Image Filters
(Noise removal in Image processing)

Contents

Order Statistics Filters 2

FIRST: Alpha-trim filter 2

SECOND: Adaptive Median Filter 4

Implementation steps of Adaptive Median filter on Image 6

Project Requirements 8

FIRST: alpha-trim filter with TWO different algorithms: 8

SECOND: adaptive median filter with TWO different algorithms: 8

THIRD: timing graphs 8

How to calculate execution time? 8

How to draw the graph? 8

Given 9

Input 9

Deliverables 9

Implementation 9

Documentation 10

Grades 10

BONUS TASK 10

Order Statistics Filters

In image processing, filter is usually necessary to perform a high degree of noise reduction in an image before performing higher-level processing steps. The **order statistics filter** is a non-linear digital filter technique, often used to remove speckle (salt and pepper) noise from images. We target two common filters in this project:

1. Alpha-trim filter
2. Adaptive median filter

The main idea of both filters is to sort the pixel values in a neighborhood region with certain window size and then chose/calculate the single value from them and places it in the center of the window in a new image, see figure 1. This process is repeated for all pixels in the original image.

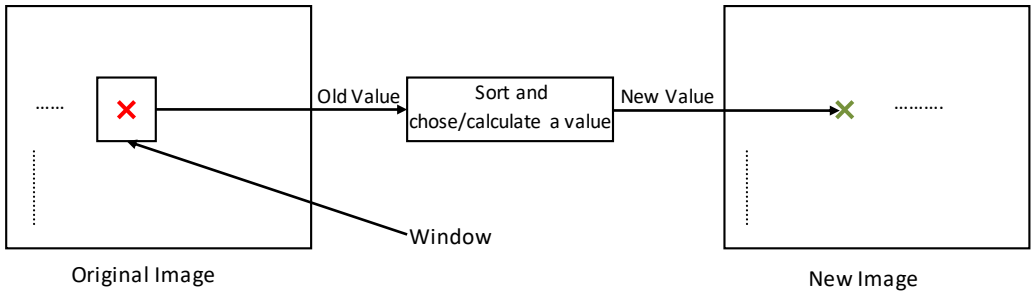


Figure 1: Main idea of order-statistics filters

As the window size increased, the effect of the filter is increased, as shown in figure 3.

FIRST: Alpha-trim filter

The idea is to calculate the average of some neighboring pixels' values after trimming out (excluding) the smallest ***T*** pixels and largest ***T*** pixels. This can be done by repeating the following steps for **each pixel** in the image:

1. Store the values of the neighboring pixels in an array. The array is called the window, and it should be odd sized.
2. Sort the values in the window in ascending order.
3. Exclude the first ***T*** values (smallest) and the last ***T*** values (largest) from the array.
4. Calculate the **average** of the remaining values as the new pixel value and place it in the center of the window in the new image, see figure 1.

This filter is usually used to remove both salt & pepper noise and random noise. See figure 2

Notes

- We work on gray-level images. So, each pixel has a value ranged from 0 to 255. Where 0 is the black pixel and 255 is the white pixel.

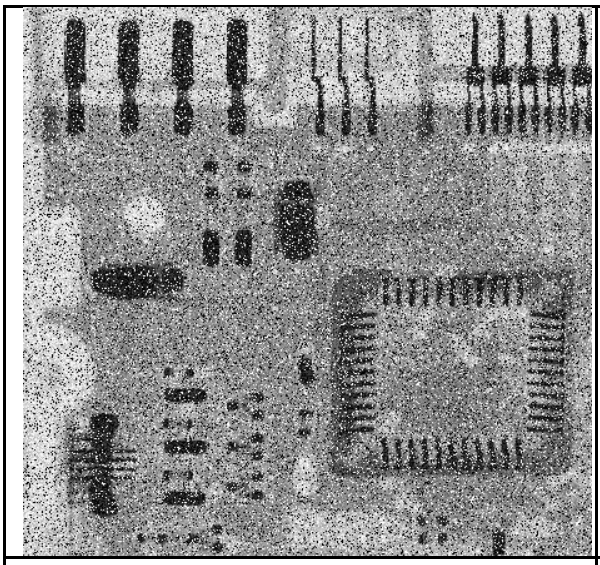
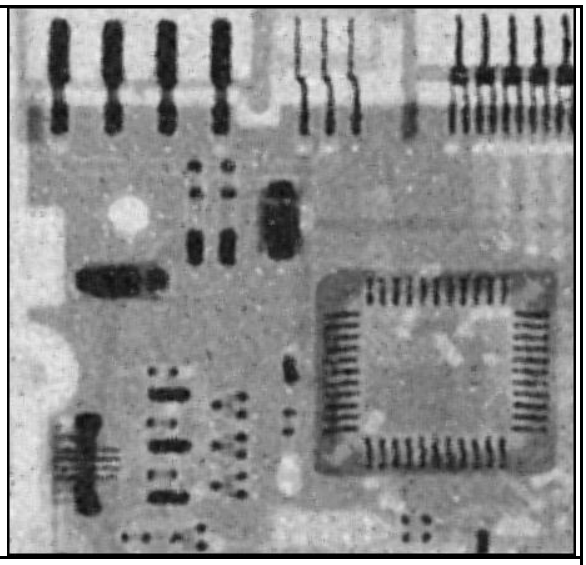


