March 31, 2023

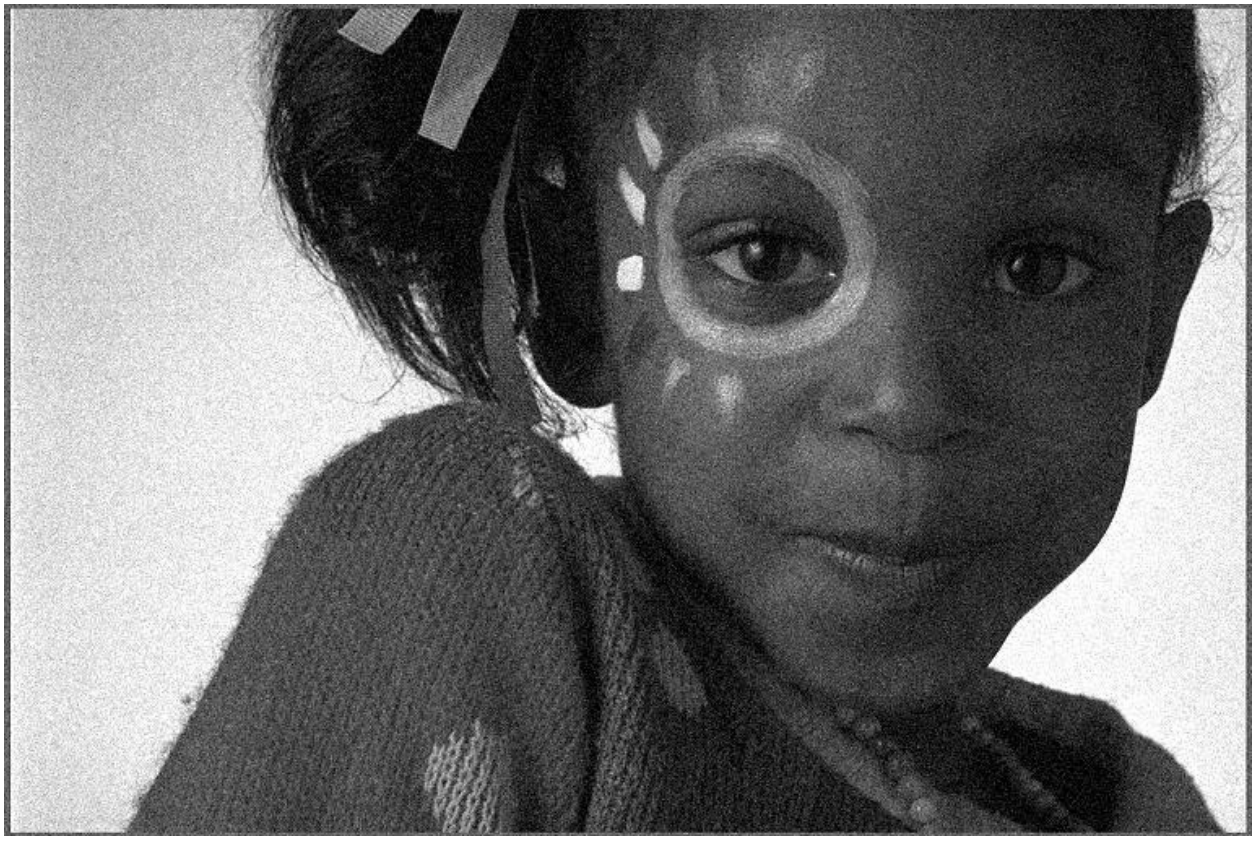
Section 1: MMSE Linear Filters

Deliverable 1: Four original images:

Img14g.tif:

Img14bl.tif:

