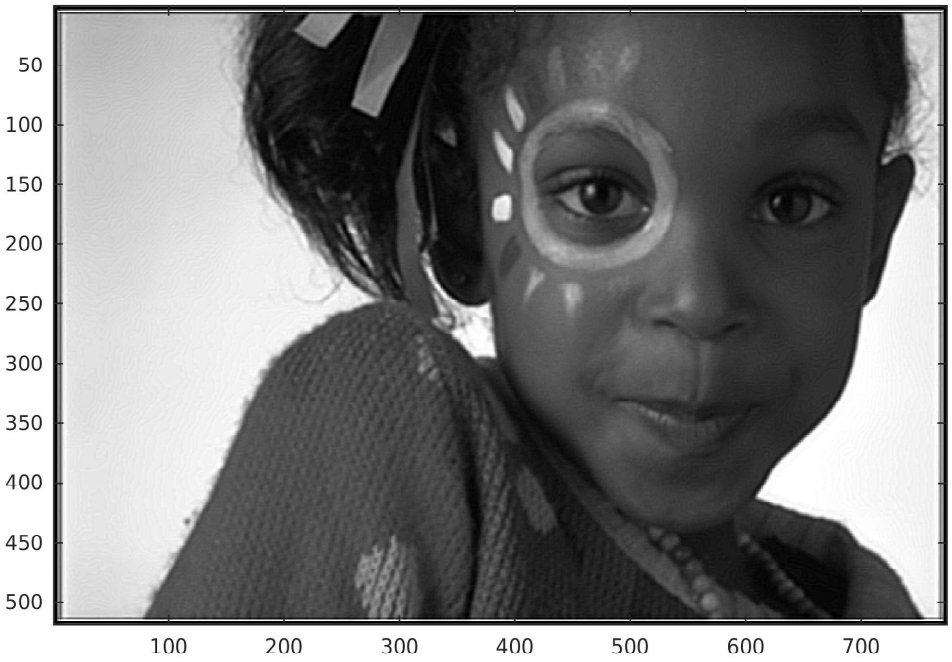
1.1.3 Image *img14gn*



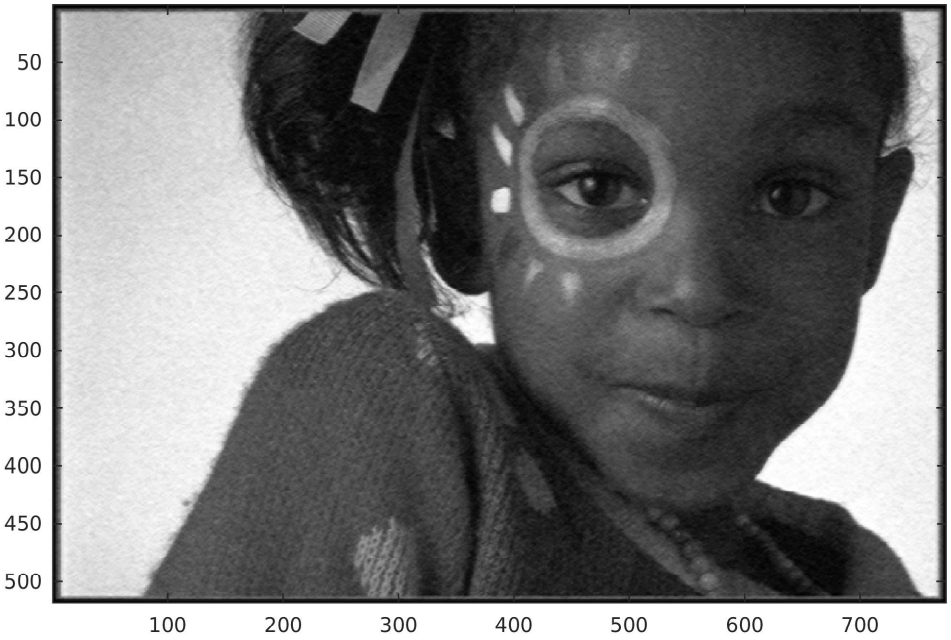
1.1.4 Image *img14sp*

1.2 Filtered Images

1.2.1 Filtered Image *img14bl*



1.2.2 Filtered Image *img14gn*



1.2.3 Filtered Image *img14sp*

1.3 Theta*

1.3.1 For *img*14*bl*

$$\begin{pmatrix} 0.37203 & 0.20523 & -0.96822 & 1.0572 & 0.19607 & -1.002 & 0.92537 \\ -0.043143 & 0.40695 & -1.2219 & -0.02805 & -0.61461 & -1.3229 & 0.4024 \\ -0.35409 & -0.32423 & -0.48103 & 0.33209 & 0.758 & -0.087075 & -0.79227 \\ 1.1089 & -2.4308 & 1.9317 & 3.7782 & 1.5691 & -0.070071 & 0.061508 \\ 0.3791 & -0.45903 & -1.1045 & 1.2263 & 0.83581 & -1.471 & 0.39048 \\ -1.099 & -0.18022 & -0.29437 & 1.0624 & -1.8928 & -1.9628 & 0.81266 \\ 1.156 & 0.47759 & -1.7439 & 0.64825 & 0.29486 & 0.26038 & 0.30417 \end{pmatrix}$$

1.4 For *img*14*gn*

1.4.1 For *img*14*gn*

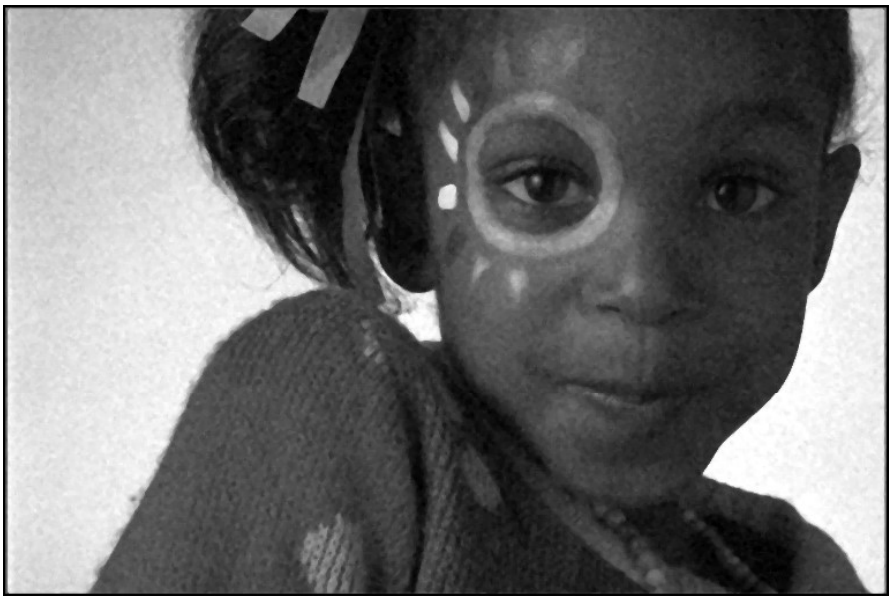
$$\begin{pmatrix} 0.016529 & 0.02589 & 0.0044091 & 0.0050055 & -0.0080127 & 0.030237 & -0.02594 \\ -0.0055497 & 0.0052631 & 0.035469 & 0.020481 & 0.046448 & 0.0090524 & 0.0065885 \\ -0.01053 & -0.012467 & 0.067433 & 0.073091 & 0.047025 & 0.028984 & -0.0030232 \\ -0.0090564 & -0.015337 & 0.047646 & 0.23062 & 0.089143 & -0.01753 & 0.0011056 \\ -0.0049524 & -0.022205 & 0.042308 & 0.11174 & 0.064951 & -0.011783 & 0.0068565 \\ -0.0043627 & 0.0078819 & 0.030727 & 0.026825 & 0.0087753 & -0.0063344 & 0.019169 \\ -0.0052977 & -0.0043493 & 0.015414 & 0.012697 & 0.014011 & 0.018338 & 0.0053806 \end{pmatrix}$$

1.4.2 For *img*14*sp*

$$\begin{pmatrix} 0.0080265 & 0.0048375 & -0.0016264 & -0.0049539 & 0.025652 & -0.020898 & -0.018546 \\ 0.0016665 & -0.0015614 & 0.055793 & 0.026715 & 0.043518 & 0.02138 & 0.019605 \\ -0.0010291 & 0.0042167 & 0.041335 & 0.096846 & 0.021245 & -0.019637 & 0.019868 \\ -0.0013909 & -0.020329 & 0.035006 & 0.26519 & 0.14924 & -0.028651 & 0.008286 \\ 0.025169 & 0.0023109 & 0.061241 & 0.096504 & 0.015397 & -0.041243 & 0.023308 \\ -0.0098941 & -0.00060704 & 0.031284 & 0.049733 & 0.014341 & 0.0038189 & 0.01314 \\ -0.040718 & 0.016156 & -0.0068141 & 0.0099791 & 0.0079258 & 0.012909 & -0.011003 \end{pmatrix}$$

