NTNU
Norwegian University of
Science and Technology

Nonlinear filtering in digtal image processing

Michael Johansen

Master of Science in Electronics
Submission date: August 2011
Supervisor: Tor Audun Ramstad, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

With a plethora of digital imaging devices on the market today, and the trend going toward higher density imaging (more pixel elements on smaller imaging sensors), the need for novel and known filtering techniques are evident. As imaging sensors get smaller, less light per surface area is captured, and as the amount of light decreases in comparison to the thermal noise detected by the imaging sensor, the capturing of an image becomes difficult. Some of this noise can be removed by filters, either by applying filtering circuitry on the imaging device or as post processing. Traditional linear filtering techniques are not always sufficient for such as they are prone to smoothing the edges in the image. Alternate approaches can be found in nonlinear filtering techniques, such as median filtering, which can preserve edges and can be more visually pleasing.

For this purpose a software based imaging editor needs to be developed that is able to analyze and apply different filtering algorithms. Filtering operations will utilize general purpose computing on graphics processing units(GPGPU) to allow parallelization of the process.

This project will be done at IET, with supervisor Tor Audun Ramstad.

Abstract

This project has produced a software suite for image processing of noisy digital images. The software utilizes a series of nonlinear filtering techniques in the attempt to remove noise from images. The noise included might be forms of white noise, impulsive noise or uniform noise. To make the software use its given cpu-cycles in an efficient manor, the utilization of general purpose computing on graphics processing units was adopted. Using hardware in such a manor allowed parallel processing techniques to be utilized, which suits image filtering well. This parallel processing was done with Microsoft Accelerator libraries, a API for processing arrays in parallel.

The user interface of this software was implemented using the WPF subsystem of the .NET framework. WPF allowed for both rich and powerful user experiences, but also allowed for rapid development and easy extensibility. The use of this software aimed to provide a straightforward, but also powerful, solution to image noise filtering.

The method for developing this software was a form of adapted agile development. The development process was iterative and development of modules was done upon need of that module. This approach allowed a working version of the software always to be available during development. As such the final documentation is always the source code itself, but this thesis will do its best to describe all the features and implementations of the project. And lastly we present results of filtering performed in the application.

Preface

This thesis is the result of a project performed during a master thesis in the spring of 2011. The thesis have been completed at Department of Electronics and Telecommunications at the Norwegian University of Science and Technology. I would like to thank my supervisor Tor Audun Ramstad for guidance on this project.

Michael Johansen
Trondheim, July 28. 2011

Contents

List of Figures

List of Tables

Algorithms

Nomenclature

DWMFilter Distance Weighted Median Filter

HMMFilter Hybrid Mean-Median Filter

HMDWMFilter Hybrid Mean-Distance Weighted Median Filter

KNNFilter K-Nearest Neighbor Filter

MSE Mean Squared Error

MTMFilter Modified Trimmed Mean Filter

RGB Red, Green and Blue. A common way to represent colors digitally.

WMFilter Winsorized Mean Filter

WPF Windows Presentation Foundations

1 Introduction

Digital images surrounds us everywhere nowadays, with an increasing amount of devices capable of delivering, capturing and sharing multimedia sources. With increased interconnectivity it is easier to share experiences, often in form of an image or an video. Often this kind of media is shared on social network sites such as Facebook™[1] or Twitter™[2]. More devices is being able to connect to the Internet so media can be shared from anywhere, at anytime.

The devices used to share images is ofttimes mobile phones, and their capturing device is in form of an integrated charge-coupled device (CCD). As keeping the size of mobile devices small is often essential, not much space is allowed for the image capturing device. With ever increasing resolutions being demanded of the CCD's the amount of light per pixel becomes diminishingly small. Mobile devices is often required to capture high mobility situations, which require high shutter speed, and/or dark scenes. Either situation making it difficult to capture enough light.

Capturing images under such difficult working conditions, will introduce sensor errors that may be observed as noise in the resulting image. Two kinds of noise or often introduced under such conditions: Gaussian white noise and/or impulsive noise.

There are many techniques for handling such noise, called image filtering. Some of these filtering techniques are explained in section 2. In this thesis the main focus is mainly on Nonlinear Filters (and median-based filters in particular), because they have certain capabilities that makes them interesting for image filtering, such as being able to preserve edges while still suppressing noise.

In this thesis an image filtering software will be developed, which have been given the name ImageLab, to apply these filters. The software will be capable of common image operations, such as gamma correction and contrast/brightness alterations, and addition to being able to apply the different filters.

The software will not aim to replace Photoshop™or other professionally developed photo editors, but simply act as an alternative when more powerful filtering techniques are required. As such it is planed to release the source code when an adequate license is located.

[1] service found on http://www.facebook.com/
[2] service found on http://www.twitter.com/

Since image processing often is a repetition of the same series of operations for each pixel of the image, there is a potential to utilize parallelization. Thus this software uses Microsoft™Accelerator v2, to utilize the additional computing power of modern graphics processing units from a managed environment.

This thesis will be partitioned as follows: Section 2 will give an introduction to the different filtering techniques that will be used in the software. Section 3 will describe the tools that will be used to develop ImageLab, and why they are suited for the purpose. Section 4 will describe the development of the software. It will give an in-depth, or in certain instances as much as is reasonable, description of the implementation of ImageLab. In Section 5 the results of filtering operations performed with the software will be presented. And lastly, Section 6 will conclude this thesis and give suggestions for future or similar work.

2 Background

In this section the various techniques and filtering methods will be defined. Pseudo-code will be given where it is appropriate, and a few examples will be given of how the code changes when we instead uses parallel programing paradigms.

2.1 Windowing

Often it is impractical to work on whole pictures at once, for many filtering operations only pixel data from nearby pixels is used. For such a windowing operation is needed. In this thesis only square windowing functions are considered. For a given radius r the resulting window W_r will be:

$$W_r = \begin{pmatrix} x_{(i-r,j-r)} & \cdots & x_{(i-r,j)} & \cdots & x_{(i-r,j+r)} \\ \vdots & \ddots & \vdots & & \vdots \\ x_{(i,j-r)} & \cdots & x_{(i,j)} & \cdots & x_{(i,j+r)} \\ \vdots & & \vdots & \ddots & \vdots \\ x_{(i+r,j-r)} & \cdots & x_{(i+r,j)} & \cdots & x_{(i+r,j+r)} \end{pmatrix} \tag{1}$$

where $x_{(a,b)}$ is pixel elements from the current image. When this window is close to the edges of the image some of the elements $x_{(a,b)}$ may be undefined. There are several ways to handle these undefined elements, some are:

Replicate Use the closest defined element.

Default Value Use a constant value instead.

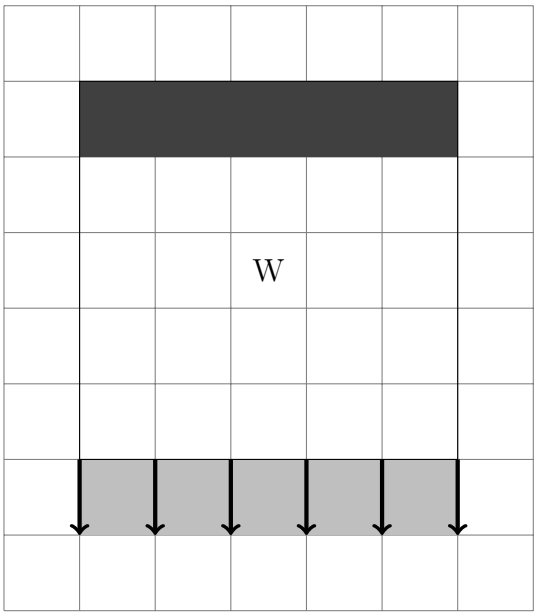
Ignore Do not use undefined elements. Effectively making the window smaller as it approaches the edges.