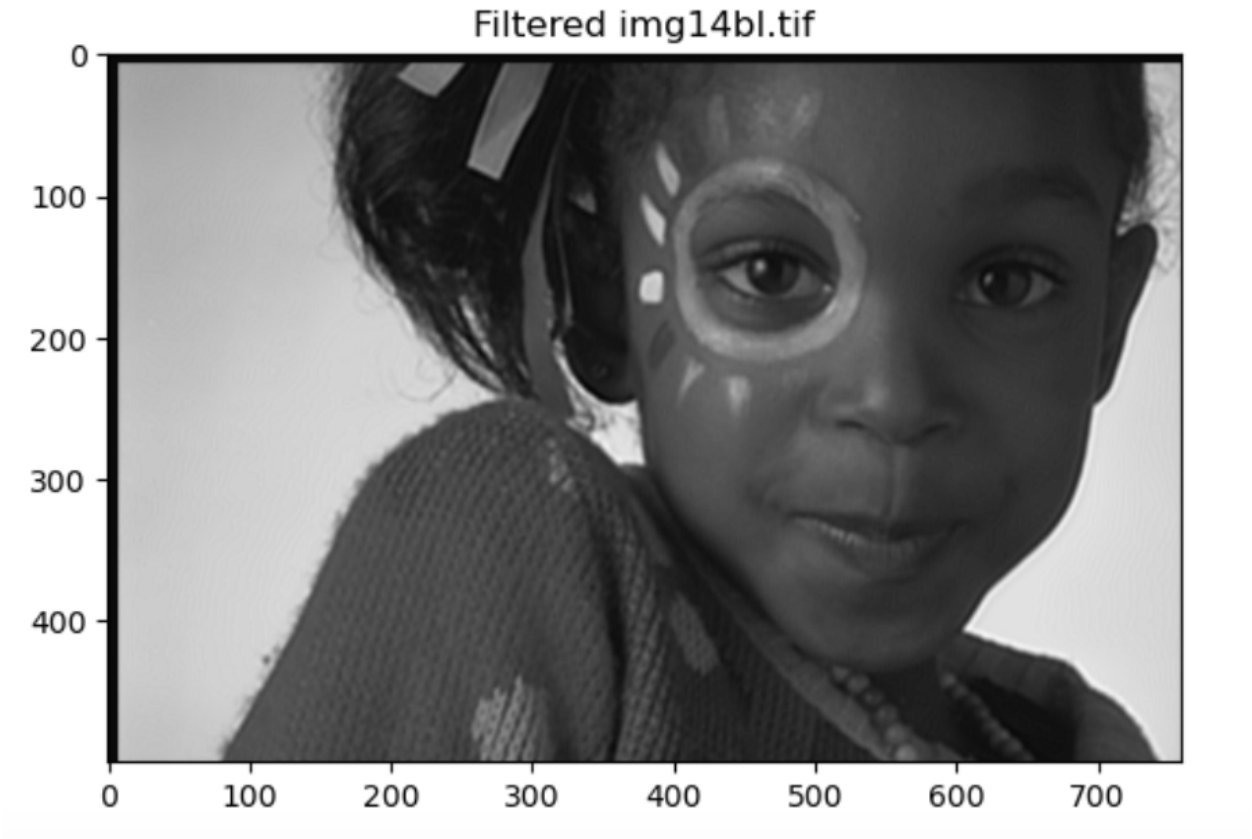
Img14gn.tif:



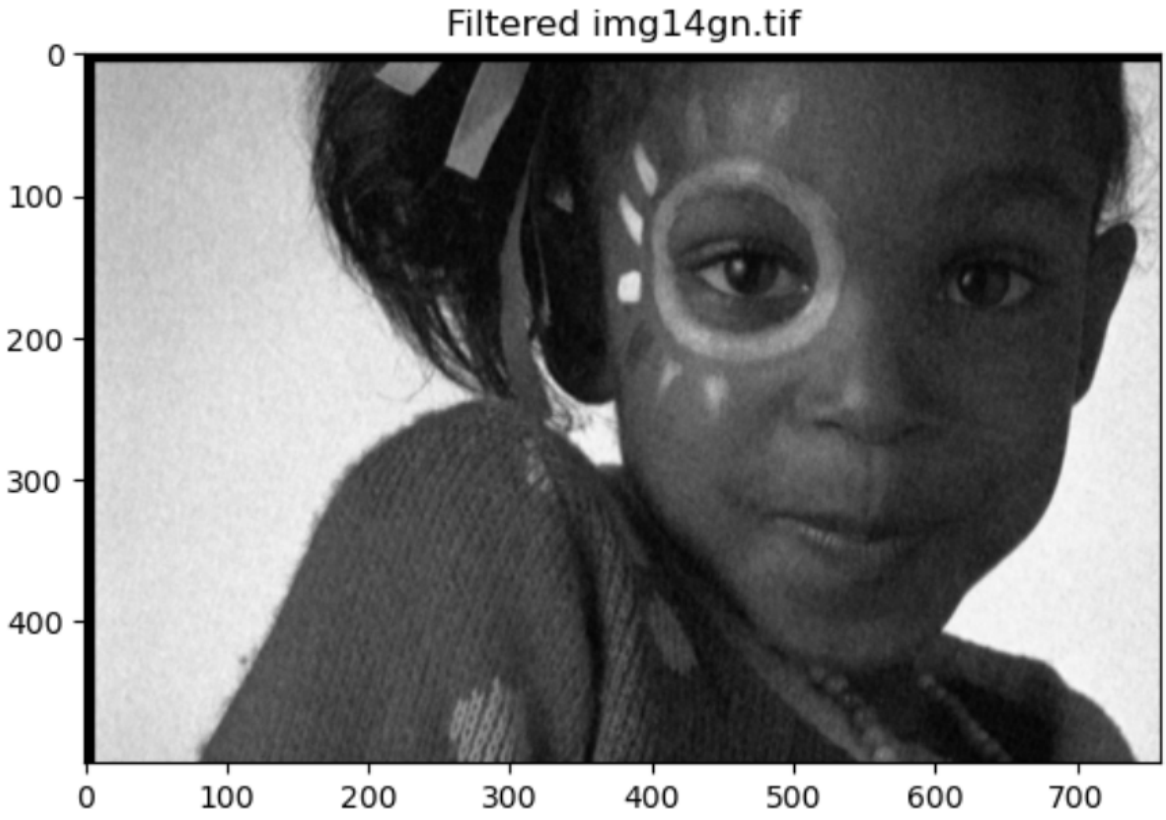
Img14sp.tif:

