Report Computer Vision TP2

Choules, Louis
louis.choules@etu.univ-grenoble-alpes.fr

Lkhagvajav, Dagvanorov
dagvanorov.lkhagvajav@etu.univ-grenoble-alpes.fr

October 2022

Contents

List of Code

List of Figures

1 Introduction

This is the report for the TP2 Computer Vision. For this project, we have several files as describe below.

1.1 filtering_3x3

This file is for creating the image with ***Gaussian Filter*** with filter 3x3.

How to compile

```
make filtering_3x3
```

List of Code 1: Compile filtering3x3

How to use

```
./filtering_3x3 [filename]>[file_output] [times_of_repetition]
```

List of Code 2: Use filtering3x3

you need to pass some arguments in here.

- **filename** is the picture that you want to modified
- **file_output** is the name of output picture
- **times_of_repetition** is the total times of filter using Gaussian Filter (3x3)

1.2 filtering_5x5

This file is for creating the image with ***Gaussian Filter*** with filter 5x5.

How to compile

```
make filtering_5x5
```

List of Code 3: Compile filtering5x5

How to use

```
./filtering_5x5 [filename]>[file_output] [times_of_repetition]
```

List of Code 4: Use filtering5x5

you need to pass some arguments in here.

- **filename** is the picture that you want to modified
- **file_output** is the name of output picture
- **times_of_repetition** is the total times of filter using Gaussian Filter (5x5)

1.3 filtering

This file is for creating the image with ***Gaussian Filter*** where you can specify the size of the filter manually.

How to compile

```
make filtering
```

List of Code 5: Compile filtering

How to use

```
./filtering [filename]>[file_output] [size_of_the_blur] [
    times_of_repetition]
```

List of Code 6: Use filtering

you need to pass some arguments in here.

- **filename** is the picture that you want to modified
- **file_output** is the name of output picture
- **size_of_the_blur** is the size of filter (size_of_the_blur x size_of_the_blur) expected in odd number and greater than 0
- **times_of_repetition** is the total times of filter

1.4 filter_median

This file is for creating the image with ***Median Filter*** where you can specify the size of the filter manually.

How to compile

```
make filter_median
```

List of Code 7: Compile filter_median

How to use

```
./filter_median [filename]>[file_output] [size_of_the_blur] [
    times_of_repetition]
```

List of Code 8: Use filter_median

you need to pass some arguments in here.

- **filename** is the picture that you want to modified
- **file_output** is the name of output picture
- **size_of_the_blur** is the size of filter (size_of_the_blur x size_of_the_blur) expected in odd number and greater than 0
- **times_of_repetition** is the total times of filter using median

1.5 histogram

This file is for creating the image with ***Gaussian Filter*** where you can specify the size of the filter manually and also add ***Histogram Modification*** at the result of the image.

How to compile

```
make histogram
```

List of Code 9: Compile histogram

How to use

```
./histogram [filename]>[file_output] [size_of_the_blur] [
    times_of_repetition] [histogram_mode]
```

List of Code 10: Use histogram

you need to pass some arguments in here.

- **filename** is the picture that you want to modified
- **file_output** is the name of output picture
- **size_of_the_blur** is the size of filter (size_of_the_blur x size_of_the_blur) expected in odd number and greater than 0
- **times_of_repetition** is the total times of filter using Gaussian Filter
- **histogram_mode** is the mode of histogram transformation expected value "s" or "e"

1.6 Image Test Case

Figure 1: boat.png

Figure 2: boat_noise1.png

Figure 3: boat_noise2.png

2 Program Description

2.1 Local Filter

Local filtering is consist of replacing a pixel value by a weighted combination of values in local neighborhood of the pixel. The weight is based on the filter that we will use.

2.1.1 Gaussian Filter

for Gaussian Filter we have 3 files such as, *filtering_3x3*, *filtering_5x5*, and *filtering*. For this three files, we using the same concept of it. We define a binomial filter with n size from a Pascal Triangle.

$$
\begin{array}{ll}
n=0 & \qquad\qquad\qquad\quad 1 \\
n=1 & \qquad\qquad\quad 1 \quad 1 \\
n=2 & \qquad\qquad 1 \quad 2 \quad 1 \\
n=3 & \qquad\quad 1 \quad 3 \quad 3 \quad 1 \\
n=4 & \qquad 1 \quad 4 \quad 6 \quad 4 \quad 1 \\
n=5 & \quad 1 \quad 5 \quad 10 \quad 10 \quad 5 \quad 1 \\
n=6 & \; 1 \quad 6 \quad 15 \quad 20 \quad 15 \quad 6 \quad 1 \\
 & \overline{0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6}
\end{array}
$$

We will create a Pascal triangle like above, we need to get the filter by using the n at size-1; For example we will using 3×3 filter, then we will using the $n = 2$. We need to create a matrix from it by matrix multiplication. The divider is comes from the sum of all the values in the filter.

$$h = 1/16 \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

For example we have a pixel with the neighbors like below.

$$\begin{bmatrix} 32 & 12 & 4 \\ 21 & 41 & 23 \\ 41 & 32 & 13 \end{bmatrix}$$

then we apply the filter above with current matrix. we will get

$$h = 1/16 \begin{bmatrix} 1 \times 32 & 2 \times 12 & 1 \times 4 \\ 2 \times 21 & 4 \times 41 & 2 \times 23 \\ 1 \times 41 & 2 \times 32 & 1 \times 13 \end{bmatrix}$$

$$h = \frac{1 \times 430}{16}$$

$$h = 26.875$$

We have 3 option about how to change it into an integer, either we can use either floor(), ceiling(), or round(). But for our code, we only using the integer directly so it similar to floor(). So for the new pixel will be 26.

2.1.2 Median Filter

for Median Filter, we have 1 file called, filter_median. The concept of this filter is take all the value in it's pixel value and it's neighbor's pixel values. Then we sort all the values and for the new value is will be the one in the middle.

For example we have a pixel with the neighbors like below.

$$\begin{bmatrix} 32 & 12 & 4 \\ 21 & 41 & 23 \\ 41 & 32 & 13 \end{bmatrix}$$

Then we sort all the value so it become, [4, 12, 13, 21, 23, 32, 32, 41, 41]. Then we will take the value in the middle of the sorted value as the new value in that pixel. So for the new pixel in that pixel is 23.

2.2 Histogram

PGM formatted images are formatted as 2-dimensional arrays. Each element of the array is in the range between 0 and 255 in regard of the its intensity. Histogram of images is used to representing the distribution of the values between the range.

2.2.1 Histogram Stretching

In order to make image details more clear and perceivable, there should be gaps between the intensity values in the histogram. Histogram stretching is used to enhancing the contrast of an image. By considering the histogram of the image, the contrast is the difference between the maximum and the minimum pixel intensity. In other words, the histogram stretching expands the entire range of the pixel intensity value. It should be noted that the min and max pixel intensity of an image should not be the same as possible min and max intensities (0, 255).