In this thesis replicate is the preferred method for defining undefined elements.

The complexity of the naive approach to selecting the window W_r is $O(r^2)$. So doubling the window size will quadruple the number of instructions needed. This limitation makes larger window sizes impractical, but there are solutions to this complexity problem.

When moving the window one step, not all of the elements needs to be selected anew, most of the elements will be the same. Observing that there are $2 * r + 1$ new elements and the same amount of old elements, it is possible to just exchange these, see figure 1. This will yield a complexity of $O(r)$, which is further described

Figure 1: Figure showing the movement of windowing function W_2, if only operating on the marked pixels, a complexity of $O(r)$ can be achieved.



by Huang[3].

Further improvements may be achieved if one observe that for each row, the elements are moved only one step per loop. Thus only one new element needs to be added and one element be removed[4], achieving a complexity of $O(1)$.

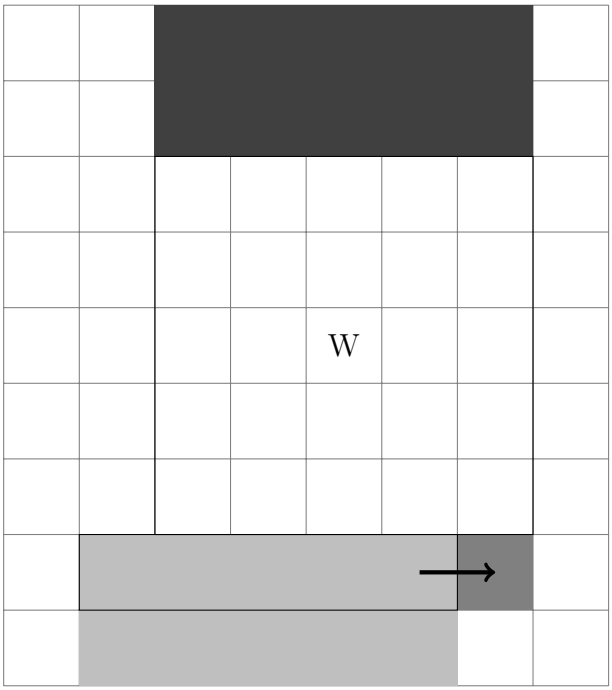
These improvements work well for sequential processing units, but cannot easily be applied on parallel processing units. This project is thus often forced to utilize the naive approach to windowing, making it impractical for very large window sizes.

2.2 Mean Filter

Mean filtering is one of the simplest techniques for smoothing images. For each pixel in an image, average the output of the windowing function for that pixel. This technique can remove noise, as long as the noise source is additive and is zero-mean. Consider the 1 dimensional noise signal x_{noise} of length N:

$$x_{noise}(i) = X_i, \quad \forall i \in 1 \leq \mathbb{N} \leq N \tag{2}$$

Figure 2: Showing the overview of the constant time windowing ($O(1)$).



where X_i is a random number with $E[X] = 0$, then applying the 1 dimensional version of Window W_r on the signal x_{noise} resulting in:

$$W_r(i) = (x_{noise}(i-r), \ldots, x_{noise}(0), \ldots, x_{noise}(i+r)) \tag{3}$$

Taking the mean of the Window W_r:

$$mean(W_r(i)) = \frac{\sum_{j=i-r}^{i+r} x_{noise}(j)}{(2r+1)}, \tag{4}$$

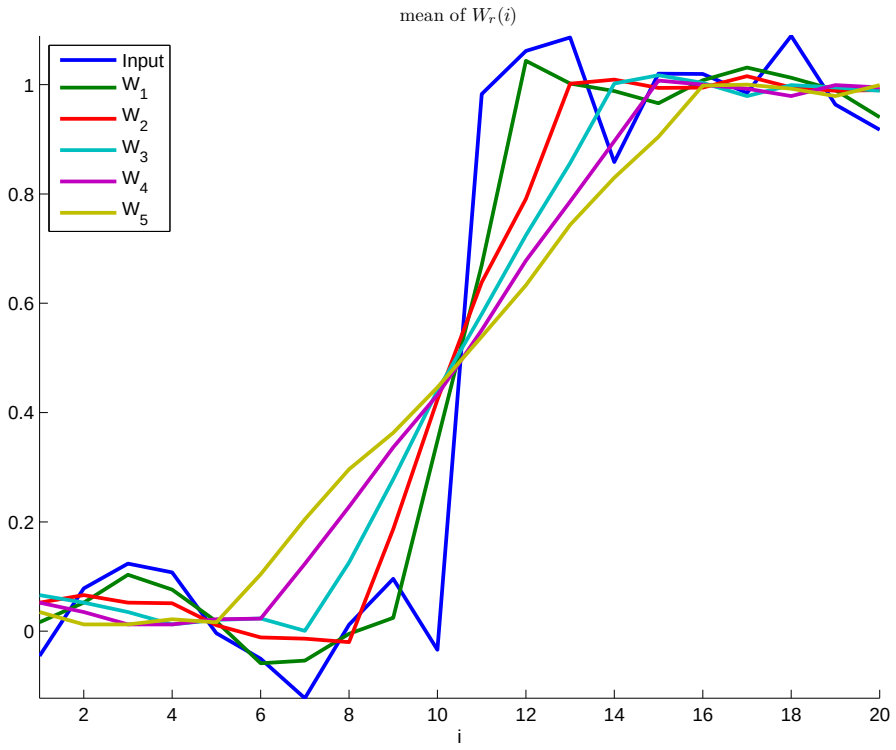
As the window size r increases we observe that:

$$\lim_{r\to\infty} mean(W_r(i)) = \frac{0}{\infty} = 0 \tag{5}$$

The requirement that expectation value $E[X] = 0$ is not strictly required, and more usefully the filtering is capable of removing the noise entirely. But outside this example its not reasonable to expect a signal entirely consisting of noise. The filter will also remove information from the original signal, in addition to the noise.

Even with this drawback, which exists in pretty much all filters, it is still an useful filtering technique. As long as the signal is approximately uniform over

Figure 3: Mean Filtering of edge with Gaussian noise for different window sizes r.



an area, the mean filtering works well. This observation is utilized in some the nonlinear filters later in this thesis, by using nonlinear techniques for selecting only relevant pixels, it is possible to utilize that the image is somewhat piecewise uniform.

The filter also display some undesired behavior near edges, as can be seen in figure 3. As the window size r increases the initially sharp edge becomes gradually more sloped. The figure also displays the inherent trade off between image detail and noise removal (for larger r the signal is much smoother), as a function of window size r.

The implementation in algorithm 2 is inspired by Braünl[2], where they made the $O(r^2)$ into $O(r)$ through the use of separability. Unfortunately the median filter is not separable.

Algorithm 1: Mean Filter

```
input: Image, radius
output: Result

N = (2 * radius + 1) * (2 * radius + 1)
for i = 1 to Image.Width do
    for j = 1 to Image.Height do
        place window W_r at (i, j)
        accumulator = 0
        for each element x ∈ W_r
            accumulator = accumulator + x
        Result(i, j) = accumulator/N
```

Algorithm 2: Mean Filter Parallel

```
input: Image, radius
output: Result

N = (2 * radius + 1) * (2 * radius + 1)
for i = -radius to radius do
        rowsum = rowsum + Image.Shift(0, i)
for j = -radius to radius do
        sum = sum + rowsum.Shift(j, 0)
Result = sum / N
```

2.3 Median Filter

While the mean filter is a basic linear filter, the median filter is one of the most basic nonlinear filtering techniques around. They are very similar in both function and construction, but yield significantly differing results. Where the mean filter averages the output of the sliding window, the median filter instead uses the median of the sliding window output.

