



UNIVERSITEIT  
GENT

# DE IMPLEMENTATIE EN VERGELIJKING VAN WISKUNDIGE MODELLEN IN HET KADER VAN BEELDVERWERKING

**Bjarne Dewilde**

Promotor: Prof. Dr. Marian Slodička  
Copromotor: Dr. Karel Van Bockstal

Masterproef ingediend tot het behalen van de academische graad van  
Master of Science in de industriële wetenschappen: informatica

Academiejaar: 2018 - 2019



# DE IMPLEMENTATIE EN VERGELIJKING VAN WISKUNDIGE MODELLEN IN HET KADER VAN BEELDVERWERKING

**Bjarne Dewilde**

Promotor: Prof. Dr. Marian Slodička  
Copromotor: Dr. Karel Van Bockstal

Masterproef ingediend tot het behalen van de academische graad van  
Master of Science in de industriële wetenschappen: informatica

Academiejaar: 2018 – 2019



## Woord vooraf

In februari begon ik ietwat onzeker doch nieuwsgierig aan mijn masterproef. Nu, vier maand later, schrijf ik de laatste regels van mijn thesis. Een semester consequent en zelfstandig doorwerken heeft me niet alleen op professioneel maar ook op persoonlijk vlak veel bijgeleerd. Zo transformeerden mijn initiële onzekerheid en afwachtende houding al snel in zelfvertrouwen en een jeugdig enthousiasme. Ik wil elke toekomstige masterstudent dan ook geruststellen dat, mits het vereiste doorzettingsvermogen, een masterproef een project kan zijn waar met plezier aan gewerkt kan worden.

Uiteraard zou deze masterproef niet succesvol afgerond kunnen zijn zonder een aantal belangrijke personen. Daarom zou ik graag mijn promotor professor Marian Slodička willen bedanken voor de begeleiding en wiskundige uitleg doorheen het semester. Ook mijn copromotor, doctor Karel Van Bockstal, verdient een bedanking voor het meermaals nalezen van deze thesis en het geven van constructieve feedback.

Verder wil ik ook mijn ouders, zus en proefpersonen bedanken om tijd vrij te maken voor het gedeeltelijk nalezen en uittesten van de applicatie.

Tot slot bedank ik graag nog mijn vrienden, met in het bijzonder Jonathan en Dik Druifje, voor de vele mooie momenten in de laatste jaren van de opleiding.

# Toelating tot bruikleen

“De auteur geeft de toelating deze scriptie voor consultatie beschikbaar te stellen en delen van de scriptie te kopiëren voor persoonlijk gebruik.

Elk ander gebruik valt onder de beperkingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze scriptie.”

Bjarne Dewilde, juni 2019

# The implementation and comparison of mathematical models in the context of image processing

Bjarne Dewilde

Supervisor(s): Marian Slodička, Karel Van Bockstal

*Abstract-* Basic mathematical models to reduce the amount of noise in an image cannot retain fine edges and details, which limits their practical applications. To solve this problem more advanced nonlinear diffusion equations like Perona-Malik can be used. The goal of this thesis is to implement such models, experiment with their parameters and compare them with each other.

*keywords-* image processing, Gaussian noise, salt and pepper noise, Perona-Malik, fractional diffusion

## I. INTRODUCTION

An often-recurring problem in the development of digital images is the phenomenon of noise. By this is meant any arbitrary deviation in brightness or colour damaging an image. Noise can occur when taking a photo in unfavourable conditions such as low light, a low shutter speed or a too high ambient temperature. With the necessary precautions and good camera settings, the amount of noise can be limited. However, a completely noise-free image can never be guaranteed. Therefore, some mathematical models were developed to specifically restore damaged images due to noise. An example of such model is to simply replace every pixel-value with the mean pixel-value of its neighbours. Although this technique effectively reduces the amount of noise in an image, it also blurs sharp edges and details, thus further reducing the quality of the image. This problem can be avoided by including information of structures that must be preserved, a property that e.g. the Perona-Malik (nonlinear diffusion) equation possesses.

The goal of this thesis is to implement more advanced mathematical models like Perona-Malik and fractional diffusion, in order to reduce the amount of noise in an image whilst retaining edges and details. Hereafter, the developed application will be used to compare the implemented models and examine their input parameters.

## II. TYPES OF NOISE

This thesis will focus on the reduction of two well-known types of noise: Gaussian noise and “salt and pepper noise”.

### A. Gaussian noise

Gaussian noise is a form of statistical noise whose probability density function is the well-known normal distribution or Gaussian distribution (hence the name). In other words, all damaged pixels will assume a value that is normally distributed as shown in Figure 1. This form of noise is very important in practice since many natural phenomena such as blood pressure and IQ-scores show the same Gaussian-distributed pattern.

### B. Salt-and-pepper noise

Another less common form of noise is the so-called “salt-and-pepper noise”, also known as impulse noise. This occurs due to sharp or sudden disturbances in the analogue signal of the image, which after conversion results into completely black or white pixels. An example of such a damaged picture is shown in Figure 2.

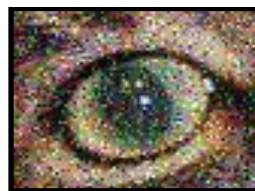


Figure 1: Gaussian noise



Figure 2: salt-and-pepper noise

### III. MATHEMATICAL MODELS

#### A. Perona-Malik

The Perona-Malik equation is a nonlinear diffusion equation that is used in image processing to filter out noise and sharpen edges. The original equation was introduced in [1] as follows:

$$u_t = \nabla \cdot (g(|\nabla u|)\nabla u) \quad (1)$$

with

$$u : [0, T] \times \Omega \rightarrow \mathbb{R} : (t, x) \mapsto u(t, x) \quad (2)$$

the image as a function of time and place and

$$g : \mathbb{R} \rightarrow \mathbb{R}^+ : x \mapsto g(x) \quad (3)$$

the diffusion coefficient.

The gradient  $\nabla u(x, t)$  is used to determine the location of the edges in an image. This is a mathematical concept that represents the measure of deviation. In the context of image processing, the gradient of a pixel will give an indication to what extent the pixel value deviates from the pixel values of its neighbours. Specifically, a large gradient will indicate a lot of deviation, thus the pixel can be considered as part of an edge. A small gradient means little to no change in neighbouring pixel values indicating a smooth surface. With the gradient as an estimator, the diffusion coefficient can therefore be selected as a function of  $|\nabla u|$ . In [1], the following diffusion functions were proposed:

$$g_1(x) = e^{-\frac{1}{2}\left(\frac{x}{k}\right)^\alpha} \quad (4)$$

and

$$g_2(x) = \frac{1}{1 + (x/k)^\alpha} \quad (5)$$

where  $k$  is called the contrast parameter who serves as a threshold (from which value a gradient will be regarded as an edge). This implementation uses the in [2] proposed value

$$k = 1.4826 \text{ MAD} (|\nabla u_0|), \quad (6)$$

where MAD (= median absolute variation) is a robust estimator for standard deviation and is defined as

$$\text{MAD} (|\nabla u_0|) = \text{median}(|\nabla u_0 - \text{median}(|\nabla u_0|)|) \quad (7)$$

A numerical scheme needs to be established to approximate the solution of (1). This can be done by approximating the time derivative by backward Euler method, which results in an elliptic equation at every time step. Hereafter, these linear equations are

discretised in space using the finite volume method. A more in-depth elaboration can be found in [3].

#### B. Regularised Perona-Malik

The Perona-Malik equation can behave itself locally as a backward heat equation. This results in an ill-posed problem because such equations do not have a unique solution that continuously depends on the given data. As a result, the Perona-Malik comparison is expected to yield poor results. However, in practice this is not the case and it works much better than is theoretically expected. The only observed instability is the so-called step-shaped effect, in which a smooth edge evolves into individual segments separated by jumps.

This problem can be converted into a well-defined problem through regularization. This can be achieved by replacing  $g(|\nabla u|)$  with  $g(|\nabla G_\sigma * u|)$ , where  $G_\sigma$  is the Green function given by

$$G_\sigma(x) = \frac{1}{4\pi\sigma} \exp\left(-\frac{|x|^2}{4\sigma}\right) \quad (8)$$

and ‘\*’ denotes the convolution product.

The regularized Perona-Malik equation is supplemented with an additional parameter  $\sigma$  (sigma) in comparison with the classical Perona-Malik model. This regularized model approximates the original model because  $\nabla G_\sigma * u \rightarrow \nabla u$  as  $\sigma \rightarrow 0$ . It also retains all the practical features of the original Perona-Malik model. An extensive study on this regularization, using the Rothe method, can be found in [3].

#### C. Fractional diffusion

A different approach uses fractional calculus to remove noise from images. The global adaptive fractional integral algorithm (GAFIA) described in [4] will be implemented in this thesis. Herein, fractional integration is applied to each pixel whereby the optimum orders per pixel are obtained via the average gradient in four or eight directions depending on the used mask.

The three most common expressions used in fractional diffusion are Grunwald-Letnikov (G-L), Riemann-Liouville (R-L) and Caputo. In this work, the formulation according to G-L based on integer-order differentials is used. The Gamma function is used to extend this integer order to a rational order. If the order  $\nu < 0$ , then the fractional integral according to G-L is approximated as



$$I_{G-L}^{\nu} f(t) \approx f(t) + (-\nu)f(t-1) + \frac{(-\nu)(-\nu+1)}{2}f(t-2) + \dots + \frac{\Gamma(-\nu+1)}{n! \Gamma(-\nu-n+1)}f(t-n) \quad (9)$$

Since the partial fractional integral of a two-dimensional digital image is defined in eight directions, a mask can be set up in four or eight dimensions. To distinguish these possibilities, the simplest mask (four dimensions) is assigned the name “std”, while the mask in eight dimensions is named after one of the authors of [4]: “boli”. Both masks can be found in Figure 3 and Figure 4, respectively. The parameters  $\xi$  are represented by the different coefficients from (10):

$$\begin{cases} \xi_1 = 1 \\ \xi_2 = -\nu \\ \xi_3 = \frac{(-\nu)(-\nu+1)}{2} \\ \vdots \\ \xi_n = \frac{\Gamma(-\nu+1)}{n! \Gamma(-\nu-n+1)} \end{cases}, (n \in N, G-L \text{ definition}) \quad (10)$$

0	0	...	0	$\xi_n$	0	...	0	0
0	.	.	:	:	:	.	.	0
:	.	0	0	$\xi_3$	0	0	.	:
0	...	0	0	$\xi_2$	0	0	...	0
$\xi_n$	...	$\xi_3$	$\xi_2$	$4\xi_1$	$\xi_2$	$\xi_3$	...	$\xi_n$
0	...	0	0	$\xi_2$	0	0	...	0
:	.	0	0	$\xi_3$	0	0	.	:
0	.	.	:	:	:	.	.	0
0	0	...	0	$\xi_n$	0	...	0	0

Figure 3: std-mask

$\xi_n$	0	...	0	$\xi_n$	0	...	0	$\xi_n$
0	.	.	:	:	:	.	.	0
:	.	$\xi_3$	0	$\xi_3$	0	$\xi_3$	.	:
0	...	0	$\xi_2$	$\xi_2$	$\xi_2$	0	...	0
$\xi_n$	...	$\xi_3$	$\xi_2$	$8\xi_1$	$\xi_2$	$\xi_3$	...	$\xi_n$
0	...	0	$\xi_2$	$\xi_2$	$\xi_2$	0	...	0
:	.	$\xi_3$	0	$\xi_3$	0	$\xi_3$	.	:
0	.	.	:	:	:	.	.	0
$\xi_n$	0	...	0	$\xi_n$	0	...	0	$\xi_n$

Figure 4: boli-mask

The fractional order  $\nu$  is variable per pixel and is determined by the average gradient of each pixel in four or eight directions depending on the used mask. This calculation in eight dimensions is represented as

$$M(i, j) = \frac{1}{8}(|f(i, j) - f(i-1, j-1)| + |f(i, j) - f(i-1, j)| + |f(i, j) - f(i-1, j+1)| + |f(i, j) - f(i, j-1)| + |f(i, j) - f(i, j+1)| + |f(i, j) - f(i+1, j-1)| + |f(i, j) - f(i+1, j)| + |f(i, j) - f(i+1, j+1)|) \quad (11)$$

wherein  $f(i, j)$  represents the pixel value of the pixel with coordinates  $(i, j)$ .

If the smallest and largest average gradient of all pixels in the image are denoted by X and Y respectively, then the fractional order of any pixel with coordinates  $(i, j)$  in eight directions is represented by:

$$\nu_{GAFIA} = (-1) \times \frac{M(i, j) - X}{Y - X} \quad (12)$$

A damaged pixel produces a large average gradient resulting in a small, negative order and large blur. Before the std- or boli-mask can be used, each element must be divided by four or eight times the sum of all the different coefficients in the mask. Only after this can the image be processed by convolving each pixel with the selected mask and the optimum order.

#### IV. IMPLEMENTATION

A console application written in Python was developed. All mentioned models are implemented and can be used to enhance a damaged image. Console applications are usually faster but less user friendly than graphical user interfaces. Since the application will be used by non-computer scientists, it is very important to avoid long and difficult commands. This was achieved by implementing a series of consecutive menus through which the user specifies its choice by typing the corresponding number. An example of such menu can be found in Figure 5. A series of user tests proved the efficiency and user friendliness of this approach.

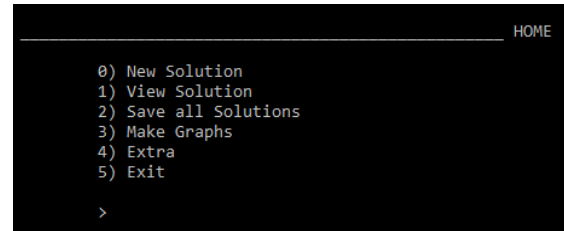


Figure 5: The Home menu

In total six mathematical models were implemented: three basic techniques whom do not preserve edges

and details (mean, Gaussian blur & median) and the three in Section III described advanced models (Perona-Malik, regularised Perona-malik & fractional diffusion). All models are capable to enhance both colour and greyscale images.

To investigate the influence of the different models and associated parameters, a quantitative assessment method must be established. After all, it is not possible to visually compare the different end results with each other and to objectively determine the best results. Instead, the so-called "reference Image Quality Assessment" algorithms can be used to obtain steady assessments. These methods compare the enhanced image with the ideal non-damaged image and provide a score based on their similarity. The application implements the peak signal-to-noise ratio (PSNR), mean squared error (MSE) and structural similarity index (SSI) algorithms. Throughout the experiments both MSE and SSI values are mostly used to examine the quality of the end results. The MSE value can be interpreted as the amount of deviating pixel values between two images, whilst the SSI value can be interpreted as a percentage that represents the amount of structural equality between two images. Therefore, a low MSE and high SSI (max=1=100%) value are desired. The application offers some premade graphs and tables that allows the user to quickly examine the quality of the resulted images.

Moreover, processed images can be exported or saved as Python objects for later use.

A working MATLAB implementation of both the standard and regularized Perona-Malik model was already available in [3]. This code had to be translated into Python to achieve a working Python implementation. This went quite smoothly because the most essential MATLAB methods are also available in Python. Consequently, large pieces of code could be copied after changing the basic syntax.

The GAFIA-method was more difficult to implement because each pixel had to be treated individually. Therefore, a double "for-loop" was necessarily, which makes the implementation rather slow for large images.

## V. COMPARISON

In this section, all implemented models are compared in order to discover which method provides the best image enhancements. This is achieved by performing three experiments, which sequentially focus on gaussian noise removal, salt-and-pepper noise removal and preserving fine edges and details.

### A. Gaussian noise removal

In this experiment all models will be used to enhance the damaged image shown in Figure 1. Keep in mind that this image is severely damaged and such amounts of noise rarely occur in reality. Each model continues its calculations until both MSE and SSI assessments reach their optimal values.

The improved images corresponding with the best MSE values are shown in Figure 6. The optimal MSE and SSI values themselves are shown in Table 1, supplemented with the iteration and time (in seconds) in which they were found.

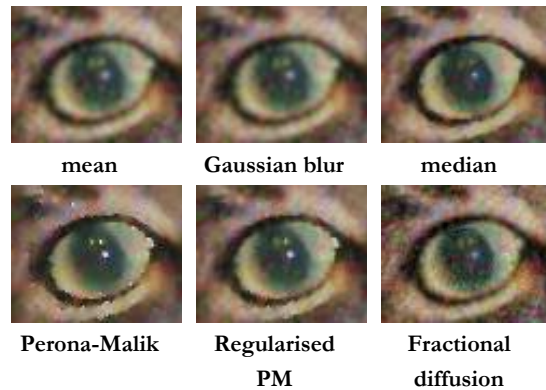


Figure 6: End results of the image damaged by Gaussian noise

Table 1: The best MSE and SSI values after improving the image damaged by Gaussian noise

Variable	Best mse	iteration	time
Method=median	216.4893	2	1.856
Method=PeronaMalikReg	228.6822	3	25.454
Method=PeronaMalik	239.7885	6	37.379
Method=gaussian	259.0605	3	3.296
Method=mean	260.007	2	2.411
Method=fractionalDiffusion	277.2311	12	429.636

Variable	Best ssi	iteration	time
Method=median	0.6585	9	9.41
Method=PeronaMalik	0.6572	8	49.917
Method=PeronaMalikReg	0.644	5	43.098
Method=gaussian	0.6123	7	7.324
Method=mean	0.6118	5	5.687
Method=fractionalDiffusion	0.5348	15	553.404

Surprisingly enough these results indicate that the basic method using medians achieves the highest improvement rate. Both its MSE (lower=better) and SSI (higher=better) values are better than those of the other models. The regularised Perona-Malik equation delivers the second-best MSE value whilst the default Perona-Malik equation delivers the second-best SSI value.

Another interesting observation is that fractional diffusion results in the worst MSE and SSI values.

### B. Salt-and-pepper noise removal

This second experiment focusses on the reduction of salt-and-pepper noise. As before all models are tasked to enhance a damaged image (Figure 2) until both MSE and SSI reach their optimal values. The best enhancements (according to MSE) are once again collected in Figure 7. The corresponding MSE and SSI values are shown in Table 2.

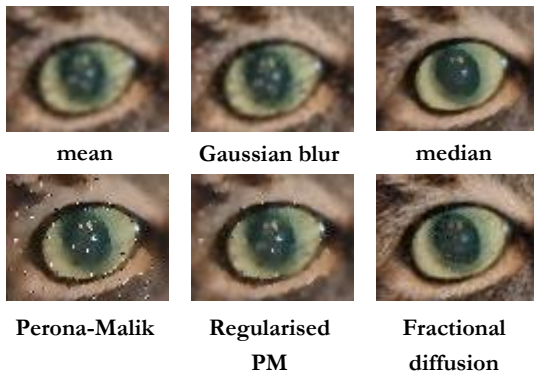


Figure 7: End results of the image damaged by salt-and-pepper noise

Table 2: The best MSE and SSI values after improving the image damaged by salt-and-pepper noise

Variable	Best mse	iteration	time
Method=fractionalDiffusion	79.4844	13	91.104
Method=median	89.6748	1	1.103
Method=PeronaMalikReg	184.8851	4	32.141
Method=gaussian	189.0585	2	2.157
Method=mean	197.4266	2	1.943
Method=PeronaMalik	324.2425	7	38.885

Variable	Best ssi	iteration	time
Method=median	0.8764	1	1.031
Method=fractionalDiffusion	0.8269	21	172.724
Method=PeronaMalikReg	0.711	5	39.72
Method=gaussian	0.6793	4	4.258
Method=mean	0.678	3	3.3
Method=PeronaMalik	0.6216	7	43.503

In contrast to the experiment in the previous section, fractional diffusion delivers the best MSE value and the second-best SSI value, whilst in this case the default Perona-Malik equation delivers the worst results. A possible explanation can be found in the fact that salt-and-pepper noise only damages a limited number of pixels. In Figure 2 for example, 90% of all pixels are not damaged, thus they do not have to be “improved”. Because the fractional diffusion implementation runs through each pixel individually, it can decide to only change pixels with very high gradients and thus effectively ignoring undamaged pixels. The other models, including Perona-Malik, will alter all pixels and thus already perfect pixel values are changing, which results in worse MSE and SSI values. As before, medians once again yield remarkably good results. The other two basic methods (“mean” and “Gaussian”) are less effective.

### C. Preserving fine edges and details

Previous experiments showed how the simple method based on medians produces good results when removing both Gaussian and salt-and-pepper noise. However, in practical applications (like MRI scans) it’s also very important to retain fine edges and details. For this reason, a final experiment examines how well each method retains such edges and details. The image shown in Figure 8 was specially designed for this test and is composed of several fine edges that represent a simple pattern.

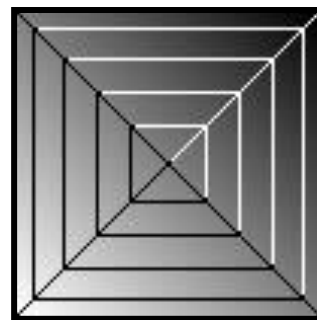


Figure 8: an image with fine lines

Unlike the previous experiments, no noise is added to this image. As a result, the MSE and SSI values are no longer relevant. Specifically, each model will be used to improve this image as if it were damaged by noise. Since this isn’t the case, the “improved” images shouldn’t deviate from the original one.

The various results obtained after two iterations are shown for each method in Figure 9.

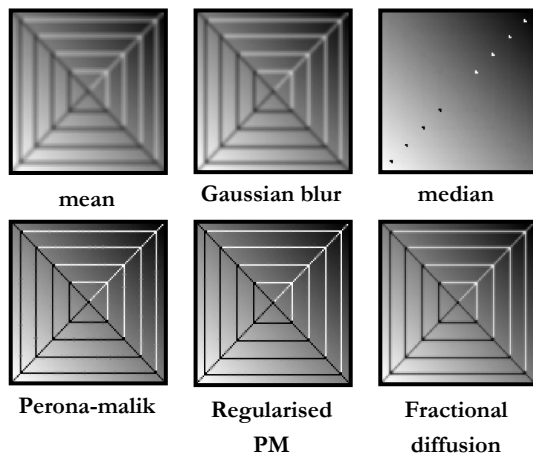


Figure 9: End results of the image with fine lines

These results show how the basic methods fail to retain the simple pattern. Gaussian blurring and averaging result in blurry edges, whilst using medians leads to their overall removal. It isn't difficult to imagine that such effects can have catastrophic consequences in medical applications. Fortunately, these edges are well preserved using the (regularized) Perona-Malik equation and fractional diffusion. Therefore, it's clear how these advanced models, despite their long calculation time, can certainly come in handy in practical applications where the retention of fine edges and details is an absolute necessity.

## VI. CONCLUSION

In this thesis a user-friendly console application written in Python was developed. The user is able to reduce the amount of noise in images by aid of six implemented mathematical models. This are three basic methods based on averages, Gaussian blurring and medians; and three advanced techniques based on the Perona-Malik equation, a regularized Perona-Malik equation and fractional diffusion. In addition, the application offers the possibility to experiment and investigate the implemented models using three objective assessment methods: the Peak signal-to-noise ratio (PSNR), the mean squared error (MSE) and the structural similarity index (SSI). Using these assessment methods, the program is able to plot graphs and tables that can be used to compare the various end results.

In a second phase, the developed application was used to compare the implemented models. This showed

how the (regularised) Perona-Malik equation excels in removing Gaussian noise whilst fractional diffusion specializes in the removal of salt-and-pepper noise. Both types of noise can effectively be reduced by using the basic method based on medians. However, a final experiment showed how fine edges and details cannot be retained using such basic models thus making them useless in most practical applications.

## REFERENCES

- [1] P. Perona en J. Malik, „Scale-space and edge detection using anisotropic diffusion,” IEEE Transactions on pattern analysis and machine intelligence, 1990.
- [2] M. Black, G. Sapiro, D. Marimont en D. Heeger, „Robust anisotropic diffusion,” in IEEE Transactions on image processing, 1998, p. 421–432.
- [3] A. Devos, „digitale beeldverwerking met behulp van partiële differentiaalvergelijkingen,” UGent, Gent, België, 2017.
- [4] B. Li en W. Xie, „Image denoising and enhancement based on adaptive fractional calculus of small probability strategy,” South China University of Technology, Guangzhou 510640, China, 2015.