Let P_i be a pixel intensity value.

Let a, b be the current minimum and maximum values of the image, respectively.

Let MIN, MAX be the possible minimum and maximum pixel intensity.

Thus, the modified value of the pixel intensity after stretching is P_st.

$$P_st = \frac{(P_i - a)(MAX - MIN)}{(b - a)} + MIN$$

Since MIN equals 0, we can retrieve the following formula.

$$P_st = \frac{(P_i - a) * MAX}{(b - a)}$$

2.2.2 Histogram Equalization (Normalization)

On the other side, the histogram equalization is used to normalize the histogram density into a uniform distribution. The expected result is for the cumulative histogram of the image becomes as flat as possible.

Let *hist(j)* is the histogram value of j intensity.

Let *P_i* be a pixel intensity value.

Let *Max* be the maximum pixel intensity.

$$P_st = Max\frac{\Sigma_{j=0}^{P_i} hist(j)}{\Sigma_{j=0}^{Max} hist(j)}$$

3 Program

3.1 Code Explanation

3.1.1 get_byte_by_pos

```
int get_byte_by_pos(int col, int row){
  if(row < 0)
    row = 0;
  if(col < 0)
    col = 0;
  if(col >= cols)
    col = cols-1;
  if(row >= rows)
    row = rows-1;

  return graymap[row * cols + col];
}
```

List of Code 11: get_byte_by_pos

This line of code is for take the byte based on coordinate passed in the arguments. if the coordinate is out of the image, it will return the nearest pixel.

3.1.2 createFilter

```
void createFilter(int filter_size){
  int coef, i, j, temp_pascal[filter_size+2];
  divider = 0;
  for(i = 0; i<= filter_size; i++){
    if(i==0)
      coef = 1;
    else
      coef = coef * (filter_size-i)/i;

    temp_pascal[i] = coef;
  }

  filter = malloc(sizeof(int) * filter_size * filter_size);
  for(i = 0; i < filter_size; i++){
    for(j = 0; j < filter_size; j++){
      int x = temp_pascal[i] * temp_pascal[j];
      filter[i*filter_size+j] = x;
      divider += x;
    }
  }
}
```

List of Code 12: createFilter

This line of code is for creating the filter dynamically, it will return the filter based on the size passed on the argument.

3.1.3 repeat_run

```
void repeat_run(int total_runs, int filter_size){
  gray temp_graymap[cols * rows];
  int k;
```

```
  int i, j;
  for(k = 0; k < total_runs; k++){
    for(i=0; i < rows; i++){
      for(j=0; j < cols ; j++){
        temp_graymap[i * cols + j] = get_new_byte(j,i, filter_size)
    ;
      }
    }
    for(i=0; i < rows; i++){
      for(j=0; j < cols ; j++){
        graymap[i * cols + j] = temp_graymap[i * cols + j];
      }
    }
  }
}
```

List of Code 13: repeat_run

This line of code is for do the filtering repeatedly.

3.1.4 get_new_byte Gaussian Filter Version

```
int get_new_byte(int col, int row, int filter_size){
  int temp = 0;
  int start_col = col - (filter_size/2);
  int start_row = row - (filter_size/2);
  for(int i = 0; i < filter_size; i++){
    for(int j = 0; j < filter_size; j++){
      temp += get_byte_by_pos(start_col + i, start_row + j) *
    filter[i *filter_size + j];
    }
  }

  return temp / divider;
}
```

List of Code 14: get_new_byte Gaussian Filter Version

This line of code is for get the new pixel based on Gaussian Filter that has been created before.

3.1.5 get_new_byte Median Filter Version

```
int get_new_byte(int col, int row, int size_filter){
  int temp[size_filter*size_filter];
  int i, j;
  for(i = -(size_filter/2); i <= (size_filter/2); i++){
    for(j= -(size_filter/2); j <= (size_filter/2); j++){
      temp[i * size_filter + j] = get_byte_by_pos(col-j, row-i);
    }
  }

  for(i = 1; i < size_filter*size_filter; i++){
    for(j=0; j < i ; j++){
      if(temp[j] > temp[j+1]){
        int x = temp[j];
        temp[j] = temp[j+1];
        temp[j+1] = x;
      }
```

```
    }
  }
```

List of Code 15: get_new_byte Median Filter Version

This line of code is for get the new pixel based on Median Filter, where it will take the byte in the specific coordinate then sort it then take the median of sorted area.

3.1.6 clear_histogram

```
void clear_histogram(int maxval){
  for(int i = 0; i <= maxval; i++){
    histogram[i] = 0;
  }
}
```

List of Code 16: clear_histogram

This line of code is for create an empty histogram array.

3.1.7 create_histogram

```
void create_histogram(int col, int row, int maxval){
  clear_histogram(maxval);
  int i,j;

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      histogram[graymap[i * cols + j]] ++;
    }
  }
}
```

List of Code 17: create_histogram

This line of code is for create the histogram array.

3.1.8 getMaxMin

```
void getMaxMin(int col, int row){
  max = __INT_MAX__ * -1;
  min = __INT_MAX__;
  int i,j;

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      int x = graymap[i * cols + j];
      if(max < x)
        max = x;
      if(min > x)
        min = x;
    }
  }
}
```

List of Code 18: getMaxMin

This line of code is for define the maximum and the minimum of value in the histogram.

3.1.9 histogram_stretching

```
void histogram_stretching(int col, int row, int maxval){
  create_histogram(col, row, maxval);
  getMaxMin(col, row);
  int i, j;

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      graymap[i * cols + j] = maxval * (graymap[i * cols + j] - min
    ) / (max-min);
    }
  }
}
```

List of Code 19: histogram_stretching

This line of code is for image improvement using histogram linear transformation or histogram stretching.

3.1.10 make_Fi

```
void make_Fi(int maxval){
  Fi = (int *) malloc(sizeof(int) * (maxval + 2));
  int temp = 0;
  for(int i = 0; i <= maxval; i++){
    temp += histogram[i];
    Fi[i] = temp;
  }
}
```

List of Code 20: make_Fi

This line of code is for creating the F(i) values or $\Sigma_{j=0}^{P_i} hist(j)$.

3.1.11 histogram_equalization

```
void histogram_equalization(int col, int row, int maxval){
  create_histogram(col, row, maxval);
  make_Fi(maxval);

  int total_pixel = row * col;
  int i, j;

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      graymap[i * cols + j] = maxval * Fi[graymap[i * cols + j]]/
    total_pixel;
    }
  }
}
```

List of Code 21: histogram_equalization

This line of code is for image improvement using histogram non linear transformation or histogram equalization.