As the median is described by:

$$MED(X) = x_{median} \quad \text{where} \quad P(X \geq x_{median}) \geq \frac{1}{2} \wedge P(X \leq x_{median}) \geq \frac{1}{2} \quad (6)$$

It can be shown for any symmetrical zero-mean random function that the median will remove such noise. Given a normal distributed signal(such as in 2) with probability density function $f(x)$ described by $N(0, 1)$, similar result as in section 2.2 may be achieved for removing the noise.

As the rest of the filters in this thesis is a variation of either mean or median filtering, we can generally assume that the ability to totally remove the image noise also holds for these filters.

Comparing figure 4 with figure 3 the median filters properties are evident. Where the mean filter smoothed the edge, the median filter preserved it while still suppressing the noise. Clearly this is a useful property for image processing.

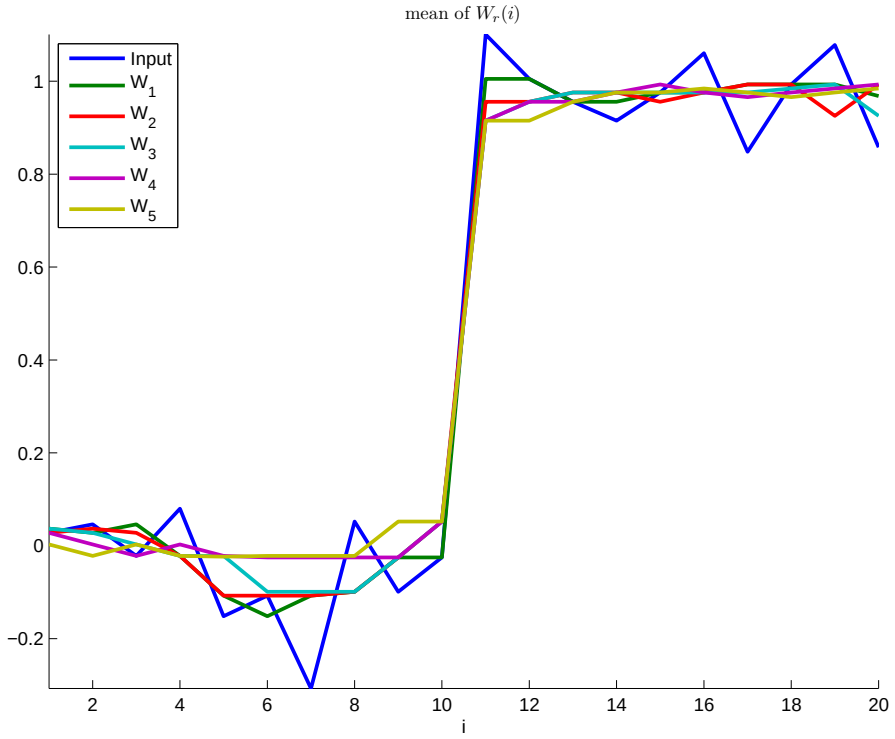
It is evident from figure 5 that edges may be moved with median filter. This total loss of detail and variably moving of edges can make median filters a bit unpredictable compared to mean filters, where loss of detail is more gradual with increasing window sizes. Other filters in this thesis will try to combine the best from both worlds.

Upon visual inspection of filtered images, the result of median filtering often seems more visually pleasing than the result obtained from mean filtering. This may be due to how perception of images work. Ie. while moving an edge is hardly noticed, blurring it may cause it to seem bigger.

2.4 Distance Weighted Median Filter

A Distance Weighted Median Filter (DWMFilter) utilizes a weighted median. The weighted median can be seen as an extension of the median function, and if all

Figure 4: Median Filtering of edge with Gaussian noise for different window sizes r.

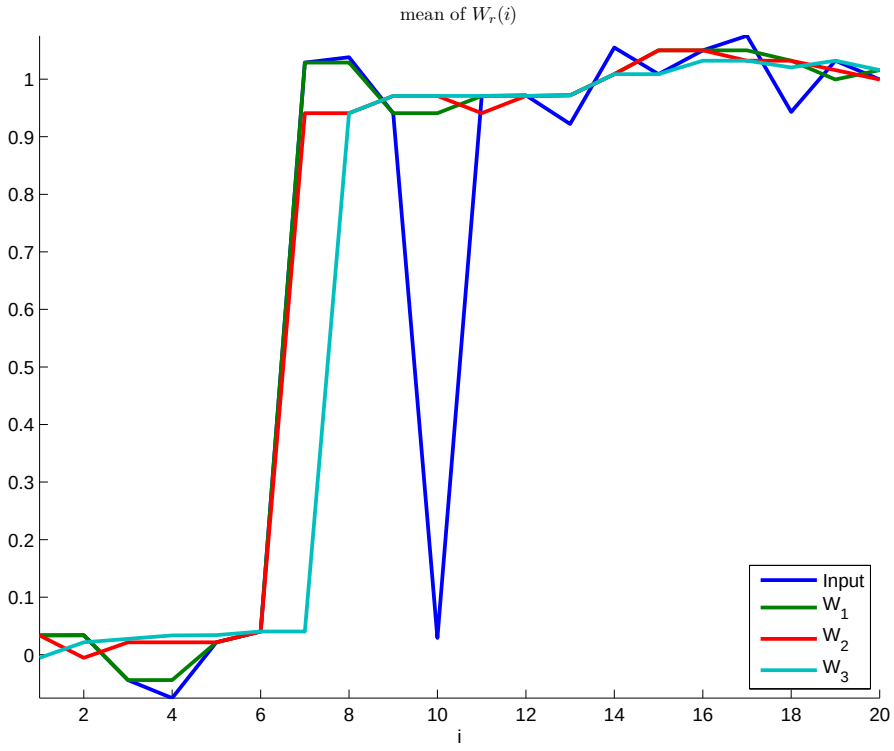


Algorithm 3: Median Filter

```
input: Image, radius
output: Result

N = (2 * radius + 1) * (2 * radius + 1)
for i = 1 to Image.Width do
    for j = 1 to Image.Height do
        place window W_r at (i, j)
        Result(i, j) = median(W_r)
```

Figure 5: Median Filtering of edge with Gaussian noise for different window sizes r, showing how structure may be lost.



Algorithm 4: Weighted Median

```
input: x, w
output: Result

indices = SortIndices(x)

w_sum = 0
for each w_i ∈ w do w_sum = w_sum + w_i

sum = 0
for i = 1 to x.Length do
    sum = w[ indices[i] ]
    if sum ≥ w_sum
        Result = x[ indices[i] ]
        return
```

weights are equal to one, they are the same. Given a signal $x \in \mathbb{R}$ with weights in a corresponding signal $w \in \mathbb{R}_{\geq 0}$ it is possible to follow the pseudo-code in algorithm 4 to obtain the weighted median.

In the DWMFilter the weights are set to be inversely proportional with the distance to the center of the current windowing function, thus taking advantage of the inherent spatial relationship between nearby pixels. It seems reasonable to chose such weights, because closer pixels is more likely to resemble the window center. As such the following weights have been chosen:

$$w_{i,j} = \begin{cases} \frac{3}{2} & \text{if } i = 0 \text{ and } j = 0 \\ \frac{1}{\sqrt{i^2+j^2}} & \text{else} \end{cases} \tag{7}$$

The selection of constant value 3/2 in equation 7 is quite arbitrary and may be substituted in different implementations. The center weight has to be implemented different than the rest of the weights, or else the center weight will be positive ∞, and no other pixel than the central pixel would ever be used. Any value between 0 and $\frac{\sum w_{i,j}}{2} \quad \forall i, j$ will suffice.

