ECE 63700 Laboratory:

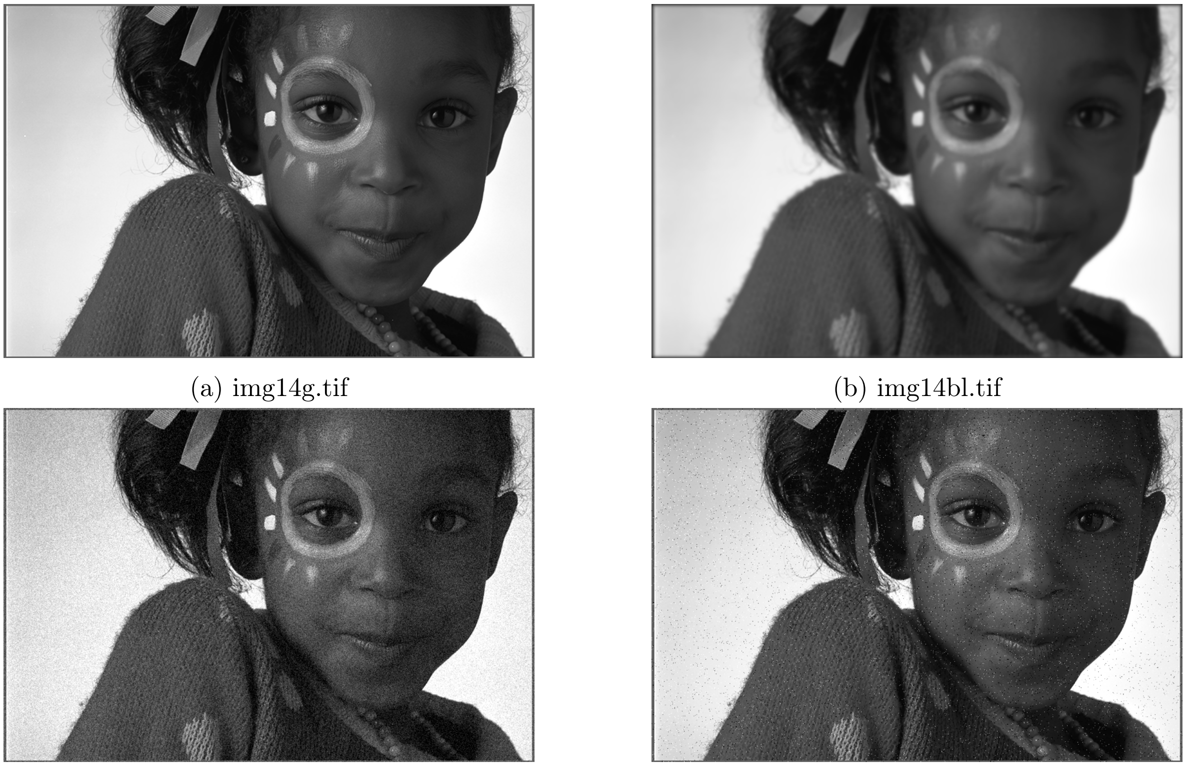
Image Restoration

Shang Hsuan (Sean) Lee

March 27, 2024

1 Minimum Mean Square Error (MMSE) Linear Filters

1.1 Four Original Images



(a) img14g.tif

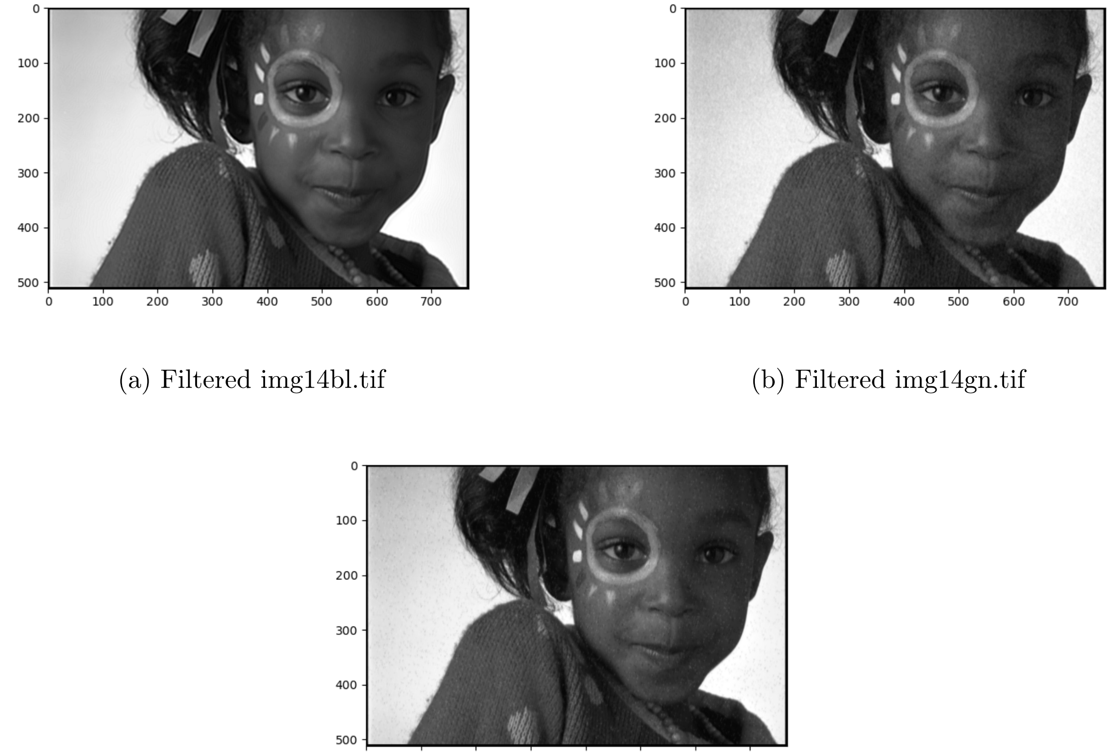
(b) img14bl.tif

(c) img14gn.tif

(d) img14sp.tif

Figure 1: Four Original Images

1.2 Filtered Images



(a) Filtered img14bl.tif

(b) Filtered img14gn.tif

(c) Filtered img14sp.tif

Figure 2: Filtered Images

1.3 MMSE filters

1.3.1 img14bl.tif

$$\theta^* = \begin{bmatrix} 1.5208 & -1.6833 & -0.0147 & 0.9993 & -0.2206 & 0.0671 & 0.5672 \\ -0.2197 & -1.3117 & -0.3552 & -0.4407 & -1.1395 & -0.4067 & -0.4302 \\ 1.3415 & -0.97 & 0.7195 & 2.3861 & 0.6888 & -1.1164 & 0.4174 \\ -1.1754 & -0.0361 & 1.2484 & 1.8966 & 0.5231 & -0.6413 & -0.3591 \\ 0.3286 & -0.8734 & 0.1385 & 0.9866 & 0.4403 & -1.1208 & 0.733 \\ -0.578 & -0.7402 & 0.5593 & 0.1056 & -0.636 & -2.1178 & 1.3349 \\ 1.0864 & -0.8139 & 0.2833 & -0.9683 & 1.0474 & -0.0205 & -0.0308 \end{bmatrix}$$

1.3.2 img14gn.tif

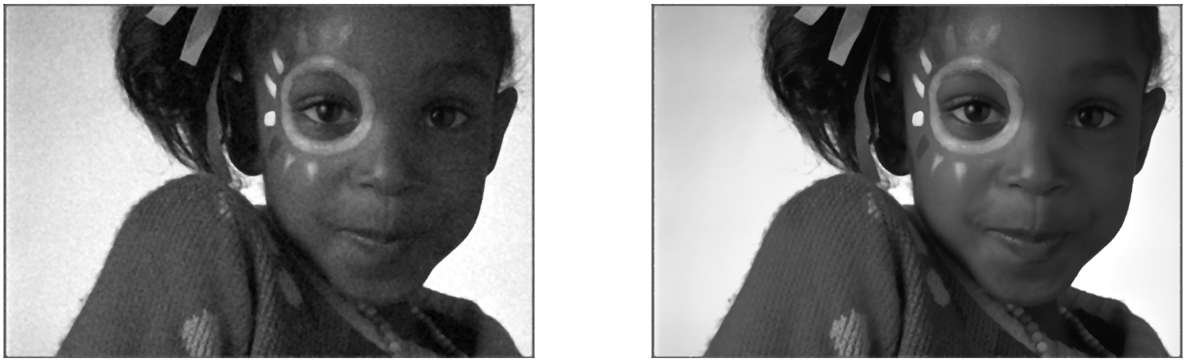
$$\theta^* = \begin{bmatrix} -0.0106 & -0.0088 & -0.0079 & 0.0365 & 0.0359 & 0.0115 & 0.0035 \\ 0.0354 & 0.0035 & 0.0111 & 0.0405 & 0.0461 & -0.0169 & -0.0016 \\ -0.0184 & 0.0214 & 0.0657 & 0.0963 & 0.0337 & -0.0189 & -0.0201 \\ -0.0106 & 0.0043 & 0.1015 & 0.1991 & 0.0717 & 0.0155 & -0.0015 \\ -0.0004 & 0.0407 & 0.0371 & 0.0851 & 0.0333 & 0.007 & 0.0119 \\ -0.0285 & 0.0128 & 0.0258 & 0.0179 & 0.0019 & 0.025 & -0.0105 \\ -0.0284 & 0.0007 & 0.0164 & 0.012 & 0.0117 & 0.0066 & 0.0102 \end{bmatrix}$$