# Inhoudsopgave

<b>Inhoudsopgave</b>	<b>7</b>	
<b>Lijst van figuren</b>	<b>8</b>	
<b>Lijst van tabellen</b>	<b>10</b>	
<b>Inleiding</b>	<b>11</b>	
<b>1</b>	<b>Digitale afbeeldingen</b>	<b>13</b>
1.1	Pixel-structuur	13
1.2	Voorstelling in het geheugen	15
1.3	Bestand extensies	17
<b>2</b>	<b>Probleemstelling</b>	<b>18</b>
2.1	Gaussische ruis	18
2.2	Salt-and-pepper noise	18
2.3	Eenvoudige technieken	19
2.4	Doel masterproef	20
<b>3</b>	<b>Wiskundige modellen</b>	<b>21</b>
3.1	De Perona-Malik vergelijking	21
3.2	De geregulariseerde Perona-Malik vergelijking	24
3.3	Fractionele diffusie	25
<b>4</b>	<b>Implementatie</b>	<b>28</b>
4.1	Gebruikte technologieën	28
4.2	Console Applicatie	31
4.3	Testen	39
<b>5</b>	<b>Functionaliteiten en Gebruikershandleiding</b>	<b>40</b>
5.1	Home-menu	40
5.2	New solution	41
5.3	View solution	44
5.4	Save all solutions	45
5.5	Make graphs	45
<b>6</b>	<b>Experimenteren en parameterstudie</b>	<b>49</b>
6.1	Standaard parameterwaarden	49
6.2	Experimenteren: Perona-Malik	51
6.3	Experimenteren: geregulariseerde Perona-Malik	77
6.4	Experimenteren: Fractionele Diffusie	82
6.5	Vergelijking Perona-Malik en Fractionele Diffusie	88
<b>7</b>	<b>Algemeen besluit</b>	<b>98</b>
<b>Referenties</b>	<b>99</b>	

# Lijst van figuren

<b>Figuur 1.1:</b> De pixel representatie van een digitale afbeelding [3] .....	13
<b>Figuur 1.2:</b> De invloed van de gebruikte bit-diepte [5].....	14
<b>Figuur 1.3:</b> De voorstelling van een kleurenafbeelding in het geheugen met een bit-diepte van 8 bits .....	15
<b>Figuur 1.4:</b> De voorstelling in het geheugen van de digitale afbeelding na het verticaal spiegelen .....	16
<b>Figuur 2.1:</b> Een voorbeeld van Gaussische ruis.....	18
<b>Figuur 2.2:</b> Een voorbeeld van salt-and-pepper noise .....	18
<b>Figuur 2.3:</b> Een genormaliseerde box filter .....	19
<b>Figuur 2.4:</b> Een Gaussische kern.....	19
<b>Figuur 3.1:</b> Het std-masker .....	26
<b>Figuur 3.2:</b> Het boli-masker .....	26
<b>Figuur 4.1:</b> De main functie.....	32
<b>Figuur 4.2:</b> De _home functie.....	32
<b>Figuur 4.3:</b> Het berekenen van een gradiënt in MATLAB (links) en Python (rechts).....	33
<b>Figuur 4.4:</b> De onderverdeling van het boli-masker in triviale componenten .....	34
<b>Figuur 4.5:</b> De MSE- en SSI-waarden bij verschillende helderheid .....	38
<b>Figuur 4.6:</b> De MSE- en SSI-waarden bij verschillende structuur .....	38
<b>Figuur 5.1:</b> Het home-menu .....	40
<b>Figuur 5.2:</b> Het new solution-menu.....	42
<b>Figuur 5.3:</b> De parameterwaarden voor de generatie van vijf oplossingen waarbij de a-waarde verhoogd wordt van één tot vijf in stappen van één.....	42
<b>Figuur 5.4:</b> Een voorbeeld van een correct opgesteld <i>solution file</i> .....	42
<b>Figuur 5.5:</b> Het view solution-menu.....	45
<b>Figuur 5.6:</b> Het make graphs-menu .....	45
<b>Figuur 5.7:</b> Een voorbeeld van de MSE- <i>assessment graph</i> en bijhorende tabel.....	46
<b>Figuur 5.8:</b> Een voorbeeld van de MSE <i>extremum graph</i> en bijhorende tabel.....	46
<b>Figuur 5.9:</b> Een voorbeeld van de MSE <i>improvement graph</i> en bijhorende tabel.....	47
<b>Figuur 5.10:</b> Een voorbeeld van de <i>CPU time graph</i> en bijhorende tabel.....	47
<b>Figuur 6.1:</b> De standaardafbeelding flor.jpg gebruikt doorheen de experimenten .....	50
<b>Figuur 6.2:</b> flor.jpg beschadigd met 50% Gaussische ruis .....	50
<b>Figuur 6.3:</b> Eindresultaat flor.jpg met a=1 .....	52
<b>Figuur 6.4:</b> Eindresultaat flor.jpg met a=5 .....	52
<b>Figuur 6.5:</b> Het verloop van diffusie-functie (3.8) met veranderlijke a-waarden.....	52
<b>Figuur 6.6:</b> Verschillende hoeveelheden initiële ruis .....	53
<b>Figuur 6.7:</b> Eindresultaten bekomen met verschillende hoeveelheden initiële ruis .....	53
<b>Figuur 6.8:</b> De MSE- <i>assessment graph</i> met veranderlijke hoeveelheid initiële ruis .....	54
<b>Figuur 6.9:</b> De MSE <i>improvement graph</i> met veranderlijke hoeveelheid initiële ruis .....	55
<b>Figuur 6.10:</b> De CPU-tijden (in seconden) met veranderlijke hoeveelheid initiële ruis .....	55
<b>Figuur 6.11:</b> De beschadigde en verbeterde afbeelding voor verschillende afmetingen.....	57
<b>Figuur 6.12:</b> De rekentijd per iteratie (in seconden) voor verschillende afmetingen .....	57
<b>Figuur 6.13:</b> De MSE- <i>assessment graph</i> voor verschillende afmetingen .....	58
<b>Figuur 6.14:</b> De SSI- <i>assessment graph</i> voor verschillende afmetingen .....	59
<b>Figuur 6.15:</b> borders.jpg (links) en noborders.jpg (rechts).....	60
<b>Figuur 6.16:</b> Eindresultaten borders.jpg met a=2.....	63
<b>Figuur 6.17:</b> Eindresultaten noborders.jpg met a=1 .....	63
<b>Figuur 6.18:</b> De vier afbeeldingen met veranderlijke blokgroottes .....	64
<b>Figuur 6.19:</b> colors.jpg in grijstinten (links) en kleur (rechts) .....	66

<b>Figuur 6.20:</b> Eindresultaten colors.jpg in kleur (links) en zwart-wit (rechts) .....	67
<b>Figuur 6.21:</b> De MSE- <i>assessment graph</i> voor de verschillende diffusie-functies (3.8) en (3.9) .....	71
<b>Figuur 6.22:</b> Eindresultaten flor.jpg voor beide diffusie-functies met $a=4$ .....	71
<b>Figuur 6.23:</b> Het verloop van beide diffusie-functies (3.8) en (3.9) met $a=4$ .....	72
<b>Figuur 6.24:</b> De MSE- <i>assessment graph</i> met veranderlijke tijdstap $< 1$ .....	73
<b>Figuur 6.25:</b> De MSE- <i>assessment graph</i> met veranderlijke tijdstap $> 1$ .....	74
<b>Figuur 6.26:</b> De MSE <i>extremum graph</i> met veranderlijke tijdstap van een afbeelding met 20% (boven), 50% (midden) en 80% (onder) initiële ruis .....	75
<b>Figuur 6.27:</b> De SSI- <i>assessment graph</i> met veranderlijke eindtijd (finaltime) .....	76
<b>Figuur 6.28:</b> De afbeelding frul.jpg, gebruikt om de invloed van de gekozen sigma-waarde na te gaan .....	78
<b>Figuur 6.29:</b> De rektijden per iteratie (in seconden) voor de gewone en geregulariseerde PM-vergelijking .....	78
<b>Figuur 6.30:</b> Een vergelijking tussen de geregulariseerde (links) en de gewone PM-vergelijking (rechts) .....	80
<b>Figuur 6.31:</b> Eindresultaten flor.jpg veranderlijke maskergrootte (maskSize) .....	83
<b>Figuur 6.32:</b> De rektijd per iteratie met veranderlijke maskergrootte (maskSize) .....	83
<b>Figuur 6.33:</b> Eindresultaten flor.jpg voor beide maskers .....	85
<b>Figuur 6.34:</b> De rektijd per iteratie (in seconden) met veranderlijke schatting hoeveelheid ruis (noiseEstimation) .....	86
<b>Figuur 6.35:</b> Eindresultaten flor.jpg beschadigd met 10% salt-and-pepper noise met een schatting van 10% (links) en 100% hoeveelheid initiële ruis (rechts) .....	87
<b>Figuur 6.36:</b> Eindresultaten flor.jpg met 20% Gaussische ruis via alle modellen (I) .....	90
<b>Figuur 6.37:</b> Eindresultaten flor.jpg met 20% Gaussische ruis via alle modellen (II) .....	90
<b>Figuur 6.38:</b> De MSE- <i>assessment graph</i> van alle modellen met veranderlijke afmetingen .....	91
<b>Figuur 6.39:</b> De SSI- <i>assessment graph</i> van alle modellen met veranderlijke afmetingen .....	91
<b>Figuur 6.40:</b> Het verlies aan contrast bij Perona-Malik t.o.v. de eenvoudige methode op basis van medianen .....	92
<b>Figuur 6.41:</b> flor.jpg beschadigd met 10% salt-and-pepper noise .....	93
<b>Figuur 6.42:</b> Eindresultaten flor.jpg met 10% salt-and-pepper noise via alle modellen (I) .....	95
<b>Figuur 6.43:</b> Eindresultaten flor.jpg met 10% salt-and-pepper noise via alle modellen (II) .....	95
<b>Figuur 6.44:</b> De afbeelding lines.png gebruikt om het behoud van fijne randen en details te testen. ....	96
<b>Figuur 6.45:</b> Eindresultaten lines.png na 2 iteraties via alle modellen .....	97

## Lijst van tabellen

<b>Tabel 5.1:</b> De verschillende invoerparameters .....	43
<b>Tabel 6.1:</b> De gebruikte standaardparameterwaarden.....	49
<b>Tabel 6.2:</b> De beste MSE- en SSI-waarden bekomen met veranderlijke a-waarden .....	51
<b>Tabel 6.3:</b> De beste MSE-waarden met veranderlijke hoeveelheid initiële ruis .....	54
<b>Tabel 6.4:</b> De totale MSE-verbetering met veranderlijke hoeveelheid initiële ruis .....	55
<b>Tabel 6.5:</b> De invloed van de parameter a bij verschillende hoeveelheden initiële ruis.....	56
<b>Tabel 6.6:</b> De totale CPU-tijden (in seconden) voor verschillende afmetingen .....	57
<b>Tabel 6.8:</b> De invloed van de parameter a bij verschillende afmetingen .....	59
<b>Tabel 6.9:</b> De beste MSE- en SSI- waarden voor scherpe en vloeiende randen met veranderlijke a-waarden.....	61
<b>Tabel 6.10:</b> De totale MSE-verbetering voor scherpe en vloeiende randen .....	62
<b>Tabel 6.11:</b> De beste MSE- en SSI-waarden voor veranderlijke blokgroottes.....	64
<b>Tabel 6.12:</b> De totale SSI-verbetering voor veranderlijke blokgroottes .....	64
<b>Tabel 6.13:</b> De invloed van de parameter a bij verschillende blokgroottes.....	65
<b>Tabel 6.14:</b> De beste MSE-waarden van colors.jpg in kleur en zwart-wit met veranderlijke a-waarden.....	66
<b>Tabel 6.15:</b> De beste MSE-waarden van flor.jpg in kleur en zwart-wit .....	68
<b>Tabel 6.16:</b> De totale CPU-tijden (in seconden) van colors.jpg in kleur en zwart-wit met veranderlijke a-waarden ..	68
<b>Tabel 6.17:</b> De optimale a-waarde volgens MSE en SSI in kleur, zwart-wit en monochroom .....	69
<b>Tabel 6.18:</b> De beste MSE- en SSI-waarden per diffusie functie met veranderlijke a-waarden .....	70
<b>Tabel 6.19:</b> De beste a en bijhorende SSI-waarden bekomen via optie 0 met veranderlijke hoeveelheden ruis.....	72
<b>Tabel 6.20:</b> De beste a en bijhorende SSI-waarden bekomen via optie 1 met veranderlijke hoeveelheden ruis .....	72
<b>Tabel 6.21:</b> De beste MSE-waarden met veranderlijke tijdstap < 1 .....	73
<b>Tabel 6.22:</b> De beste MSE-waarden met veranderlijke tijdstap > 1.....	74
<b>Tabel 6.23:</b> De beste rekentijd (in seconden) met veranderlijke tijdstap.....	74
<b>Tabel 6.25:</b> De beste MSE- en SSI- waarden met veranderlijke diffusie-functie en a-waarden voor de geregulariseerde (links) en gewone PM-vergelijking (rechts) .....	81
<b>Tabel 6.26:</b> Het MSE-verschil tussen het beste en slechtste resultaat met veranderlijke a-waarden per diffusie functie .....	81
<b>Tabel 6.30:</b> De beste MSE-waarden met veranderlijke schatting hoeveelheid ruis (noiseEstimation).....	86
<b>Tabel 6.31:</b> De beste MSE-waarden met veranderlijke schatting hoeveelheid ruis (noiseEstimation) voor 10% salt- and-pepper noise.....	87
<b>Tabel 6.32:</b> De beste MSE- en SSI-waarden voor alle modellen met 20, 50 en 80 procent Gaussische ruis .....	89
<b>Tabel 6.33:</b> De beste MSE- en SSI-waarden voor alle modellen met 1, 5 en 10 procent salt-and-pepper noise.....	94



# Inleiding

Beeld u in: een adembenemende zonsondergang op een zwoele zomeravond. Het ongeziene schouwspel van licht en kleur wordt enthousiast vastgelegd met de beste digitale fotocamera voorhanden. Helaas blijkt het eindresultaat een slechte kopie van de werkelijkheid te zijn. Het is niet ongewoon dat het nemen van een foto resulteert in een afbeelding van matige tot slechte kwaliteit ten gevolge van ongunstige omgevingsfactoren of camera-instellingen. Een hardnekkig, wederkerend probleem is beschadiging door ruis.

Met behulp van partiële differentiaalvergelijkingen kan de hoeveelheid ruis in een afbeelding gereduceerd worden. De bekomen resultaten zijn volledig afhankelijk van het gebruikte model. De relatief eenvoudige technieken op basis van de lineaire warmtevergelijking zijn niet in staat een onderscheid te maken tussen effectieve schade ten gevolge van ruis en belangrijke details waardoor ze een weliswaar ruisvrije maar wazige afbeelding opleveren. Dit probleem kan vermeden worden door a priori informatie op te nemen van structuren die behouden moeten blijven, een eigenschap die de niet lineaire diffusievergelijking, zoals Perona-Malik, bezit. Het eerste aspect van deze thesis focust op de implementatie van dergelijke geavanceerde wiskundige modellen die in staat zijn ruis uit afbeeldingen te verwijderen terwijl randen en belangrijke details behouden blijven. In een tweede luik wordt geëxperimenteerd met de verschillende parameters om het effect op het bekomen eindresultaat na te gaan. Tot slot worden de basismodellen (“mean”, “gaussian”, “median”) en de geavanceerde modellen (“Perona-malik”, “geregulariseerde Perona-Malik”, “Fractionele diffusie”) onderling vergeleken om na te gaan of deze effectief betere resultaten opleveren.

## Inhoud

Alvorens de structuur van deze thesis te overlopen is het belangrijk te vermelden dat deze voortbouwt op de thesis van Arnaud Devos [1]. Hoewel het interessant kan zijn is het geen absolute noodzaak deze door te nemen aangezien de relevantste besluiten hieronder kort aan bod komen. Bijgevolg zullen sommige onderdelen overeenkomen en zal af en toe verwezen worden naar deze thesis. In een eerste hoofdstuk wordt de opbouw en efficiënte opslag van digitale afbeeldingen in het geheugen aangekaart. Ook enkele relevante bestandsextensies waarmee de eindgebruiker in contact kan komen worden kort besproken. Hoofdstuk twee behandelt de hierboven vermelde probleemstelling waarbij dieper ingegaan wordt op verschillende soorten ruis, eenvoudige technieken en doelstellingen van de masterproef.

In hoofdstuk drie volgt een relatief eenvoudige wiskundige beschrijving van de geïmplementeerde wiskundige modellen op niveau van een derde bachelor industrieel ingenieur. De belangrijkste concepten van de eigenlijke implementatie worden besproken in hoofdstuk vier. Hierbij wordt dieper ingegaan op de gebruikte technologieën en gemaakte beslissingen doorheen de ontwikkeling

van de applicatie. Een basiskennis van de elementaire programmeer concepten zoals functies en variabelen is hierbij aangewezen. Een gebruikershandleiding die eveneens de werking alsook alle functionaliteiten van het programma overloopt is terug te vinden in hoofdstuk vijf. Deze werd toegevoegd als naslagwerk voor de uiteindelijke eindgebruiker.

In hoofdstuk zes wordt de ontwikkelde applicatie gebruikt om de invloed van de verschillende parameters op het eindresultaat na te gaan. Tot slot wordt nog een algemeen besluit geformuleerd.

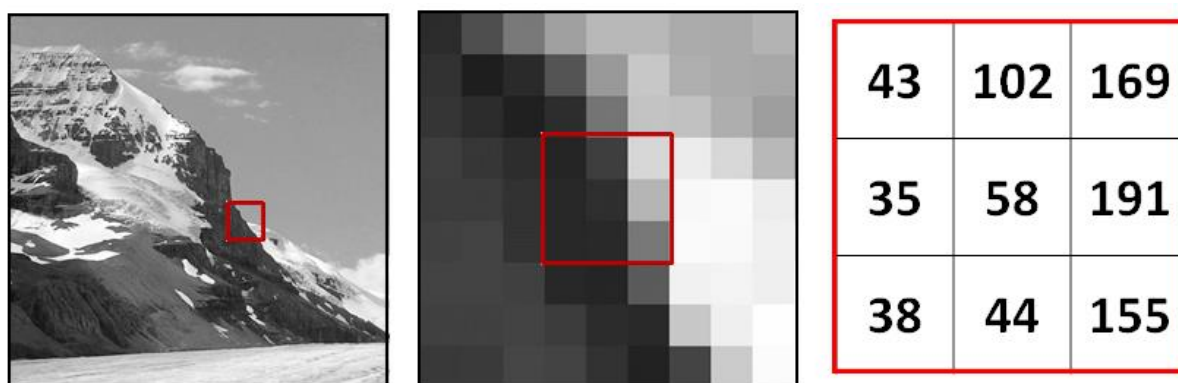
# 1 Digitale afbeeldingen

Afbeeldingen gemaakt op een computer of gedigitaliseerd uit andere digitale bronnen zoals digitale fotografie worden beschouwd als digitale afbeeldingen. De Nederlandse encyclopedie [2] beschrijft dergelijke afbeeldingen als volgt: “elektronische afbeeldingen opgeslagen in de vorm van elektronisch gecodeerde beeldelementen”.

In dit hoofdstuk wordt gefocust op de pixel-structuur en de verschillende relevante voorstellingen van digitale afbeeldingen. Verder wordt besproken hoe deze structuur efficiënt opgeslagen wordt in het geheugen en wat de meest voorkomende bestands-extensies zijn.

## 1.1 Pixel-structuur

Pixels, of *picture elements*, zijn de kleinste fundamentele basisblokken waaruit elke digitale afbeelding opgebouwd is. Praktisch zijn dit binaire woorden voorgesteld in het computergeheugen door een vast aantal bits  $k$ . Dit aantal wordt in de literatuur vaak de bit-diepte genoemd en zal bijgevolg bepalen hoeveel waarden elke pixel kan aannemen ( $= 2^k$ ). Visueel wordt elke pixel voorgesteld als een klein, vierkant oppervlak met egale kleur afhankelijk van de pixelwaarde. Na digitalisatie kan een afbeelding voorgesteld worden als een 2D matrix van pixelwaarden waarbij het aantal kolommen  $N$  en het aantal rijen  $M$  respectievelijk de breedte en hoogte van de afbeelding voorstellen. De afbeelding wordt dus beschouwd als een rooster van discrete elementen, geordend van boven naar onder en van links naar rechts, zoals geïllustreerd in Figuur 1.1.



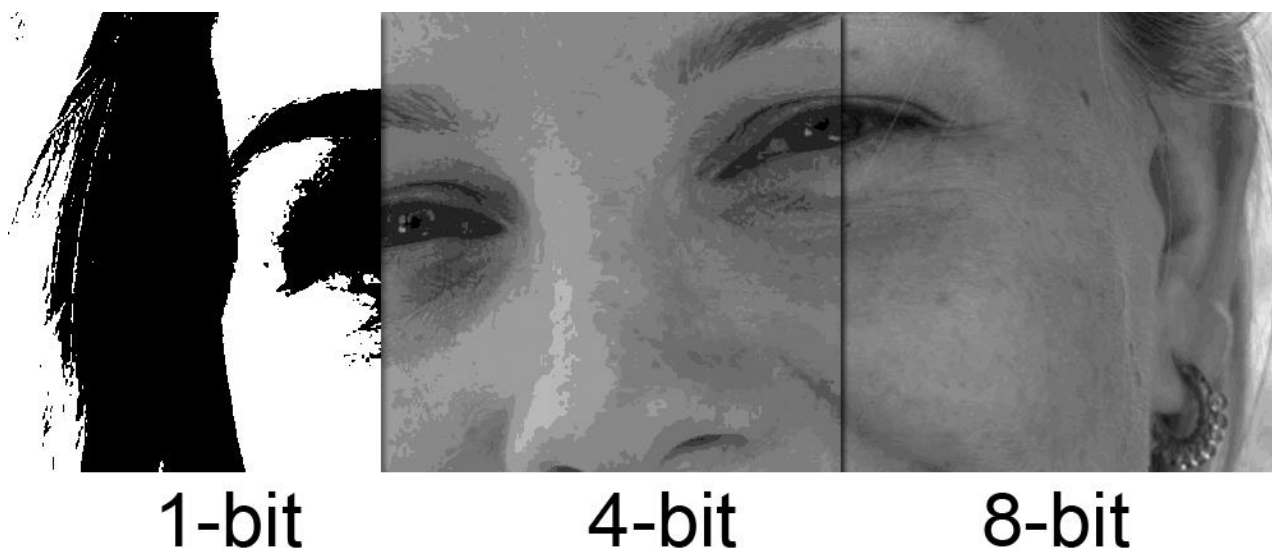
**Figuur 1.1:** De pixel representatie van een digitale afbeelding [3]

Een digitale afbeelding kan op verschillende manieren voorgesteld worden. De relevantste representaties zijn: binair, zwart-wit of RGB (kleur).

De eenvoudigste van deze weergaven is de binaire of monochrome vorm. Dergelijke afbeeldingen zijn opgebouwd uit pixels die slechts twee mogelijke waarden aannemen: 0 (zwart) of 1 (wit).

Bijgevolg zal een bit-diepte van  $k = 1$  volstaan. Deze afbeeldingen worden vaak gebruikt bij het voorstellen van lijnen, archiveren van documenten en elektronisch printen. In toepassing van beeldverwerking worden ze vaak als maskers gebruikt.

In zwart-wit afbeeldingen kunnen de pixels naast zwart of wit ook alle tussengelegen grijsntinten aannemen. De exacte hoeveelheid grijsntinten wordt bepaald door de gehanteerde bit-diepte. In praktijk volstaat voor de meeste toepassingen een  $k = 8$  bits per pixel waardoor de intensiteit een waarde aanneemt tussen 0 en 255, waarbij 0 de minimum intensiteit voorstelt (zwart) en 255 de maximum intensiteit (wit). Indien de helderheid en het contrast van de afbeelding bewerkt moeten worden kan het interessant zijn te kiezen voor hogere bit-dieptes zoals 16 of 32 bits per pixel om bruuske kleurovergangen te vermijden [4]. In Figuur 1.2 wordt de invloed van de gebruikte bit-diepte visueel voorgesteld.