3.2 Question

3.2.1 Exercise 1 : Filtering

- Write a c program that reads a PGM image and compute the image smoothed with the following binomial filter:

```c
#include <stdlib.h>
#include <stdio.h>
#include "Util.h"

int rows, cols;
gray *graymap;

int get_byte_by_pos(int col, int row){
  if(row < 0)
    row = 0;
  if(col < 0)
    col = 0;
  if(col >= cols)
    col = cols-1;
  if(row >= rows)
    row = rows-1;

  return graymap[row * cols + col];
}

int get_new_byte(int col, int row){
  int temp = 0;
  temp += get_byte_by_pos(col-1,row-1) * 1;
  temp += get_byte_by_pos(col-1,row) * 2;
  temp += get_byte_by_pos(col-1,row+1) * 1;

  temp += get_byte_by_pos(col,row-1) * 2;
  temp += get_byte_by_pos(col,row) * 4;
  temp += get_byte_by_pos(col,row+1) * 2;

  temp += get_byte_by_pos(col+1,row-1) * 1;
  temp += get_byte_by_pos(col+1,row) * 2;
  temp += get_byte_by_pos(col+1,row+1) * 1;

  return temp / 16;
}

void repeat_run(int total_runs){
  gray temp_graymap[cols * rows];
  int k;
  int i, j;
  for(k = 0; k < total_runs; k++){
    for(i=0; i < rows; i++){
      for(j=0; j < cols ; j++){
        temp_graymap[i * cols + j] = get_new_byte(j,i);
      }
    }
    for(i=0; i < rows; i++){
      for(j=0; j < cols ; j++){
        graymap[i * cols + j] = temp_graymap[i * cols + j];
      }
```

```
    }
  }
}

int main(int argc, char* argv[]){
  FILE* ifp;
  int ich1, ich2, maxval=255, pgmraw, total_runs;
  int i, j;

  /*
    Argument Handler
  */
  if ( argc != 3 ){
    printf("\nUsage: %s file repetition \n\n", argv[0]);
    exit(0);
  }

  ifp = fopen(argv[1],"r");
  if (ifp == NULL) {
    printf("error in opening file %s\n", argv[1]);
    exit(1);
  }

  total_runs = atoi(argv[2]);
  if(total_runs < 0){
    printf("Filter need to run at least once\n");
    exit(1);
  }

  ich1 = getc( ifp );
  if ( ich1 == EOF )
    pm_erreur( "EOF / read error / magic number" );
  ich2 = getc( ifp );
  if ( ich2 == EOF )
    pm_erreur( "EOF /read error / magic number" );
  if(ich2 != '2' && ich2 != '5')
    pm_erreur(" wrong file type ");
  else
    if(ich2 == '2')
      pgmraw = 0;
    else
      pgmraw = 1;

  cols = pm_getint( ifp );
  rows = pm_getint( ifp );
  maxval = pm_getint( ifp );

  graymap = (gray *) malloc(cols * rows * sizeof(gray));

  for(i=0; i < rows; i++)
    for(j=0; j < cols ; j++)
      if(pgmraw)
        graymap[i * cols + j] = pm_getrawbyte(ifp) ;
      else
        graymap[i * cols + j] = pm_getint(ifp);

  /*
```

```
    Processing
  */
  repeat_run(total_runs);

  if(pgmraw)
    printf("P2\n");
  else
    printf("P5\n");

  printf("%d %d \n", cols, rows);
  printf("%d\n",maxval);

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      if(pgmraw){;
        printf("%d ", graymap[i * cols + j]);
      }
      else{
        printf("%c", graymap[i * cols + j]);
      }
    }
  }

  fclose(ifp);
  return 0;
}
```

List of Code 22: filtering_3x3

- Add the option to smooth n times as well as the option to smooth with a filter of dimension 5x5 (binomial coefficients can be obtained from the pascal triangle).

```
#include <stdlib.h>
#include <stdio.h>
#include "Util.h"

int rows, cols;
gray *graymap;

int get_byte_by_pos(int col, int row){
  if(row < 0)
    row = 0;
  if(col < 0)
    col = 0;
  if(col >= cols)
    col = cols-1;
  if(row >= rows)
    row = rows-1;

  return graymap[row * cols + col];
}

int get_new_byte(int col, int row){
  int temp = 0;
  temp += get_byte_by_pos(col-2,row-2) * 1;
  temp += get_byte_by_pos(col-2,row-1) * 4;
```

```
  temp += get_byte_by_pos(col-2,row) * 6;
  temp += get_byte_by_pos(col-2,row+1) * 4;
  temp += get_byte_by_pos(col-2,row+2) * 1;

  temp += get_byte_by_pos(col-1,row-2) * 4;
  temp += get_byte_by_pos(col-1,row-1) * 16;
  temp += get_byte_by_pos(col-1,row) * 24;
  temp += get_byte_by_pos(col-1,row+1) * 16;
  temp += get_byte_by_pos(col-1,row+2) * 4;

  temp += get_byte_by_pos(col,row-2) * 6;
  temp += get_byte_by_pos(col,row-1) * 24;
  temp += get_byte_by_pos(col,row) * 36;
  temp += get_byte_by_pos(col,row+1) * 24;
  temp += get_byte_by_pos(col,row+2) * 6;

  temp += get_byte_by_pos(col+1,row-2) * 4;
  temp += get_byte_by_pos(col+1,row-1) * 16;
  temp += get_byte_by_pos(col+1,row) * 24;
  temp += get_byte_by_pos(col+1,row+1) * 16;
  temp += get_byte_by_pos(col+1,row+2) * 4;

  temp += get_byte_by_pos(col+2,row-2) * 1;
  temp += get_byte_by_pos(col+2,row-1) * 4;
  temp += get_byte_by_pos(col+2,row) * 6;
  temp += get_byte_by_pos(col+2,row+1) * 4;
  temp += get_byte_by_pos(col+2,row+2) * 1;

  return temp / 256;
}

void repeat_run(int total_runs){
  gray temp_graymap[cols * rows];
  int k;
  int i, j;
  for(k = 0; k < total_runs; k++){
    for(i=0; i < rows; i++){
      for(j=0; j < cols ; j++){
        temp_graymap[i * cols + j] = get_new_byte(j,i);
      }
    }
    for(i=0; i < rows; i++){
      for(j=0; j < cols ; j++){
        graymap[i * cols + j] = temp_graymap[i * cols + j];
      }
    }
  }
}

int main(int argc, char* argv[]){
  FILE* ifp;
  int ich1, ich2, maxval=255, pgmraw, total_runs;
  int i, j;

  /*
    Argument Handler
  */
```

```
82  if ( argc != 3 ){
83    printf("\nUsage: %s file repetition\n\n", argv[0]);
84    exit(0);
85  }
86
87  ifp = fopen(argv[1],"r");
88  if (ifp == NULL) {
89    printf("error in opening file %s\n", argv[1]);
90    exit(1);
91  }
92
93  total_runs = atoi(argv[2]);
94  if(total_runs < 0){
95    printf("Filter need to run at least once\n");
96    exit(1);
97  }
98
99  ich1 = getc( ifp );
100 if ( ich1 == EOF )
101   pm_erreur( "EOF / read error / magic number" );
102 ich2 = getc( ifp );
103 if ( ich2 == EOF )
104   pm_erreur( "EOF /read error / magic number" );
105 if(ich2 != '2' && ich2 != '5')
106   pm_erreur(" wrong file type ");
107 else
108   if(ich2 == '2')
109     pgmraw = 0;
110   else
111     pgmraw = 1;
112
113 cols = pm_getint( ifp );
114 rows = pm_getint( ifp );
115 maxval = pm_getint( ifp );
116
117 graymap = (gray *) malloc(cols * rows * sizeof(gray));
118
119 for(i=0; i < rows; i++)
120   for(j=0; j < cols ; j++)
121     if(pgmraw)
122       graymap[i * cols + j] = pm_getrawbyte(ifp) ;
123     else
124       graymap[i * cols + j] = pm_getint(ifp);
125
126 /*
127   Processing
128 */
129 repeat_run(total_runs);
130
131 if(pgmraw)
132   printf("P2\n");
133 else
134   printf("P5\n");
135
136 printf("%d %d \n", cols, rows);
137 printf("%d\n",maxval);
138
```

```
  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      if(pgmraw){;
        printf("%d ", graymap[i * cols + j]);
      }
      else{
        printf("%c", graymap[i * cols + j]);
      }
    }
  }

  fclose(ifp);
  return 0;
}
```