1.3.3 img14sp.tif

$$\theta^* = \begin{bmatrix} 0.026 & -0.035 & 0.025 & 0.043 & 0.046 & 0.005 & -0.0001 \\ 0.023 & 0.005 & -0.003 & 0.024 & 0.037 & -0.014 & -0.002 \\ -0.001 & -0.017 & 0.069 & 0.129 & 0.006 & -0.017 & -0.008 \\ 0.018 & -0.012 & 0.079 & 0.183 & 0.086 & -0.006 & 0.007 \\ -0.011 & 0.029 & 0.057 & 0.111 & 0.061 & -0.011 & 0.014 \\ -0.023 & 0.013 & 0.018 & 0.023 & 0.028 & 0.001 & -0.02 \\ -0.037 & 0.016 & 0.018 & -0.012 & -0.002 & 0.017 & 0.016 \end{bmatrix}$$

2 Weighted Median Filtering

2.1 Median Filtering Results



(a) Filtered img14gn.tif

(b) Filtered img14sp.tif

Figure 3: Filtered Images

2.2 C code Listing

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

void error(char *name);

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

void sort(double arr1[], double arr2[], int n){
        for (int i=0; i<n-1; i++){
                for (int j=0; j<n-i-1; j++){
                        if (arr1[j] < arr1[j+1]){
                                swap(&arr1[j], &arr1[j+1]);
                                swap(&arr2[j], &arr2[j+1]);
                        }
                }
        }
}

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, output_img;
  double **img, **img_padded;
        int32_t i, j, a, b, idx;
        double sum1, sum2;
        double X[25];
        double weight[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};

  char *input_file = "img14sp.tif";

  /* open image file */
  if ( ( fp = fopen ( input_file, "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot␣open␣file␣%s\n", input_file );
    exit ( 1 );
  }

  /* read image */
  if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error␣reading␣file␣%s\n", input_file );
    exit ( 1 );
  }

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

  /* check the type of image data */
  if ( input_img.TIFF_type != 'g' ) {
    fprintf ( stderr, "error:␣␣image␣must␣be␣24-bit␣color\n" );
    exit ( 1 );
```

```
}

/* set up structure for output grey image */
/* Note that the type is 'g' rather than 'c' */
get_TIFF ( &output_img, input_img.height, input_img.width, 'g' );

/* Allocate image and a padded image */
      img = (double **)get_img(input_img.width, input_img.height, sizeof(double
      img_padded = (double **)get_img(input_img.width+4, input_img.height+4, si

/* Pad image */
      for (i=0; i<input_img.height+2; i++){
  for (j=0; j<input_img.width+2; j++){
    img_padded[i][j] = 0;
  }
      }

for (i=0; i<input_img.height; i++){
  for (j=0; j<input_img.width; j++){
    img_padded[i+2][j+2] = input_img.mono[i][j];
  }
      }


for (i=2; i<input_img.height+2; i++){
              for (j=2; j<input_img.width+2; j++){
                      idx = 0;
                      for (a=i-2; a<=i+2; a++){
                              for (b=j-2; b<=j+2; b++){
                                      X[idx] = img_padded[a][b];
                                      idx += 1;
                              }
                      }

                      sort(X, weight, sizeof(X)/sizeof(X[0]));

                      /* Find median index a*/
                      idx = 1;
                      sum1 = 0;
                      sum2 = 0;
                      while(1) {
                              for (b=0; b<=idx; b++){
                                      sum1 = sum1 + weight[b];
                              }
                              for (b=idx+1; b<sizeof(weight)/sizeof(weight[0]);
                                      sum2 = sum2 + weight[b];
                              }
                              if (sum1 >= sum2) {
                                      break;
                              }
```

```
                                sum1 = 0;
                                sum2 = 0;
                                idx += 1;
                        }

                        img[i-2][j-2] = X[idx];
                }
        }

  for (i=0; i<input_img.height; i++){
                for (j=0; j<input_img.width; j++){
                        // pixel = (int32_t)img[i][j];
                        if(img[i][j] > 255) {
                                img[i][j] = 255;
                        }
                        if(img[i][j]<0) {
                                img[i][j] = 0;
                        }
                        output_img.mono[i][j] = (int32_t)img[i][j];
                }
        }


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

  /* write output image */
  if ( write_TIFF ( fp, &output_img ) ) {
    fprintf ( stderr, "error␣writing␣TIFF␣file␣wmf_sp.tif\n");
    exit ( 1 );
  }

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

  /* de-allocate space which was used for the images */
  free_img((void *)img);
  free_img((void *)img_padded);
  free_TIFF ( &(input_img) );
  free_TIFF ( &(output_img) );

  return(0);
}

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 pass the input image through a low pass filter,\n");
        printf("and writes out the result as an 8-bit image\n");
        printf("with the name 'lpf_output.tiff'.\n");
        exit(1);
    }
```