**Figuur 1.2:** De invloed van de gebruikte bit-diepte [5]

RGB of kleurafbeeldingen zijn opgebouwd uit drie verschillende lagen of kanalen. Elke laag komt overeen met de rode, groene of blauwe tinten die samen een kleur voorstellen. Voor elke laag worden in de praktijk meestal 8 bits voorzien wat resulteert in 16 miljoen mogelijke pixelwaarden. Dit vereist 24 bits per pixel (8 per kanaal) om een kleur voor te stellen.

Om een RGB-afbeelding om te zetten naar een zwart-wit afbeelding volstaat het om het wiskundig gemiddelde te nemen van de pixelwaarden in de drie verschillende lagen en dit gemiddelde te kiezen als nieuwe pixelwaarde.

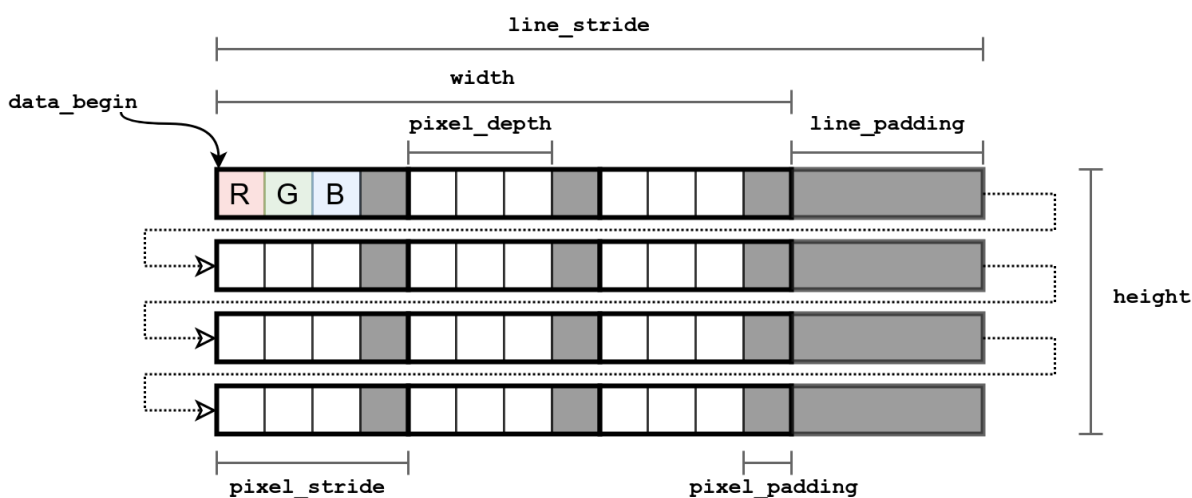
## 1.2 Voorstelling in het geheugen

In [6] en [7] wordt beschreven hoe digitale afbeeldingen efficiënt opgeslagen kunnen worden. Veronderstel hiervoor een niet-gecomprimeerde, digitale RGB-afbeelding met een bit-diepte van 8 bits per kanaal. Om snel de individuele pixelwaarden te kunnen raadplegen of aanpassen is een snelle en efficiënte voorstelling nodig in het geheugen van de computer. Een schematische weergave van dergelijke structuur is weergegeven in Figuur 1.3.

In theorie worden 24 bits per pixel gereserveerd maar aangezien de klassieke 32 en 64-bit processoren het efficiëntst data verwerken in blokken van 32 of 16-bit groot, wordt aan elke pixel doelbewust een informatie loze *pixel padding* van 8 bits groot toegevoegd. Hoewel het opslaan van een afbeelding nu meer geheugen vereist zullen de verschillende operaties zoals het wijzigen van een specifieke pixel waarde sneller uitgevoerd kunnen worden. De totale hoeveelheid bytes per pixel in het geheugen wordt een *pixel stride* genoemd.

De volledige afbeelding wordt pixel per pixel, rij per rij beginnend linksboven, opgeslagen in een één-dimensionale *array*. Analoog als hiervoor worden na het opslaan van alle pixels op eenzelfde rij een aantal informatie loze bytes toegevoegd, de *line padding* genaamd. De grootte van deze padding is vooral afhankelijk van de gebruikte *hardware* en is opnieuw noodzakelijk om efficiëntie te verhogen. De totale hoeveelheid bytes die toegevoegd moeten worden aan het adres van de eerste pixel van een willekeurige rij om het adres van de eerste pixel van de daaropvolgende rij te bekomen wordt de *line stride* genoemd. Dit mag echter niet verward worden met de breedte van de afbeelding. Deze wordt namelijk uitgedrukt in pixels en beschrijft een kenmerk van de afbeelding terwijl de *line stride* uitgedrukt wordt in bytes en afhankelijk is van de representatie van de afbeelding in het geheugen.

Tot slot wordt vooraan een *header* toegevoegd die belangrijke *metadata* zoals de breedte, hoogte en type van de afbeelding alsook een *pointer* naar de eerste pixel in het geheugen bevat. In Figuur 1.3 wordt het adres van de eerste pixel voorgesteld als *data begin*.



**Figuur 1.3:** De voorstelling van een kleurenafbeelding in het geheugen met een bit-diepte van 8 bits

Deze structuur maakt het programmatisch opzoeken van het adres van een willekeurige pixel (x,y) in het geheugen erg eenvoudig:

$$pixel\_address = data\_begin + y * line\_stride + x * pixel\_stride \quad (1.1)$$

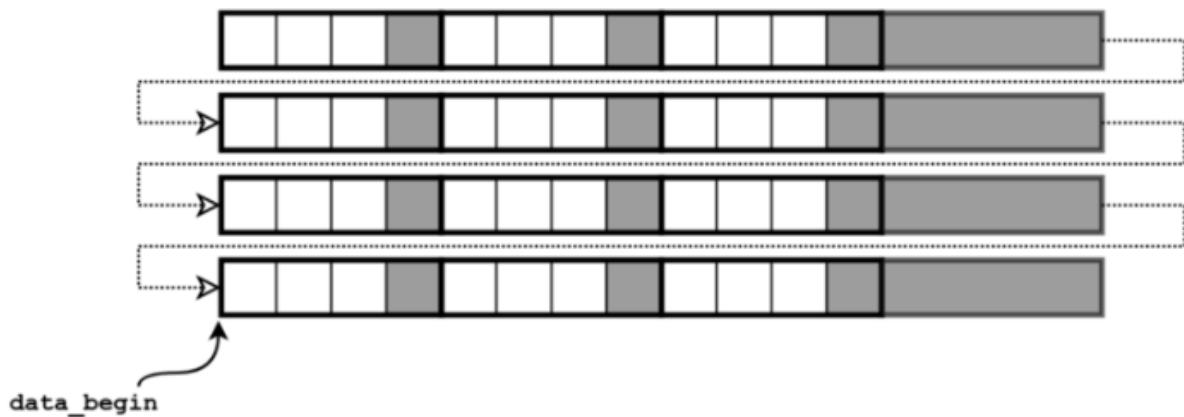
Ook het uitvoeren van complexere operaties zoals verticaal spiegelen kunnen via enkele eenvoudige berekeningen uitgevoerd worden.

$$data\_begin = data\_begin + (height - 1) * line\_stride, \quad (1.2)$$

verplaatst de *data\_begin* pointer naar de eerste pixel van de laatste rij zoals weergegeven in Figuur 1.4. Een tweede wijziging:

$$line\_stride = -line\_stride \quad (1.3)$$

zal in combinatie met (1.1) ervoor zorgen dat de afbeelding van onder naar boven gelezen kan worden waardoor efficiënt een verticale spiegeling bekomen wordt.



**Figuur 1.4:** De voorstelling in het geheugen van de digitale afbeelding na het verticaal spiegelen

## 1.3 Bestand extensies

Het niet-gecomprimeerd opslaan van afbeeldingen is in veel toepassingen niet erg efficiënt. Er bestaan namelijk een grote hoeveelheid algoritmen die de vereiste opslagcapaciteit drastisch kunnen verlagen zonder een visueel verschillend resultaat op te leveren. Dergelijke compressiealgoritmes kunnen onderverdeeld worden in twee categorieën: “*losy*” en “*lossless*” compressie [8]. De eerste groep bekommt een lagere bestandsgrootte ten koste van kwaliteit door bijvoorbeeld de bit-diepte van de pixels te verlagen. Deze wijziging resulteert in een lager aantal verschillende pixelwaarden met minder kleurvariatie tot gevolg. *Lossless* compressie daarentegen behoudt een kopie van de originele niet-gecomprimeerde afbeelding maar slaagt erin de opslagcapaciteit te verlagen door bijvoorbeeld een wederkerend patroon slechts éénmalig op te slaan. Afhankelijk van het gebruikte compressie algoritme krijgen digitale afbeeldingen een specifieke bestands extensie. Om vertrouwd te raken met de populairste extensies worden deze kort besproken zonder in te gaan op de specifieke werking van de gebruikte algoritmes.

### JPG (of JPEG)

Dit is de geprefereerde bestands extensie voor de meeste op het web gepubliceerde afbeeldingen dankzij de relatief goede kwaliteit bij hoge compressie waardoor de afbeeldingen snel geladen kunnen worden. JPEG maakt gebruik van *losy* compressie waardoor het herhaaldelijk bewerken en opslaan van eenzelfde JPG-afbeelding resulteert in een (zwaar) beschadigde afbeelding. Indien dit niet het geval is, zal het eenmalig comprimeren visueel geen verschillend resultaat opleveren. JPG is een goede keuze voor foto's en wordt vaak als standaard gebruikt in de digitale fotografie. Afbeeldingen met grote uniform gekleurde oppervlakken en scherpe lijnen daarentegen leveren vrij slechte resultaten op waardoor andere compressie algoritmen zoals PNG hier aangewezen zijn.

### PNG

PNG is een *lossless* compressie formaat waardoor geen data verloren gaat. Het nadeel hiervan is dat de bestandsgrootte gemiddeld groter zal zijn dan van JPG-afbeeldingen waardoor ze minder geschikt zijn voor het gebruik in onlineapplicaties. In tegenstelling tot JPG ondersteunt het PNG-bestandsformaat transparantie en gaat geen data verloren.

### TIFF (of TIF)

Dit is een zeer flexibel formaat dat zowel *losy* als *lossless* kan comprimeren afhankelijk van de gebruikte techniek. Verder kunnen per kleur acht of zestien bits voorzien worden. De bestandsgrootte is doorgaans groter dan eenzelfde JPG of PNG-afbeelding en niet alle webbrowsers ondersteunen dit formaat. TIFF is daarentegen wel een ideale keuze indien de afbeelding afgedrukt moet worden en wordt in de printindustrie als standaard gebruikt. De *lossless* variant is bovendien uitermate geschikt om een afbeelding meermaals te bewerken en herhaaldelijk op te slaan.

## 2 Probleemstelling

Een wederkerend probleem bij het ontwikkelen van digitale afbeeldingen is het fenomeen van ruis. Hieronder verstaat men elke willekeurige afwijkingen in helderheid of kleur die een afbeelding beschadigt. Ruis kan onder meer optreden bij het nemen van een foto in ongunstige omstandigheden zoals: weinig lichtinval, een lage sluitertijd of een te hoge omgevingstemperatuur. Mits de nodige voorzorgsmaatregelen en goede camera-instellingen kan de hoeveelheid ruis in veel gevallen beperkt blijven. Een volledig ruisvrije afbeelding kan echter nooit gegarandeerd worden. Gelukkig bestaan er allerlei eenvoudige technieken die proberen een door ruis beschadigde afbeelding te herstellen. In dit hoofdstuk zullen deze technieken en hun tekortkomingen aangekaart worden wat vervolgens leidt tot het doel van deze masterproef. Eerst worden echter twee specifieke vormen van ruis nader besproken: Gaussische ruis en *salt-and-pepper noise*.

### 2.1 Gaussische ruis

Volgens [9] is Gaussische ruis een vorm van statistische ruis met als kansdichtheidsfunctie de welbekende normaalverdeling of Gaussverdeling (vandaar de naam). Alle beschadigde pixels zullen met andere woorden een waarde aannemen die normaal verdeeld is zoals weergegeven in Figuur 2.1. Deze vorm van ruis is in praktijk zeer belangrijk aangezien veel natuurlijke fenomenen zoals bloeddruk en IQ-scores eenzelfde Gauss-verdeeld patroon vertonen.

### 2.2 Salt-and-pepper noise

Een andere, minder voorkomende, vorm van ruis is de zogenaamde *salt-and-pepper noise* ook wel impuls ruis genoemd. Deze treedt op door scherpe of plotse verstoringen in het analoge signaal van de afbeelding waardoor, na conversie, enkele willekeurige pixels in het eindresultaat foutief een zwarte of witte kleur aannemen. Een voorbeeld van dergelijke beschadiging is weergegeven in Figuur 2.2.



**Figuur 2.1:** Een voorbeeld van Gaussische ruis



**Figuur 2.2:** Een voorbeeld van salt-and-pepper noise



## 2.3 Eenvoudige technieken

Aangezien Gaussische ruis frequenter voorkomt dan *salt-and-pepper noise* zal vooral dit type ruis uitvoerig behandeld worden. Er bestaan een drietal eenvoudige technieken om snel de hoeveelheid Gaussische ruis in een digitale afbeelding te verlagen.

### Gemiddelden

Een eerste eenvoudige methode convolueert de afbeelding met een genormaliseerde box filter zoals weergegeven in Figuur 2.3. Deze berekening zorgt ervoor dat elke pixelwaarde vervangen wordt door de gemiddelde pixelwaarde van diens burens. Op deze manier kunnen beschadigde pixels hersteld worden op basis van nabijgelegen, mogelijks, onbeschadigde pixels. Hoewel deze werkwijze plausibel klinkt blijken de eindresultaten in realiteit vooral wazig, en dus onbruikbaar, te zijn.

### Gaussiaans vervagen

De tweede techniek doet eveneens een beroep op een convolutie berekening. Ditmaal wordt echter een Gaussische kern (Figuur 2.4) i.p.v. een box filter gebruikt. Deze kern (soms ook masker genoemd) zal een gewogen gemiddelde nemen van de nabijgelegen pixelwaarden waarbij de originele pixel de grootste invloed uitoefent op de nieuwe pixelwaarde. De gebruikte numerieke waarden in de kern zijn afgeleid uit de Gaussische functie in twee dimensies. Gaussiaans vervagen heeft de eigenschap om de hoogfrequente onderdelen van de afbeelding te verlagen waardoor dit proces beschouwt kan worden als een laagdoorlaatfilter. Desondanks deze aanpassing resulteert ook deze techniek in eindresultaten met wazige randen.

$$\frac{1}{9}$$

1	1	1
1	1	1
1	1	1

**Figuur 2.3:** Een genormaliseerde box filter

$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

**Figuur 2.4:** Een Gaussische kern

## Medianen

Een laatste veelgebruikte, niet-lineaire, filter techniek vervangt elke pixelwaarde door de mediaan van de nabijgelegen pixelwaarden binnen een bepaald bereik. Hieruit volgt dat, in tegenstelling tot de hiervoor vermelde technieken, elke pixel een waarde toegekend zal krijgen die zeker voorkomt in de oorspronkelijke afbeelding. Hierdoor blijven onder andere kleur en helderheid goed behouden. Bijkomend zullen ook de overgangen tussen verschillende oppervlakken relatief goed bewaard blijven waardoor het uiteindelijke eindresultaat niet wazig oogt. Desondanks deze verbeteringen is ook deze methode niet feilloos. In sommige gevallen kan het namelijk voorkomen dat details en texturen op een monochrome achtergrond verdwijnen alsof ze nooit aanwezig waren. Het is vanzelfsprekend dat in medische toepassingen zoals CT-scans dergelijk effect catastrofale gevolgen kan hebben.

## 2.4 Doel masterproef

De hierboven vermelde technieken slagen er niet in een door (Gaussische) ruis beschadigde afbeelding te verbeteren zonder randen uit te vegen of cruciale details te verliezen. Er bestaan echter enkele geavanceerde wiskundige modellen die wel in staat zijn ruis weg te werken zonder de hierboven vermelde nadelen. In [1] worden uitgebreid de Perona-Malik en geregulariseerde Perona-Malik vergelijking wiskundig verklaart en besproken. In [10] wordt een derde techniek op basis van fractionele diffusie voorgesteld.

Op dit moment is binnen de Faculteit Wetenschappen geen programma beschikbaar waarmee deze modellen uitgebreid getest kunnen worden. Het is dan ook de bedoeling een gebruiksvriendelijke applicatie te ontwikkelen die deze wiskundige modellen implementeert en de eindgebruiker in staat stelt hiermee te experimenteren. In een tweede fase wordt vervolgens aan de slag gegaan met deze applicatie om de invloeden van de verschillende parameters van de verscheidene modellen te onderzoeken. Tot slot kunnen deze geavanceerde technieken onderling vergeleken worden om na te gaan welke methode de beste eindresultaten oplevert.

### 3 Wiskundige modellen

In dit hoofdstuk worden drie geavanceerde wiskundige modellen besproken die allen de tekortkomingen van de in Hoofdstuk 2.3 besproken eenvoudige technieken oplossen. Voor een uitgebreide wiskundige uitwerking van deze modellen wordt verwezen naar [1] en [10]. Wel wordt het idee en de globale werkwijze van deze modellen besproken.

#### 3.1 De Perona-Malik vergelijking

De Perona-Malik vergelijking is een niet-lineaire diffusievergelijking die in beeldverwerking gebruikt wordt om ruis weg te filteren en randen te verscherpen.

De originele vergelijking werd in [11] als volgt geïntroduceerd:

$$u_t = \nabla \cdot (g(|\nabla u|)\nabla u) \quad (3.1)$$

met

$$u : [0, T] \times \Omega \rightarrow \mathbb{R} : (t, x) \mapsto u(t, x) \quad (3.2)$$

de afbeelding als functie van de tijd en plaats en

$$g : \mathbb{R} \rightarrow \mathbb{R}^+ : x \mapsto g(x) \quad (3.3)$$

de diffusiecoëfficiënt als functie van de ruimte. Hierbij wordt de functie  $g$  beperkt tot de klasse van monotoon dalende functies en wordt  $\Omega \subset \mathbb{R}^2$  als een begrensd, rechthoekig domein verondersteld.

Vergelijking (3.1) gaat gepaard met een beginconditie, i.e. de initiële afbeelding, en een Neumann randconditie

$$\nabla u(t, x) \cdot \nu = 0 \quad \text{op } (0, T) \times \partial\Omega, \quad (3.4)$$

$$u(0, x) = u_0(x) \text{ in } \Omega, \quad (3.5)$$

waarbij  $\nu$  de uitwaartse normaalvector is aan  $\partial\Omega$ . Randconditie (3.4) zorgt ervoor dat de totale intensiteit van de afbeelding bewaard blijft. De beginconditie  $u_0(x)$  stelt de afbeelding voor die we wensen te bewerken.

Om het idee achter deze vergelijking te verduidelijken wordt vertrokken vanuit een diffusievergelijking. Dit is een partiële differentiaalvergelijking die vaak optreedt bij fysische processen. De algemene vorm wordt voorgesteld als:

$$\partial_t u(x, t) = \nabla \cdot (C(u, x, t)) \nabla u(x, t), \quad (3.6)$$

waarbij  $C$  de diffusiecoëfficiënt wordt genoemd. Indien deze coëfficiënt de waarde 1 aanneemt wordt de lineaire warmtevergelijking bekomen. Het is welbekend dat deze een *smoothing* effect uitoefent waardoor initieel ruwe data glad gemaakt wordt.

Het idee achter de Perona-Malik vergelijking is om dit *smoothing* effect te bekomen in het binnenste van elke regio en te vermijden bij de randen. Dit gedrag zou verwezenlijkt kunnen worden door de diffusiecoëfficiënt  $C$  gelijk te stellen aan 1 in aaneengesloten oppervlakken en gelijk te stellen aan 0 langs de randen. Omdat deze diffusiecoëfficiënt duidelijk afhankelijk is van de oplossing wordt gesproken van een niet-lineaire diffusievergelijking.

Om de locatie van de randen in een afbeelding na te gaan kan gebruik gemaakt worden van de gradiënt  $\nabla u(x, t)$ . Dit is een wiskundig begrip die de maat van afwijking t.o.v. de omgeving voorstelt. In kader van beeldverwerking zal de gradiënt van een pixel een indicatie geven in hoeverre de pixelwaarde afwijkt ten opzichte van de pixelwaarden van zijn burens. Concreet zal een gradiënt met grote magnitude duiden op veel afwijking waardoor een pixel als onderdeel van een rand beschouwd kan worden terwijl een kleine gradiënt duidt op een egaal oppervlak aangezien nabijgelegen pixelwaarden weinig tot niet verschillen.

Met  $\nabla u$  als schatter, kan de diffusiecoëfficiënt bijgevolg gekozen worden in functie van de grootte van de  $\nabla u$ , i.e.

$$C = g(|\nabla u|). \quad (3.7)$$

Hierbij is  $g$  een niet-negatieve monotoon dalende functie met  $g(0) = 1$ . In [11] werden volgende twee diffusie-functies voorgesteld:

$$g_1(x) = e^{-\frac{1}{2} \left(\frac{x}{k}\right)^\alpha} \quad (3.8)$$

en

$$g_2(x) = \frac{1}{1 + (x/k)^\alpha} \quad (3.9)$$

waarbij  $k$  de contrast parameter genoemd wordt. Deze doet dienst als *threshold* en bepaalt vanaf welke waarde een gradiënt als rand beschouwd zal worden. De invloed van deze parameter op het

bekomen eindresultaat is niet verwaarloosbaar en moet bijgevolg goed gekozen worden. In deze implementatie wordt gebruik gemaakt van de in [12] voorgestelde waarde

$$k = 1.4826 \text{ MAD } (|\nabla u_0|), \quad (3.10)$$

waarbij MAD (*median absolute variation*) een robuuste schatter is voor de standaarddeviatie en gedefinieerd wordt als:

$$\text{MAD } (|\nabla u_0|) = \text{median}(|\nabla u_0 - \text{median}(|\nabla u_0|)|) \quad (3.11)$$

De  $\alpha \in \mathbb{R}$  in functies (3.8) en (3.9) is een extra parameter die in de literatuur doorgaans een waarde van 2 aanneemt. In de implementatie wordt deze echter als een veranderlijke beschouwd om ook de invloed van deze waarde te kunnen bestuderen. Vanaf nu wordt deze parameter genoteerd door 'a'.

Om de oplossing van (3.1) te benaderen moet een numeriek schema opgesteld worden. Hiervoor wordt gestart met een discretisatie in tijd van (3.1) waarbij het interval  $[0, T]$ , met  $T$  als eindtijd, uniform verdeeld wordt in  $N_t$  deel-intervallen  $[t_{l-1}, t_l]$  met  $l = 1, \dots, N_t$ . De tijdstap  $\tau$  wordt gedefinieerd als  $\tau = T/N_t$ . De tijdsafgeleide in (3.1) wordt vervangen door de achterwaartse Euler differentie. De niet-lineaire termen van de vergelijking worden geëvalueerd op het vorige tijdstip terwijl de lineaire termen beschouwd worden op het huidige tijdstip. Op deze manier ontstaat een semi-discreet lineair schema.

Op elk tijdstip moet de oplossing worden bepaald. Daartoe worden vervolgens de ruimtelijke coördinaten gediscetiseerd via de eindige volume methode (EVM). Hierbij wordt  $\Omega = (0, n) \times (0, m)$  gediscetiseerd in een rooster van  $mn$  elementen. Deze elementen zijn vierkanten (lees: pixels) waarvan de oppervlakte de basiseenheid definieert. De eindige volumes komen dus overeen met deze pixels. Waarbij  $(i, j)$  met  $1 \leq i \leq m$  en  $1 \leq j \leq n$ , de pixelindex voorstelt. De coördinaten van het midden van deze pixel zijn gegeven door  $(x_j, y_i) := (j - 1/2, i - 1/2)$ . Dit midden wordt als representatief punt gebruiken van  $(i, j)$ . De pixelindexen kunnen als volgt afgebeeld worden op de oppervlakte die ze beschrijven in  $\mathbb{R}^2$ ,

$$\alpha : \{1, \dots, m\} \times \{1, \dots, n\} \rightarrow \Omega. \quad (3.12)$$

Beschouw vervolgens een pixel  $p = \alpha(i, j)$ . Deze heeft ten hoogste vier burens in het rooster, die genoteerd worden als  $e, w, s$  en  $n^1$ . De respectievelijke punten van deze burens worden gegeven door

---

<sup>1</sup> East, west, south en north

$$(x_j + 1, y_i), (x_j - 1, y_i), (x_j, y_i + 1), (x_j, y_i - 1).$$

Deze verzameling wordt genoteerd als  $\mathcal{N}(p)$ .