List of Code 23: filtering_5x5

- Implement a median filter and compare it with the binomial filters.

```
#include <stdlib.h>
#include <stdio.h>
#include "Util.h"

int rows, cols;
gray *graymap;

int get_byte_by_pos(int col, int row){
  if(row < 0)
    row = 0;
  if(col < 0)
    col = 0;
  if(col >= cols)
    col = cols-1;
  if(row >= rows)
    row = rows-1;

  return graymap[row * cols + col];
}

int get_new_byte(int col, int row, int size_filter){
  int temp[size_filter*size_filter];
  int i, j;
  for(i = -(size_filter/2); i <= (size_filter/2); i++){
    for(j= -(size_filter/2); j <= (size_filter/2); j++){
      temp[i * size_filter + j] = get_byte_by_pos(col-j, row-i
    );
    }
  }

  for(i = 1; i < size_filter*size_filter; i++){
    for(j=0; j < i ; j++){
      if(temp[j] > temp[j+1]){
        int x = temp[j];
        temp[j] = temp[j+1];
        temp[j+1] = x;
      }
    }
  }
```

```
39
40    return temp[size_filter*size_filter/2];
41  }
42
43  void repeat_run(int total_runs, int size_filter){
44    gray temp_graymap[cols * rows];
45    int k;
46    int i, j;
47    for(k = 0; k < total_runs; k++){
48      for(i=0; i < rows; i++){
49        for(j=0; j < cols ; j++){
50          temp_graymap[i * cols + j] = get_new_byte(j,i,
      size_filter);
51        }
52      }
53      for(i=0; i < rows; i++){
54        for(j=0; j < cols ; j++){
55          graymap[i * cols + j] = temp_graymap[i * cols + j];
56        }
57      }
58    }
59  }
60
61  int main(int argc, char* argv[]){
62    FILE* ifp;
63    int ich1, ich2, maxval=255, pgmraw, total_runs, size_filter;
64    int i, j;
65
66    /*
67      Argument Handler
68    */
69    if ( argc != 4 ){
70      printf("\nUsage: %s file \n\n", argv[0]);
71      exit(0);
72    }
73
74    ifp = fopen(argv[1],"r");
75    if (ifp == NULL) {
76      printf("error in opening file %s\n", argv[1]);
77      exit(1);
78    }
79
80    size_filter = atoi(argv[2]);
81    if(size_filter % 2 == 0 || size_filter <= 0){
82      printf("error in size of filter %s\n", argv[2]);
83      printf("expecting %s is greater than 0 and an odd number\n
      ", argv[3]);
84      exit(1);
85    }
86
87    total_runs = atoi(argv[3]);
88    if(total_runs < 0){
89      printf("Filter need to run at least once\n");
90      exit(1);
91    }
92
93    ich1 = getc( ifp );
```

```
  if ( ich1 == EOF )
    pm_erreur( "EOF / read error / magic number" );
  ich2 = getc( ifp );
  if ( ich2 == EOF )
    pm_erreur( "EOF /read error / magic number" );
  if(ich2 != '2' && ich2 != '5')
    pm_erreur(" wrong file type ");
  else
    if(ich2 == '2')
      pgmraw = 0;
    else
      pgmraw = 1;

  cols = pm_getint( ifp );
  rows = pm_getint( ifp );
  maxval = pm_getint( ifp );

  graymap = (gray *) malloc(cols * rows * sizeof(gray));

  for(i=0; i < rows; i++)
    for(j=0; j < cols ; j++)
      if(pgmraw)
        graymap[i * cols + j] = pm_getrawbyte(ifp) ;
      else
        graymap[i * cols + j] = pm_getint(ifp);

  /*
    Processing
  */
  repeat_run(total_runs, size_filter);

  if(pgmraw)
    printf("P2\n");
  else
    printf("P5\n");

  printf("%d %d \n", cols, rows);
  printf("%d\n",maxval);

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      if(pgmraw){;
        printf("%d ", graymap[i * cols + j]);
      }
      else{
        printf("%c", graymap[i * cols + j]);
      }
    }
  }

  fclose(ifp);
  return 0;
}
```