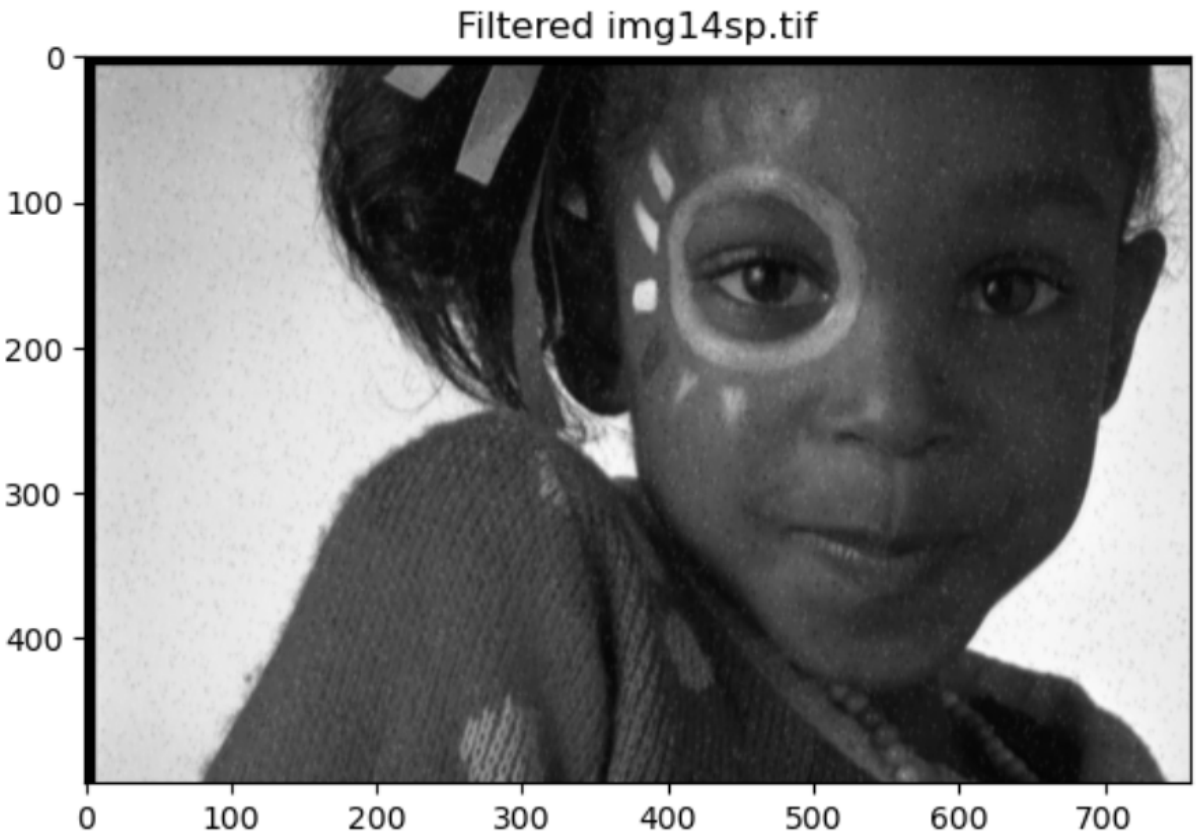
Deliverable 2: Output of optimal filtering for blurred image and the 2 noisy images

Blurred, followed by the gn and sp noise images:

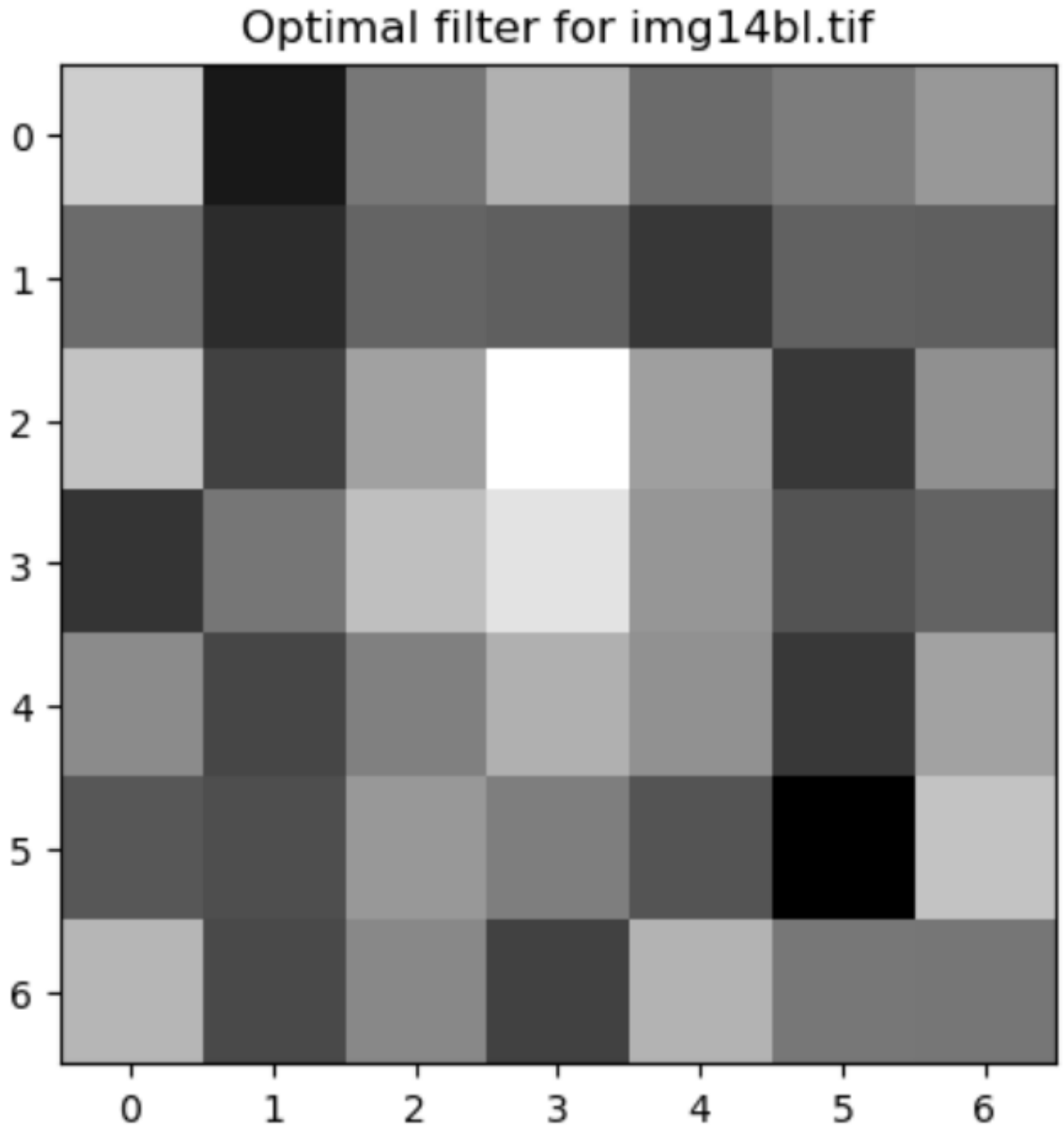

Filtered img14bl.tif
0
100
200
300
400
0
100
200
300