De benaderde waarde van de oplossing op tijdstip  $t_l$  binnenin het eindige volume  $p$  wordt genoteerd met  $u_p^l$ .

Het volledig discreet eindige volume schema voor het oplossen van de vergelijking wordt als volgt gedefinieerd:

$$u_p^l + \tau \sum_{q \in \mathcal{N}(p)} g_q^{l-1}(p)(u_p^l - u_q^l) = u_p^{l-1}, \quad (3.13)$$

startend met  $u^0$  gegeven door (3.5).

Een uitgebreide uitwerking van dit schema is terug te vinden in [1].

## 3.2 De geregulariseerde Perona-Malik vergelijking

De Perona-Malik vergelijking kan zich soms lokaal gedragen als een achterwaartse warmtevergelijking. Deze vergelijking is echter een wiskundig slecht gesteld probleem omdat deze geen unieke oplossing bezit die continu afhangt van de data. Hierdoor wordt verwacht dat de Perona-Malik vergelijking slechte resultaten oplevert. In praktijk is dit echter niet het geval en werkt deze veel beter dan theoretisch verwacht wordt. De enige geobserveerde instabiliteit is het zogenaamde trapvormig effect, waarbij een gladde rand evolueert in stuksgewijze segmenten die gescheiden worden door sprongen.

Via regularisatie kan dit probleem omgezet worden naar een goed gedefinieerd probleem.

Dit kan bekomen worden door  $g(|\nabla u|)$  te vervangen door  $g(|\nabla G_\sigma * u|)$ , waarbij  $G_\sigma$  een Greense functie is,

$$G_\sigma(x) = \frac{1}{4\pi\sigma} \exp\left(-\frac{|x|^2}{4\sigma}\right). \quad (3.14)$$

Bijgevolg bezit de geregulariseerde Perona-Malik vergelijking dezelfde parameters als de gewone Perona-Malik vergelijking aangevuld met een nieuwe parameter, sigma.

Doordat  $\nabla G_\sigma * u \rightarrow \nabla u$  als  $\sigma \rightarrow 0$ , leunt dit model zeer dicht aan bij de originele formulering voor kleine waarden van  $\sigma$ . Hierdoor behoudt het ook alle praktische eigenschappen van het originele Perona-Malik model.

Een uitgebreide studie omtrent deze regularisatie, waarbij gebruik gemaakt wordt van de Rothemethode [13], is terug te vinden in [1].

### 3.3 Fractionele diffusie

Een vrij recente andere benadering maakt gebruik van fractionele calculus om ruis uit afbeeldingen te verwijderen. Concreet wordt het in [10] beschreven *global adaptive fractional integral algorithm* (GAFIA) in deze thesis geïmplementeerd. Hierin wordt fractionele integratie toegepast op elke pixel waarbij de optimale ordes per pixel bekomen worden via de gemiddelde gradiënt in vier of acht richtingen afhankelijk van het gebruikte masker.

Er bestaat geen uniforme definitie van fractionele calculus. Een handvol wiskundigen hebben dit probleem op unieke wijze geanalyseerd en beschreven waardoor een aantal verschillende formuleringen zijn ontstaan. De drie meest voorkomende uitdrukkingen zijn: Grunwald-Letnikov (G-L), Riemann-Liouville (R-L) en Caputo. In deze thesis wordt uitsluitend gewerkt met de formulering volgens G-L op basis van gehele-orde differentiaal. De Gammafunctie wordt gebruikt om deze gehele-orde uit te breiden naar een rationale-orde. Indien de orde  $\nu < 0$ , dan wordt de fractionele integraal volgens G-L benaderend voorgesteld als:

$$I_{G-L}^{\nu} f(t) \approx f(t) + (-\nu)f(t-1) + \frac{(-\nu)(-\nu+1)}{2}f(t-2) + \dots \quad (3.15)$$

$$+ \frac{\Gamma(-\nu+1)}{n! \Gamma(-\nu-n+1)}f(t-n)$$

waarbij het tijdsinterval opgedeeld wordt in  $n$  deelintervallen van lengte 1.

Het is deze integraal die instaat voor het verwijderen van ruis in de beschadigde afbeelding.

Aangezien de gedeeltelijke fractionele integraal in acht richtingen van een tweedimensionaal digitaal beeld gedefinieerd is kunnen hiermee een masker in vier of acht dimensies opgesteld worden. Om deze te onderscheiden krijgt het eenvoudigste masker in vier dimensies de naam “std” toegewezen terwijl het masker in acht dimensies vernoemd wordt naar één van de auteurs van [10]: “boli”.

Beide maskers zijn respectievelijk terug te vinden in Figuur 3.1 en Figuur 3.2. De parameters  $\xi$  worden voorgesteld door de verschillende coëfficiënten uit (3.16):

$$\begin{cases} \xi_1 = 1 \\ \xi_2 = -\nu \\ \xi_3 = \frac{(-\nu)(-\nu+1)}{2} \\ \vdots \\ \xi_n = \frac{\Gamma(-\nu+1)}{n! \Gamma(-\nu-n+1)} \end{cases}, (n \in \mathbb{N}, G-L \text{ definition}) \quad (3.16)$$

0	0	...	0	$\xi_n$	0	...	0	0
0	$\ddots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\ddots$	0
$\vdots$	$\ddots$	0	0	$\xi_3$	0	0	$\ddots$	$\vdots$
0	...	0	0	$\xi_2$	0	0	...	0
$\xi_n$	...	$\xi_3$	$\xi_2$	$4\xi_1$	$\xi_2$	$\xi_3$	...	$\xi_n$
0	...	0	0	$\xi_2$	0	0	...	0
$\vdots$	$\ddots$	0	0	$\xi_3$	0	0	$\ddots$	$\vdots$
0	$\ddots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\ddots$	0
0	0	...	0	$\xi_n$	0	...	0	0

Figuur 3.1: Het std-masker

$\xi_n$	0	...	0	$\xi_n$	0	...	0	$\xi_n$
0	$\ddots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\ddots$	0
$\vdots$	$\ddots$	$\xi_3$	0	$\xi_3$	0	$\xi_3$	$\ddots$	$\vdots$
0	...	0	$\xi_2$	$\xi_2$	$\xi_2$	0	...	0
$\xi_n$	...	$\xi_3$	$\xi_2$	$8\xi_1$	$\xi_2$	$\xi_3$	...	$\xi_n$
0	...	0	$\xi_2$	$\xi_2$	$\xi_2$	0	...	0
$\vdots$	$\ddots$	$\xi_3$	0	$\xi_3$	0	$\xi_3$	$\ddots$	$\vdots$
0	$\ddots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\ddots$	0
$\xi_n$	0	...	0	$\xi_n$	0	...	0	$\xi_n$

Figuur 3.2: Het boli-masker

De gebruikte fractionele orde  $\nu$  is veranderlijk per pixel en wordt bepaald door de gemiddelde gradiënt van elke pixel in vier of acht richtingen afhankelijk van het gebruikte masker. Deze berekening in acht dimensies wordt voorgesteld als:

$$M(i, j) = \frac{1}{8} (|f(i, j) - f(i-1, j-1)| + |f(i, j) - f(i-1, j)| + |f(i, j) - f(i-1, j+1)| + |f(i, j) - f(i+1, j-1)| + |f(i, j) - f(i+1, j)| + |f(i, j) - f(i+1, j+1)|), \quad (3.17)$$

waarbij  $f(i, j)$  de pixelwaarde voorstelt van de pixel met coördinaten  $(i, j)$ .

De grootte van het masker is constant voor alle pixels in de afbeelding.

Stel de kleinste en grootste gemiddelde gradiënt van alle pixels in de afbeelding respectievelijk gelijk aan  $X$  en  $Y$ , dan wordt de fractionele orde van een willekeurige pixel met coördinaten  $(i, j)$  in acht richtingen voorgesteld door:

$$\nu_{GAFIA} = (-1) \times \frac{M(i, j) - X}{Y - X} \quad (3.18)$$

Deze orde bevindt zich steeds in het interval  $[-1, 0]$ . Een door ruis beschadigde pixel levert bijgevolg een grote gemiddelde gradiënt wat resulteert in een kleine, negatieve orde en grote vervaging. Alvorens het std- en boli-masker gebruikt kunnen worden moet elk element respectievelijk gedeeld worden door vier of acht maal de som van alle verschillende coëfficiënten in het masker. Pas hierna kan de door ruis beschadigde afbeelding verwerkt worden door elke pixel te convolueren met het geselecteerde masker en de optimale orde. Langs de rand van de afbeelding worden de pixelwaarden gespiegeld zodat ook hier het masker opgesteld kan worden. De gebruikte maskers moeten steeds een vierkante vorm aannemen met een oneven aantal elementen per zijde zodat deze ten alle tijden één centraal middelpunt bevatten.



Om de implementatie te verduidelijken worden voorgaande stappen samengevat in een overzichtelijk stappenplan:

1. Selecteer een masker en zijn grootte.
2. Selecteer het aantal uit te voeren iteraties.
3. Overloop elke pixel in de afbeelding.
  - a. Bereken per pixel de benodigde masker-coëfficiënten via (3.16) en (3.18).
  - b. Deel elk element door vier of acht maal de som van alle verschillende coëfficiënten in het masker.
  - c. Convolueer de geselecteerde pixel met het masker.
4. Verhoog de huidige iteratie en herhaal vanaf stap drie tot het opgegeven aantal iteraties bereikt wordt.

## 4 Implementatie

In dit hoofdstuk worden de gebruikte technologieën en technieken besproken die geleid hebben tot de uiteindelijke implementatie van de verschillende wiskundige modellen en experimentatie mogelijkheden. Grote stukken code zullen hierbij niet letterlijk besproken worden, wel worden zowel de denk- en werkwijze als het idee achter de gemaakte beslissingen verduidelijkt. De enige vereiste omtrent de implementatie was het opleveren van een gebruiksvriendelijke, werkende applicatie waarmee de in Hoofdstuk 3 beschreven modellen uitgetest kunnen worden, bij voorkeur geschreven in MATLAB of Python. De exacte uitwerking van de precieze functionaliteiten, werkwijzen en vorm waren vrij te kiezen.

### 4.1 Gebruikte technologieën

Als programmeertaal werd resoluut gekozen voor Python aangezien deze krachtiger en flexibeler is dan MATLAB. Bovendien zorgde een weliswaar geringe ervaring met Python ervoor dat de basis syntax snel opgefrist kon worden. MATLAB daarentegen was volledig onbekend, het aanleren van syntax en werkwijze zou bijgevolg de implementatie enkel maar bemoeilijken.

#### 4.1.1 Python *libraries*

In realiteit bouwen nieuwe uitvindingen vaak voort op voorgaande ontdekkingen. Zo heeft de uitvinder van de auto, Karl Benz, bijvoorbeeld gebruik gemaakt van een aantal reeds gekende technologieën zoals het wiel om een nieuw product te maken. Dit principe doet zich ook voor binnen de softwareontwikkeling waar een beroep gedaan kan worden op door anderen reeds geïmplementeerde functionaliteiten om dubbel werk te vermijden. Dergelijke onafhankelijke stukken code die gebruikt kunnen worden om een complexer geheel op te bouwen worden “modules” genoemd. Een verzameling modules uit eenzelfde toepassingsgebied worden vaak ondergebracht in een “*library*”. Python beschikt over een brede waaier verschillende *libraries* die functionaliteiten aanbieden variërend van artificiële intelligentie tot het verwerken van http-berichten. Ook in kader van beeldverwerking en data representatie zijn een heleboel handige *libraries* beschikbaar. Hieronder worden de belangrijkste in de applicatie gebruikte *libraries* beschreven alsook enkele concrete *use cases*.

#### NumPy

Elk Python-script die wetenschappelijke berekeningen uitvoert of grote hoeveelheden data verwerkt, wordt vroeg of laat genoodzaakt gebruik te maken van de NumPy *library*. In essentie biedt deze een multidimensionaal *array* object aan waarmee erg efficiënt wiskundige berekeningen uitgevoerd kunnen worden. Uit [14] blijkt dat NumPy maar liefst acht keer sneller de som van 1

miljoen integers kan berekenen dan de standaard Python implementatie. Deze enorme snelheidswinst is onder meer te danken aan het feit dat NumPy achter de schermen gebruikmaakt van voorgecompileerde C-code.

Aangezien een afbeelding van middelmatige grootte, veronderstel bijvoorbeeld 1024 x 768 pixels, reeds opgebouwd is uit 789432 pixelwaarden kunnen enorme snelheidswinsten geboekt worden indien deze opgeslagen worden in een NumPy *array* i.p.v. een standaard Python *list*. Concreet wordt in de implementatie dan ook zoveel mogelijk gebruik gemaakt van deze *library* om de snelheid te verhogen.

## SciPy

SciPy is een *library* die onder meer gebruik maakt van NumPy om geavanceerde wiskundige functionaliteiten aan te bieden in kader van bijvoorbeeld numerieke integratie, interpolatie en lineaire algebra. Vooral dit laatste toepassingsgebied komt tijdens de implementatie van de Perona-Malik vergelijking goed van pas. SciPy voorziet onder andere een efficiënte functie waarmee een lineair systeem opgelost kan worden via de toegevoegde gradiëntmethode (*conjugate gradient method*). Dankzij dergelijke modules is een diepgaande kennis omtrent de precieze werking van deze wiskundige modellen niet langer een vereiste.

De voorstelling van een lineair systeem in klassieke Python-code bracht een ander probleem met zich mee. Deze wordt namelijk opgeslagen als een enorme vierkante matrix waarbij elke rij evenveel elementen bevat als het totaal aantal pixels in de afbeelding. Indien opnieuw een afbeelding van 789432 pixels gebruikt wordt zal het lineair systeem voorgesteld worden door een matrix van 60 miljard elementen groot. In praktijk zullen dergelijke matrices jammer genoeg resulteren in *stackoverflow* fouten wegens onvoldoende intern geheugen om alle elementen op te slaan. Wanneer deze gigantische matrix bekeken wordt, blijken echter het overgrote deel van deze elementen nullen te zijn. Indien bijgevolg enkel de niet-nul elementen opgeslagen zouden worden kan de grootte van deze matrix enorm gereduceerd worden. In de implementatie wordt hiervoor gebruik gemaakt van de SciPy methode “*spdiags*”.

## OpenCV

De *open source computer vision library* bevat meer dan 2500 geoptimaliseerde algoritmen voor het herkennen van objecten in afbeeldingen en het classificeren van acties in een video. Het is een onmisbare *library* in elke computervisie gerelateerde applicatie en is naast Python ook beschikbaar in Java, C++ en MATLAB. Hoewel deze applicatie geen gebruik maakt van computervisie bevat deze *library* ook een implementatie van de in Hoofdstuk 2.3 vermelde eenvoudige technieken om ruis weg te werken. Concreet worden de “*blur()*”, “*gaussianBlur()*” en “*medianBlur()*” methodes gebruikt om een afbeelding ruisvrij te maken via respectievelijk: gemiddelden, Gaussiaans vervagen en medianen.

## Pillow

De Pillow *library* is een voortzetting van de verouderde *Python Imaging Library* (PIL). Het voorziet methodes voor het openen, bewerken en opslaan van digitale afbeeldingen en is compatibel met alle in Hoofdstuk 1.3 vermelde bestands extensies. Concreet worden de “Image.open()” en “Image.save()” methodes gebruikt om respectievelijk een nieuwe afbeelding te openen en exporteren.

## Matplotlib

Via de Matplotlib *library* kan een hoeveelheid data visueel voorgesteld worden aan de hand van verschillende grafieken en tabellen. Bijkomend kunnen ook afbeeldingen snel en eenvoudig in een venster weergegeven worden. De programmeur is in staat de stijl, vorm en type van de grafieken naar wens aan te passen. In de implementatie wordt vooral gebruik gemaakt van de klassieke lijngrafieken en bijhorende tabellen. Een uitgebreide beschrijving van deze geïmplementeerde grafieken volgt later in Hoofdstuk 5.

## Pickle

De Pickle-module implementeert een aantal binaire protocollen die instaan voor de (de)serialisatie van Python objecten. Dit betekent dat Python objecten geconverteerd kunnen worden naar een *byte stream* waarna ze opgeslagen kunnen worden als Pickle-bestanden. Deze bestanden kunnen vervolgens gemakkelijk opnieuw gevormd worden tot volwaardige Python objecten. Deze functionaliteiten bieden de mogelijkheid om efficiënt Python objecten op te slaan zodat deze ook na het heropstarten van de applicatie beschikbaar zijn. Het Pickle-bestandstype is Python specifiek waardoor andere programmeertalen niet in staat zijn dergelijke bestanden naar objecten om te vormen.

Concreet worden de “Pickle.dump()” en “Pickle.load()” functies gebruikt om respectievelijk een object op te slaan en opnieuw te laden.

### 4.1.2 PyInstaller

Er mag niet verondersteld worden dat de eindgebruiker de hierboven vermelde *libraries* wil of kan installeren op zijn/haar computer. Om deze reden wordt in de praktijk meestal niet de broncode maar een gebruiksvriendelijke distributie van de ontwikkelde applicatie afgeleverd aan de eindgebruikers. Dit vermijdt de noodzaak om alle gebruikte *libraries* alsook de gehanteerde programmeertaal te installeren op de computer. Elke distributie bevat een *executable* waarmee de applicatie gestart kan worden.

Python applicaties kunnen op eenvoudige wijze gevormd worden tot een gebruiksklare distributie door gebruik te maken van PyInstaller. Het grote voordeel van PyInstaller zit hem in het feit dat geen externe plug-ins of aanpassingen nodig zijn om alle gebruikte *libraries* samen te laten werken.

## 4.2 Console Applicatie

Met oog op resultaat werd in overleg met de promotoren beslist in eerste instantie een sobere console applicatie te ontwikkelen. Op deze manier kan volledig gefocust worden op de eigenlijke implementatie zonder tijd te verspillen aan *lay-out* wat bij een klassieke *graphical user interface* (GUI) of webapplicatie wel het geval zou zijn.

### 4.2.1 Gebruiksvriendelijkheid

Hoewel het programmeren van een console applicatie relatief snel verloopt kan de complexiteit die grote projecten teweegbrengt de gebruiksvriendelijkheid nadelig beïnvloeden. Dit komt vooral omdat Python scripts doorgaans als commando uitgevoerd worden waarbij de gebruiker eenmalig een reeks input variabelen meegeeft. Wanneer een console applicatie echter veel verschillende functionaliteiten aanbiedt zullen ook deze input variabelen variëren naargelang het gewenste gedrag met een groot aantal mogelijkheden tot gevolg. Tenzij de eindgebruiker elke mogelijke permutatie onthoudt zal veel opgezocht moeten worden hoe bepaalde resultaten bekomen kunnen worden. Om de gebruiksvriendelijkheid te vergroten werd daarom beslist een console applicatie te ontwikkelen waarmee de gebruiker stapsgewijs het gewenste gedrag duidelijk maakt. Om de hoeveelheid tekstuele gebruikersinput te beperken wordt zoveel mogelijk gebruik gemaakt van numerieke invoer waardoor fouten ten gevolge van schrijffouten vermeden kunnen worden. Concreet zal doorheen de applicatie genavigeerd worden via een reeks opeenvolgende keuzemenu's waarin acties gekoppeld worden aan gehele getallen. De eindgebruiker kan vervolgens het gewenste gedrag selecteren door eenvoudigweg de corresponderende numerieke waarde door te geven waarna eventueel een volgend specifiek menu verschijnt.

### 4.2.2 Werking keuzemenu's

De opeenvolging van verschillende keuzemenu's doet denken aan de werking van het "*state design pattern*"<sup>1</sup>. Hierbij verandert het gedrag van een object naargelang de toestand van diens interne staat. Denk ter illustratie bijvoorbeeld aan de werking van een drankautomaat; eerst wordt gewacht op een klant, daarna wordt een keuze gemaakt, hierna wordt het geld ontvangen en tot slot wordt het drankje afgegeven waarna het hele proces zich van vooraf aan herhaald.

Het idee achter de implementatie van de opeenvolgende keuzemenu's is vrij gelijkaardig aan die van het klassieke *state design pattern* met als grote verschil dat het gedrag niet langer opgeslagen wordt als kenmerk van een object maar toegekend wordt aan een variabele als lambda-functie. Bij conventie begint de naam van alle mogelijke lambda-functies met een *underscore*. Deze implementatie leidt tot een erg korte en eenvoudige *main function* weergegeven in Figuur 4.1. Hierop is te zien hoe aan de globale variabele "*state*" de functie "*\_home*" wordt toegekend. Hierna belandt

---

<sup>1</sup> Een *design pattern* is een theoretisch implementeerbaar model voor een in de praktijk vaak voorkomend probleem dat leidt tot efficiënte en makkelijk uitbreidbare code.

het programma in een lus waarin de functie gekoppeld aan de *state* variabele uitgevoerd wordt totdat deze overeenkomt met de “*\_exit*” functie.

Het effectief toekennen en veranderen van toestand wordt afgehandeld binnen elke functie zelf door eerst een aantal opties naar het scherm te schrijven en vervolgens te wachten op gebruikersinput. Dit gedrag wordt verwezenlijkt door gebruik te maken van een “*CliMenu*” object. Deze verwacht bij creatie een Python *dictionary* met daarin een aantal opties en bijbehorende nieuwe toestand als *key-value pairs*. Vervolgens wordt via de “*askInput()*” methode gewacht op een keuze van de eindgebruiker die ervoor zorgt dat de huidige *state* verandert. Een voorbeeld van deze declaratie en werkwijze voor de “*\_home*” functie bevindt zich in Figuur 4.2.

```
if __name__ == "__main__":
    state = _home
    while(state != _exit):
        os.system('cls' if os.name == 'nt' else 'clear')
        state()
```

Figuur 4.1: De main functie

```
def _home():
    homeMenu = CliMenu({"New Solution":_newSolution,
                        "View Solution":_viewSolution,
                        "Save all Solutions":_saveAllSolutions,
                        "Make Graphs":_makeGraphs,
                        "Extra":_extra,
                        "Exit":_exit},
                       "HOME")

    global state
    state = homeMenu.askInput()
```

Figuur 4.2: De *\_home* functie

### 4.2.3 Wiskundige modellen

#### (Geregulariseerde) Perona-Malik

Aangezien een werkende MATLAB-implementatie van zowel de standaard als geregulariseerde Perona-Malik vergelijking reeds voorhanden was in [1] moest deze code enkel nog vertaald worden naar Python. Dit verliep vrij vlot omdat de meest essentiële MATLAB-methoden ook in Python beschikbaar zijn, hoewel deze vaak een andere naam gekregen hebben. Bijgevolg konden grote stukken, mits syntax wijziging, gewoon overgenomen worden. Een goed voorbeeld van deze equivalentie is weergegeven in Figuur 4.3 waarin de gradiënt van een pixel wordt berekend.

<code>[Gx, Gy] = imgradientxy(diff1,'central');</code>	<code>(Gy, Gx) = np.gradient(diff1.astype('float32'))</code>
<code>Gx(:,1)=0;</code>	<code>Gx[:, 0] = 0</code>
<code>Gx(:,end)=0;</code>	<code>Gx[:, -1] = 0</code>
<code>Gy(1,:)=0;</code>	<code>Gy[0, :] = 0</code>
<code>Gy(end,:)=0;</code>	<code>Gy[-1, :] = 0</code>
<code>grad = (Gx.^2+Gy.^2).^(1/2);</code>	<code>grad = (Gx ** 2 + Gy ** 2) ** 0.5</code>

**Figuur 4.3:** Het berekenen van een gradiënt in MATLAB (links) en Python (rechts)

Uiteraard biedt de reeds geïmplementeerde MATLAB-code niet genoeg functionaliteiten waardoor de Python implementatie de reeds beschikbare code uitbreidt. Zo wordt telkens wanneer een beschadigde afbeelding verbeterd wordt, een nieuw “*Solution*” object aangemaakt. Dit object bevat alle gebruikte parameters alsook een lijst van “*result*” objecten die de eindresultaten, bekomen in de verschillende iteraties, voorstelt. Doordat elke oplossing als object opgeslagen wordt, kunnen gemakkelijk meerdere bewerkingen op verschillende afbeeldingen na elkaar uitgevoerd worden zonder dat voorgaande resultaten verloren gaan. Bijkomend kunnen, indien gewenst, deze oplossingen ook permanent opgeslagen worden als Pickle-bestanden waardoor deze na het heropstarten van de applicatie nog steeds beschikbaar zijn. Aangezien sommige methoden aanzienlijk veel CPU-tijd vereisen is deze functionaliteit geen overbodige luxe.