Image with uniform and salt & pepper noises



Alpha-trim filter 5x5 and T = 5

Figure 2: Effect of the alpha-trim filter with window size = 5x5 and trim value T = 5

Image with salt & pepper noise

Median filter with window 3×3

Median filter with window 5×5

Median filter with window 7×7

Figure 3: Effect of the standard median filter with different window size

SECOND: Adaptive Median Filter

The idea of the **standard median filter** is similar to alpha-trim filter but instead we calculate the median of neighboring pixels' values (middle value in the window array after sorting).

It's usually used to remove the salt and pepper noise, see figure 3.

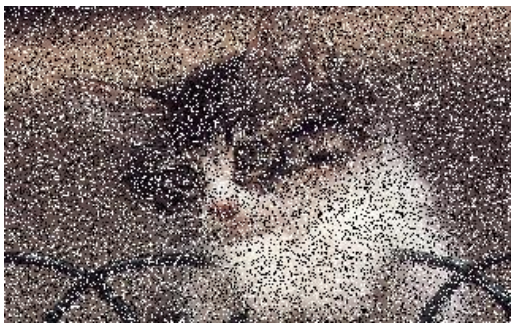
However, the standard median filter has the following drawbacks:

1. It fails to remove salt and pepper noise with large percentage (greater than 20%) without causing distortion in the original image.
2. It usually has a side-effect on the original image especially when it's applied with large mask size, see figure 2 with window 7×7.

Adaptive median filter is designed to handle these drawbacks by:

1. Seeking a median value that's not either salt or pepper noise by increasing the window size until reaching such median.
2. Replace the noise pixels only. (i.e. if the pixel is not a salt or a pepper, then leave it).

This is clear in figure 4 and figure 5. Compare the effect of both filters in each case. Note that both can remove the noise, but adaptive filter don't cause large distortion on the original image as the standard filter do.

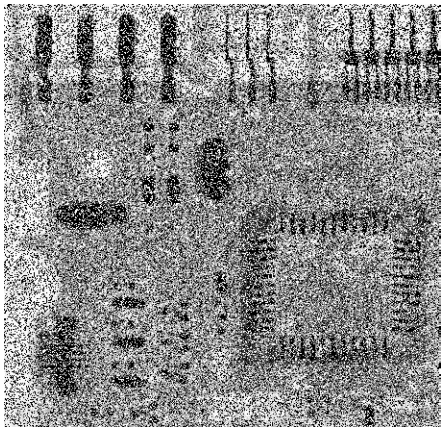


Original image corrupted with salt and pepper noise with percentage ≈ 30%

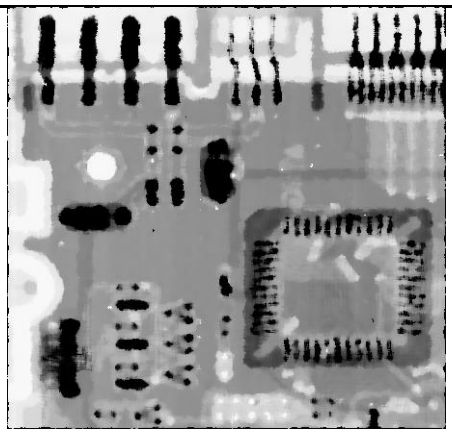
Standard median filter with window 7×7

Adaptive median with max window 7×7

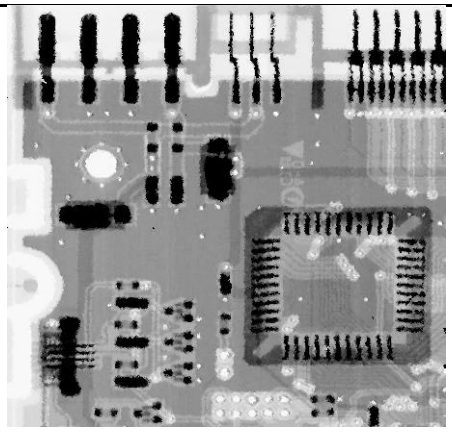
Figure 4: Effect of adaptive vs. standard median filter on small percentage of salt and pepper noise



Original image corrupted with salt and pepper noise with percentage ≈ 50%



Standard median filter with window 7×7



Adaptive median with max window 7×7

Figure 5 Effect of adaptive vs. standard median filter on large percentage of salt and pepper noise

Implementation steps of Adaptive Median filter on Image

Adaptive median filter has variable window size $\mathbf{W_S}$, and the procedure of updating the pixel value is as follows:

For each pixel in the image:

Try window sizes ranging from 3×3 to $W_S \times W_S$, where W_S is the maximum window size entered by the user, as follows:

Step 0: Start by window size 3×3

Step 1: Chose a non-noise median value

Sort the current window, and denote the following:

1. Z_{xy} is the gray value of the current pixel value at location (x, y)
2. Z_{max} is the maximum gray value in the window.
3. Z_{min} is the minimum gray value in the window.
4. Z_{med} is the median gray value in the window.