List of Code 24: filter_median

Bonus modified filtering universal

```c
#include <stdlib.h>
#include <stdio.h>
#include "Util.h"

int rows, cols;
gray *graymap;
int *filter;
int divider;

int get_byte_by_pos(int col, int row){
  if(row < 0)
    row = 0;
  if(col < 0)
    col = 0;
  if(col >= cols)
    col = cols-1;
  if(row >= rows)
    row = rows-1;

  return graymap[row * cols + col];
}

/*
Create Pascal triangle
Courtesy: https://www.faceprep.in/c/program-to-print-pascals-triangle/
*/
void createFilter(int filter_size){
  int coef, i, j, temp_pascal[filter_size+2];
  divider = 0;
  for(i = 0; i<= filter_size; i++){
    if(i==0)
      coef = 1;
    else
      coef = coef * (filter_size-i)/i;

    temp_pascal[i] = coef;
  }

  filter = malloc(sizeof(int) * filter_size * filter_size);
  for(i = 0; i < filter_size; i++){
    for(j = 0; j < filter_size; j++){
      int x = temp_pascal[i] * temp_pascal[j];
      filter[i*filter_size+j] = x;
      divider += x;
    }
  }
}


int get_new_byte(int col, int row, int filter_size){
  int temp = 0;
  int start_col = col - (filter_size/2);
  int start_row = row - (filter_size/2);
  for(int i = 0; i < filter_size; i++){
    for(int j = 0; j < filter_size; j++){
      temp += get_byte_by_pos(start_col + i, start_row + j) *
```

```
    filter[i *filter_size + j];
    }
  }

  return temp / divider;
}

void repeat_run(int total_runs, int filter_size){
  gray temp_graymap[cols * rows];
  int k;
  int i, j;
  for(k = 0; k < total_runs; k++){
    for(i=0; i < rows; i++){
      for(j=0; j < cols ; j++){
        temp_graymap[i * cols + j] = get_new_byte(j,i,
    filter_size);
      }
    }
    for(i=0; i < rows; i++){
      for(j=0; j < cols ; j++){
        graymap[i * cols + j] = temp_graymap[i * cols + j];
      }
    }
  }
}

int main(int argc, char* argv[]){
  FILE* ifp;
  int ich1, ich2, maxval=255, pgmraw, filter_size, total_runs;
  int i, j;

  /*
    Argument Handler
  */
  if ( argc != 4 ){
    printf("\nUsage: %s file filter_size repetition \n\n",
    argv[0]);
    exit(0);
  }

  ifp = fopen(argv[1],"r");
  if (ifp == NULL) {
    printf("error in opening file %s\n", argv[1]);
    exit(1);
  }

  filter_size = atoi(argv[2]);
  if(filter_size < 0 || filter_size %2 != 1){
    printf("error in size of filter %s\n", argv[2]);
    printf("expecting %s is greater than 0 and an odd number\n
    ", argv[3]);
    exit(1);
  }

  total_runs = atoi(argv[3]);
  if(total_runs < 0){
    printf("Filter need to run at least once\n");
```

```
110     exit(1);
111   }
112
113   ich1 = getc( ifp );
114   if ( ich1 == EOF )
115     pm_erreur( "EOF / read error / magic number" );
116   ich2 = getc( ifp );
117   if ( ich2 == EOF )
118     pm_erreur( "EOF /read error / magic number" );
119   if(ich2 != '2' && ich2 != '5')
120     pm_erreur(" wrong file type ");
121   else
122     if(ich2 == '2')
123       pgmraw = 0;
124     else
125       pgmraw = 1;
126
127   cols = pm_getint( ifp );
128   rows = pm_getint( ifp );
129   maxval = pm_getint( ifp );
130
131   graymap = (gray *) malloc(cols * rows * sizeof(gray));
132
133   for(i=0; i < rows; i++)
134     for(j=0; j < cols ; j++)
135       if(pgmraw)
136         graymap[i * cols + j] = pm_getrawbyte(ifp) ;
137       else
138         graymap[i * cols + j] = pm_getint(ifp);
139
140   /*
141     Processing
142   */
143   createFilter(filter_size);
144   repeat_run(total_runs, filter_size);
145
146   if(pgmraw)
147     printf("P2\n");
148   else
149     printf("P5\n");
150
151   printf("%d %d \n", cols, rows);
152   printf("%d\n",maxval);
153
154   for(i=0; i < rows; i++){
155     for(j=0; j < cols ; j++){
156       if(pgmraw){;
157         printf("%d ", graymap[i * cols + j]);
158       }
159       else{
160         printf("%c", graymap[i * cols + j]);
161       }
162     }
163   }
164
165   fclose(ifp);
166   return 0;
```

```
}
```

List of Code 25: filtering

3.2.2 Exercise 2 : Histograms

- Add to the previous program a function that computes the intensity histogram. This histogram will be represented by an array of dimension the maximal number of grayscale intensities (i.e. 255).

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "Util.h"

int rows, cols;
gray *graymap;
int *filter;
int divider;
int *histogram;
int *Fi;
int max = __INT_MAX__ * -1;
int min = __INT_MAX__;

int get_byte_by_pos(int col, int row){
  if(row < 0)
    row = 0;
  if(col < 0)
    col = 0;
  if(col >= cols)
    col = cols-1;
  if(row >= rows)
    row = rows-1;

  return graymap[row * cols + col];
}

void clear_histogram(int maxval){
  for(int i = 0; i <= maxval; i++){
    histogram[i] = 0;
  }
}

void create_histogram(int col, int row, int maxval){
  clear_histogram(maxval);
  int i,j;

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      histogram[graymap[i * cols + j]] ++;
    }
  }
}
/*
Create Pascal triangle
Courtesy: https://www.faceprep.in/c/program-to-print-pascals-
    triangle/
```

```
*/
void createFilter(int filter_size){
  int coef, i, j, temp_pascal[filter_size+2];
  divider = 0;
  for(i = 0; i<= filter_size; i++){
    if(i==0)
      coef = 1;
    else
      coef = coef * (filter_size-i)/i;

    temp_pascal[i] = coef;
  }

  filter = malloc(sizeof(int) * filter_size * filter_size);
  for(i = 0; i < filter_size; i++){
    for(j = 0; j < filter_size; j++){
      int x = temp_pascal[i] * temp_pascal[j];
      filter[i*filter_size+j] = x;
      divider += x;
    }
  }
}


int get_new_byte(int col, int row, int filter_size){
  int temp = 0;
  int start_col = col - (filter_size/2);
  int start_row = row - (filter_size/2);
  for(int i = 0; i < filter_size; i++){
    for(int j = 0; j < filter_size; j++){
      temp += get_byte_by_pos(start_col + i, start_row + j) *
    filter[i *filter_size + j];
    }
  }

  return temp / divider;
}

void repeat_run(int total_runs, int filter_size){
  gray temp_graymap[cols * rows];
  int k;
  int i, j;
  for(k = 0; k < total_runs; k++){
    for(i=0; i < rows; i++){
      for(j=0; j < cols ; j++){
        temp_graymap[i * cols + j] = get_new_byte(j,i,
    filter_size);
      }
    }
    for(i=0; i < rows; i++){
      for(j=0; j < cols ; j++){
        graymap[i * cols + j] = temp_graymap[i * cols + j];
      }
    }
  }
}

```

```
int main(int argc, char* argv[]){
  FILE* ifp;
  int ich1, ich2, maxval=255, pgmraw, filter_size, total_runs;
  int i, j;
  char *mode;

  /*
    Argument Handler
  */
  if ( argc != 5 ){
    printf("\nUsage: %s file filter_size repetition mode\n\n",
     argv[0]);
    exit(0);
  }

  ifp = fopen(argv[1],"r");
  if (ifp == NULL) {
    printf("error in opening file %s\n", argv[1]);
    exit(1);
  }

  filter_size = atoi(argv[2]);
  if(filter_size < 0 || filter_size %2 != 1){
    printf("error in size of filter %s\n", argv[2]);
    printf("expecting %s is greater than 0 and an odd number\n
    ", argv[3]);
    exit(1);
  }

  total_runs = atoi(argv[3]);
  if(total_runs < 0){
    printf("Filter need to run at least once\n");
    exit(1);
  }

  mode = argv[4];
  if(strcmp(mode, "e") != 0 && strcmp(mode, "s") != 0){
    printf("Filter mode must be s[Stretching] or e[
    Equalization]\n");
    exit(1);
  }

  ich1 = getc( ifp );
  if ( ich1 == EOF )
    pm_erreur( "EOF / read error / magic number" );
  ich2 = getc( ifp );
  if ( ich2 == EOF )
    pm_erreur( "EOF /read error / magic number" );
  if(ich2 != '2' && ich2 != '5')
    pm_erreur(" wrong file type ");
  else
    if(ich2 == '2')
      pgmraw = 0;
    else
      pgmraw = 1;

  cols = pm_getint( ifp );
```

```
  rows = pm_getint( ifp );
  maxval = pm_getint( ifp );

  graymap = (gray *) malloc(cols * rows * sizeof(gray));
  histogram = (int *) malloc((maxval+1) * sizeof(int));

  for(i=0; i < rows; i++)
    for(j=0; j < cols ; j++)
      if(pgmraw)
        graymap[i * cols + j] = pm_getrawbyte(ifp) ;
      else
        graymap[i * cols + j] = pm_getint(ifp);