400
500
600
700

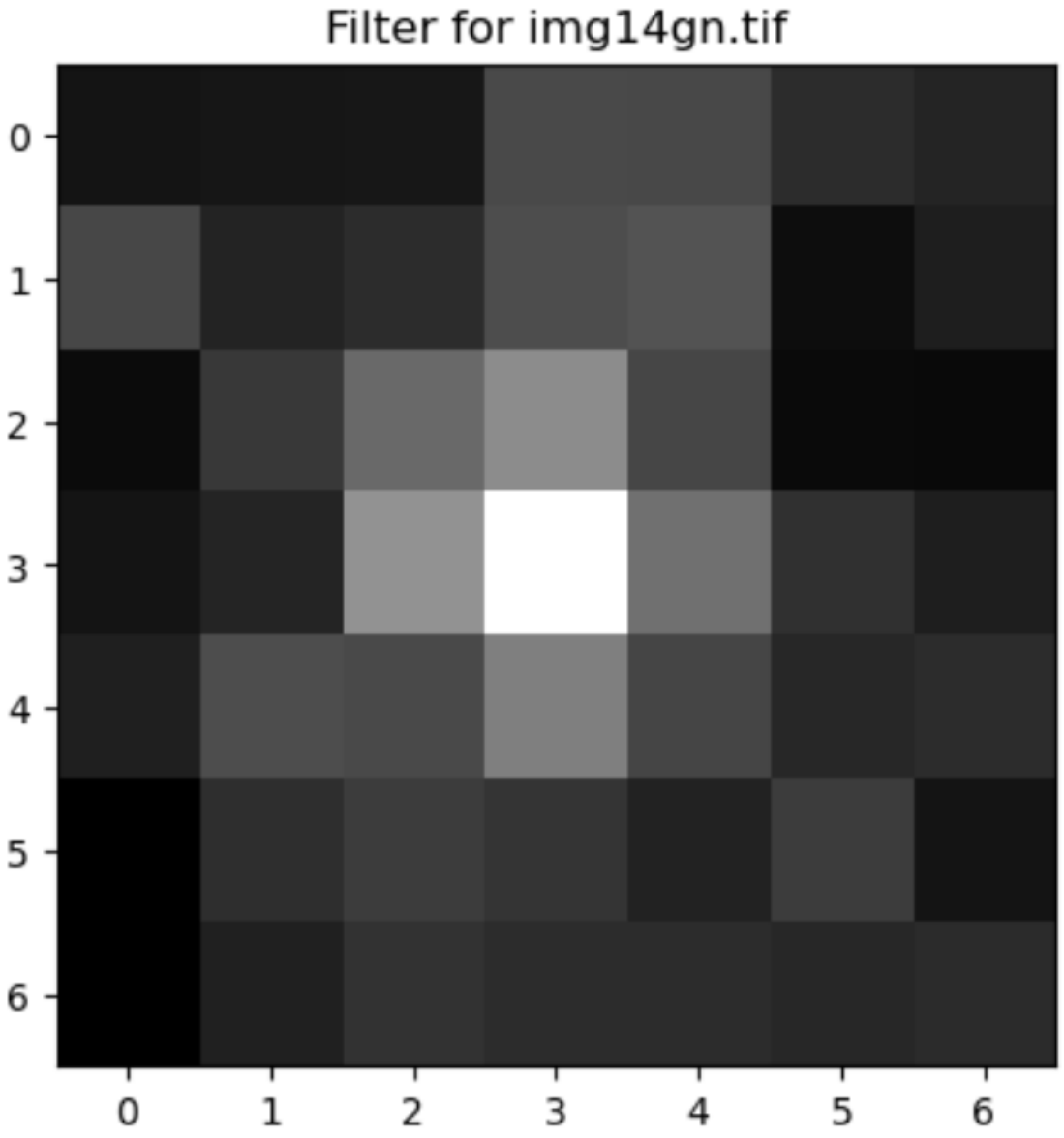
March 31, 2023

