Digital Signal Processing I
Purdue University
Spring 2021

Lab 2 Report

Jerry Wang

1 Minimum Mean Square Error (MMSE

Linear Filteres

1. Hand in the original four images

Figure 1: Original Image

Figure 2: Blurred Version of Original Image

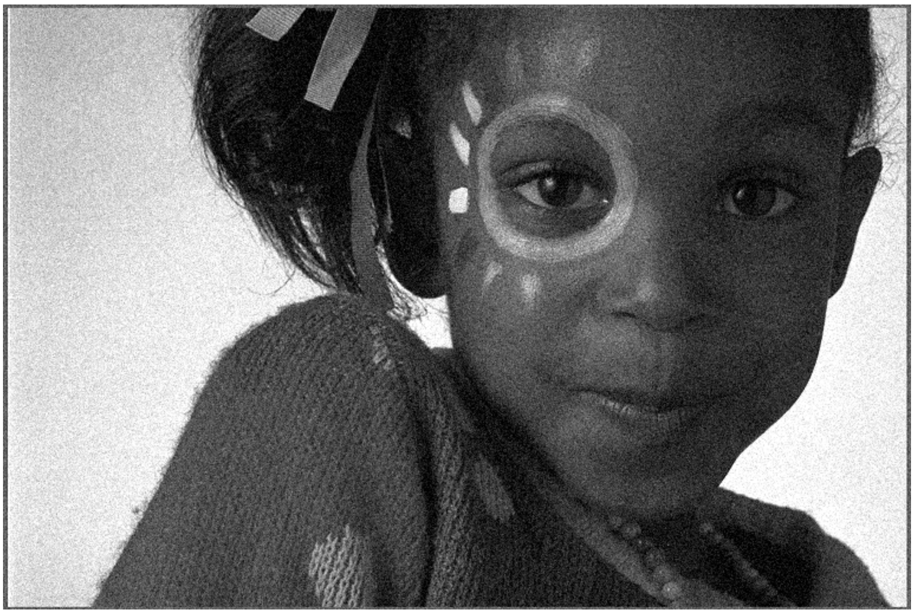

Figure 3: Noisy Version of Original Image 1

Figure 4: Noisy Version of Original Image 2

2. Output of the optimal filtering for the blurred image.

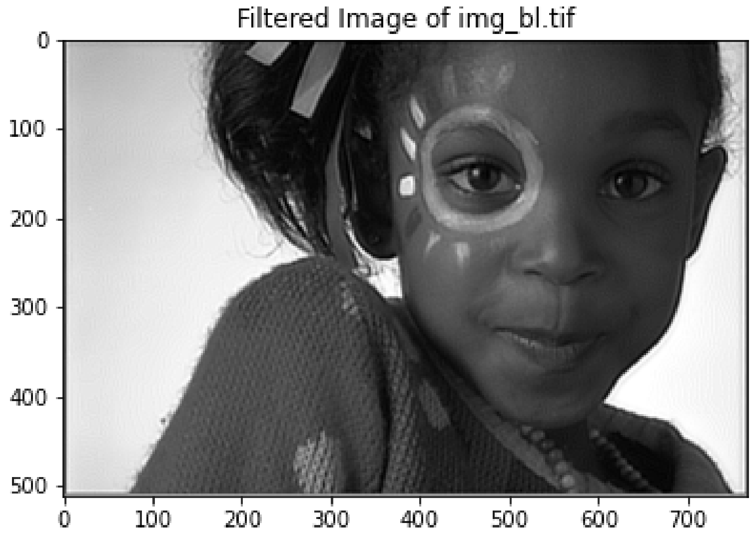

Figure 5: Filtered Image of *img14bl.tif*

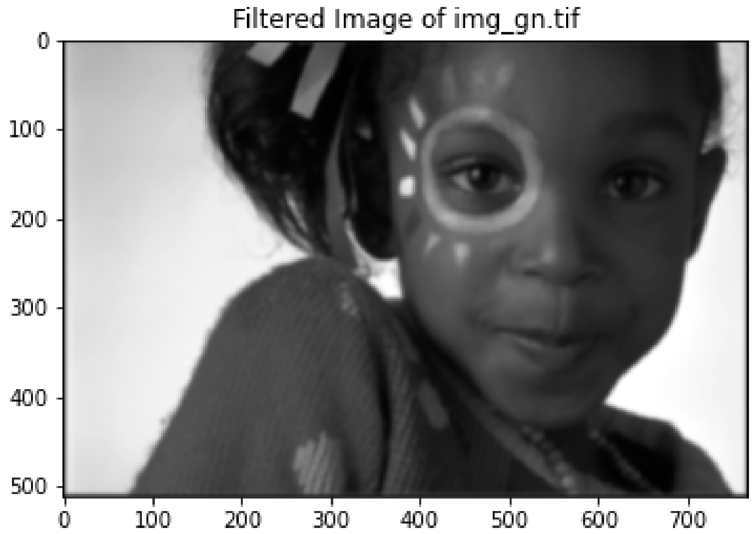

Figure 6: Filtered Image of *img14gn.tif*

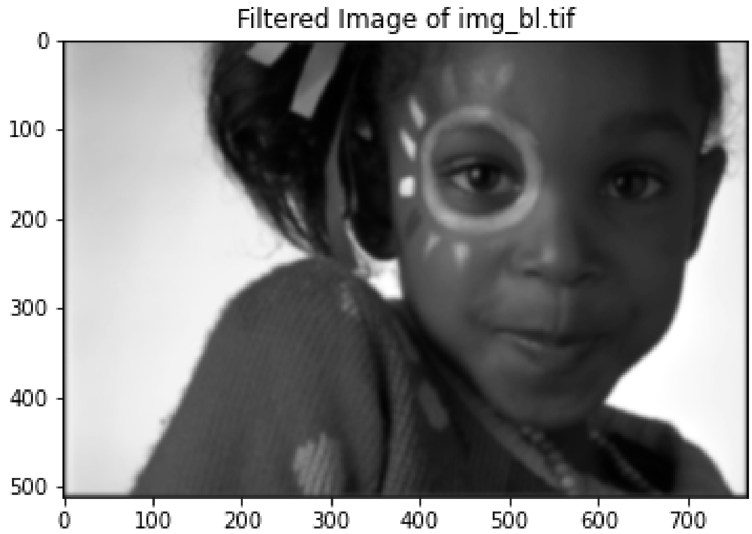

Figure 7: Filtered Image of *img14sp.tif*

3. MMSE Filter

MMSE Filter for the Blured Image:

$$\theta = \begin{bmatrix} 1.712 & 0.740 & 0.928 & 0.815 & -0.937 & -1.813 & 1.807 \\ -1.486 & -1.809 & -0.900 & -0.582 & -2.983 & 0.582 & 