Deze implementatie vereist zeven of acht parameters naargelang de gewone of de geregulariseerde Perona-Malik vergelijking gebruikt wordt. De *startImgName* en *idealImgName* parameters stellen respectievelijk een padnaam naar de beschadigde en onbeschadigde afbeelding voor. Via de *rgb* parameter kan opgegeven worden of de afbeelding al dan niet in kleur verwerkt moet worden. Deze parameter kan twee mogelijke waarden aannemen: 0 of 1 die respectievelijk grijs- of kleurtinten voorstellen. De *option* parameter stelt de gebruikte diffusie-functie voor. Ook deze parameter kan twee waarden aannemen: 0 en 1 die respectievelijk de functies in (3.8) en (3.9) voorstellen. De macht in de geselecteerde diffusie-functie wordt bepaald door de parameter ‘a’. In de literatuur wordt hieraan steeds een waarde van 2 toegekend. Indien een waarde van 0 wordt opgegeven, wordt de diffusie homogeen waardoor de afbeelding globaal vervaagd wordt.

De *timestep* en *finaltime* parameters bepalen hoeveel iteraties uitgevoerd moeten worden (totaal aantal iteraties = eindtijd/tijdstap). Indien een string<sup>1</sup> opgegeven wordt als *finaltime* zal de applicatie de afbeelding blijven verbeteren tot zowel de beste MSE- als SSI-waarde bekomen zijn. Dit kan eenvoudig weg bereikt worden door elke iteratie de bekomen MSE- en SSI-waarden te vergelijken met de waarden uit de voorgaande iteratie. De Sigma parameter tot slot is enkel van toepassing voor de geregulariseerde methode en komt overeen met de gelijknamige parameter in (3.14).

### Fractionele diffusie

Van de in Hoofdstuk 3.3 beschreven GAFIA-methode op basis van fractionele diffusie was nog geen werkende implementatie beschikbaar. Aan de hand van de beschrijvingen in [10] kon ook dit model aan de applicatie toegevoegd worden hoewel hierbij wel enkele opmerkingen aangekaart moeten worden.

Ten eerste werd beslist om het std- (Figuur 3.1) en boli-masker (Figuur 3.2) programmatisch voor te stellen als tweedimensionale lijsten (ook wel matrices genaamd). Het genereren van deze lijsten bleek echter niet zo eenvoudig aangezien een logisch verband gezocht moest worden tussen de grootte van het masker en de locatie van de verschillende masker-waarden. Een oplossing voor dit probleem bestaat eruit het masker op te splitsen in een aantal kleinere, triviale matrices waarin dit verband erg eenvoudig is. Door vervolgens deze eenvoudige componenten te spiegelen en samen te voegen kan het gewenste masker bekomen worden. Een voorbeeld van dergelijke onderverdeling voor het boli-masker is weergegeven in Figuur 4.4 waarbij de verschillend gekleurde oppervlakken een triviaal onderdeel voorstellen.

$\xi_n$	0	...	0	$\xi_n$	0	...	0	$\xi_n$
0	$\ddots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\ddots$	0
$\vdots$	$\ddots$	$\xi_3$	0	$\xi_3$	0	$\xi_3$	$\ddots$	$\vdots$
0	...	0	$\xi_2$	$\xi_2$	$\xi_2$	0	...	0
$\xi_n$	...	$\xi_3$	$\xi_2$	$8\xi_1$	$\xi_2$	$\xi_3$	...	$\xi_n$
0	...	0	$\xi_2$	$\xi_2$	$\xi_2$	0	...	0
$\vdots$	$\ddots$	$\xi_3$	0	$\xi_3$	0	$\xi_3$	$\ddots$	$\vdots$
0	$\ddots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\ddots$	0
$\xi_n$	0	...	0	$\xi_n$	0	...	0	$\xi_n$

**Figuur 4.4:** De onderverdeling van het boli-masker in triviale componenten

<sup>1</sup> Een gegevenstype dat gebruikt wordt bij het programmeren om een stuk tekst voor te stellen.



Het voordeel van deze manier van werken is dat dezelfde code gebruikt kan worden om een masker te genereren ongeacht zijn grootte. In praktijk blijkt het spiegelen en samenvoegen van de verschillende componenten echter relatief traag waardoor de totale rekentijd nog hoger zal liggen dan reeds het geval is. Om dit probleem gedeeltelijk te voorkomen werd beslist de maskers met een grootte van drie, vijf en zeven pixels breed hard te coderen waardoor deze tijdrovende operaties vermeden kunnen worden. Bijgevolg kunnen de reketijden van maskers groter dan zeven pixels niet zomaar vergeleken worden met die van kleinere maskers.

Een tweede, onoplosbaar gegeven, die deze implementatie een pak trager maakt dan die van de Perona-Malik vergelijking vloeit voort uit het feit dat elke pixel in de convolutie berekening een eigen, unieke kern hanteert. Dit is een probleem aangezien de erg efficiënte “convolve2d” methode uit de SciPy *library* geen veranderlijke kern toelaat waardoor elke pixel afzonderlijk overlopen moet worden via een trage dubbele *for-lus*. De efficiëntie van deze operatie is bijgevolg afhankelijk van het aantal pixels in een afbeelding.

Bovendien zullen dergelijke operaties in *high-level* programmeertalen zoals Python van nature reeds trager verlopen dan in *mid* en *low-level* talen zoals C en assembly.

In sommige applicaties kan de rekentijd programmatisch verkort worden door gebruik te maken van *multithreading*. In deze gevallen worden de berekeningen namelijk parallel uitgevoerd waardoor onafhankelijke berekeningen gelijktijdig gestart kunnen worden. *Multithreading* biedt echter geen snelheidswinst op de hierboven beschreven wiskundige modellen aangezien elke iteratie voortbouwt op zijn voorganger en bijgevolg sowieso sequentieel berekend moet worden.

Deze implementatie maakt gebruik van zeven parameters waarvan drie (*startImgName*, *idealImgName* en *rgb*) reeds beschreven werden. De vierde parameter *mask* duidt aan welk masker gebruikt moet worden en kan twee mogelijk waarden aannemen: “std” (Figuur 3.1) en “boli” (Figuur 3.2). De grootte van het masker en het aantal iteraties worden respectievelijk meegegeven via de *maskSize* en *iterations* parameters. Indien de *iterations* parameter een string-waarde toegewezen krijgt wordt de afbeelding verbeterd totdat de beste MSE- en SSI-waarden bereikt zijn. De *noiseEstimation* parameter dient als een indicatie voor de hoeveelheid ruis in de beschadigde afbeelding en zal bepalen hoeveel procent van de pixels uiteindelijk geconvolveerd zullen worden. Concreet worden hiervoor eerst alle pixels gerangschikt volgens dalende gradiënt waarna enkel de eerste x procent, aangeduid door de *noiseEstimation* parameter, effectief behandeld worden. Een goed geschatte waarde kan voorkomen dat onbeschadigde pixels foutief aangepast worden waardoor het eindresultaat minder afwijkt van de ideale afbeelding. Deze parameter neemt een waarde aan tussen 0 en 1 en wordt als een percentage geïnterpreteerd (0.5 = 50%).

Een samenvattende opsomming van alle mogelijke parameters van alle geïmplementeerde modellen is terug te vinden in de gebruikershandleiding (Hoofdstuk 5, Tabel 5.1).

#### 4.2.4 Beoordelingsmethoden

Om de invloed van de verschillende modellen en bijbehorende parameters te kunnen onderzoeken moet een kwantitatieve beoordelingsmethode vastgelegd worden. Het is namelijk niet mogelijk om consistent de verschillende eindresultaten visueel met elkaar te vergelijken en hieruit objectief te bepalen welke het beste resultaat oplevert. Om standvastige beoordelingen te bekomen zal gebruik gemaakt worden van zogenaamde “*reference Image Quality Assessment*” algoritmen. Deze zullen een praktisch bekomen eindresultaat vergelijken met het theoretisch optimale eindresultaat (lees: onbeschadigde afbeelding). Op basis hiervan wordt een numerieke waarde berekend die indiceert hoe gelijkend beide oplossingen zijn. Deze technieken kunnen bijgevolg enkel toegepast worden in testomgevingen waar een onbeschadigde afbeelding voorhanden is. In praktijk beschikt men namelijk niet over dergelijk ideaal eindresultaat aangezien het verbeteren van een beschadigde afbeelding dan triviaal zou zijn. Als alternatief zouden “*no-reference Image Quality Assessment*” algoritmen gebruikt kunnen worden. Deze methoden zijn echter veel ingewikkelder en liggen buiten het kader van deze masterproef.

Een eerste, veel voorkomende, beoordelingsmethode is de zogenaamde *peak signal-to-noise ratio* (PSNR) [15]:

$$PSNR = 10 \cdot \log_{10} \left( \frac{MAX_I^2}{MSE} \right) \quad (4.1)$$

met

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2. \quad (4.2)$$

Dit is een technische term voor de verhouding tussen de maximale intensiteit ( $MAX_I$ ) van een signaal en de kracht van de hoeveelheid ruis die de betrouwbaarheid van de weergave beïnvloedt ( $MSE$ ). Omdat veel signalen een zeer breed dynamisch bereik hebben, wordt PSNR meestal uitgedrukt in termen van de logaritmische decibelschaal. Deze beoordelingsmethode is grotendeels afhankelijk van de in (4.2) weergegeven *mean squared error* (MSE) wat de cumulatief gekwadraterde fout tussen de beschadigde ( $K$ ) en het originele beeld ( $I$ ) voorstelt. Twee nagenoeg gelijke afbeeldingen leveren bijgevolg een lage MSE-waarde op wat resulteert in een hoge PSNR wat duidt op een beter resultaat.

Aangezien de PSNR uitgedrukt wordt in decibel zal tijdens de experimenten louter rekening gehouden worden met de intuïtievare MSE-waarden. Per conventie wordt een verschil tussen twee MSE-waarden van tien of minder fouten geïnterpreteerd als twee visueel identieke oplossingen.

Hoewel deze beoordelingsmethode een goede indicatie vormt omtrent het aantal afwijkende pixelwaarden heeft deze ook enkele belangrijke tekortkomingen [16]. Indien bijvoorbeeld het bekomen eindresultaat, op kleur of helderheid na, identiek is aan de ideale afbeelding zal de MSE-waarde toch enorm groot zijn (Figuur 4.5). De omgekeerde situatie kan zich ook voordoen; indien het eindresultaat visueel verschilt maar toch dezelfde kleuren en helderheid als de originele afbeelding heeft, kan het voorkomen dat de MSE-waarde toch relatief laag is (Figuur 4.6). Om deze reden is het interessant om een aanvullende beoordelingsmethode te gebruiken die deze zwakten aanvult: de *structural similarity index* (SSI of SSIM) [17].

$$SSIM(x, y) = [l(x, y)^\alpha \cdot c(x, y)^\beta \cdot s(x, y)^\gamma] \quad (4.3)$$

met

$$l(x, y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \quad (4.4)$$

$$c(x, y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \quad (4.5)$$

$$s(x, y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3} \quad (4.6)$$

Deze techniek zal, in tegenstelling tot de MSE, beide afbeeldingen opsplitsen in kleinere gebieden ( $x$  en  $y$ ) en deze vervolgens onderling vergelijken. Hierbij wordt rekening gehouden met de belichting ( $l$ ), contrast ( $c$ ) en structurele gelijkheid ( $s$ ) van een groter aantal pixels. De symbolen  $\mu$  en  $\sigma$  stellen respectievelijk het gemiddelde en de variantie van de pixel-waarden in het oppervlak voor. De variabelen  $c_1$ ,  $c_2$  en  $c_3$  zijn ingevoerd om een deling met zwakke deler te stabiliseren en worden als volgt gedefinieerd:

$$c_1 = (k_1 L^2) \quad (4.7)$$

$$c_2 = (k_2 L^2) \quad (4.8)$$

$$c_3 = c_2/2 \quad (4.9)$$

waarbij

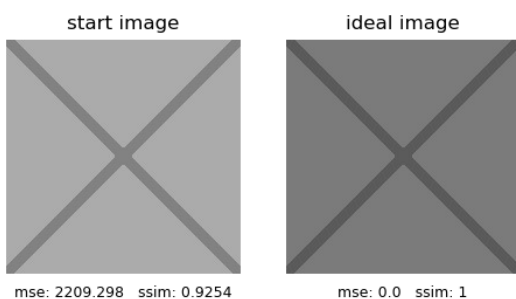
$$L = 2^{\#bits \text{ per pixel}} - 1 \quad (4.10)$$

$$k_1 = 0.01 \quad (4.11)$$

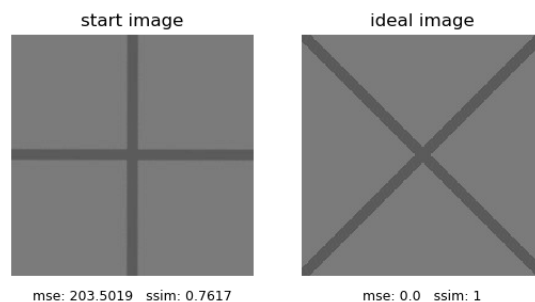
$$k_2 = 0.03 \quad (4.12)$$

De SSI levert een resultaat gelegen tussen 0 en 1 op dat geïnterpreteerd kan worden als een percentage van gelijkheid. Zo zullen twee identieke afbeeldingen een SSI-waarde opleveren van 1 wat overeenkomt met 100% gelijkheid. In Figuur 4.5 is te zien is hoe de SSI-waarde desondanks het kleurverschil erg goed is terwijl ze In Figuur 4.6, zoals verwacht, een pak lager ligt. De implementatie van deze beoordelingsmethode maakt gebruik van de “`ssim()`” methode uit de *skimage library*.

Ook deze beoordelingsmethode is niet altijd perfect. Zo zal het eindresultaat met optimale SSI-waarde vaak iets waziger zijn dan het eindresultaat met optimale MSE-waarde waardoor dit resultaat visueel minder goed oogt. Het is daarom belangrijk in te zien dat beide beoordelingsmethoden elkaars zwakten aanvullen en samen een objectief beeld kunnen schetsen over de kwaliteit van het bekomen eindresultaat. Er wordt dus gestreefd naar een eindresultaat met een zo laag mogelijke MSE-waarde en zo hoog mogelijke SSI-waarde.



**Figuur 4.5:** De MSE- en SSI-waarden bij verschillende helderheid



**Figuur 4.6:** De MSE- en SSI-waarden bij verschillende structuur

Een laatste, niet onbelangrijke, beoordelingsmethode is de vereiste rekestijd per iteratie; de “*calculationTime*” genaamd. Deze wordt tijdens de berekeningen programmatisch opgemeten en kan doorslaggevend zijn indien twee afbeeldingen visueel gelijke resultaten opleveren.

## 4.3 Testen

Zoals bij elke softwareapplicatie is het belangrijk het ontwikkelde programma uitvoerig te testen alvorens te publiceren. Een goede manier om dit te verwezenlijken is door de applicatie te laten uitproberen door een divers aantal proefpersonen die a priori geen ervaring hebben met de ontwikkelde applicatie. Deze test is belangrijk omdat de programmeur op deze manier doelgericht feedback ontvangt omtrent de moeilijkheden van de eindgebruikers. Concreet werd aan vijf proefpersonen gevraagd enkele beschadigde afbeeldingen te verbeteren. Aan drie van de vijf proefpersonen werd bovendien gevraagd de gebruikershandleiding (Hoofdstuk 5) door te nemen, de twee anderen moesten op geheel intuïtieve wijze de werking van de applicatie achterhalen. Na wat *trial and error* slaagden beide groepen erin de beschadigde afbeelding te verbeteren. Niet verwonderlijk had de groep die de gebruikershandleiding niet gelezen had meer tijd nodig om dit te verwezenlijken. De grootste initiële ergernis was het feit dat de muis niet gebruikt kon worden bij het aanduiden van de gewenste optie. Helaas zijn dergelijke interacties onmogelijk in console applicaties waardoor dit “probleem” niet opgelost kan worden. Alle proefpersonen vonden na afloop de numerieke navigatie een goed en efficiënt alternatief voor het gebruik van een muis.

Een tweede type test maakt gebruik van het feit dat de ontwikkelde applicatie de mogelijkheid biedt een aantal oplossingen te genereren vanuit een correct opgesteld Json-bestand<sup>1</sup>, een “*solution file*” genaamd (meer hierover in onderdeel 5.2). Dit kan namelijk gebruikt worden om de robuustheid van de applicatie in kader van foute gebruikersinput uit te testen. Concreet wordt het programma gevraagd een aantal oplossingen te genereren vanuit een specifiek Json-bestand waarin sommigen parameters een doelbewust foutieve waarde opgegeven krijgen. Denk hierbij bijvoorbeeld aan een numerieke input waar een tekstuele waarde verwacht wordt. De bedoeling van deze test is om na te gaan of de applicatie deze fouten correct opvangt. Indien dit niet het geval is zal de applicatie namelijk crashen waardoor deze automatisch afgesloten wordt en alle niet-opgeslagen oplossingen onherroepelijk verloren gaan. Deze test werd doorheen de ontwikkeling van de applicatie verscheidene malen herhaald om zo snel mogelijk problemen op te sporen.

---

<sup>1</sup> Ook wel *JavaScript Object Notation* genoemd, wordt gebruikt om data uit te wisselen.

## 5 Functionaliteiten en Gebruikershandleiding

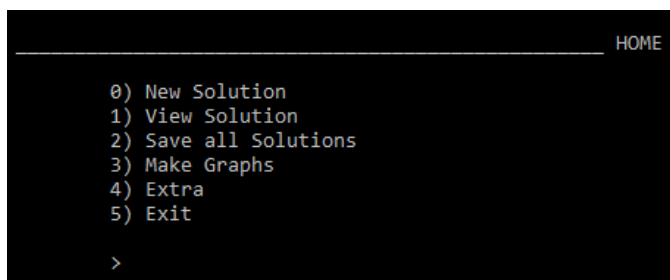
In volgend onderdeel worden alle geïmplementeerde functionaliteiten van het programma beschreven alsook hoe de eindgebruiker hiermee overweg moet gaan. Zoals reeds vermeld werd in overleg met de promotoren besloten een *command line interface* applicatie te ontwikkelen geschreven in Python. Hierbij werd geprobeerd een zo intuïtief mogelijke bediening te ontwikkelen die minimale gebruikersinput vereist.

### 5.1 Home-menu

De applicatie kan opgestart worden door het “main.exe” bestand in de installatiefolder tweemaal aan te klikken. Dit zal een console venster openen dat na het laden van de applicatie het “Home” menu (Figuur 5.1) weergeeft. Deze operatie kan, naargelang het systeem, enkele seconden in beslag nemen. Indien het lettertype te groot of te klein weergegeven wordt kan deze op Windows aangepast worden door een rechtermuisklik op de dikke, witte balk bovenaan en vervolgens de optie “eigenschappen” te selecteren.

In het Home-menu kan de gebruiker een keuze maken uit zes mogelijke acties door het nummer, overeenkomstig met de gewenste operatie, te typen en vervolgens op de enter-toets te duwen. Op deze manier kan met minimale input genavigeerd worden doorheen de applicatie.

Nadat het programma een eerste maal gestart wordt, bevindt zich in de installatie folder een configuratiebestand genaamd “config.ini”. Hierin bevinden zich enkele default locaties die onder andere gebruikt worden bij het opslaan van oplossingen (“./\_data/saves”), laden van afbeeldingen (“./\_data/images”) en exporteren van grafieken (“./\_data/graphs”). Al deze paden leiden naar een locatie binnen de “\_data” map in de installatie folder. Hoewel dit niet aangeraden wordt, kunnen deze locaties, indien gewenst, manueel aangepast worden door het configuratie bestand aan te passen. Moest dit bestand verwijderd of verplaatst worden, dan wordt deze automatisch opnieuw gegenereerd. Dit bestand vermijdt het herhaaldelijk intypen van eenzelfde pad-locatie. Wanneer bijvoorbeeld een beschadigde afbeelding opgegeven moet worden dat zich in de default locatie “./\_data/images” bevindt, volstaat het de naam en extensie van deze afbeelding als parameter mee te geven in plaats van de volledige absolute padnaam. Hieronder worden de verschillende beschikbare opties in het home-menu verder besproken.



```
HOME
0) New Solution
1) View Solution
2) Save all Solutions
3) Make Graphs
4) Extra
5) Exit
>
```

Figuur 5.1: Het home-menu

## 5.2 New solution

Het “*new solution*” menu, weergegeven in Figuur 5.2, stelt de gebruiker in staat een door ruis beschadigde afbeelding te herstellen door middel van zes geïmplementeerde wiskundige modellen: Perona-Malik, geregulariseerde Perona-Malik, fractionele diffusie en de drie in Hoofdstuk 2.3 besproken basismethoden (“Mean”, “Gaussian” & “Median”). Nadat de gebruiker het gewenste model geselecteerd heeft via opties 0 tot 3 zal gevraagd worden de vereiste parameterwaarden op te geven. De verschillende parameters zijn opgesomd in Tabel 5.1 aangevuld met het datatype, een korte beschrijving en de eventuele mogelijke waarden. Indien een string opgegeven wordt waarin zich spaties bevinden moet deze tussen dubbele aanhalingstekens geplaatst worden.

Tijdens de experimenten zou men het gedrag van een parameter kunnen bestuderen door verschillende oplossingen te genereren waarin deze parameter verschillende waarden aanneemt. Deze werkwijze vereist echter dat voor elke oplossing ook steeds de andere, onveranderde parameters opnieuw ingevuld moeten worden. Om dit vervelend en tijdrovend werk te vermijden werd voor integer en float parameters een sneller alternatief geïmplementeerd. In plaats van één waarde op te geven kunnen dergelijke parameters ook uit drie waarden, gescheiden door een spatie (of komma in een *solution file*; zie later) bestaan. De betekenis van deze drie waarden (a,b en c) is als volgt: “Bereken  $(b-a+1) / c$  oplossingen waarbij de parameterwaarde veranderd van a tot b in stappen van c”. Ter illustratie bevinden zich in Figuur 5.3 de parameterwaarden voor de generatie van vijf oplossingen waarbij de a-waarde verhoogd wordt van één tot vijf in stappen van één.