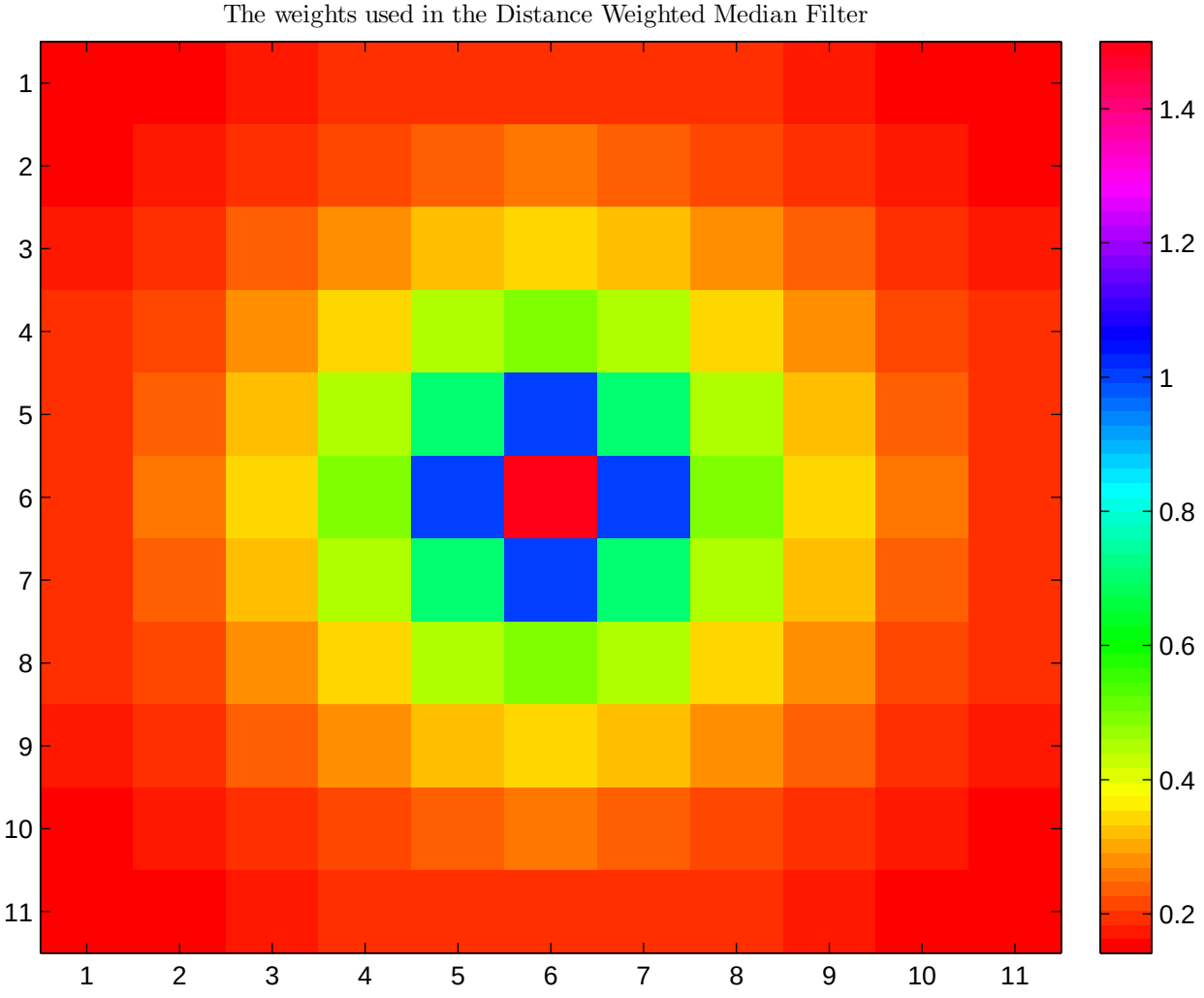


Figure 6: Distanse Weighted Median Filter Weights

Algorithm 5: Hybrid Mean-Median Filter

```
input: Image, radius, selectionpoint
output: Result

Image_variance = var(Image, radius)
Image_mean = mean(Image, radius)
Image_median = median(Image, radius)

for i = 1 to Image.Width do
    for j = 1 to Image.Height do
        if Image_variance(i, j) ≥ selectionpoint
        do Result(i, j) = Image_median(i, j)
        else do Result(i, j) = Image_mean(i, j)
```

2.5 Hybrid Mean-Median Filter

A Hybrid Mean-Median Filter is exactly what it sounds like, it is a filter that utilizes both Mean Filtering and Median Filtering depending on a set selection criteria. The idea is to utilize the Mean Filter in uniform portions of the image and the Median Filter in non-uniform portions. In this implementation the selection criteria is the variance of the sliding window, see algorithm 5 and Astola[1].

2.6 Hybrid Mean-Distance Weighted Median Filter

The Hybrid Mean-Distance Weighted Median Filter (HMDWMFilter) is also what it sounds like, a hybrid between a Mean Filter and a DWMFilter. See section 2.5. It is an attempt to improve on the HMMFilter, and utilizing the the similarity between nearby pixels.

2.7 Winsorized Mean Filter

The idea behind the Winsorized Mean Filter (WMFilter) is to only use a subset of the windowed pixels. The subset contains the usual pixels from the windowing function without the n_- smallest and n_+ largest elements, see equation 8. This approach allows us to remove outliers from the set of pixels, especially impulsive noise, that otherwise would have a significant impact on the Mean Filter.

Algorithm 6: Modified Trimmed Mean Filter

```
input: Image, radius, α
output: Result

for i = 1 to Image.Width do
    for j = 1 to Image.Height do
        place window W_r at (i, j)
        x_median = median(W_r)
        count = 0
        sum = 0
        for each x ∈ W_r do
            if |x_median − x| ≤ α
                count = count + 1
                sum = sum + x
        Result(i, j) = sum/count
```

$$Y_{WinsorizedMean} = \frac{\sum_{i=n_-}^{n-n_+} X_{sorted}(i)}{n - n_- - n_+} \tag{8}$$

This is a compromise between the median and mean filters, attempting to improve on the existing filters. Discarding subsets due to not meeting some criteria is an approach we will use in the next filters also.

2.8 Modified Trimmed Mean Filter

Similarly to the WMFilter the Modified Trimmed Mean Filter (MTMFilter) discards a subset of the windowed pixels. The pixels that are more than α different from the median x_{median} of the set, are discarded, and the rest are averaged, see algorithm 6.

The MTMFilter introduces the use of a variable amount of pixels per position. And some filter input may present problems, i.e. for certain input all of the pixels may be discarded (the median itself is not necessarily a member of the set, unless the set consist of an odd amount of members). In this thesis the window sizes available prevents this, see table 1. As seen in the table all the sizes is odd, and thus the problem is avoided.

r	N
0	1
1	9
2	25
3	49
4	81
5	121
...	...

Table 1: Window radius vs. elements used.

2.9 K-Nearest Neighbor Filter

The K-Nearest Neighbor Filter (KNNFilter) takes a slightly different approach to the discarding of pixels. It uses the k least different pixels in the window and averages them. This may make the filter easier to set up, as the input don't need much tweaking.

3 Tools

This section will describe the different tools used during the development of ImageLab. The coming subsections will clarify why these tools where used, instead of other tools. Often the choice of one tool limits or even dictates what other tools must be used in conjunction with it.

3.1 .NET Framework

The .NET Framework is a software framework, mainly for windows. It contains a large library with many utilities that allow for quite rapid (high level) development. There is also a choice of languages that are compatible with the framework, these include C#, C++, Visual Basic and F#. Programs authored for the .NET Framework is run on a virtual machine (managed space), and has utilities like garbage handling and memory management built-in.