1.353 \\ -0.943 & -2.762 & 0.306 & 2.978 & 0.714 & -2.734 & -0.818 \\ 2.028 & -0.553 & 3.635 & 3.448 & 3.236 & -3.158 & 0.677 \\ 1.626 & -3.088 & -0.413 & 5.077 & -0.159 & -1.426 & 0.799 \\ -0.303 & -1.874 & -2.093 & -2.220 & -0.0368 & -1.503 & 0.925 \\ 1.263 & -0.731 & 1.238 & 1.873 & -1.149 & -1.288 & 1.007 \end{bmatrix}$$

MMSE Filter for the Noisy Image *img14gn.tif*:

$$\theta = \begin{bmatrix} 0.01 & -0.0288 & 0.022 & 0.044 & -0.0404 & -0.0058 & 0.0049 \\ 0.0132 & -0.0156 & 0.0079 & 0.0326 & -0.0110 & -0.0078 & 0.0097 \\ -0.0307 & -0.0294 & 0.0330 & 0.1277 & 0.0224 & -0.0320 & -0.0389 \\ 0.0340 & 0.0474 & 0.1362 & 0.2890 & 0.0849 & 0.0021 & 0.0244 \\ -0.0161 & -0.0004 & 0.0903 & 0.1353 & 0.0413 & -0.0035 & 0.0127 \\ -0.0147 & 0.0197 & -0.0215 & 0.0678 & -0.0216 & -0.0028 & 0.0277 \\ 0.0189 & 0.0103 & -0.0148 & 0.0547 & -0.0400 & -0.0379 & -0.0049 \end{bmatrix}$$