$A1 = Z_{med} - Z_{min}$
$A2 = Z_{max} - Z_{med}$
If $A1 > 0$ and $A2 > 0$ then we found a non-noise median
    Go to Step 2
Else
    Increase window size by 2
    If new window size ≤ W_S then
        Repeat Step 1 again
    Else
        NewPixelVal = Z_{med}

Step 2: Replace the center with the median value, or leave it

$B1 = Z_{xy} - Z_{min}$
$B2 = Z_{max} - Z_{xy}$
If $B1 > 0$ and $B2 > 0$ then
    NewPixelVal = Z_{xy} //leave the center pixel as it is
Else
    NewPixelVal = Z_{med} //replace the center pixel with the median value

Step 3: repeat the process for the next pixel starting from step 0 again

The steps above summarize what's done through adaptive median filter implementation. The meaning of these steps is as follows:

Step 1: Search for a true median

IF the current window has a true median (i.e. Z_{med} is different from Z_{min} and Z_{max}) THEN
    //Execute Step 2
ELSE
    IF the current window size is not the maximum
        Increase it and repeat Step 1
    ELSE
        Let the output pixel be Z_{med} and move to the next pixel.
    ENDIF
EndIF

Step 2: if we have a true median

IF (Z_{xy} is different from Z_{min} and Z_{max}) (i.e. not noise)
THEN
    Let the output pixel be Z_{xy} (i.e. not changed) and move to the next pixel.
ELSE
    Let the output pixel be Z_{med} and move to the next pixel.

Project Requirements

FIRST: alpha-trim filter with TWO different algorithms:

1. Counting Sort
2. Selecting K^{th} smallest element in the array without sorting it (Textbook sec. 9.2). where K = T to exclude the smallest T values, then on the remaining array, apply the algorithm again to exclude the largest T values. Finally, calculate the average of the remaining values

SECOND: adaptive median filter with TWO different algorithms:

1. Quick Sort
2. Counting Sort

THIRD: timing graphs

1. Display two graphs to show the execution time **against different window sizes** (3, 5, 7,… W_{max}), where $\mathbf{W_{max}}$ is user input.
    1. One graph for alpha-trim filter to compare its two methods (counting & selecting k^{th} element)
    2. Another graph for adaptive median filter to compare its two methods (quick & counting).

How to calculate execution time?

- To calculate time of certain peace of code:
    1- Get the system time before the code
    2- Get the system time after the code
    3- Subtract both of them to get the time of your code

    To get system time in milliseconds, you can use `System.Environment.TickCount`

How to draw the graph?

- See the example in the given code that draw a graph for two functions: N & N Log(N), using the Z-graph library.
- To draw the timing graph:
    1. Create new object from `ZGraphForm`
    2. Construct x-axis by storing the values of different window sizes (3, 5, 7,… W_{max}) in a `double[]` array.
    3. Construct y-axis by calculating the execution times of the filter at different window sizes (3, 5, 7,… W_{max}) and store them in a `double[]` array.
    4. Add a new curve with x-axis and y-axis to the `ZGraphForm` using the `add_curve` function

Given

- TEMPLATE C# Code to
  1. Open image & load it in 2D array stored in a global variable of type `byte[,]` called `ImageMatrix`, using the following function inside `ImageOperations` class

     ```
     byte [,] OpenImage(string ImagePath)
     ```

  1. Get width and height of the image matrix

     ```
     int GetHeight(byte[,] ImageMatrix)
     int GetWidth(byte[,] ImageMatrix)
     ```

  2. Display an image on a given `PictureBox` control using the following function inside `ImageOperations` class

     ```
     void DisplayImage(byte[,] ImageMatrix, PictureBox PicBox)
     ```

- `ZGraphForm` to use it for drawing the graph, with sample code showing how to use it.

Input

1. Noisy image

Alpha-Trim Filter

1. Window size
2. Trim value
3. Max window size for graph (W_{max})

Adaptive Med Filter

1. Max window size for the filter (W_s)
2. Max window size for the graph (W_{max})

Deliverables

Implementation

1. Alpha-trim filter using two methods:
   a. Counting sort
   b. Select K^{th} smallest/largest element **(Sec.9.2)**
2. Adaptive median filter using two methods:
   a. Counting sort
   b. Quick sort

3. Display two graphs (one for the alpha-trim and other for adaptive median) to show the execution time **against different window sizes** (3, 5, 7,…) of different methods.

Documentation

1. Determine which method is better in each filter based on your results? Explain **why**?

Grades

Deliverable	Grade
Implementation of adaptive median filter using 2 methods	35%
Implementation of Alpha-trim filter using 2 methods	25%
Plotting two graphs for different window size	20%
Document	20%

BONUS TASK

There are some heuristics that can achieve implementing the median filter in image processing in **MUCH** better performance than any sorting/selection algorithm. Try to find/search the idea and implement it.