1.5 Weighted Median Filtering

1.5.1 Filtered Image *img14gn*



1.5.2 Filtered Image *img14sp*

1.5.3 C-Code

```c
#include<stdio.h>

void error(char *name);
unsigned int find_median(struct TIFF_img in, int i, int j);
unsigned int** median_filter(struct TIFF_img in);
```

```c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "tiff.h"
#include "allocate.h"
#include "medfilter.h"
```

```
7
8  int main (int argc, char **argv){
9    FILE *fp;
10   struct TIFF_img input_img;
11   unsigned int** median_fil_img;
12   int i, j;
13   if (argc != 3) {
14     error(argv[0]);
15   }
16   /* open image file */
17   if ((fp = fopen(argv[1], "rb")) == NULL) {
18     fprintf(stderr, "cannot open file %s\n", argv[1]);
19     exit(1);
20   }
21   /* read image */
22   if (read_TIFF(fp, &input_img)) {
23     fprintf(stderr, "error reading file %s\n", argv[1]);
24     exit(1);
25   }
26   /* close image file */
27   fclose(fp);
28   /* check the type of image data */
29   if (input_img.TIFF_type != 'g') {
30     fprintf(stderr, "error: image must be grayscale\n");
31     exit(1);
32   }
33   median_fil_img = median_filter(input_img);
34   for (i = 0; i < input_img.height; i++) {
35     for (j = 0; j < input_img.width; j++) {
36     input_img.mono[i][j] = median_fil_img[i][j];
37     }
38   }
39   /* open image file */
40   if ((fp = fopen(argv[2], "wb")) == NULL) {
41     fprintf(stderr, "cannot open file %s\n", argv[3]);
42     exit(1);
43   }
44   /* write image */
45   if (write_TIFF(fp, &input_img)) {
46     fprintf(stderr, "error writing TIFF file %s\n", argv[3] );
47     exit(1);
48   }
49   /* close image file */
50   fclose(fp);
51   /* de-allocate space which was used for the tiff */
52   free_TIFF(&input_img);
53   return(0);
54 }
55
56 void error(char *name){
57     printf("usage:  %s image.tiff output.tiff\n\n", name);
58   printf("This program reads in a 24-bit color TIFF image.\n");
59   printf("Then it filters the image with a weighted median fitler\n");
60   printf("Then writes out the filtered image \n");
61     exit(1);
62 }
63
64 unsigned int** median_filter(struct TIFF_img in) {
65   int i, j;
66
```

```
  unsigned int** med_fil = (unsigned int**) get_img(in.width,
                            in.height,
                            sizeof(unsigned int));
  for (i = 2; i < in.height - 2; i++) {
    for (j = 2; j < in.width - 2; j++) {
      med_fil[i][j] = find_median(in, i, j);
    }
  }

  return med_fil;
}

unsigned int find_median(struct TIFF_img in, int i, int j){
  int m, n;
  int k=0;
  int sum1, sum2;
  int sum_w=0;
  int window[25];
  int weighted_fil[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 (int m = 0; m < 25: m++){
    sum_w+=weighted_fil[m]
  }
  /* Get the window to which you want to apply filter */
  for (m = i - 2; m < i + 3; m++) {
    for (n = j - 2; n < j + 3; n++) {
      window[k] = in.mono[m][n];
      k++;
    }
  }
  /* Sort window values, with them sort the median filter weighted values as well */
  for (m = 0; m < 25; m++) {
    for (n = m + 1; n < 25; n++) {
      if (window[m] < window[n]) {

        window[m] = window[m] + window[n];
        window[n] = window[m] - window[n];
        window[m] = window[m] - window[n];

        weighted_fil[m] = weighted_fil[m] + weighted_fil[n];
        weighted_fil[n] = weighted_fil[m] - weighted_fil[n];
        weighted_fil[n] = weighted_fil[m] - weighted_fil[n];
    }
  }
  sum1 = weighted_fil[0];
  sum2 = sum_w - sum1;
  for (m = 0; m < 25; m++) {
    if (sum1 > sum2) {
      return window[m];
    }
    sum1 += weighted_fil[m+1];
    sum2 -= weighted_fil[m+1];
  }
  return window[m];
}
```