Indien men liever alle parameterwaarden voor een groot aantal oplossingen a priori ingeeft, kan gekozen worden voor optie vier: “From file”. Deze keuze vereist een correct opgesteld Json-bestand waarin alle noodzakelijk parameters van de gekozen methode ingevuld zijn. Een voorbeeld van zo’n *solution file* is weergegeven in Figuur 5.4. Hierin bevinden zich alle mogelijke parameters van alle geïmplementeerde modellen gevolgd door een dubbele punt en vierkante-haakjes. Tussen deze haakjes dient, indien van toepassing, een parameterwaarde ingevuld te worden. De extra parameter “*method*” zal bepalen welk wiskundig model gebruikt wordt en bijgevolg welke parameters daadwerkelijk een waarde moeten krijgen. De drie mogelijke waarden die deze “*method*” parameter aanneemt zijn: “peronamalik”, “fractional” of “basic”. Indien, zoals in het voorbeeld, gebruik gemaakt wilt worden van de geregulariseerde Perona-Malik implementatie kent men een waarde toe aan de sigma parameter, anders laat men deze leeg. Ook alle andere parameters die niet van toepassing zijn hoeven geen waarde toegekend te krijgen. Opnieuw kunnen float of integer parameters drie waarden toegekend krijgen om snel meerdere oplossingen te genereren (ditmaal gescheiden door een komma). Bijkomend wordt vermeld dat in Json-bestanden strings tussen aanhalingstekens geplaatst moeten worden. Indien gewenst kan een leeg Json-bestand met correcte structuur automatisch gegenereerd worden via het Extra-menu (optie vier in het Home-menu).

```
NEW
0) PeronaMalik
1) PeronaMalik-Regularized
2) Fractional diffusion
3) Basic noise removal
4) From file
5) Home
>
```

**Figuur 5.2:** Het new solution-menu

```
> noisy Image = flor-noise50.jpg
> ideal Image = flor.jpg
> a = 1 5 1
> timestep = 0.5
> finaltime = 10
> option = 0
> RGB = 0
```

**Figuur 5.3:** De parameterwaarden voor de generatie van vijf oplossingen waarbij de a-waarde verhoogd wordt van één tot vijf in stappen van één

```
{
  "solutions": [
    {
      "method": ["peronamalik"],
      "rgb": [0],
      "startImgName": ["flor-noise50.jpg"],
      "idealImgName": ["flor.jpg"],
      "a": [2, 6, 2],
      "timestep": [0.5],
      "finaltime": [10],
      "option": [0],
      "sigma": [0.5],
      "mask": [],
      "maskSize": [],
      "iterations": [],
      "noiseEstimation": []
    }
  ]
}
```

**Figuur 5.4:** Een voorbeeld van een correct opgesteld *solution file*



**Tabel 5.1:** De verschillende invoerparameters

Model	Parameter	Type	Beschrijving
/	Method	string	Enkel noodzakelijk indien met een <i>solution file</i> gewerkt wordt. Bevat de naam van het gewenste wiskundig model. Mogelijke waarden: "peronamalik", "fractional" of "basic"
All	startImgName	string	De volledige padnaam met extensie van de oorspronkelijke afbeelding of enkel de naam met extensie indien de afbeelding zich in de default locatie bevindt.
All	ideallmgName	string / none	De volledige padnaam met extensie van de onbeschadigde afbeelding of enkel de naam met extensie indien de afbeelding zich in de default locatie bevindt.  Indien dergelijke afbeelding niet beschikbaar is, gewoon leeg laten.
All	rgb	integer	Twee mogelijke waarden: 0 (= zwart wit) of 1 (= kleur)
Perona-Malik / Reg Perona-Malik	a	float	De macht gebruikt in de geselecteerde diffusie-functie. Bijzondere waarden: a = 2 → standaard Perona-Malik a = 0 → homogene diffusie
Perona-Malik / Reg Perona-Malik	timestep	float	De tijdstap.
Perona-Malik / Reg Perona-Malik	finaltime	float/ string	De eindtijd. Indien een string opgegeven wordt zal pas gestopt worden indien zowel MSE als SSI een optimale waarde bereikt hebben. Wanneer echter geen ideale afbeelding opgegeven werd kunnen geen MSE/SSI-waarden berekend worden en zal een eindtijd van 10 verondersteld worden.
Perona-Malik / Reg Perona-Malik	option	integer	Bepaalt de te gebruiken diffusie-functie. Twee mogelijke waarden: 0 (= formule (3.8)) of 1 (= formule (3.9))
Reg Perona-Malik	sigma	float	De regularisatie parameter gebruikt in de warmtevergelijking.
Basic method / Fractional diffusion	mask	string	Het masker gebruikt in de concolutieberekening. Mogelijke waarden voor de basismethoden: "mean", "gaussian" en "median". Mogelijke waarden voor fractionele diffusie: "std" en "boli".
Basic method / Fractional diffusion	maskSize	integer	De grootte van het gebruikte masker. Dit moet steeds een oneven geheel getal zijn groter dan één.
Basic method / Fractional diffusion	iterations	integer / string	Het aantal keren dat de berekeningen herhaald worden alvorens gestopt wordt. Indien een string opgegeven wordt zal pas gestopt worden indien zowel de MSE als SSI een optimale waarde bereikt hebben. Wanneer echter geen ideale afbeelding opgegeven werd kunnen geen MSE/SSI-waarden berekend worden en zullen drie iteraties verondersteld worden.
Fractional diffusion	noiseEstimation	float	Een waarde tussen nul en één die bepaald hoeveel procent van het totaal aantal pixels aangepast zullen worden.

## 5.3 View solution

Via de “*view solution*” optie in het Home-menu kan de gebruiker een reeds gegenereerde oplossing selecteren waarna men een gedetailleerd overzicht te zien krijgt van de gekozen oplossing. Een voorbeeld van dergelijk overzicht is weergegeven in Figuur 5.5. ook na het succesvol genereren van één of meerdere nieuwe oplossingen zal eenzelfde overzicht verschijnen van de laatst gegenereerde oplossing.

In dit overzicht bevinden zich bovenaan alle opgegeven parameters gevolgd door een opsomming van de gegenereerde resultaten. Per iteratie worden de MSE, SSI, PSNR en *calculationTime* waarden weergegeven. Onderaan bevindt zich een menu met enkele mogelijke acties.

### 5.3.1 View results

Indien voor de eerste optie: “*View results*” gekozen wordt kunnen één of meerdere gegenereerde resultaten (lees: verbeterde afbeeldingen) bekeken worden. Het programma verwacht hiervoor één of meerdere iteratie nummers, gescheiden door een spatie. Indien de resultaten met extreme MSE en SSI-waarden bekeken willen worden, kan ook de string “best” opgegeven worden. Indien bij generatie geen onbeschadigde afbeelding meegegeven werd, zullen de MSE- en SSI-waarden niet langer een indicatie zijn voor de kwaliteit van het eindresultaat maar voor de afwijking t.o.v. de beschadigde afbeelding. Bijgevolg zal het programma een schatting proberen maken van het beste eindresultaat. Deze is echter onbetrouwbaar wegens gebrek aan verfijning waardoor een visuele beoordeling van de verschillende eindresultaten ten zeerste aangewezen is. Eender welke andere input brengt de gebruiker terug naar het overzicht.

### 5.3.2 Export results as images

Deze optie werkt gelijkaardig als hierboven en zal de geselecteerde resultaten exporteren als afbeeldingen naar de opgegeven locatie met de opgegeven naam. Wanneer de naam niet voorafgegaan wordt door een padnaam wordt de afbeelding geëxporteerd naar de in het configuratiebestand vermelde default locatie. Indien de opgegeven naam geen extensie bezat zal de afbeelding automatisch als Tiff-bestand geëxporteerd worden. Deze extensie comprimeert een afbeelding zonder dataverlies waardoor geen kwaliteit verloren gaat.

### 5.3.3 Save solution

Wanneer een oplossing gegenereerd wordt, wordt deze eerst in het intern geheugen opgeslagen. Dit wil zeggen dat de oplossing beschikbaar is zolang de applicatie actief is. Wanneer het programma afgesloten wordt, zullen alle oplossingen die zich in het geheugen bevinden verwijderd worden. Via optie 2: “*Save solution*” kan een oplossing permanent opgeslagen worden als een Pickle-bestand waardoor deze na het heropstarten van de applicatie nog steeds beschikbaar zal zijn.

```

SOLUTION
- Used parameters:
  'Method' = 'PeronaMalik'
  'startImgName' = 'flor-noise50.jpg'
  'a' = 2.0
  'timestep' = 0.5
  'finaltime' = 3.0
  'option' = 0
  'rgb' = 0
  'idealImgName' = 'flor.jpg'
  'callback' = <function progresscallback at 0x0000017AAFD61268>
- Results:
Iteratie: 0      mse  1045.4019      ssi  0.3072      psnr  17.938      calculationTime  1.694
Iteratie: 1      mse  550.0311      ssi  0.4056      psnr  20.7269      calculationTime  1.694
Iteratie: 2      mse  336.3043      ssi  0.5031      psnr  22.8635      calculationTime  1.9309
Iteratie: 3      mse  243.9466      ssi  0.5879      psnr  24.2579      calculationTime  1.758
Iteratie: 4      mse  207.3112      ssi  0.6483      psnr  24.9646      calculationTime  1.6526
Iteratie: 5      mse  197.0126      ssi  0.6829      psnr  25.1859      calculationTime  1.6417
Iteratie: 6      mse  199.6089      ssi  0.698       psnr  25.129      calculationTime  2.0548

----- SOLUTION
0) View results
1) Export results as images
2) Save solution
3) Home
>

```

Figuur 5.5: Het view solution-menu

## 5.4 Save all solutions

Indien een groot aantal gegenereerde oplossingen opgeslagen moeten worden kan dit eenvoudigweg via de “*Save all solutions*” optie in het home-menu. Dit zorgt ervoor dat alle nog niet opgeslagen oplossingen opgeslagen worden als Pickle-bestanden.

## 5.5 Make graphs

Via de “*Make graphs*” optie in het home-menu kunnen de gegevens van één of meerdere oplossingen in grafieken weergegeven worden. De gebruiker heeft de keuze uit een viertal geïmplementeerde grafieken, weergegeven in Figuur 5.6. Bijkomend bevindt zich rechts van elke grafiek een tabel met relevante waarden. Houdt rekening dat de meerderheid van deze grafieken steunen op MSE-, SSI- of PSNR-waarden. Bijgevolg moet hiervoor in de oplossingen een onbeschadigde afbeelding beschikbaar zijn.

```

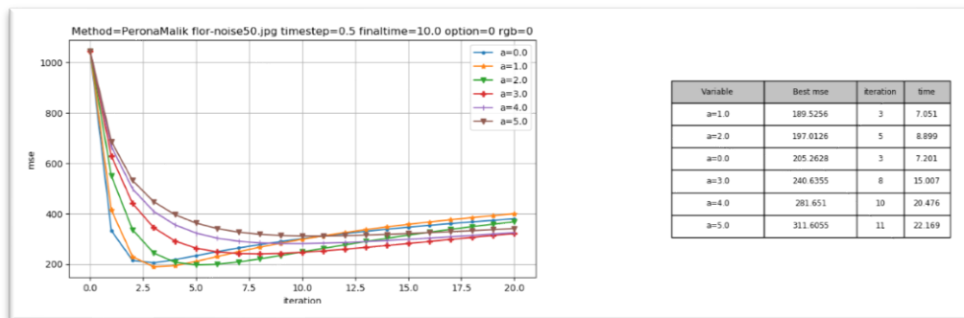
----- MAKE GRAPH
0) assessment graph
1) extremum graph
2) improvement graph
3) CPU time graph
4) generate all graphs & save
5) Home
>

```

Figuur 5.6: Het make graphs-menu

### 5.5.1 Assessment graph

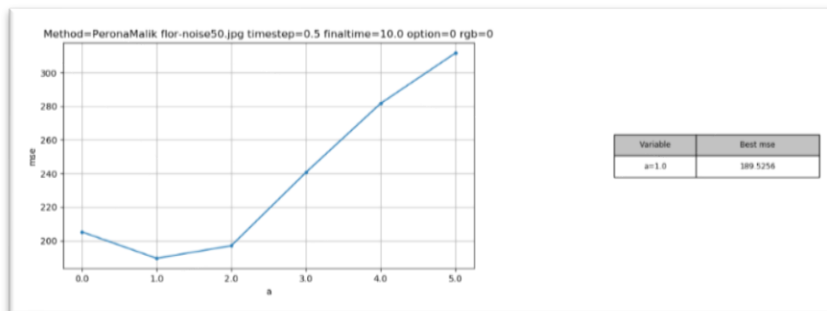
Op dergelijk grafieken wordt een, door de gebruiker opgegeven, *assessment* van één of meerdere oplossingen weergegeven in functie van de iteratie. De mogelijke *assessments* zijn: “mse”, “ssi”, “psnr” of “*calculationTime*”. Indien meerdere *assessments* opgegeven worden (gescheiden door een spatie), zullen de verschillende *assessment graphs* één voor één weergegeven worden in een nieuw venster. Een voorbeeld van de MSE-*assessment graph* en bijhorende tabel van zes oplossingen met variërende a-waarde is terug te vinden in Figuur 5.7 In de tabel bevinden zich per oplossing de extreme *assessment* waarde aangevuld met de iteratie waarin deze optreedt alsook de vereiste rekestijd alvorens deze gevonden werd.



**Figuur 5.7:** Een voorbeeld van de MSE-*assessment graph* en bijhorende tabel

### 5.5.2 Extremum graph

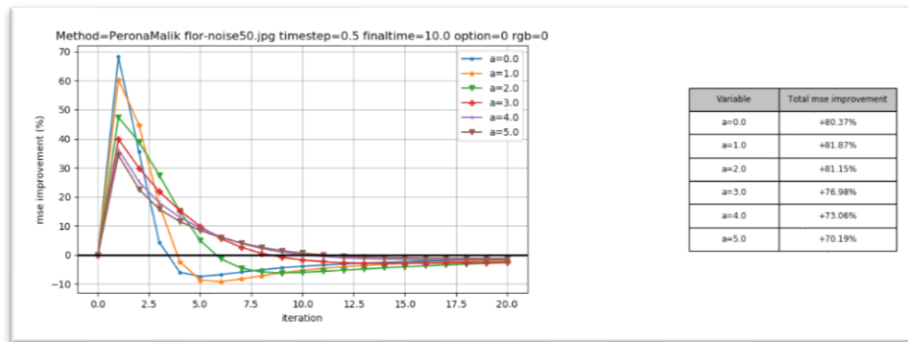
Vaak is enkel de extreme *assessment* waarde interessant aangezien deze wijst op een optimaal eindresultaat. Indien de extreme *assessment* waarden van veel verschillende oplossingen vergeleken willen worden kan men hiervoor de *extremum graph* gebruiken. Deze zal de verschillende optimale *assessment* waarden van verschillende oplossingen weergegeven in één curve in functie van de veranderlijke variabelen. Op deze manier kan snel onderzocht worden welk effect een veranderende parameterwaarde heeft op het best bekomen eindresultaat. Opnieuw moet de gebruiker eerst een aantal oplossingen selecteren. Daarna moet het te gebruiken *assessment* (“mse”, “ssi”, “psnr” of “*calculationTime*”) opgegeven worden. De *extremum graph* en bijhorende tabel van zes oplossingen met veranderlijke a-waarde is weergegeven in Figuur 5.8. In de tabel bevindt zich de beste extreme waarde van alle geselecteerde oplossingen.



**Figuur 5.8:** Een voorbeeld van de MSE *extremum graph* en bijhorende tabel

### 5.5.3 Improvement graph

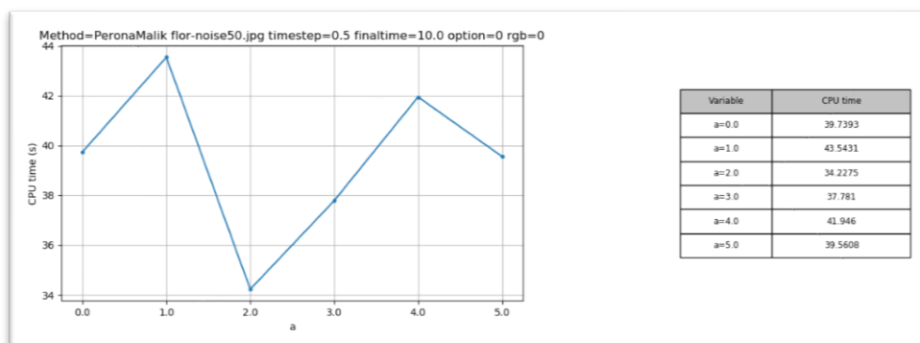
Indien bijvoorbeeld de beschadigde afbeelding veranderd doorheen de geselecteerde oplossingen kunnen de verschillende MSE- en SSI-waarden niet zomaar vergeleken worden. In dergelijk gevallen kan de *improvement graph* gebruikt worden om toch een beeld te krijgen over welke oplossing de oorspronkelijke afbeelding het meest verbeterd heeft. Deze grafiek geeft per iteratie de verbetering van het geselecteerde *assessment* ten opzichte van voorgaande iteratie uitgedrukt in procent weer. Een positieve waarde duidt op een verbetering, een negatieve waarde op een verslechtering. Bijgevolg zal de totale oppervlakte gevormd door het positieve deel van de grafiek en de x-as een indicatie geven voor de totale verbetering ten opzichte van de beginsituatie. Deze totale verbetering per oplossing is weergegeven in de bijhorende tabel. In Figuur 5.9 bevinden zich de MSE *improvement graph* en grafiek van zes oplossingen met variërende a-waarden.



**Figuur 5.9:** Een voorbeeld van de MSE *improvement graph* en bijhorende tabel

### 5.5.4 CPU time graph

Een niet onbelangrijk detail is de benodigde CPU-tijd om een aantal iteraties uit te voeren. Zeker in de beeldverwerking kunnen, afhankelijk van de gebruikte afbeelding, de totale rekestijden hoog oplopen. Indien het effect van een parameterwaarde op de totale CPU-tijd onderzocht wilt worden kan men gebruik maken van de *CPU time graph*. Deze sommeert de verschillende rekestijden van elke iteratie per oplossing en geeft deze weer op één curve in functie van de variabele parameters. De gebruiker dient hiervoor enkel de te gebruiken oplossingen op te geven. Een voorbeeld voor zes oplossingen bekomen via Perona-Malik met variabele a-waarde is weergegeven in Figuur 5.10.



**Figuur 5.10:** Een voorbeeld van de *CPU time graph* en bijhorende tabel

### 5.5.5 Generate all graphs & save

Het manueel genereren van alle bovenstaande grafieken voor alle mogelijke assessments is een lang en repetitief proces dat gelukkig vermeden kan worden via de *Generate all graphs & save* optie. Deze zal, zoals de naam al doet vermoeden, alle mogelijke grafieken genereren en opslaan als afbeeldingen in de opgegeven locatie. Ten eerste wordt aan de gebruiker gevraagd welke oplossingen gebruikt moeten worden om de grafieken te genereren. Vervolgens kan een locatie doorgegeven worden naar waar de grafieken geëxporteerd moeten worden. Indien de gebruiker geen pad meegeeft wordt de in het configuratiebestand vermelde *default* locatie gebruikt.

Tot slot wordt een primaire en secundaire folder naam gevraagd. Deze zullen als volgt aan het opgegeven pad toegevoegd worden: pad/primair/secundair. Op deze manier kan de gebruiker een overzichtelijke structuur creëren en snel de opgeslagen grafieken terugvinden.

## 6 Experimenteren en parameterstudie

In dit Hoofdstuk wordt de ontwikkelde applicatie gebruikt om de invloed van de verschillende parameters op het bekomen eindresultaat te bestuderen. Dit gebeurt aan de hand van een reeks gerichte testen waarin één of meerdere parameters als variabelen beschouwd worden. Deze testen worden geformuleerd aan de hand van een onderzoeksvraag, eventueel aangevuld met bijkomende bedenkingen of testen naargelang het resultaat.

### 6.1 Standaard parameterwaarden

Om de invloed van een parameter op het eindresultaat te onderzoeken is het belangrijk de andere parameters constant te houden. Om deze reden wordt aan elke mogelijke veranderlijke op voorhand een standaard parameterwaarde toegekend, weergegeven in Tabel 6.1.

**Tabel 6.1:** De gebruikte standaardparameterwaarden

Parameter	Standaardwaarde
startImgName	"flor-noise50.jpg"
ideallmgName	"flor.jpg"
rgb	0
a	2
timestep	0.5
finaltime	10
option	0
sigma	0.5
mask	"std"
maskSize	5
iterations	"best"
noiseEstimation	0.5

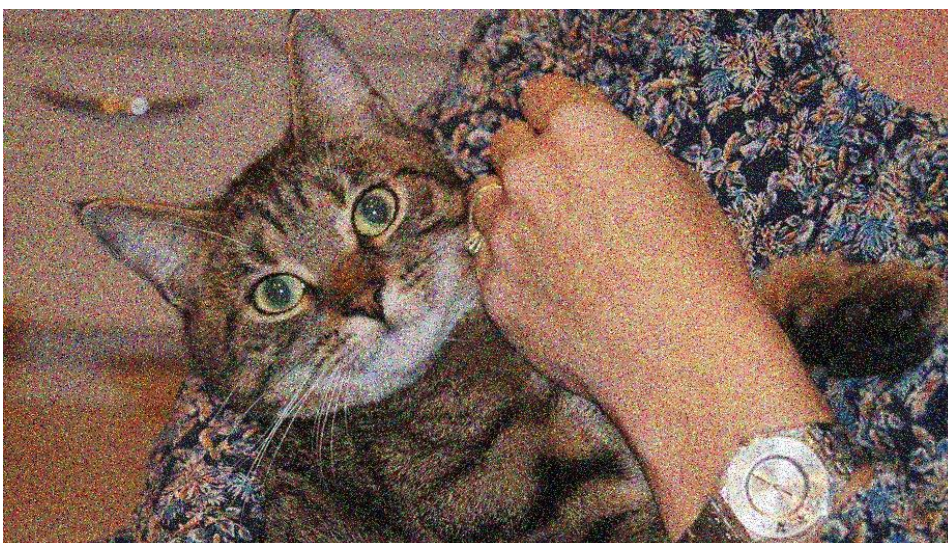
De afbeelding waarmee het meest gewerkt zal worden is “flor.jpg”, weergegeven in Figuur 6.1. Deze afbeelding is interessant vanwege de grote variëteit aan verschillende texturen en oppervlakken. De snorharen van de kat doen dienst als fijne dunne lijnen, het motief van de blouse en vacht voorzien voldoende randen en details, het hand vormt in essentie een egaal oppervlak en de achtergrond is wazig waardoor deze geen randen bezit. Ter beschadiging wordt aan deze afbeelding 50% Gaussische ruis toegevoegd (Figuur 6.2). In praktijk komen dergelijke extreme gevallen van beschadiging gelukkig zelden voor.

Om de productiviteit te verhogen wordt standaard zoveel mogelijk gewerkt met zwart-wit afbeeldingen aangezien deze een pak sneller verwerkt kunnen worden.

Tijdens de testen worden de in Tabel 6.1 vermelde standaard parameters verondersteld indien een bepaalde parameter waarde niet expliciet vernoemd wordt.



**Figuur 6.1:** De standaardafbeelding flor.jpg<sup>1</sup> gebruikt doorheen de experimenten



**Figuur 6.2:** flor.jpg beschadigd met 50% Gaussische ruis

<sup>1</sup> De afbeelding werd vernoemd naar de naam van de kat.



## 6.2 Experimenteren: Perona-Malik

### 6.2.1 Wat is de invloed van parameter $a$ op het eindresultaat?

In de standaard Perona-Malik vergelijking wordt de gebruikte  $a$ -waarde gelijkgesteld aan twee. Om het effect van deze parameter op het eindresultaat na te gaan zal de standaard afbeelding (Figuur 6.2) enkele malen verbeterd worden met  $a$ -waarden variërend van nul tot vijf in stappen van één. Ter herhaling wordt vermeld dat een  $a$ -waarde van nul overeenkomt met homogene diffusie wat resulteert in een globaal wazige afbeelding.

De beste MSE en SSI-waarden per permutatie zijn weergegeven in Tabel 6.2.

Hieruit volgt dat een  $a = 1$  of  $a = 2$  de beste resultaten oplevert als respectievelijk de MSE- of SSI-waarden doorslaggevend zijn. Een  $a$ -waarde van nul doet het, zoals verwacht, minder goed aangezien scherpe randen en details verloren zijn gegaan. Opvallend is hoe een hogere  $a$ -waarden slechtere resultaten oplevert en hiervoor bovendien meer rekentijd vereist. Dit laatste is te wijten aan het feit dat een hogere  $a$ -waarde meer iteraties vereist alvorens een extreme waarde gevonden wordt.