  /*
    Processing
  */
  createFilter(filter_size);
  repeat_run(total_runs, filter_size);

  if(pgmraw)
    printf("P2\n");
  else
    printf("P5\n");

  printf("%d %d \n", cols, rows);
  printf("%d\n",maxval);

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      if(pgmraw){;
        printf("%d ", graymap[i * cols + j]);
      }
      else{
        printf("%c", graymap[i * cols + j]);
      }
    }
  }

  fclose(ifp);
  return 0;
```

List of Code 26: histogram

- Modify the program so that it computes the image transformed with a histogram stretching.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "Util.h"

int rows, cols;
gray *graymap;
int *filter;
int divider;
int *histogram;
int *Fi;
```

```
12 int max = __INT_MAX__ * -1;
13 int min = __INT_MAX__;
14
15 int get_byte_by_pos(int col, int row){
16   if(row < 0)
17     row = 0;
18   if(col < 0)
19     col = 0;
20   if(col >= cols)
21     col = cols-1;
22   if(row >= rows)
23     row = rows-1;
24
25   return graymap[row * cols + col];
26 }
27
28 void clear_histogram(int maxval){
29   for(int i = 0; i <= maxval; i++){
30     histogram[i] = 0;
31   }
32 }
33
34 void create_histogram(int col, int row, int maxval){
35   clear_histogram(maxval);
36   int i,j;
37
38   for(i=0; i < rows; i++){
39     for(j=0; j < cols ; j++){
40       histogram[graymap[i * cols + j]] ++;
41     }
42   }
43 }
44
45 void getMaxMin(int col, int row){
46   max = __INT_MAX__ * -1;
47   min = __INT_MAX__;
48   int i,j;
49
50   for(i=0; i < rows; i++){
51     for(j=0; j < cols ; j++){
52       int x = graymap[i * cols + j];
53       if(max < x)
54         max = x;
55       if(min > x)
56         min = x;
57     }
58   }
59 }
60
61 void histogram_stretching(int col, int row, int maxval){
62   create_histogram(col, row, maxval);
63   getMaxMin(col, row);
64   int i, j;
65
66   for(i=0; i < rows; i++){
67     for(j=0; j < cols ; j++){
68       graymap[i * cols + j] = maxval * (graymap[i * cols + j]
```

```
    - min) / (max-min);
    }
  }
}
/*
Create Pascal triangle
Courtesy: https://www.faceprep.in/c/program-to-print-pascals-triangle/
*/
void createFilter(int filter_size){
  int coef, i, j, temp_pascal[filter_size+2];
  divider = 0;
  for(i = 0; i<= filter_size; i++){
    if(i==0)
      coef = 1;
    else
      coef = coef * (filter_size-i)/i;

    temp_pascal[i] = coef;
  }

  filter = malloc(sizeof(int) * filter_size * filter_size);
  for(i = 0; i < filter_size; i++){
    for(j = 0; j < filter_size; j++){
      int x = temp_pascal[i] * temp_pascal[j];
      filter[i*filter_size+j] = x;
      divider += x;
    }
  }
}


int get_new_byte(int col, int row, int filter_size){
  int temp = 0;
  int start_col = col - (filter_size/2);
  int start_row = row - (filter_size/2);
  for(int i = 0; i < filter_size; i++){
    for(int j = 0; j < filter_size; j++){
      temp += get_byte_by_pos(start_col + i, start_row + j) *
    filter[i *filter_size + j];
    }
  }

  return temp / divider;
}

void repeat_run(int total_runs, int filter_size){
  gray temp_graymap[cols * rows];
  int k;
  int i, j;
  for(k = 0; k < total_runs; k++){
    for(i=0; i < rows; i++){
      for(j=0; j < cols ; j++){
        temp_graymap[i * cols + j] = get_new_byte(j,i,
    filter_size);
      }
    }
```

```
122     for(i=0; i < rows; i++){
123       for(j=0; j < cols ; j++){
124         graymap[i * cols + j] = temp_graymap[i * cols + j];
125       }
126     }
127   }
128 }
129
130 int main(int argc, char* argv[]){
131   FILE* ifp;
132   int ich1, ich2, maxval=255, pgmraw, filter_size, total_runs;
133   int i, j;
134   char *mode;
135
136   /*
137     Argument Handler
138   */
139   if ( argc != 5 ){
140     printf("\nUsage: %s file filter_size repetition mode\n\n",
       argv[0]);
141     exit(0);
142   }
143
144   ifp = fopen(argv[1],"r");
145   if (ifp == NULL) {
146     printf("error in opening file %s\n", argv[1]);
147     exit(1);
148   }
149
150   filter_size = atoi(argv[2]);
151   if(filter_size < 0 || filter_size %2 != 1){
152     printf("error in size of filter %s\n", argv[2]);
153     printf("expecting %s is greater than 0 and an odd number\n
      ", argv[3]);
154     exit(1);
155   }
156
157   total_runs = atoi(argv[3]);
158   if(total_runs < 0){
159     printf("Filter need to run at least once\n");
160     exit(1);
161   }
162
163   mode = argv[4];
164   if(strcmp(mode, "e") != 0 && strcmp(mode, "s") != 0){
165     printf("Filter mode must be s[Stretching] or e[
      Equalization]\n");
166     exit(1);
167   }
168
169   ich1 = getc( ifp );
170   if ( ich1 == EOF )
171     pm_erreur( "EOF / read error / magic number" );
172   ich2 = getc( ifp );
173   if ( ich2 == EOF )
174     pm_erreur( "EOF /read error / magic number" );
175   if(ich2 != '2' && ich2 != '5')
```

```
    pm_erreur(" wrong file type ");
  else
    if(ich2 == '2')
      pgmraw = 0;
    else
      pgmraw = 1;

  cols = pm_getint( ifp );
  rows = pm_getint( ifp );
  maxval = pm_getint( ifp );

  graymap = (gray *) malloc(cols * rows * sizeof(gray));
  histogram = (int *) malloc((maxval+1) * sizeof(int));

  for(i=0; i < rows; i++)
    for(j=0; j < cols ; j++)
      if(pgmraw)
        graymap[i * cols + j] = pm_getrawbyte(ifp) ;
      else
        graymap[i * cols + j] = pm_getint(ifp);

  /*
    Processing
  */
  createFilter(filter_size);
  repeat_run(total_runs, filter_size);

  if(strcmp(mode, "s") == 0){
    histogram_stretching(cols, rows, maxval);
  }

  if(pgmraw)
    printf("P2\n");
  else
    printf("P5\n");

  printf("%d %d \n", cols, rows);
  printf("%d\n",maxval);

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      if(pgmraw){;
        printf("%d ", graymap[i * cols + j]);
      }
      else{
        printf("%c", graymap[i * cols + j]);
      }
    }
  }

  fclose(ifp);
  return 0;
```

List of Code 27: histogram_stretching

- Modify the program so that it computes the image transformed with a histogram equalization.

