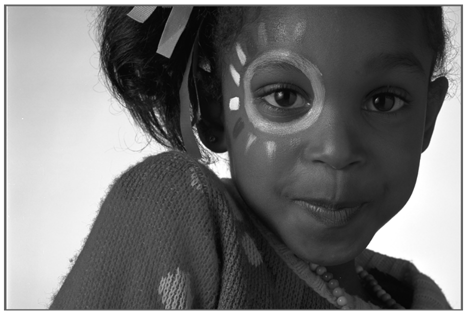
ECE 637 Digital Image Processing Laboratory: Image Restoration

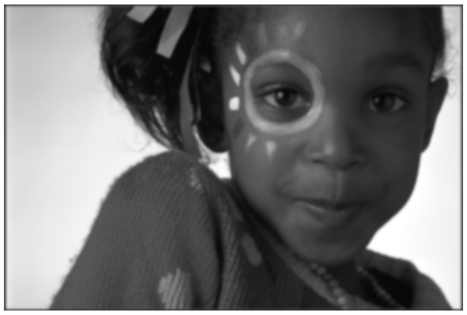
Yang Wang

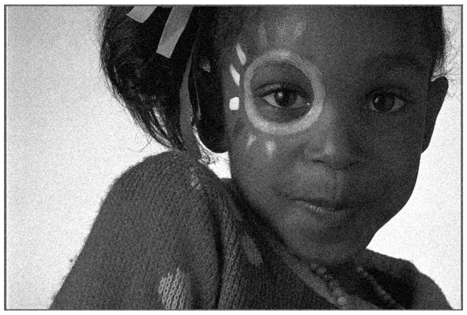
March 27, 2016

1 Minimum Mean Square Error (MMSE) Linear Filters

1.1 Plot Original Four Images


(a) img14g.tif


(b) img14bl.tif


(c) img14gn.tif

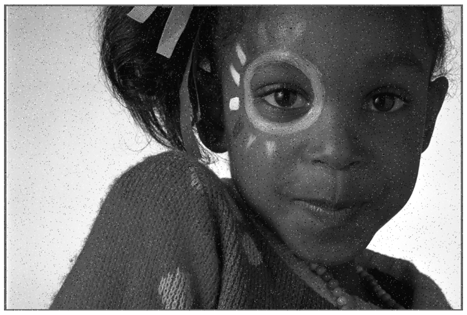
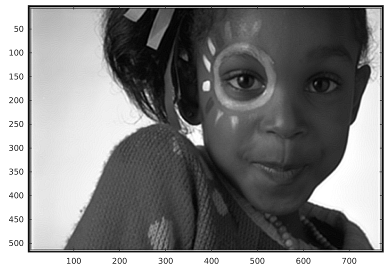
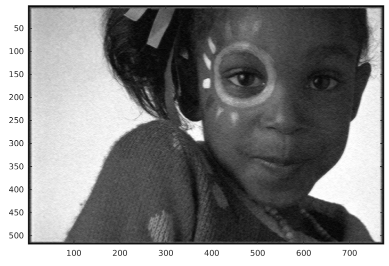

(d) img14sp.tif

Figure 1: Orignal Fabulous Four Images

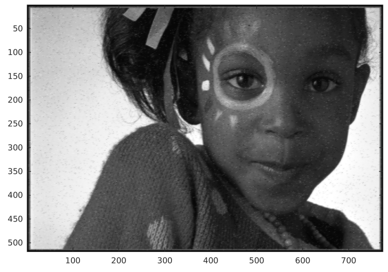
1.2 Plot the Blurred Image, and Two Noisy After Optimal Filtering



(a) Filtered Blurred Image



(b) Filtered Noisy Image



(c) Filtered Noisy Image with Spots

Figure 2: Distorted Images After Optimal Filtering

1.3 Results of MMSE Filters, $\theta^\star$

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 spots,

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

2 Weighted Median Filtering

2.1 Results of Median Filtering

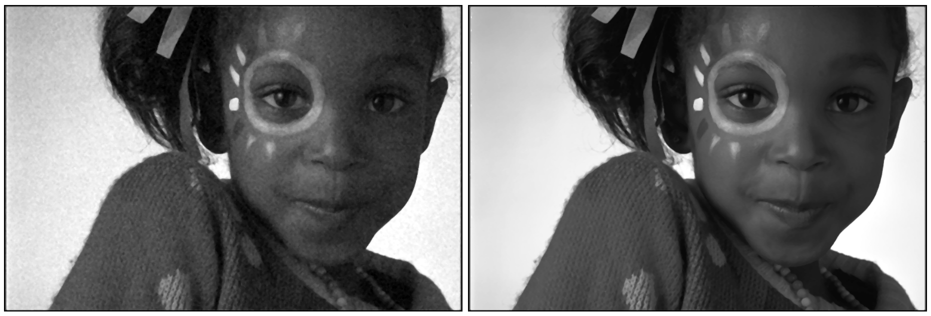

(a) Median Filtered Blurred Image (b) Median Filtered Noisy Image

Figure 3: Weighted Median Filtered Image

2.2 Code Listing

2.2.1 medfilter.h

```
#ifndef __MEDFILT_H_
#define __MEDFILT_H_
```

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

#endif
```

2.2.2 medfilter.c

```c
#include <math.h>
#include <string.h>

#include "tiff.h"
#include "allocate.h"
#include "medfilter.h"

int main (int argc, char **argv) {
    FILE *fp;
    struct TIFF_img tiff;
    unsigned int** filtered;
    int i, j;

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

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

    /* check the type of image data */
    if (tiff.TIFF_type != 'g') {
        fprintf(stderr, "error: image must be grayscale\n");
        exit(1);
    }
    filtered = filter(tiff);
    for (i = 0; i < tiff.height; i++) {
        for (j = 0; j < tiff.width; j++) {
        tiff.mono[i][j] = filtered[i][j];
        }
    }

    /* Free segment map */
    free_img((void *)filtered);

    /* open image file */
    if ((fp = fopen(argv[2], "wb")) == NULL) {
        fprintf(stderr, "cannot open file %s\n", argv[3]);
        exit(1);
    }

    /* write image */
    if (write_TIFF(fp, &tiff)) {
        fprintf(stderr, "error writing TIFF file %s\n", argv[3] );
        exit(1);
    }

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

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

    return(0);
}

void error(char *name) {
    printf("usage:  %s image.tiff output.tiff\n\n", name);
    printf("Filter an image with a weighted median fitler\n");
    exit(1);
}

unsigned int filterp(struct TIFF_img in, int i, int j) {
    int m, n, k;
    unsigned int sum1, sum2;
    unsigned int tmp;
    unsigned int data[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};

    k = 0;
    for (m = i - 2; m < i + 3; m++) {
        for (n = j - 2; n < j + 3; n++) {
            data[k] = in.mono[m][n];
            k++;
        }
    }

    for (m = 0; m < 25; m++) {
        for (n = m + 1; n < 25; n++) {
            if (data[m] < data[n]) {
                tmp = data[m];
                data[m] = data[n];
                data[n] = tmp;

                tmp = weights[m];
                weights[m] = weights[n];
                weights[n] = tmp;
            }
        }
    }

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

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

    return data[m];
}

unsigned int** filter(struct TIFF_img in) {
    int i, j;

    unsigned int** filtered = (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++) {
            filtered[i][j] = filterp(in, i, j);
        }
    }

    return filtered;
}
```