MMSE Filter for the Noisy Image *img14sp.tif*:

$$\theta = \begin{bmatrix} 0.0157 & 0.0106 & -0.0217 & 0.0200 & -0.0560 & -0.0010 & -0.0152 \\ -0.0133 & -0.0347 & 0.0593 & 0.0530 & -0.0266 & 0.0588 & 0.0191 \\ -0.0341 & -0.0027 & 0.0423 & 0.1107 & -0.0123 & -0.0367 & -0.0544 \\ 0.0298 & 0.0157 & 0.0954 & 0.3146 & 0.0916 & -0.0117 & 0.0086 \\ -0.0006 & 0.0084 & 0.1002 & 0.1585 & 0.0433 & 0.0073 & 0.0016 \\ 0.00095 & -0.0286 & -0.0066 & 0.0629 & -0.0170 & 0.0053 & 0.0592 \\ 0.0335 & -0.0017 & -0.00106 & 0.0466 & -0.04354 & -0.0391 & -0.0095 \end{bmatrix}$$

2 Weighted Median Filtering

1. Result of Median filtering

Figure 8: Filtered Image of *img14bl.tif*

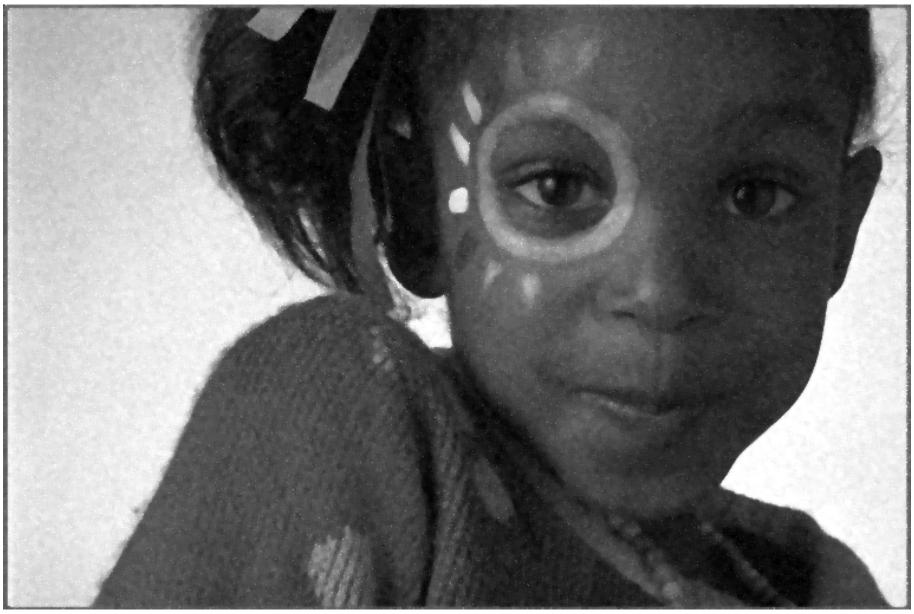

Figure 9: Filtered Image of *img14gn.tif*

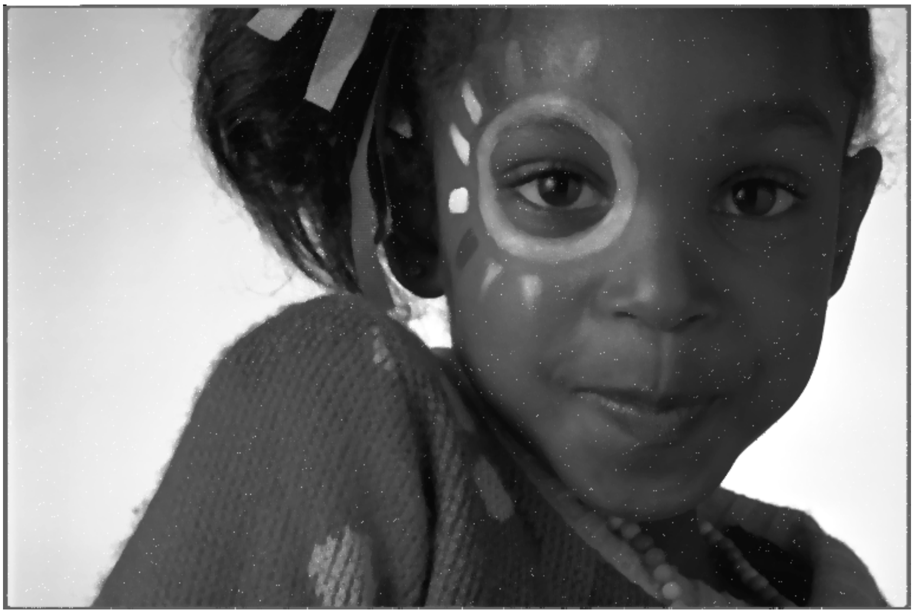

Figure 10: Filtered Image of *img14sp.tif*

2. C code of weighted median filtering

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

uint8_t weightedMeanFilter(uint8_t **img,
                           double **weight,
                           int i, int j,
                           int width, int height,
                           int weight_size);
void swap(int *a, int *b);
void selectionSort(int arr1[], int arr2[], int n);
int sum(int arr1[], int start, int end);

int main(int argc, char const *argv[])
{
  FILE *fp;
  struct TIFF_img input_img, filter_img;
  double **output;
  int weight_size = 5;
  double **weight;