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "Util.h"

int rows, cols;
gray *graymap;
int *filter;
int divider;
int *histogram;
int *Fi;
int max = __INT_MAX__ * -1;
int min = __INT_MAX__;

int get_byte_by_pos(int col, int row){
  if(row < 0)
    row = 0;
  if(col < 0)
    col = 0;
  if(col >= cols)
    col = cols-1;
  if(row >= rows)
    row = rows-1;

  return graymap[row * cols + col];
}

void clear_histogram(int maxval){
  for(int i = 0; i <= maxval; i++){
    histogram[i] = 0;
  }
}

void create_histogram(int col, int row, int maxval){
  clear_histogram(maxval);
  int i,j;

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      histogram[graymap[i * cols + j]] ++;
    }
  }
}

void getMaxMin(int col, int row){
  max = __INT_MAX__ * -1;
  min = __INT_MAX__;
  int i,j;

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      int x = graymap[i * cols + j];
      if(max < x)
        max = x;
      if(min > x)
        min = x;
    }
```

```
  }
}

void histogram_stretching(int col, int row, int maxval){
  create_histogram(col, row, maxval);
  getMaxMin(col, row);
  int i, j;

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      graymap[i * cols + j] = maxval * (graymap[i * cols + j]
    - min) / (max-min);
    }
  }
}

void make_Fi(int maxval){
  Fi = (int *) malloc(sizeof(int) * (maxval + 2));
  int temp = 0;
  for(int i = 0; i <= maxval; i++){
    temp += histogram[i];
    Fi[i] = temp;
  }
}

void histogram_equalization(int col, int row, int maxval){
  create_histogram(col, row, maxval);
  make_Fi(maxval);

  int total_pixel = row * col;
  int i, j;

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      graymap[i * cols + j] = maxval * Fi[graymap[i * cols + j
    ]]/total_pixel;
    }
  }
}

/*
Create Pascal triangle
Courtesy: https://www.faceprep.in/c/program-to-print-pascals-
    triangle/
*/
void createFilter(int filter_size){
  int coef, i, j, temp_pascal[filter_size+2];
  divider = 0;
  for(i = 0; i<= filter_size; i++){
    if(i==0)
      coef = 1;
    else
      coef = coef * (filter_size-i)/i;

    temp_pascal[i] = coef;
  }

```

```
112   filter = malloc(sizeof(int) * filter_size * filter_size);
113   for(i = 0; i < filter_size; i++){
114     for(j = 0; j < filter_size; j++){
115       int x = temp_pascal[i] * temp_pascal[j];
116       filter[i*filter_size+j] = x;
117       divider += x;
118     }
119   }
120 }
121
122
123 int get_new_byte(int col, int row, int filter_size){
124   int temp = 0;
125   int start_col = col - (filter_size/2);
126   int start_row = row - (filter_size/2);
127   for(int i = 0; i < filter_size; i++){
128     for(int j = 0; j < filter_size; j++){
129       temp += get_byte_by_pos(start_col + i, start_row + j) *
      filter[i *filter_size + j];
130     }
131   }
132
133   return temp / divider;
134 }
135
136 void repeat_run(int total_runs, int filter_size){
137   gray temp_graymap[cols * rows];
138   int k;
139   int i, j;
140   for(k = 0; k < total_runs; k++){
141     for(i=0; i < rows; i++){
142       for(j=0; j < cols ; j++){
143         temp_graymap[i * cols + j] = get_new_byte(j,i,
      filter_size);
144       }
145     }
146     for(i=0; i < rows; i++){
147       for(j=0; j < cols ; j++){
148         graymap[i * cols + j] = temp_graymap[i * cols + j];
149       }
150     }
151   }
152 }
153
154 int main(int argc, char* argv[]){
155   FILE* ifp;
156   int ich1, ich2, maxval=255, pgmraw, filter_size, total_runs;
157   int i, j;
158   char *mode;
159
160   /*
161     Argument Handler
162   */
163   if ( argc != 5 ){
164     printf("\nUsage: %s file filter_size repetition mode\n\n",
       argv[0]);
165     exit(0);
```

```
166   }
167
168   ifp = fopen(argv[1],"r");
169   if (ifp == NULL) {
170     printf("error in opening file %s\n", argv[1]);
171     exit(1);
172   }
173
174   filter_size = atoi(argv[2]);
175   if(filter_size < 0 || filter_size %2 != 1){
176     printf("error in size of filter %s\n", argv[2]);
177     printf("expecting %s is greater than 0 and an odd number\n
      ", argv[3]);
178     exit(1);
179   }
180
181   total_runs = atoi(argv[3]);
182   if(total_runs < 0){
183     printf("Filter need to run at least once\n");
184     exit(1);
185   }
186
187   mode = argv[4];
188   if(strcmp(mode, "e") != 0 && strcmp(mode, "s") != 0){
189     printf("Filter mode must be s[Stretching] or e[
      Equalization]\n");
190     exit(1);
191   }
192
193   ich1 = getc( ifp );
194   if ( ich1 == EOF )
195     pm_erreur( "EOF / read error / magic number" );
196   ich2 = getc( ifp );
197   if ( ich2 == EOF )
198     pm_erreur( "EOF /read error / magic number" );
199   if(ich2 != '2' && ich2 != '5')
200     pm_erreur(" wrong file type ");
201   else
202     if(ich2 == '2')
203       pgmraw = 0;
204     else
205       pgmraw = 1;
206
207   cols = pm_getint( ifp );
208   rows = pm_getint( ifp );
209   maxval = pm_getint( ifp );
210
211   graymap = (gray *) malloc(cols * rows * sizeof(gray));
212   histogram = (int *) malloc((maxval+1) * sizeof(int));
213
214   for(i=0; i < rows; i++)
215     for(j=0; j < cols ; j++)
216       if(pgmraw)
217         graymap[i * cols + j] = pm_getrawbyte(ifp) ;
218       else
219         graymap[i * cols + j] = pm_getint(ifp);
220
```

```
  /*
    Processing
  */
  createFilter(filter_size);
  repeat_run(total_runs, filter_size);

  if(strcmp(mode, "s") == 0){
    histogram_stretching(cols, rows, maxval);
  }
  else if(strcmp(mode, "e") == 0){
    histogram_equalization(cols, rows, maxval);
  }

  if(pgmraw)
    printf("P2\n");
  else
    printf("P5\n");

  printf("%d %d \n", cols, rows);
  printf("%d\n",maxval);

  for(i=0; i < rows; i++){
    for(j=0; j < cols ; j++){
      if(pgmraw){;
        printf("%d ", graymap[i * cols + j]);
      }
      else{
        printf("%c", graymap[i * cols + j]);
      }
    }
  }

  fclose(ifp);
  return 0;
}
```

List of Code 28: histogram_equation

3.3 Result

3.3.1 Local Filters

In this section, we find the result of applying local filters on images. First, the 3x3 binomial filter is applied on the default image.

Binomial Filters

Figure 4: Applying a 3x3 binomfial filter 1 time. (boat_3_1.png)

You can create the image using following code.