Figure 7: .NET Framework allows for ease of development

It was chosen due to familiarity with the .NET Framework, its utility functions, access to the relevant IDE[3] and its speed of development. Alternatives most likely would have been Java or C++. Either could also have served well for this project. With Java also sporting increased portability between platforms.

3.2 Windows Presentation Foundation

The Windows Presentation Foundation (WPF) is a system for rendering graphical interfaces in windows based applications. WPF allows programmers to separate the design of interfaces (which is handled in xaml-files) from application logic (which is handled by the relevant language's source files, here cs-files).

[3]Microsoft Visual Studio 2010™

The use of a graphical designer allows the interface to be easily developed. Thus allowing more time for developing the application logic.

Alternatives to WPF would most likely be to develop the interface using Windows Forms, but WPF uses DirectX technology where Windows Forms is drawn using GDI making WPF much more powerful.

3.3 Microsoft Accelerator v2 API

Microsoft Accelerator v2 allows developers to easily integrate parallel processing of arrays and images, utilizing the parallel processing capabilities of modern multi core processors (cpu)/graphical processing units (gpu).

Parallel processing is accomplished by building data structures of either arrays or images and transferring them to respective processing unit (either cpu or gpu). This of course introduces some overhead (time spent transferring data from memory to processing units memory), which is very small for cpu processing and larger for gpu prcessing. To compensate the gpu has far more processing units. Thus making the cpu the better target for small data sets, while the gpu is better for larger data sets.

Alternatives here is not much to choose between, it is possible to do general-purpose computing on graphics processing units (gpgpu) trough the use of shader programs or CUDA implementations trough managed wrappers, but none is as straightforward as Microsoft Accelerator v2 API.

4 Development

This section describes ImageLab's features and how they're implemented.

4.1 Graphical User Interface

The classes described here are all contained within the `System.Windows` namespace.

4.1.1 Main Window

Within the main window of ImageLab, see figure 8, is where most of the interaction with the user will happen. The window is implemented as an subclass of `Window`.

4.1.2 Layout

The main window lays out its subcomponents in a grid like fashion, like a table, making placement of subcomponents straightforward and easily extensible. This is implemented through one `Controls.Grid` controller, with three rows.

4.1.3 Menu

Selection of program commands/filtering is done through a menu laid out topmost in the Grid described in 4.1.2. Both the the placement and functionality of this menu should be familiar to most Windows™users. The menu system is implemented with `Controls.Menu` and `Controls.MenuItem` controls. Functionality such as Open, Save and such is implemented trough the `Input.CommandBinding` connected event handlers, whereas the other menu items is implemented through their own event handlers.

4.1.4 Image

Filling up most of ImageLab's layout space is the current image. The image will try to retain it proportions. This functionality is implemented trough a `Controls.Image` control.

4.1.5 Status Bar

Bottommost in the Grid a status bar is placed, which will give feedback to the user about the current pixel coordinate and color data. This is implemented trough a `Controls.TextBlock` within a `Controls.Primitives.StatusBar`.

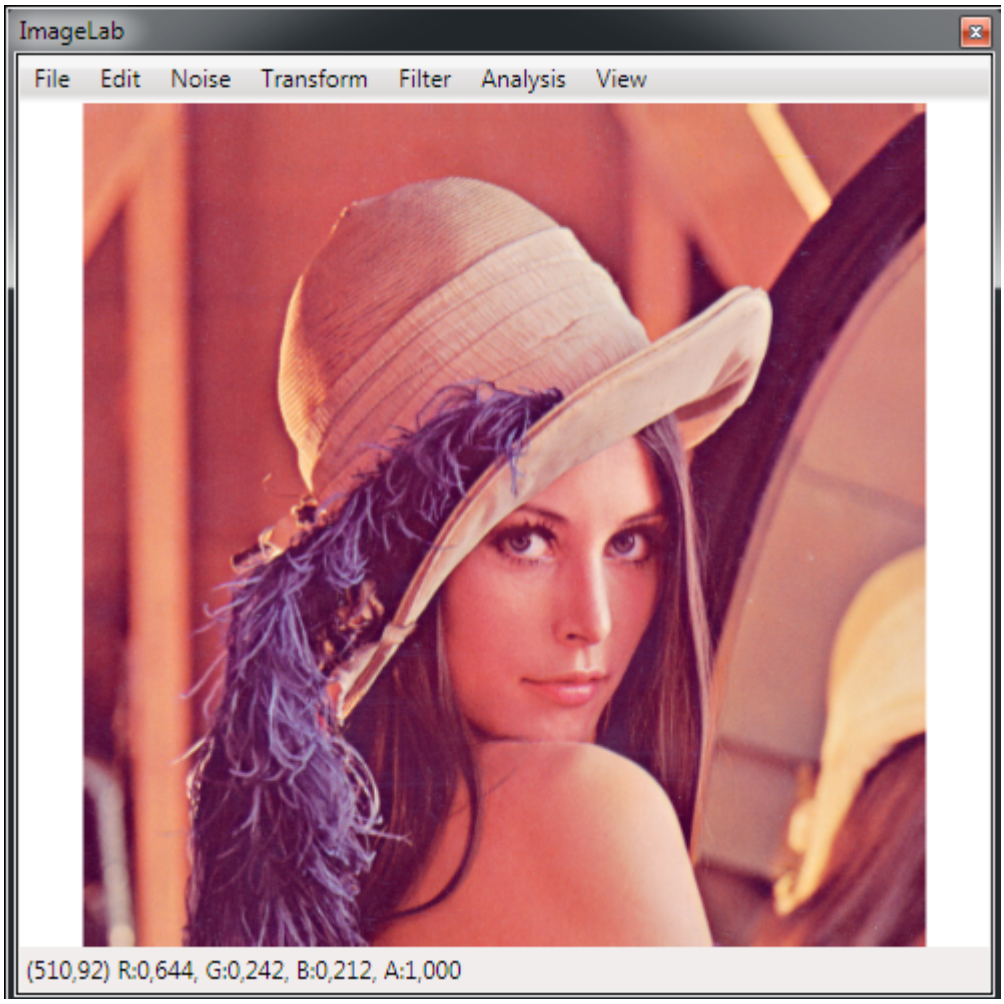


Figure 8: ImageLab's main window.

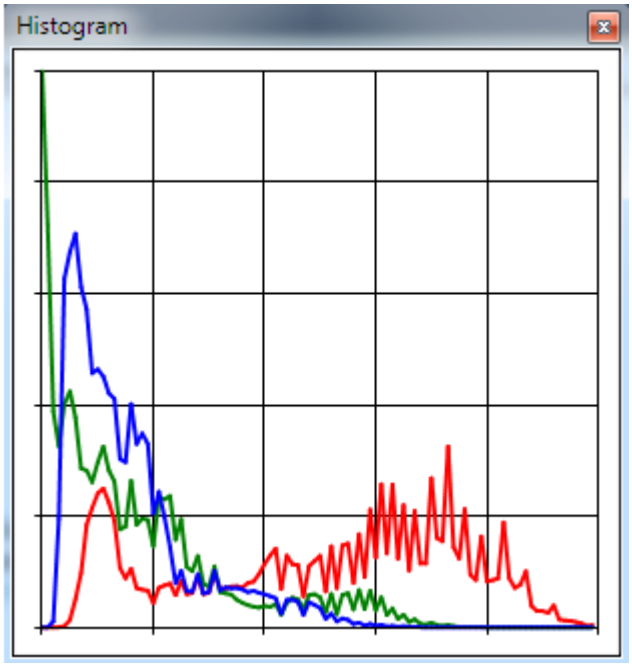


Figure 9: Histogram window

4.1.6 Histogram

Additionally a histogram window have been implemented. It can show the distribution of the RGB-color data. It also has the option to display the data frequency linearly or logarithmically (it is changed by clicking on the graph).