  // check for argument count
  if (argc != 2)
  {
    fprintf(stderr, "Missing Argument\n");
    exit(1);
```

```c
}

//check for error in reading files
if ((fp = fopen(argv[1], "rb")) == NULL)
{
  fprintf(stderr, "cannot open file %s\n", argv[1]);
  exit(1);
}

// check for reading tiff file
if (read_TIFF(fp, &input_img))
{
  fprintf(stderr, "error reading file %s\n", argv[1]);
  exit(1);
}

fclose(fp);

if (input_img.TIFF_type != 'g')
{
  fprintf(stderr, "error:  image must be greyscale\n");
  exit(1);
}

//allocate memory for the output image
output = (double **)get_img(input_img.width, input_img.height,
                            sizeof(double));
weight = (double **)get_img(weight_size, weight_size,
                            sizeof(double));

//generate weight
for (int i = 0; i < weight_size; i++)
{
  for (int j = 0; j < weight_size; j++)
  {
    if (i == 1 || i == weight_size - 1 ||
        j == 1 || j == weight_size - 1)
    {
      weight[i][j] = 1.0;
    }
    else
    {
      weight[i][j] = 2.0;
    }
  }
}

//apply the filter
```

```
  get_TIFF(&filter_img, input_img.height, input_img.width, 'g');
  for (int i = 0; i < input_img.height; i++)
  {
    for (int j = 0; j < input_img.width; j++)
    {
      filter_img.mono[i][j] = weightedMeanFilter(input_img.mono,
                                                  (double **)weight,
                                                  i, j,
                                                  input_img.width,
                                                  input_img.height,
                                                  weight_size);
    }
  }

  if ((fp = fopen("output.tif", "wb")) == NULL)
  {
    fprintf(stderr, "cannot open file output.tif\n");
    exit(1);
  }

  if (write_TIFF(fp, &filter_img))
  {
    fprintf(stderr, "cannot write to file output.tif\n");
    exit(1);
  }

  fclose(fp);

  free_img((void **)output);
  free_TIFF(&(input_img));
  free_TIFF(&(filter_img));
  return 0;
}

uint8_t weightedMeanFilter(uint8_t **img,
                           double **weight,
                           int i, int j,
                           int width, int height,
                           int weight_size)
{
  int pixels[weight_size * weight_size];
  int weights[weight_size * weight_size];
  int maxPixels = 0;

  //extact pixels and weights
  for (int k = 0; k < weight_size; k++)
  {
    for (int l = 0; l < weight_size; l++)
```

```
    {
      int loc_i = i + k - weight_size / 2;
      int loc_j = j + l - weight_size / 2;
      if (loc_i >= 0 && loc_i < height &&
          loc_j >= 0 && loc_j < width)
      {

        pixels[k * weight_size + l] = img[loc_i][loc_j];
        weights[k * weight_size + l] = weight[k][l];
        maxPixels++;
      }
    }
  }

  // sort the pixels
  selectionSort(pixels, weights, maxPixels);

  // find the median
  int median_idx;
  for (int i = 0; i < maxPixels; i++)
  {
    if (sum(weights, 0, i) >= sum(weights, i + 1, maxPixels))
    {
      median_idx = i;
      break;
    }
  }
  return pixels[median_idx];
}

void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
}

void selectionSort(int arr1[], int arr2[], int n)
{
  int i, max_idx;

  for (i = 0; i < n - 1; i++)
  {
    max_idx = i;
    for (int j = i + 1; j < n; j++)
    {
      if (arr1[j] > arr1[max_idx])
        max_idx = j;
```

```
            swap(&arr1[max_idx], &arr1[i]);
            swap(&arr2[max_idx], &arr2[i]);
        }
    }
}

int sum(int arr[], int start, int end)
{
    int sum = 0;
    for (int i = start; i <= end; i++)
    {
        sum += arr[i];
    }
    return sum;
}
```