**Tabel 6.2:** De beste MSE- en SSI-waarden bekomen met veranderlijke  $a$ -waarden

Variable	Best mse	iteration	time
a=1.0	189.5256	3	7.051
a=2.0	197.0126	5	8.899
a=0.0	205.2628	3	7.201
a=3.0	240.6355	8	15.007
a=4.0	281.651	10	20.476
a=5.0	311.6055	11	22.169

Variable	Best ssi	iteration	time
a=2.0	0.7006	7	12.33
a=1.0	0.6891	5	11.329
a=0.0	0.6695	5	11.597
a=3.0	0.6655	10	18.861
a=4.0	0.6187	11	22.903
a=5.0	0.5842	12	23.989

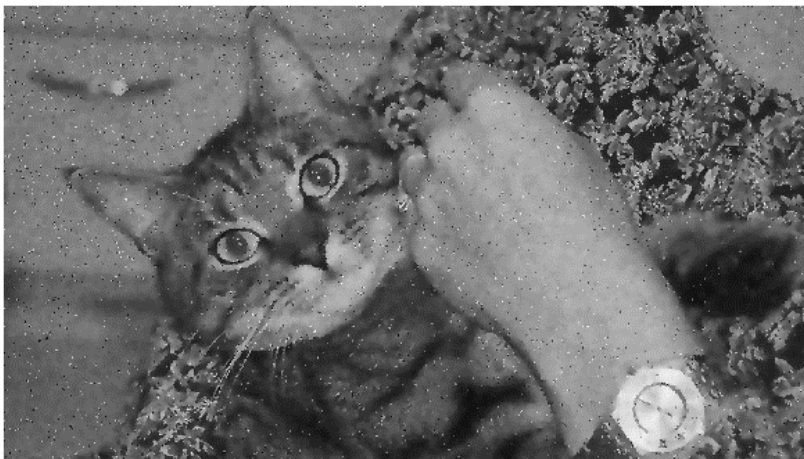
Een visuele vergelijking tussen de beste resultaten bekomen met  $a$ -waarden 1 en 5 zijn respectievelijk terug te vinden in Figuur 6.3 en Figuur 6.4. Hierop is duidelijk te zien hoe de resultaten bekomen via een hogere  $a$ -waarde resulteren in een afbeelding waarin duidelijke restanten van ruis overblijven. Dit gedrag kan wiskundig verklaard worden door de gebruikte diffusie-functie, voorgesteld door optie 0 (3.8), correct te interpreteren. In onderdeel 3.1 werd vermeld dat deze functie een pixel vervaagt of verscherpt naargelang de bekomen functiewaarde respectievelijk nadert naar 1 of 0. In Figuur 6.5 wordt het verloop van deze functie voor een aantal verschillende  $a$ -waarden uitgezet. Hierop is te zien hoe hogere  $a$ -waarden resulteren in steilere grafieken die sneller waarden dicht bij 0 of 1 opleveren wat op zijn beurt resulteert in een zeer plotse overgang tussen vervagen en verscherpen. Dit effect is duidelijk merkbaar in Figuur 6.4 waarbij enkele beschadigde pixels een te grote gradiënt bezitten en bijgevolg foutief verscherpt worden. Indien een  $a$ -waarde van 0 gekozen wordt levert de diffusie-functie ten alle tijde een relatief hoge constante waarde waardoor de afbeelding globaal vervaagd wordt. Omdat vermoed

wordt dat de optimale  $a$ -waarde vooral afhankelijk is van de gebruikte afbeelding en diffusie-functie zal in de loop van onderstaande experimenten ook deze parameter vaak opnieuw getest worden.



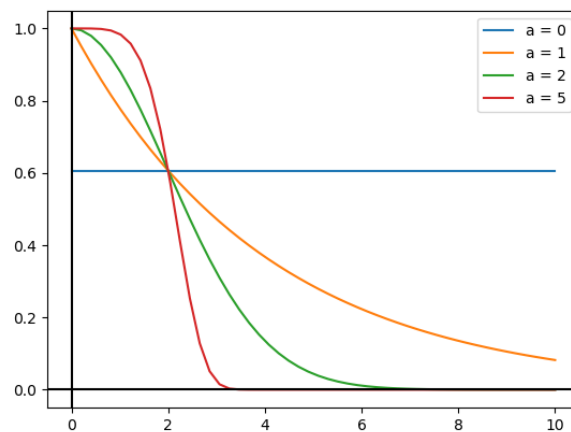
mse: 189.5256 ssim: 0.6483

**Figuur 6.3:** Eindresultaat flor.jpg met  $a=1$



mse: 311.6055 ssim: 0.584

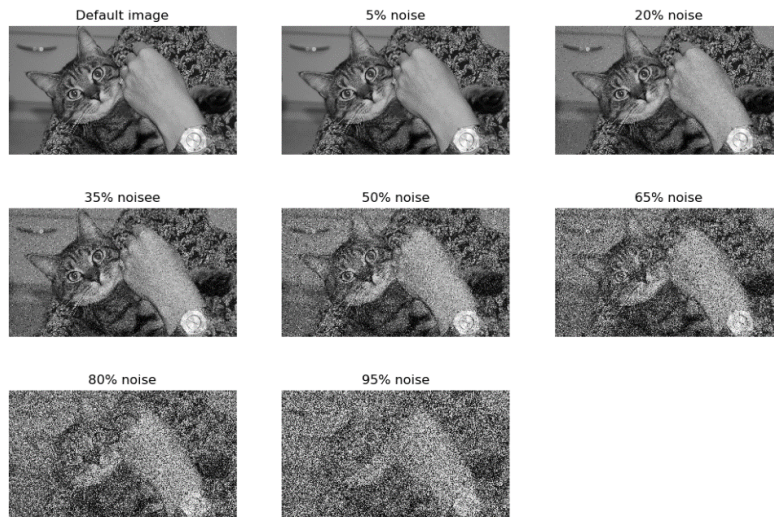
**Figuur 6.4:** Eindresultaat flor.jpg met  $a=5$



**Figuur 6.5:** Het verloop van diffusie-functie (3.8) met veranderlijke  $a$ -waarden

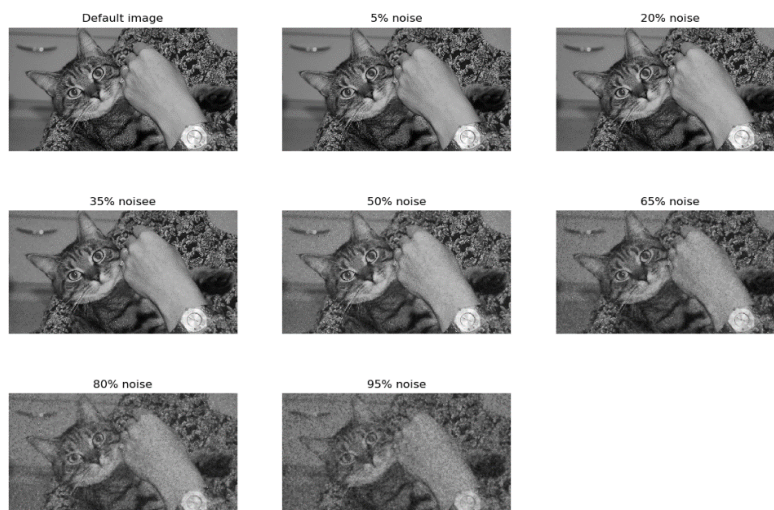
### 6.2.2 Wat is de invloed van de hoeveelheid Gaussische ruis in de initiële afbeelding op het eindresultaat?

Om een antwoord te formuleren op deze onderzoeksvraag worden aan de standaardafbeelding verschillende hoeveelheden ruis incrementeel toegevoegd. In totaal worden zeven afbeeldingen getest met 5 tot 95 procent ruis, weergegeven in Figuur 6.6. In elke opeenvolgende afbeelding wordt de hoeveelheid ruis en de sterkte ervan verhoogd met een constante factor van 15 procent.



**Figuur 6.6:** Verschillende hoeveelheden initiële ruis

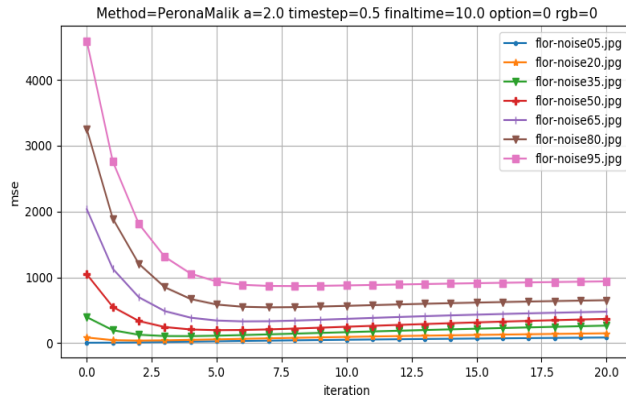
Om het effect van de hoeveelheid ruis na te gaan op het eindresultaat wordt nu op elk van bovenstaande afbeelding de geïmplementeerde Perona-Malik vergelijking toegepast. In Figuur 6.7 worden de eindresultaten met de beste (=minimale) MSE-waarden afgebeeld.



**Figuur 6.7:** Eindresultaten bekomen met verschillende hoeveelheden initiële ruis

De conclusies die genomen kunnen worden bij het bestuderen van de SSI-waarden zijn in dit experiment volledig analoog als deze bij het bestuderen van de MSE-waarden en worden bijgevolg niet afzonderlijk besproken.

In Figuur 6.8 bevindt zich de MSE-*assessment graph*. Deze zal voor elke afbeelding het verloop van de MSE-waarde in functie van de iteratie afbeelden. De interessantste resultaten, aangevuld met de iteratie en tijd waarin ze bekomen werden, zijn terug te vinden in Tabel 6.3.



**Figuur 6.8:** De MSE-*assessment graph* met veranderlijke hoeveelheid initiële ruis

**Tabel 6.3:** De beste MSE-waarden met veranderlijke hoeveelheid initiële ruis

Variable	Best mse	iteration	time
flor-noise05.jpg	5.8609	0	0
flor-noise20.jpg	38.3855	2	4.457
flor-noise35.jpg	106.1332	4	6.911
flor-noise50.jpg	197.0126	5	8.899
flor-noise65.jpg	330.4008	6	11.323
flor-noise80.jpg	543.4777	7	12.465
flor-noise95.jpg	866.8433	8	15.793

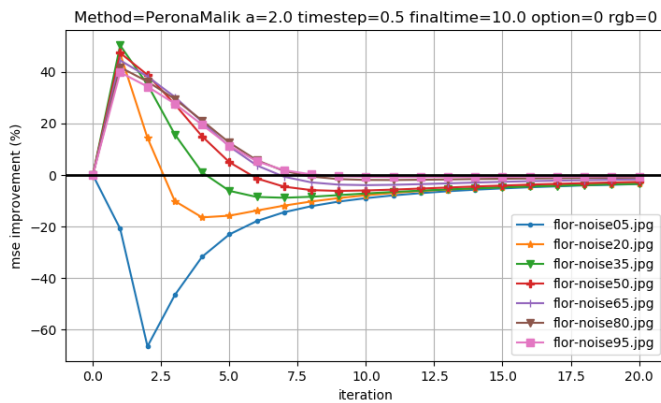
Uit de grafiek kan geconstateerd worden dat afbeeldingen met een hoge initiële ruis, een hogere initiële MSE-waarde hebben dan de afbeeldingen met een lagere initiële ruis. Dit is geen verrassing aangezien een hogere MSE-waarde duidt op een grotere afwijking ten opzichte van de onbeschadigde afbeelding.

Deze vaststelling kan vervolgens uitgebreid worden naar alle iteraties. Op de grafiek is namelijk te zien hoe op elk moment de MSE-waarde van een willekeurige afbeelding steeds hoger zal zijn dan de MSE-waarde van een initieel minder beschadigde afbeelding.

Hieruit volgt dat het eindresultaat van een willekeurige afbeelding nooit beter kan zijn dan het eindresultaat van dezelfde afbeelding met een initieel lagere hoeveelheid ruis op voorwaarde dat dezelfde parameter-waarden gebruikt worden.

Indien de resultaten uit Tabel 6.3 in acht genomen worden kan opgemerkt worden hoe de afbeelding met 5% ruis een minimale MSE-waarde bereikt in iteratie nul. Dit betekent dat er na de eerste iteratie reeds een verhoging is in het aantal afwijkingen ten opzichte van de ideale afbeelding. Bijgevolg kan een afbeelding met een lage hoeveelheid initiële ruis, afgaand op de beoordelingsmethoden, niet altijd verbeterd worden. Verder blijkt de hoeveelheid iteraties alvorens de minimale MSE-waarde gevonden wordt te stijgen naargelang de hoeveelheid ruis.

In Figuur 6.9 wordt de MSE *improvement graph* afgebeeld. Deze bevestigt opnieuw dat de afbeelding met 5% ruis reeds vanaf de eerste iteratie ongeveer 20% is verslechterd ten opzichte van voorgaande iteratie. In Tabel 6.4 bevinden zich de totale MSE-verbeteringen ten opzichte van de begintoestand (iteratie 0). Hieruit wordt geconcludeerd dat, met deze parameters, de grootste verbeteringen zich voordoen bij een ruis van 50% of meer. Houd rekening dat dit niet betekent dat het beste resultaat van 65% ruis er beter uitziet dan deze van 35% ruis.

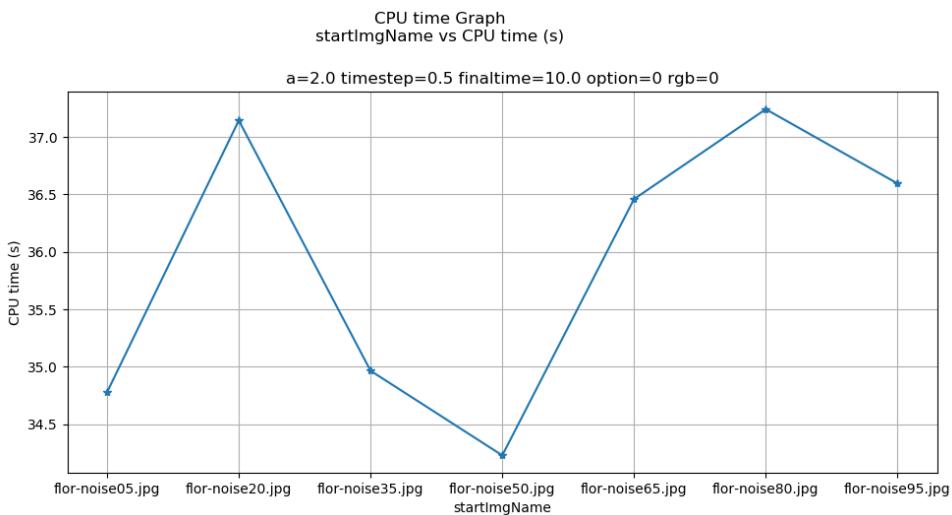


**Figuur 6.9:** De MSE *improvement graph* met veranderlijke hoeveelheid initiële ruis

**Tabel 6.4:** De totale MSE-verbetering met veranderlijke hoeveelheid initiële ruis

Variable	Total mse improvement
flor-noise65.jpg	+83.76%
flor-noise80.jpg	+83.28%
flor-noise50.jpg	+81.15%
flor-noise95.jpg	+81.11%
flor-noise35.jpg	+73.12%
flor-noise20.jpg	+54.77%
flor-noise05.jpg	+0.0%

Vervolgens wordt onderzocht of de initiële hoeveelheid ruis een invloed heeft op de vereiste CPU-tijd. Deze tijden zijn als grafiek weergegeven in Figuur 6.10. De vorm van deze curve is schijnbaar willekeurig aangezien zowel hoge als lage initiële ruis een ongeveer gelijke CPU-tijd vertonen. De schommeling bedraagt bovendien hooguit drie seconden wat perfect te wijten is aan externe factoren zoals buffer interferentie veroorzaakt door andere taken. Er kan dus besloten worden dat de CPU-tijd niet beïnvloed wordt door de initiële hoeveelheid ruis.



**Figuur 6.10:** De CPU-tijden (in seconden) met veranderlijke hoeveelheid initiële ruis

Tot slot wordt onderzocht of een grotere/kleinere  $a$ -waarde een ander effect heeft naargelang de hoeveelheid ruis in de initiële afbeelding. Om dit te onderzoeken worden de afbeeldingen met 20, 35, 65 en 80 procent Gaussische ruis opnieuw verbeterd, nu met een  $a$ -waarde van 1 en 3. Vervolgens worden de totale MSE-verbeteringen vergeleken met de hierboven bekomen resultaten. De bekomen numerieke waarden zijn weergegeven in Tabel 6.5.

**Tabel 6.5:** De invloed van de parameter  $a$  bij verschillende hoeveelheden initiële ruis

20% ruis		35% ruis	
Variable	Total mse improvement	Variable	Total mse improvement
a=2.0	+54.77%	a=1.0	+73.99%
a=1.0	+53.22%	a=2.0	+73.12%
a=3.0	+51.29%	a=3.0	+68.71%

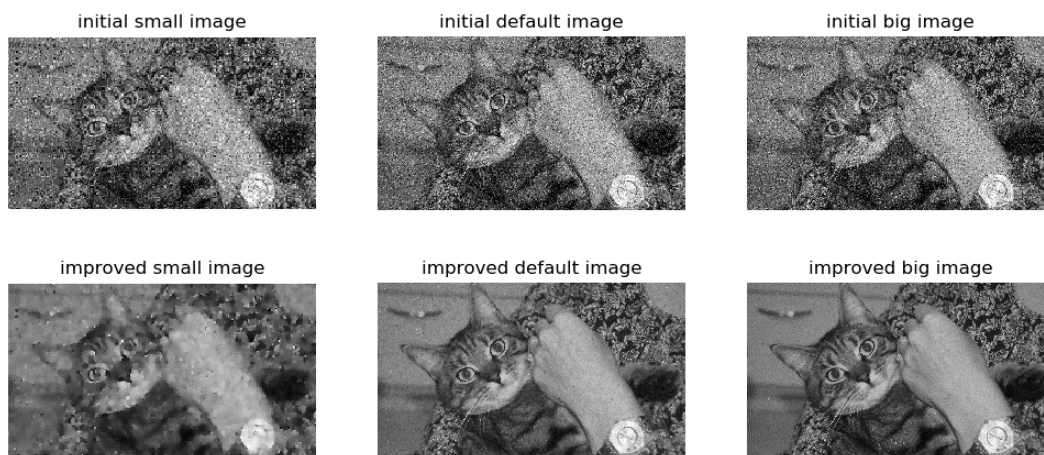
65% ruis		80% ruis	
Variable	Total mse improvement	Variable	Total mse improvement
a=1.0	+84.14%	a=1.0	+83.39%
a=2.0	+83.76%	a=2.0	+83.28%
a=3.0	+80.94%	a=3.0	+81.66%

Uit deze resultaten blijkt dat  $a=1$  meestal de beste MSE-resultaten op te leveren ongeacht de initiële hoeveelheid ruis. Indien vooral rekening gehouden wordt met SSI-waarden blijkt  $a=2$  telkens de beste resultaten op te leveren. De hoeveelheid ruis die een beschadigde afbeelding bezit is met andere woorden geen bepalende factor in de keuze voor de best mogelijke  $a$ -waarde.

### 6.2.3 Wat is de invloed van de afmeting van de initiële afbeelding op het bekomen eindresultaat?

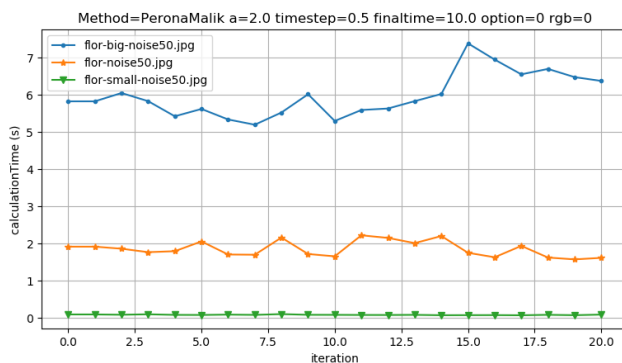
Om een antwoord te formuleren op deze onderzoeksvraag worden een grote en kleine variant van de standaard afbeelding ingevoerd. In volgorde van stijgende grootte wordt gewerkt met: ‘flor-small.jpg’ van 230x129 px, ‘flor.jpg’ van 921x518 px en ‘flor-big’ van 1612x907 px. Merk op hoe de kleine en grote afbeelding respectievelijk 4 keer kleiner en 1.75 keer groter zijn dan de middelmatige afbeelding.

Ook aan deze afbeeldingen wordt 50% Gaussische ruis toegevoegd. De initiële afbeeldingen en de beste eindresultaten volgens MSE-waarden zijn terug te vinden in Figuur 6.11.



**Figuur 6.11:** De beschadigde en verbeterde afbeelding voor verschillende afmetingen

Een eerste vaststelling die reeds tijdens de berekening duidelijk werd is dat zoals verwacht de kleine afbeeldingen veel sneller verwerkt kunnen worden dan de grotere. De grafiek in Figuur 6.12 geeft de *calculation time* voor elke iteratie weer en bevestigt deze observatie. In de Tabel 6.6 worden de totale CPU-tijden van 20 iteraties voor elke grootte opgesomd.



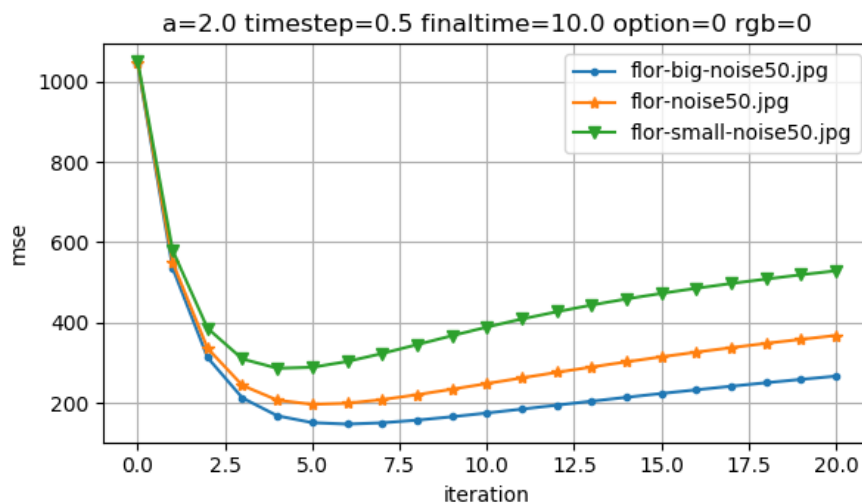
**Figuur 6.12:** De rekentijd per iteratie (in seconden) voor verschillende afmetingen

**Tabel 6.6:** De totale CPU-tijden (in seconden) voor verschillende afmetingen

Variable	CPU time
flor-big-noise50.jpg	121.3226
flor-noise50.jpg	37.521
flor-small-noise50.jpg	2.6118

Aangezien een afbeelding bestaat uit twee dimensies (breedte en hoogte) zal een verdubbeling in grootte een verviervoudiging in het totaal aantal pixels teweegbrengen. Rekening houdend hiermee kan nagegaan worden of er een kwadratisch verband bestaat tussen de grootte van de afbeelding en de benodigde CPU-tijd. De kleinste afbeelding is respectievelijk vier en zeven keer kleiner dan de middelmatige en grote afbeelding. Bijgevolg bevat de kleinste afbeelding zestien en negenenveertig keer minder pixels waardoor verwacht wordt dat diens vereiste CPU-tijd ook zestien en negenenveertig keer lager zou moeten zijn dan die van de middelmatige of grote afbeelding. Vertrekkend van de in Tabel 6.6 weergegeven 2.62 seconden voor de kleinste afbeelding worden CPU-tijden van 41.92 seconden en 128.38 seconden verwacht voor de middelmatige en grote afbeelding. In praktijk wijken de gemeten CPU-tijden enkele seconden af van de theoretisch berekende waarden. Deze liggen echter nog steeds dicht genoeg om een kwadratisch verband tussen CPU-tijd en afbeeldingsgrootte te veronderstellen.

In Figuur 6.13 wordt voor alle afmetingen de MSE-*assessment graph* weergegeven. In Tabel 6.7 bevinden zich de bijhorende MSE-minima. Op deze grafiek is te zien hoe de initiële MSE-waarden zeer dicht bij elkaar liggen (<10 verschil) maar reeds na enkele iteraties beginnen divergeren. De grootste afbeelding bereikt de laagste minimale MSE-waarde waardoor deze bijgevolg een beter eindresultaat oplevert mits een langere rekentijd. Verder blijkt dat de iteratie waarin een extreme waarde gevonden wordt opschuift naargelang de grootte van de afbeelding.