The histogram graph tries to automatically scale to any input, so it can present useful information to the user, see figure 9.

The histogram is implemented trough a `Forms.DataVisualization.Charting.Chart` control, it actually is a part of Windows Forms (not WPF as the rest of the controls). Windows Forms controls are generally not interchangeable with WPF controls, and COM Interop hosting is used to host the `Chart` control within a WPF `Controls.UserControl` control.

4.2 Image processing

All of the image processing methods are implemented after a similar pattern `F4PA FilterName(F4PA input, Object[] parameters)` where the `input` and `return` value is of the type `F4PA` (a 4 component floating point parallel structure), and

Algorithm 7: Actual implementation of Mean Filter

```
public static F4PA Mean(F4PA input,
    params object[] parameters)
{
    var radius = (int)parameters[0];

    var sum1 = new F4PA(new F4(0f), input.Shape);
    var sum2 = new F4PA(new F4(0f), input.Shape);

    var n_samples = (2 * radius + 1) * (2 * radius + 1);

    for (int i = -radius; i <= radius; i++)
    {
        sum1 += PA.Shift(input, new int[] { i, 0 });
    }
    for (int i = -radius; i <= radius; i++)
    {
        sum2 += PA.Shift(sum1, new int[] { 0, i });
    }
    return sum2 / new F4(n_samples);
}
```

`parameters` is a variable length array of various types of argument. This common pattern for all filters allow all the methods to be invoked in a similar manor. For an example of such a pattern look at algorithm 7.

The other filters are of a similar structure. For more information on the other filters, please review the attached source code.

5 Results

In this section the results from this project is presented. The results will be in the form of filtered images. Two sided impulsive noise were added to a reference image (the reference image Lena is used). The filtered images are then compared to a noise-free version of the reference image. Not all filters are presented equally exhaustively, but rather in a manner that highlights the interesting results. For completeness, a more exhaustive table is presented at the end of this section.

The results determines how effective a certain filter (with a given set of parameters) is at removing noise of this kind from this image. The reference image contains both small details, sharp edges and uniform areas, making it well suited as a test image.

5.1 An objective metric

Mean Square Error (MSE) was chosen as the metric for deciding which filtering operations performs the best. MSE may differ from what is subjectively perceived as the optimal metric, for research on perceptual quality the Q2S[5] center can be queried. But MSE it is still a robust and objective measure for signal to noise ratios, and is well suited for use with these filtering operations.

5.2 Analysis of images

The results of filtering is presented for a selection of results in figures, for an example see figure 10. To the top and left in the figure the image is presented. On the top right there are two zoomed in versions of the image, so one can properly see the effects of filtering, and two difference images for the same zoomed in area. The difference images highlights what kind of details are lost during filtering. On the bottom left there is a scan-line from the above image, which allows us to see the effects of filtering in an alternative, and perhaps more familiar way (each color component is separate). To the bottom right there is a histogram of the errors, detailing number of samples vs error (more samples to the left is better).

5.3 Analysis of a 3-by-3 mean filtered image

The first result that is presented is that of the 3-by-3 (1 radius) mean filter, and this result will be used as a benchmark result for comparison against the *novel*,

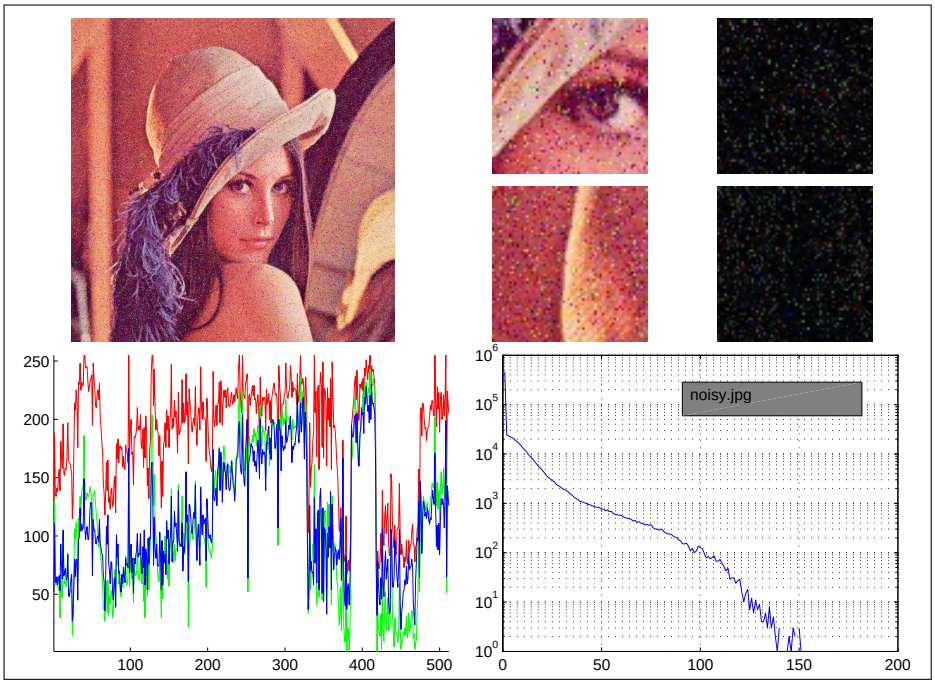


Figure 10: Analysis of the noisy image created by adding 2-sided impulsive noise to the reference image Lena.

nonlinear filters. From figure 11 it is evident that there is less noise in the image than before. In the difference images some of the details of the edges can be discerned, and some of the impulses was averaged out over a larger area (making them more noticeable in the process). Much of the variation that was observed on the scan line in figure 10 is significantly lessened by the mean filter.

5.4 Analysis of a 3-by-3 median filtered image

Of the nonlinear filters that is treated in this thesis the simplest of them is the median filter, and as such is an interesting result to present. Looking at figure12 it is clear that significantly more noise have been removed by the median filtering than by the mean filter. From the difference images there is evidence that some of the repetitive structures in the hat and eyelashes are lost, but overall the edges are well preserved. From the scan-line extraction of the image we can see the crisp contours of the underlying structure. Without yet having examined the objective metrics it is relatively safe to note that this is a significant improvement over the mean 3-by-3 filter.