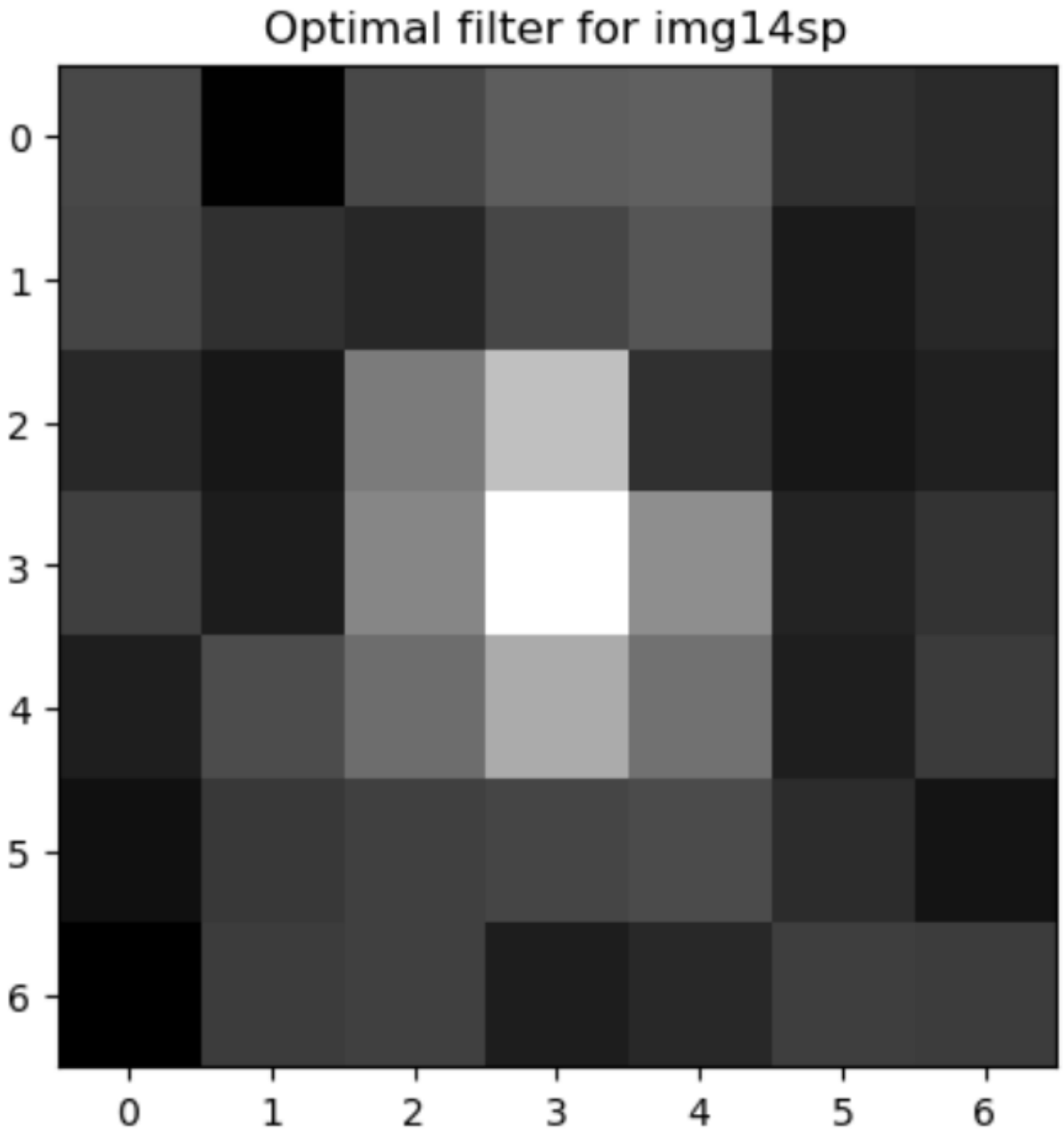




Deliverable 3: Hand in the optimal MMSE filters for each image:


Optimal filter for img14bl.tif
0
1
2
3
4
5
6
0
1
2
3
4
5
6

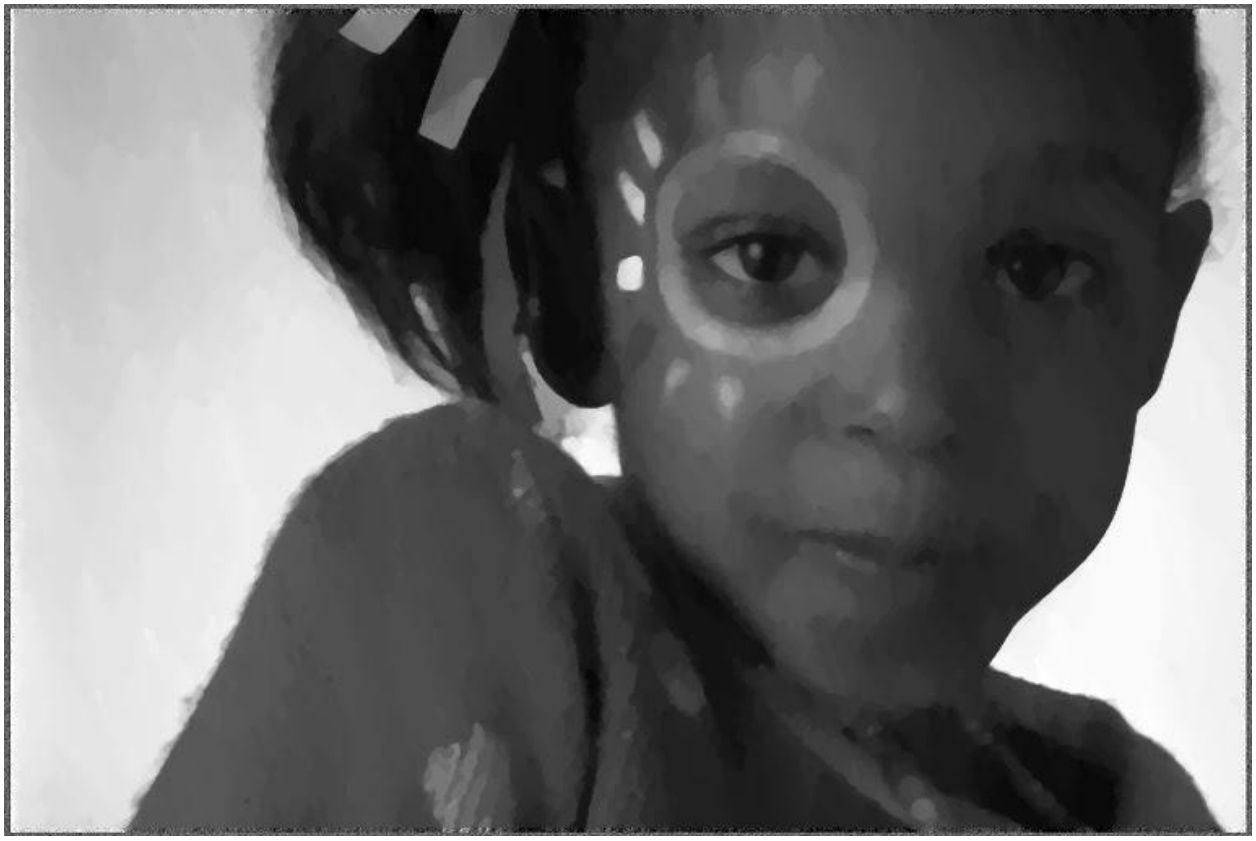

Filter for img14gn.tif
0
1
2
3
4
5
6
0
1
2
3
4
5
6



Section 2: Weighted Median Filtering

Deliverable 1: Results of median filtering:

Img14gn.tif, filtered:



Img14sp.tif, filtered:

Deliverable 2: C code:

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

void error(char *name);

int * sort(int arr[25], int weights_array[25], int return_flag);

int main (int argc, char **argv)
{
 FILE *fp;
 struct TIFF_img input_img, processed_img;
 double **img1,**img2;
 int32_t i,j,pixel;
```

```
int window[5][5];
int window_array[25];
int H,W;

if ( argc != 2 ) error( argv[0] );
printf("at the start");
/* 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 data */
if ( input_img.TIFF_type != 'g' ) {
  fprintf ( stderr, "error:  image must be grayscale\n" );
  exit ( 1 );
}
/* Allocate image of double precision floats */
img1 = (double **)get_img(input_img.width,input_img.height,sizeof(double));
img2 = (double **)get_img(input_img.width,input_img.height,sizeof(double));
 /* copy mono component to double array */
 for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
  img1[i][j] = input_img.mono[i][j];
}


int count = 0;
for (i = 0; i < input_img.height; i++)
for (j = 0; j < input_img.width; j++) {
  if ((i > 5) && (j > 5) && (i < 507) & (j < 762)) { //we are far enough in img so
that padding doesn't matter
    count = 0;
```

```
    for (H = -2; H < 3; H++)
      for (W = -2; W < 3; W++){
        //printf("H is %i\n, W is %i\n, i is %i\n, j is %i\n", H, W, i, j);
        window[H+2][W+2] = img1[i + H][j + W];
        window_array[count] = img1[i + H][j + W];
        count = count + 1;
      if ( (H == 2) && (W == 2)){
        int weights_array[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};
        //count = count + 1;
        //printf("\nunsorted, window_array:\n");
        //for (int x = 0; x < 25; x++){
        //  printf("%i ", window_array[x]);
        //}
        //printf("\nunsorted weights_array:\n");
        //for (int x = 0; x < 25; x++) { //accounted for
        //  printf("%i ", weights_array[x]); //accounted for
        //} //accounted for

        int *ptr_window;
        int *ptr_weights;
        ptr_window = sort(window_array, weights_array, 0); //return the px values
sorted
        ptr_weights = sort(window_array, weights_array, 1); //return the weights
sorted

        //printf("\nthe sorted array px values\n");
        //for (int x = 0; x < 25; x++){ //accounted for
          //printf("%i ", *(ptr_window + x));} //accounted for