```
./filtering_3x3 boat.pgm>boat_3_1.png 1
```

List of Code 29: Make Figure boat_3_1.png

or

```
./filtering boat.pgm>boat_3_1.png 3 1
```

List of Code 30: Make Figure boat_3_1.png

Figure 5: Applying a 3x3 binomfial filter 5 times. (boat_3_5.png)

You can create the image using following code.

```
./filtering_3x3 boat.pgm>boat_3_5.png 5
```

List of Code 31: Make Figure boat_3_5.png

or

```
./filtering boat.pgm>boat_3_5.png 3 5
```

List of Code 32: Make Figure boat_3_5.png

From the images above, we can observe that the boat image gets blurred over a 3x3 binomial filter.

Figure 6: Applying a 5x5 binomfial filter 1 time. (boat_5_1.png)

You can create the image using following code.

```
./filtering_5x5 boat.pgm>boat_5_1.png 1
```

List of Code 33: Make Figure boat_5_1.png

or

```
./filtering boat.pgm>boat_5_1.png 5 1
```

List of Code 34: Make Figure boat_5_1.png

Figure 7: Applying the 5x5 binomfial filter 5 times. (boat_5_5.png)

You can create the image using following code.

```
./filtering_5x5 boat.pgm>boat_5_5.png 5
```

List of Code 35: Make Figure boat_5_5.png

or

```
./filtering boat.pgm>boat_5_5.png 5 5
```

List of Code 36: Make Figure boat_5_5.png

After applying the 5x5 binomial filter, it can be noticed that blurring is stronger than the 3x3 binomial. We can see from the result images that the text ("ZENTIM") on the boat tend to be become unreadable after applying binomial filter. And, the corner details vanish after the filter.

Median Filter

Figure 8: Applying a 3x3 median filter 1 time. (boat_median_3_1.png)

You can create the image using following code.

```
./filter_median boat.pgm>boat_median_3_1.png 3 1
```

List of Code 37: Make Figure boat_median_3_1.png

Figure 9: Applying a 3x3 median filter 5 times. (boat_median_3_5.png)

You can create the image using following code.

```
./filter_median boat.pgm>boat_median_3_5.png 3 5
```

List of Code 38: Make Figure boat_median_3_5.png

Figure 10: Applying a 5x5 median filter 1 times. (boat_median_5_1.png)

You can create the image using following code.

```
./filter_median boat.pgm>boat_median_5_1.png 5 1
```

List of Code 39: Make Figure boat_median_5_1.png

Figure 11: Applying a 5x5 median filter 3 times. (boat_median_5_3.png)

You can create the image using following code.

```
./filter_median boat.pgm>boat_median_5_3.png 5 3
```

List of Code 40: Make Figure boat_median_5_3.png

From the result above, we can say that binomial and median filters are useful when removing noise from images. But, it should be noted that applying too much filter can lead to blur. In other words, filters may lose detailed information of the image, since the filters are replacing the value with neighbouring values.

3.3.2 Histogram Modifications

As it has been described on the task description, we have applied histogram modifications on images after binomial and median filter.

Histogram Stretching

Figure 12: Applying histogram stretching on the image after filtering with 3x3 binomial 1 time. (boat_histogram_stretching_3_1.png)

You can create the image using following code.

```
./histogram boat.pgm>boat_histogram_stretching_3_1.png 3 1 s
```

List of Code 41: Make Figure boat_histogram_stretching_3_1.png

Figure 13: Applying histogram stretching on the image after filtering with 3x3 binomial 5 times. (boat_histogram_stretching_3_5.png)

You can create the image using following code.

```
./histogram boat.pgm>boat_histogram_stretching_3_5.png 3 5 s
```

List of Code 42: Make Figure boat_histogram_stretching_3_5.png

Figure 14: Applying histogram stretching on the image after filtering with 5x5 binomial 1 time. (boat_histogram_stretching_5_1.png)

You can create the image using following code.

```
./histogram boat.pgm>boat_histogram_stretching_5_1.png 5 1 s
```

List of Code 43: Make Figure boat_histogram_stretching_5_1.png

Figure 15: Applying histogram stretching on the image after filtering with 5x5 binomial 5 times. (boat_histogram_stretching_5_5.png)

You can create the image using following code.

```
./histogram boat.pgm>boat_histogram_stretching_5_5.png 5 5 s
```

List of Code 44: Make Figure boat_histogram_stretching_5_5.png

First, we applied 3x3 and 5x5 binomial filters on the boat image. Then, applied histogram stretching on the filtered images. And, there is a slight difference between the input filtered images and the images after stretching.

Histogram Equalization

Figure 16: Applying histogram equalization on the image after filtering with 3x3 binomial 1 time. (boat_histogram_equalization_3_1.png)

You can create the image using following code.

```
./histogram boat.pgm>boat_histogram_equalization_3_1.png 3 1 e
```

List of Code 45: Make Figure boat_histogram_equalization_3_1.png

Figure 17: Applying histogram equalization on the image after filtering with 3x3 binomial 5 times. (boat_histogram_equalization_3_5.png)

You can create the image using following code.

```
./histogram boat.pgm>boat_histogram_equalization_3_5.png 3 5 e
```

List of Code 46: Make Figure boat_histogram_equalization_3_5.png

Figure 18: Applying histogram equalization on the image after filtering with 5x5 binomial 1 time. (boat_histogram_equalization_5_1.png)

You can create the image using following code.

```
./histogram boat.pgm>boat_histogram_equalization_5_1.png 5 1 e
```

List of Code 47: Make Figure boat_histogram_equalization_5_1.png

Figure 19: Applying histogram equalization on the image after filtering with 5x5 binomial 5 times. (boat_histogram_equalization_5_5.png)

You can create the image using following code.

```
./histogram boat.pgm>boat_histogram_equalization_5_5.png 5 5 e
```

List of Code 48: Make Figure boat_histogram_equalization_5_5.png

In addition to the histogram stretching, we have also applied histogram equalization on the images after binomial filters. We can see that after equalization, the images became darker, and details on the image (such as items on the boat) became hard to be distinguished.