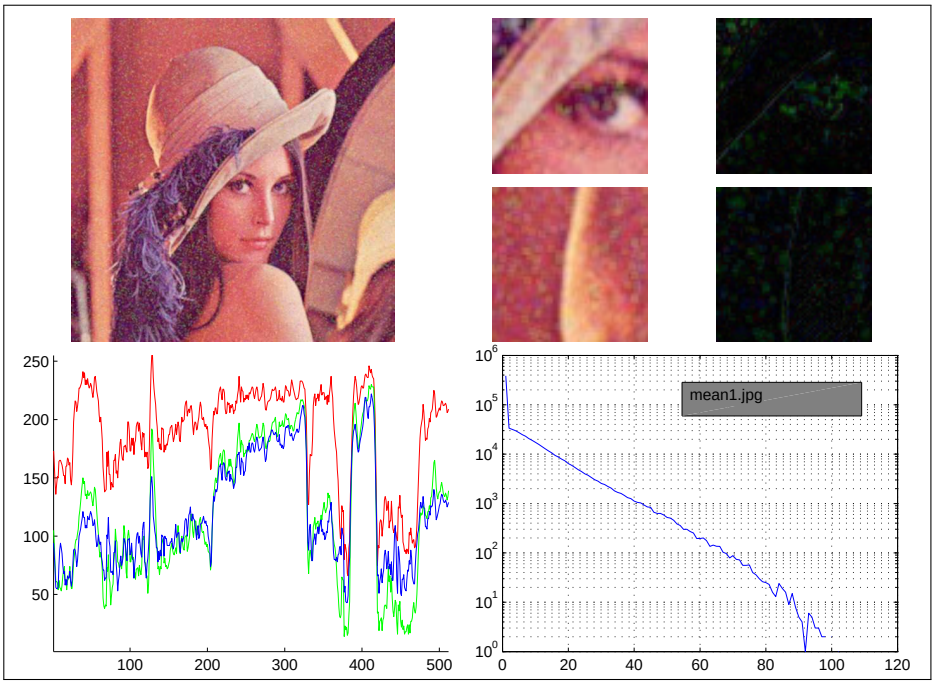


Figure 11: Mean 3-by-3 Filter

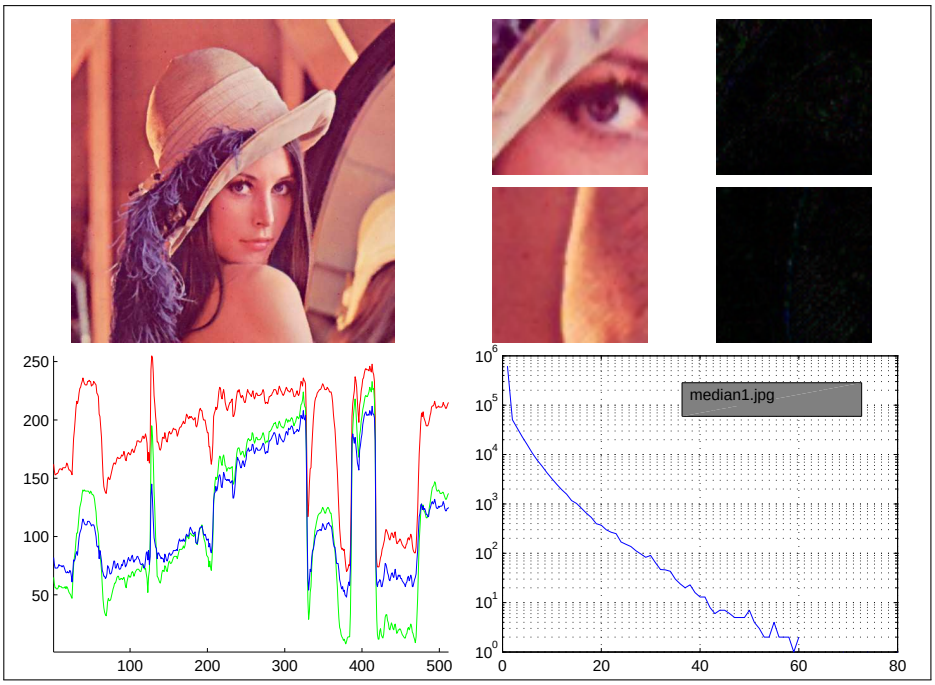


Figure 12: Median 3-by-3 Filter

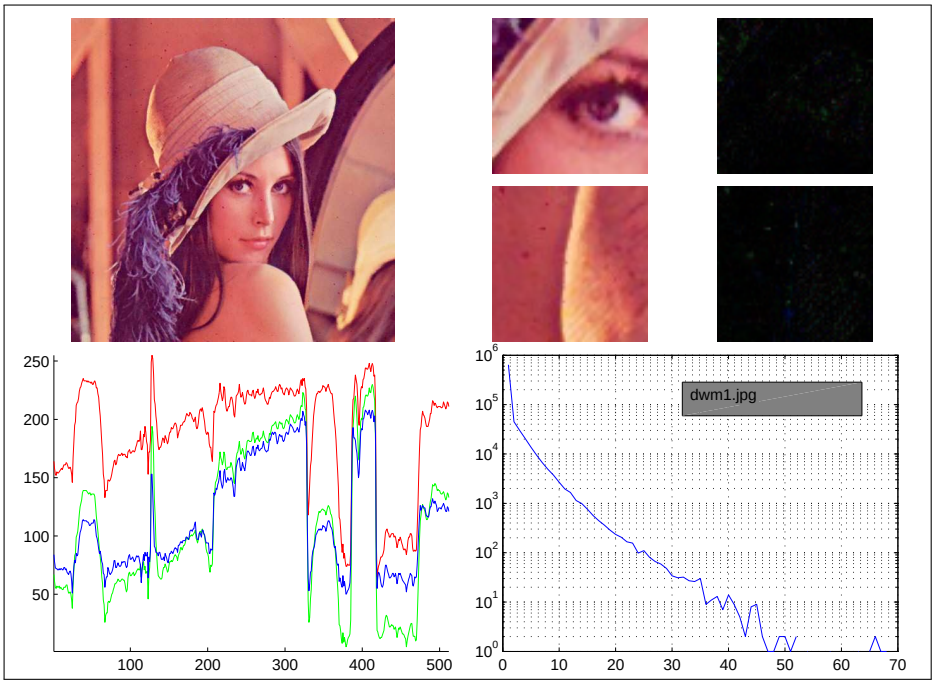


Figure 13: Distance Weighted Median 3-by-3 Filter

5.5 Analysis of a 3-by-3 DWM-filtered image

Next in line the Distance Weighted Median filter is presented, also this in the 3-by-3 variant. In figure 13 similar results as the median filtering are obtained, which is not a big surprise since the filters are very similar. From the difference images it seems like the DWMFilter is even better than the median filter at preserving edges, but also slightly worse when it comes to removing noise impulses. Both of the two last images seems to be very sharp, and almost noise free.

5.6 Analysis of a 3-by-3 MTM-filtered image

Next the Modified Trimmed Mean Filter 3-by-3 is examined, see figure 14. From the images it almost seems like the edges are enhanced, but from the difference image it is evident that they are not, they are just very sharp. The MTMFilter tries to average as many samples as possible so long as they their value is reasonably close to the median, leading us to expect great edge preservation and impulse removal, at the cost of small level structure. That is exactly what is observe in the difference images, the average error seems to have increased, but larger structures are well preserved. By tuning this filter correctly significantly better results

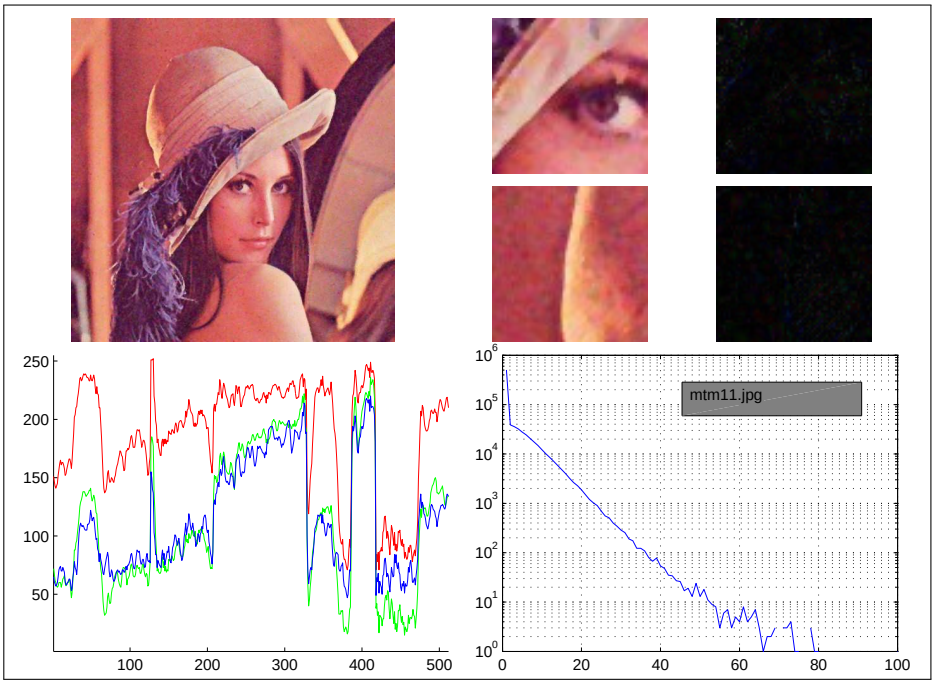


Figure 14: Modified Trimmed Mean 3-by-3 Filter

may be achieved (by matching the detail loss to the detail that is lost due to noise).