        //printf("\nthe sorted weight values:\n");
        //for (int x = 0; x < 25; x++){ //accounted for
          //printf("%i ", *(ptr_weights+x));} //accounted for

        int sum_front = 0;
        int sum_front_tracker[25];
        int sum_back_tracker[25];

        for (int x = 0; x < 25; x++) {
          sum_front = sum_front + *(ptr_weights + x); //sum of front of list
          //printf("current sum of front of list: %i\n", sum_front);
```

```
          sum_front_tracker[x] = sum_front;
        }

        int sum_back = 0;
        for (int x = 24; x > -1; x--){
          sum_back = sum_back + *(ptr_weights + x);
          //printf("current sum from back of list %i\n", sum_back);
          sum_back_tracker[x] = sum_back;
        }

        int idx = 0;
        for (int x = 0; x < 25; x++){
          //printf("this is element %i of front sum trakcer %i\n", x,
sum_front_tracker[x]);
          //printf("this is element %i of back sum trakcer %i\n", x ,
sum_back_tracker[x]);
          if (sum_front_tracker[x] >= sum_back_tracker[x]) {
            //printf("found idx of median as %i", x);
            idx = x;
            //printf("value of px to do median filter is %i", window_array[idx]);
            break;
          }
        }


        //set pixel value to the value found at the idx we found.
        img1[i][j] = (double)*(ptr_window + idx);
        //printf("value of img1 at %i, %i, is %i\n", i, j, (int)img1[i][j]);
        //printf("but at the same time, the derefed ptr is %i\n",  *(ptr_window +
idx));

          }
        }
      }
    }

 //printf("finished the copy!\n");
 //printf("count is %i", count);

 /* Filter image along horizontal direction */
 // for ( i = 0; i < input_img.height; i++ )
 // for ( j = 1; j < input_img.width-1; j++ ) {
```

```
//   img2[i][j] = (img1[i][j-1] + img1[i][j] + img1[i][j+1])/3.0;
// }

printf("finished the LPF\n");

// /* Fill in boundary pixels */
// /*
// for ( i = 0; i < input_img.height; i++ ) {
//   img2[i][0] = 0;
//   img2[i][input_img.width-1] = 0;
// }
// */

// /* Set seed for random noise generator */
//srandom2(1);

// /* Add noise to image */
// for ( i = 0; i < input_img.height; i++ )
// for ( j = 1; j < input_img.width-1; j++ ) {
//   img2[i][j] += 32*normal();
// }



// /* set up structure for output achromatic image */
// /* to allocate a full color image use type 'c' */
get_TIFF ( &processed_img, input_img.height, input_img.width, 'g' );
//printf("allocated memory for achromatic image.\n");

// /* set up structure for output color image */
// /* Note that the type is 'c' rather than 'g' */
// //get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
printf("about to write image to file?\n");
// /* copy img1 component to new image (titled "processed image") */
for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
  pixel = (uint32_t)img1[i][j];
  if(pixel>255) {
    processed_img.mono[i][j] = 255;
  }
  else {
    if(pixel<0) processed_img.mono[i][j] = 0;
```

```
    else processed_img.mono[i][j] = pixel;
  }
}
//printf("done with clipping and converting image to int values\n");
// /* Illustration: constructing a sample color image -- interchanging the red and
green components from the input color image */
// for ( i = 0; i < input_img.height; i++ )
//     for ( j = 0; j < input_img.width; j++ ) {
//         color_img.mono[i][j] = input_img.mono[i][j];
//         //color_img.color[1][i][j] = input_img.color[0][i][j];
//         //color_img.color[2][i][j] = input_img.color[2][i][j];
//     }

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

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

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


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

// /* write color image */
// if ( write_TIFF ( fp, &color_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) );
 free_TIFF ( &(processed_img) );
 //free_TIFF ( &(color_img) );
  free_img( (void**)img1 );
 free_img( (void**)img2 );

 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 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);
}

int * sort(int arr[25], int weights_array[25], int return_flag) //sort in descending
order (largest to smallest)
{
 int n = 25, x, j, t, temp_weight = 0;
 // iterates the array elements
 for (x = 0; x < n; x++){
   // iterates the array elements from index 1
   for (j = x + 1; j < n; j++) {
     // comparing the array elements, to set array
     // elements in descending order
     if (arr[x] < arr[j]) {
         t = arr[x];
         arr[x] = arr[j];
         arr[j] = t;

         //swap the corresponding weights order:
         temp_weight = weights_array[x];
         weights_array[x] = weights_array[j];
         weights_array[j] = temp_weight;}}}
```

```
if (return_flag == 0) {return arr;}
if (return_flag == 1) {return weights_array;}
}
```