5.7 Analysis of a 11-by-11 median filtered image

Lastly the use of larger window median filters is highlighted, especially the 11-by-11 median filter. The filter is not chosen because of its proves as to removing noise, but because it presents an interesting artistic effect, see figure 15. The image is significantly *softened*, but still retains its high level structures/edges.

5.8 More filters

The table 2 presents a more exhaustive selection of filters, much more than the limited selection that was presented in the previous paragraphs. From the table it is evident that the most successful filters was indeed the DWMFilter and the Median Filter. What is surprising in the table is the trend that bigger (filter window sizes) is not better. An explanation for this trend may be that as the filter sizes grow, more of the details in the original image is also lost. This is not something new, see section 2, but what is surprising is that so many of the filters tries to

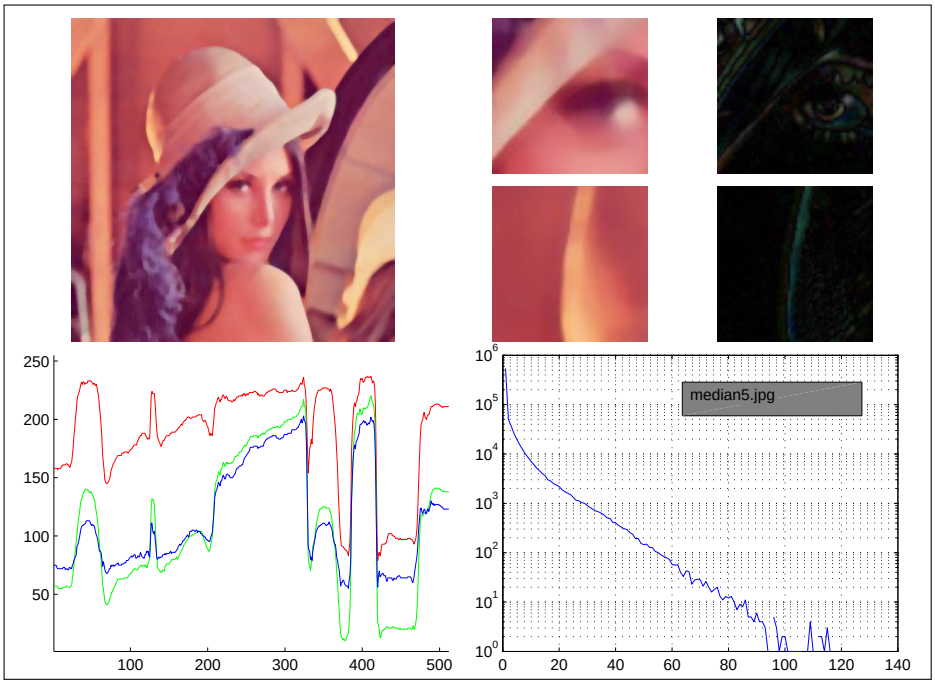


Figure 15: Median 11-by-11 Filter

incorporate as many samples as possible. When using the simplest approach, the median filter, the best *objective* result were obtained.

It is also reasonable to assume that different types of noise, a different image and different filtering parameters yields different outcomes. Mayhap some where bigger is better. Fine tuning of parameters is not something that is emphasized in this thesis, and will thus leave that problem to users of ImageLab instead.

Both the DWMFilter and the median filter is parameterless filters (not counting the window size as a parameter), and yields good results for users without the need to tweak parameters. This is good news for the applicability of the software as it does not require a lot of explanation, and means it can be deployed partially, or fully, to a broad spectrum of users.

Filter	Radius	MSE	Comment
Distance Weighted Median	1	**40.1259**	
Distance Weighted Median	2	42.9816	
HMDWM	1	131.4223	Parameter: 0.01
HMDWM	2	136.3852	Parameter: 0.01
Hybrid Mean Median	1	112.6582	Parameter: 0.01
Hybrid Mean Median	2	110.4107	Parameter: 0.01
Hybrid Mean Median	3	131.515	Parameter: 0.01
K-Nearest Neighbor	1	112.8108	Parameter: 6
K-Nearest Neighbor	2	90.6452	Parameter: 18
Mean	1	173.376	
Mean	2	181.9212	
Mean	3	217.2347	
Mean	4	256.228	
Mean	5	295.76	
Median	1	**38.5722**	
Median	2	59.0154	
Median	3	86.3629	
Median	4	112.2882	
Median	5	136.8673	
Modified Trimmed Mean	1	84.7571	Parameter: 0.1
Modified Trimmed Mean	2	88.0758	Parameter: 0.1
Modified Trimmed Mean	3	108.9378	Parameter: 0.1
Winsorized Mean	1	131.541	Parameter: 0.1
Winsorized Mean	2	102.2365	Parameter: 0.1
Winsorized Mean	3	133.4182	Parameter: 0.1
Unfiltered Image	-	533.9996	
Original Image	-	0	

Table 2: Mean Square Error for a selection of different filtering options.

6 Summary

In this thesis an application named ImageLab have been developed. ImageLab is an image filtering application focusing on the use of nonlinear filtering processes. During development of the software it became clear that ImageLab is not a full image processing suite, such as Photoshop™, but rather a complement to existing software solutions. This is true for now, but further development of the software may lead it to become an alternative full featured image processing solution.

Theory behind the use of ImageLab's image processing techniques was introduced in section 2, the same techniques was also introduced as pseudo-code. Also the different filters and their functionality were explained. In section 4.2 the same pseudo-code were turned into usable C#-code. Following this recipe it should be a straightforward process to duplicate the work done in this thesis, both for novel applications or extending the current application further.

In section 5 it was shown how the different filters performed with respect to an objective metric. While not being able to generally conclude a given filters proficiency at removing noise, we can conclude that for a given set of initial conditions (input image, the type/strength of noise and parameters) some filters did perform better than others. With 2-sided variable height impulsive noise and the Lena reference image a median-filter and a Distance Weighted Median-filter performed the best (according to the objective Mean Square Error). All the filters did improve the objective metric, the worst (Mean 11-by-11) was able to remove more than 40%[4] of the noise. The best filter (Median 1-by-1) was able to remove more than 90%[5] of the noise.

[4] $1 - 295.76/533.9996 = 45\%$
[5] $1 - 38.5722/533.9996 = 93\%$

References

[1] J. Astola and P. Kuosmanen. *Fundamentals of nonlinear digital filtering*. Electronic engineering systems series. CRC Press, 1997.

[2] T. Bräunl, S. Feyrer, W. Rapf, and M. Reinhardt. *Parallel image processing*. Springer Verlag, 2001.

[3] T. Huang, G. Yang, and G. Tang. A fast two-dimensional median filtering algorithm. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 27(1):13–18, 1979.

[4] S. Perreault and P. Hébert. Median filtering in constant time. *Image Processing, IEEE Transactions on*, 16(9):2389–2394, 2007.

[5] U. Reiter. Perceived quality in consumer electronics-from quality of service to quality of experience. In *Consumer Electronics, 2009. ISCE'09. IEEE 13th International Symposium on*, pages 958–961. IEEE.