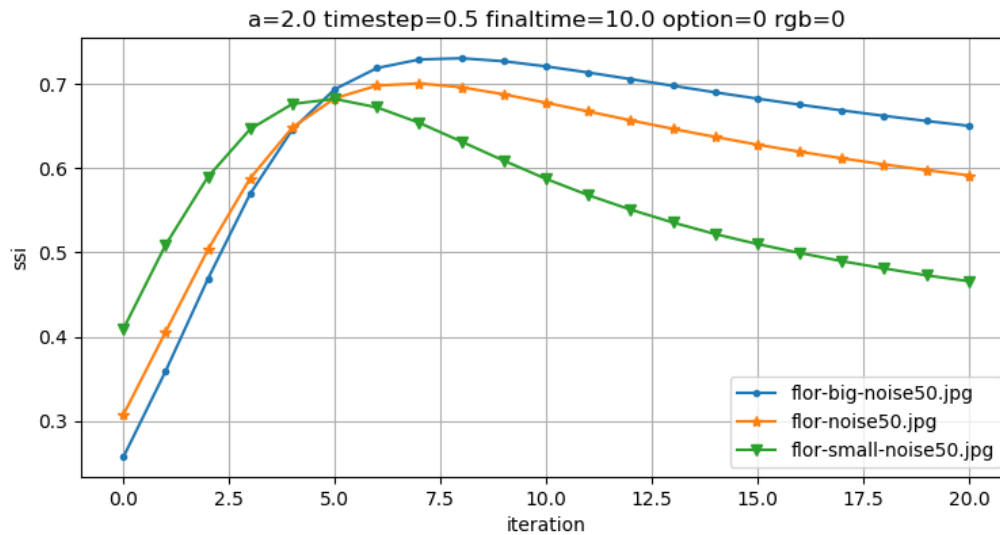
**Figuur 6.13:** De MSE-*assessment graph* voor verschillende afmetingen

**Tabel 6.7:** De beste MSE-waarden voor verschillende afmetingen

Variable	Best mse	iteration	time
flor-big-noise50.jpg	147.8053	6	34.558
flor-noise50.jpg	197.0126	5	9.255
flor-small-noise50.jpg	286.6891	4	0.372



Uit de SSI-*assessment graph* in Figuur 6.14 volgen gelijkaardige conclusies met enig verschil de ongelijke SSI-waarden in de begintoestand (iteratie nul). De SSI-waarde van een kleine afbeelding blijkt initieel beter maar zal minder goed en minder lang verbeterd kunnen worden waardoor de grote afbeelding opnieuw het beste eindresultaat oplevert.



**Figuur 6.14:** De SSI-*assessment graph* voor verschillende afmetingen

De reden waarom kleinere afbeeldingen slechtere resultaten opleveren kan mogelijks verklaard worden door het feit dat een één beschadigde pixel een relatief groot effect heeft en bijgevolg visueel beter opvalt indien de afbeelding zelf opgebouwd is uit een lage hoeveelheid pixels.

Opnieuw kan de vraag gesteld worden of de ideale  $a$ -waarde al dan niet beïnvloed wordt door de grote van de afbeelding. De totale MSE-verbeteringen voor de kleine, gemiddelde en grote afbeelding met variërende  $a$ -waarden worden met elkaar vergeleken in Tabel 6.8.

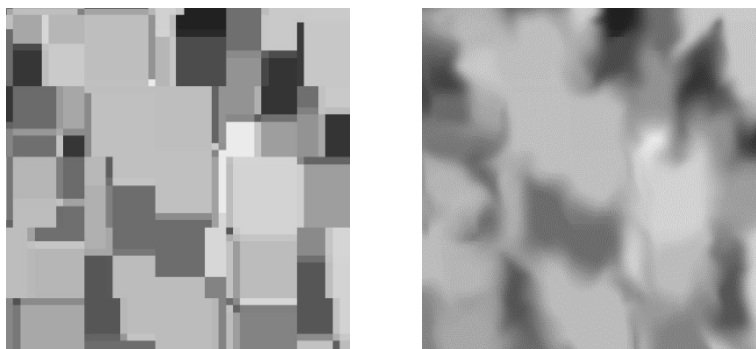
**Tabel 6.8:** De invloed van de parameter  $a$  bij verschillende afmetingen

Small		Medium		Large	
Variable	Total mse improvement	Variable	Total mse improvement	Variable	Total mse improvement
$a=1.0$	+74.44%	$a=1.0$	+81.87%	$a=1.0$	+86.38%
$a=2.0$	+72.69%	$a=2.0$	+81.15%	$a=2.0$	+85.81%
$a=3.0$	+67.52%	$a=3.0$	+76.98%	$a=3.0$	+81.97%

Opnieuw blijkt  $a=1$ , voor alle groottes, het beste resultaat op te leveren. Indien rekening gehouden wordt met SSI-waarden zal ook  $a=2$  opnieuw de beste resultaten opleveren ongeacht de grootte van de afbeelding. Dit wijst erop dat ook de grootte van de afbeelding geen bepalende factor is in de keuze voor beste  $a$ -waarde.

#### 6.2.4 Wat is de invloed van een afbeelding met scherpe of vloeiende randen op het bekomen eindresultaat?

Tot nu toe werd steeds gebruik gemaakt van dezelfde afbeelding (mits veranderlijke hoeveelheid ruis of grootte). In dit onderdeel wordt onderzocht hoe de scherpte van de randen in een afbeelding een invloed heeft op het eindresultaat. Concreet zullen hiervoor twee nieuwe afbeeldingen gebruikt worden, weergegeven in Figuur 6.15: 'borders.jpg' en 'noborders.jpg'. Zoals de naam al doet vermoeden zal de eerste afbeelding opgebouwd zijn uit een verzameling rechthoeken met duidelijk onderscheidbare randen en afgebakende gebieden. In de tweede afbeelding daarentegen zijn de randen uitgesmeerd waardoor de verschillende oppervlakken geleidelijk in elkaar overlopen.



**Figuur 6.15:** borders.jpg (links) en noborders.jpg (rechts)

Aan beide afbeeldingen wordt eerst 50% ruis toegevoegd. Hierna worden enkele oplossingen gegenereerd waarbij de  $\alpha$ -waarde varieert van 0 tot 4. De minimale MSE- en SSI-waarden die met elke permutatie bekomen kunnen worden zijn gesorteerd weergegeven in Tabel 6.9. Hieruit kunnen enkele interessante conclusies genomen worden.

**Tabel 6.9:** De beste MSE- en SSI- waarden voor scherpe en vloeiende randen met veranderlijke a-waarden  
**borders**

Variable	Best mse	iteration	time
a=2.0	78.5894	8	8.108
a=1.0	90.8441	5	6.808
a=3.0	96.5976	13	13.436
a=0.0	108.713	5	4.454
a=4.0	125.1596	17	18.321

Variable	Best ssi	iteration	time
a=2.0	0.8722	13	12.701
a=3.0	0.8515	20	20.106
a=1.0	0.8507	12	14.066
a=0.0	0.8335	16	13.298
a=4.0	0.7882	20	21.079

**noborders**

Variable	Best mse	iteration	time
a=1.0	24.2006	19	16.371
a=2.0	24.4291	19	19.053
a=0.0	24.6235	20	16.38
a=3.0	38.503	20	21.15
a=4.0	74.3348	20	19.778

Variable	Best ssi	iteration	time
a=1.0	0.9762	20	17.268
a=2.0	0.9761	20	19.874
a=0.0	0.9663	20	16.38
a=3.0	0.9364	20	21.15
a=4.0	0.8409	20	19.778

Ten eerste valt op hoe de optimale MSE- en SSI-waarden bij de afbeelding met vloeiende randen pas in de (voor)laatste iteratie gevonden worden waardoor de effectieve extreme waarden waarschijnlijk nog beter zijn, niettemin zal een latere eindtijd in dit geval amper een visueel verschil opleveren aangezien de verbeteringen reeds verwaarloosbaar klein geworden zijn. Wel duidt dit op het feit dat afbeeldingen met vloeiende randen langer verbeterd kunnen worden (lees: een extrema bereiken in een latere iteratie) dan deze met scherpe randen.

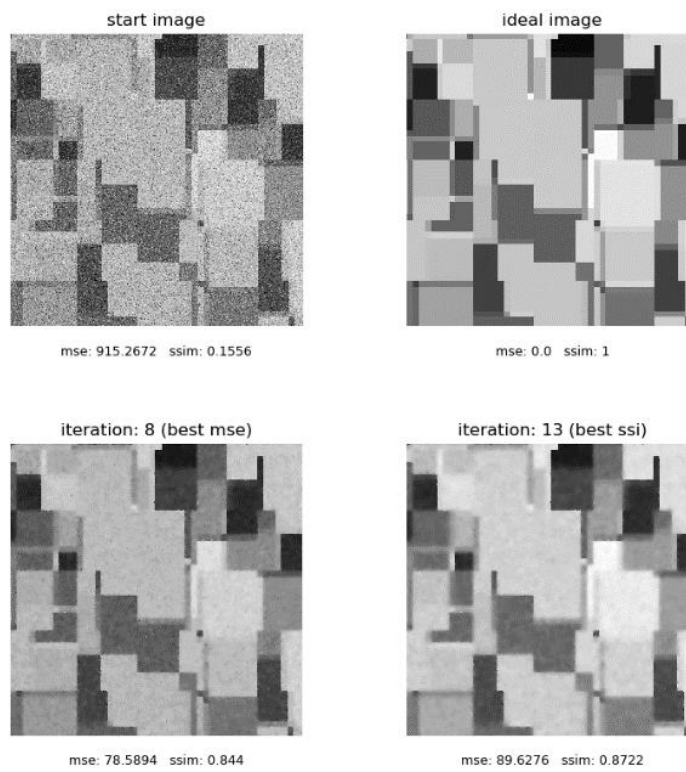
Verder kan opgemerkt worden hoe de minimale MSE-waarde zelf veel lager ligt bij afbeeldingen met vloeiende randen. Het is echter incorrect om de MSE-waarden tussen beide type afbeeldingen rechtstreeks met elkaar te vergelijken aangezien de initiële MSE-waarden van beide afbeeldingen reeds verschillen. Daarom wordt in Tabel 6.10 de totale MSE-verbetering voor elke permutatie ten opzichte van diens initiële afbeelding weergegeven. Hieruit volgt dat voor alle geteste a-waarden de afbeelding met vloeiende randen meer verbeterd wordt dan die met scherpe randen. Een potentiële verklaring kan gevonden worden in het feit dat een beschadigde afbeelding met veel scherpe randen een grotere kans heeft om foutief scherpe randen te vervagen dan een van nature wazige afbeelding.

**Tabel 6.10:** De totale MSE-verbetering voor scherpe en vloeiende randen

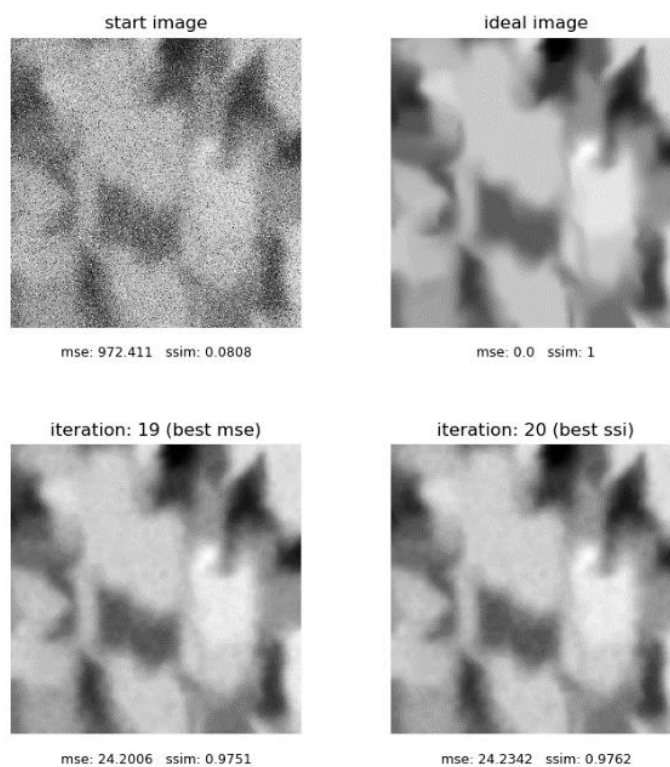
Variable	Total mse improvement
noborders-noise50.jpg a=1.0	+97.51%
noborders-noise50.jpg a=2.0	+97.49%
noborders-noise50.jpg a=0.0	+97.47%
noborders-noise50.jpg a=3.0	+96.04%
noborders-noise50.jpg a=4.0	+92.36%
borders-noise50.jpg a=2.0	+91.41%
borders-noise50.jpg a=1.0	+90.07%
borders-noise50.jpg a=3.0	+89.45%
borders-noise50.jpg a=0.0	+88.12%
borders-noise50.jpg a=4.0	+86.33%

In een laatste vaststelling wordt geconstateerd dat beide afbeeldingen een verschillende optimale  $a$ -waarde verkiezen. De afbeelding met vloeiende randen opteert een  $a$ -waarde van één hoewel een waarde van nul of twee geen merkbaar verschil zouden opleveren. De afbeelding met scherpe randen daarentegen verkiest resoluut een  $a$ -waarde gelijk aan twee. Bovendien levert een waarde van drie hier betere resultaten dan een waarde van nul wat bij vloeiende randen niet het geval is. Hieruit volgt dat de structuur van de initiële afbeelding mee bepaalt welke  $a$ -waarde de beste resultaten oplevert. Afbeeldingen met duidelijk afgebakende randen lijken een iets hogere  $a$ -waarden te verkiezen ten opzichte van afbeeldingen met vloeiende randen. Een waarde groter dan twee blijft echter slechte MSE-waarden opleveren. De beste MSE- en SSI-eindresultaten zijn voor beide afbeeldingen terug te vinden in Figuur 6.16 en Figuur 6.17.

De benodigde CPU-tijd om elke permutatie te berekenen is voor beide afbeeldingen nagenoeg gelijk en wordt dus niet beïnvloed door de structuur van de initiële afbeelding.



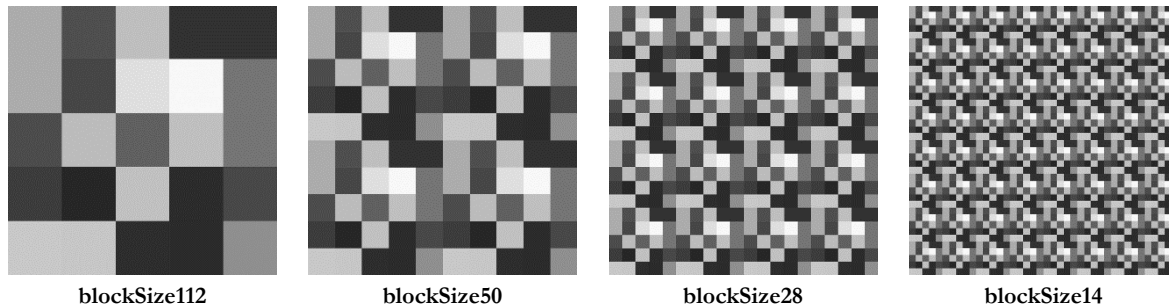
**Figuur 6.16:** Eindresultaten borders.jpg met  $\alpha=2$



**Figuur 6.17:** Eindresultaten noborders.jpg met  $\alpha=1$

### 6.2.5 Wat is de invloed van egale oppervlakken op het eindresultaat?

Om een antwoord te bieden op deze onderzoeksvraag worden vier nieuwe afbeeldingen, weergegeven in Figuur 6.18, gebruikt. Deze afbeeldingen zijn telkens 565x565 pixels groot en bestaan uit aaneengesloten rechthoeken die telkens 50% verkleind worden ten opzichte van hun voorganger. Merk op hoe erg kleine oppervlakken een willekeurige afbeelding benaderen waar elk oppervlak in essentie 1px groot is.



**Figuur 6.18:** De vier afbeeldingen met veranderlijke blokgroottes

Aan deze afbeeldingen wordt eerst 50% ruis toegevoegd waarna ze verbeterd worden via de standaard Perona-Malik vergelijking. In Tabel 6.11 bevinden zich de beste MSE- en SSI-waarden van de bekomen oplossingen. Hieruit kan afgeleid worden dat de iteratie waarin een extrema gevonden wordt stijgt volgens incremenderende blokgruotte. De resultaten in Tabel 6.12 tonen aan hoe de SSI-waarden van afbeeldingen met grote blokgroottes het meest verbeterd kunnen worden. Hetzelfde geldt voor de totale MSE-verbetering.

**Tabel 6.11:** De beste MSE- en SSI-waarden voor veranderlijke blokgroottes

Variable	Best mse	iteration	time	Variable	Best ssi	iteration	time
blockSize112-noise50.jpg	109.1357	10	13.005	blockSize112-noise50.jpg	0.9153	20	25.153
blockSize56-noise50.jpg	146.6897	8	11.755	blockSize56-noise50.jpg	0.8591	14	19.35
blockSize28-noise50.jpg	205.0521	6	7.99	blockSize28-noise50.jpg	0.796	10	12.886
blockSize14-noise50.jpg	317.0962	5	6.772	blockSize14-noise50.jpg	0.7463	6	8.028

**Tabel 6.12:** De totale SSI-verbetering voor veranderlijke blokgroottes

Variable	Total ssi improvement
blockSize112-noise50.jpg	+961.12%
blockSize14-noise50.jpg	+92.65%
blockSize56-noise50.jpg	+542.29%
blockSize28-noise50.jpg	+253.78%

Opnieuw wordt een eventueel verband gezocht tussen de gebruikte blok grootte en de parameter 'a'. Bovenstaande berekeningen worden hiervoor enkele malen herhaald met een a-waarde variërend van nul tot vier. Een beperkte selectie van de resultaten bekomen met een blok grootte van 28 en 112 pixels breed is weergegeven in Tabel 6.13. Uit alle bekomen resultaten volgt dat de beste MSE-waarden bereikt worden bij een a-waarde van twee ongeacht de gebruikte blok grootte. De beste SSI-waarde wordt voor bijna elke blok grootte bekomen indien  $a=3$ . Enkel de grootst geteste blok grootte van 112 pixels wijkt hiervan af en bekomt een optimale SSI-waarde voor een a-waarde gelijk aan twee. Wel moet opgemerkt worden dat indien toch twee gebruikt zou zijn beide eindresultaten visueel ononderscheidbaar zijn. De conclusie die hieruit genomen kan worden is dat de blok grootte weinig tot geen invloed heeft op de ideale a-waarde ongeacht een MSE of SSI extrema gezocht wordt. Wel blijkt dat voor sommige afbeeldingen de ideale a-waarde kan verschillen naargelang een MSE of SSI extrema gezocht wordt. Tot slot wordt vermeld dat a-waarden van drie voor deze afbeeldingen acceptabele resultaten opleveren.

**Tabel 6.13:** De invloed van de parameter a bij verschillende blok groottes

**Blok grootte = 28 px**

Variable	Best mse	iteration	time
a=2.0	205.0521	6	7.99
a=3.0	222.5858	10	12.714
a=4.0	253.3831	13	19.068
a=1.0	257.2559	3	3.761
a=0.0	316.6678	3	3.418

Variable	Best ssi	iteration	time
a=3.0	0.8045	14	18.043
a=2.0	0.796	10	12.886
a=4.0	0.781	18	25.495
a=1.0	0.7387	8	10.228
a=0.0	0.6965	10	12.216

**Blok grootte = 112 px**

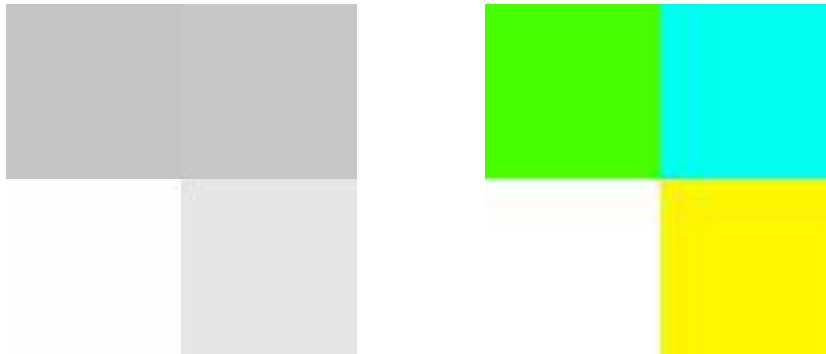
Variable	Best mse	iteration	time
a=2.0	109.1357	10	13.005
a=3.0	119.4746	19	24.901
a=1.0	134.7894	6	7.77
a=4.0	153.0564	20	25.033
a=0.0	158.9833	6	7.663

Variable	Best ssi	iteration	time
a=2.0	0.9153	20	25.153
a=3.0	0.9064	20	26.027
a=1.0	0.8924	19	22.011
a=0.0	0.8752	20	21.978
a=4.0	0.8448	20	25.033

### 6.2.6 Wat is de invloed van kleur of grijstinten op het eindresultaat?

In voorgaande experimenten werden afbeeldingen steeds verwerkt in grijstinten. Aangezien in de praktijk ook gewerkt wordt met kleuren afbeeldingen is het niet onbelangrijk na te gaan of deze zich anders gedragen dan hun zwart-wit tegenhanger.

Voor dit experiment werd een simpele afbeelding opgesteld bestaande uit 4 kleuren: groen, cyaan, geel en wit. De grijstint en kleuren variant van deze afbeelding zijn terug te vinden in Figuur 6.19.



**Figuur 6.19:** colors.jpg in grijstinten (links) en kleur (rechts)

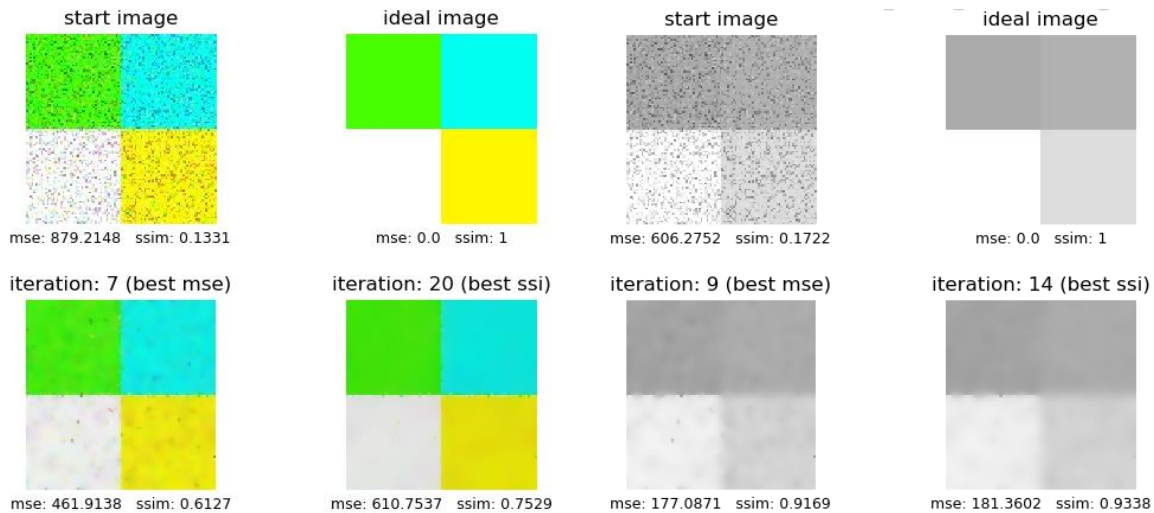
Alvorens de resultaten van het experiment te bespreken wordt eerst aangekaart hoe de MSE en SSI-berekeningen verschillen wanneer de afbeelding in kleur verwerkt wordt. Zoals reeds vermeld is een RGB-afbeelding opgebouwd uit drie verschillende lagen: een rode, groene en blauwe. Van elke laag wordt de MSE- en SSI-waarde berekend. Hieruit wordt vervolgens het wiskundig gemiddelde berekend wat gebruikt zal worden als de uiteindelijke MSE- of SSI-waarde van de afbeelding.

Aan beide afbeeldingen wordt 50% Gaussische ruis toegevoegd. Deze worden vervolgens verbeterd via de Perona-Malik vergelijking waarin de  $a$ -waarde opnieuw varieert van nul tot vier. De beste MSE-waarden per permutatie zijn terug te vinden in Tabel 6.14. Aangezien de beste MSE-waarden van de zwart-wit afbeeldingen een stuk beter zijn dan die van de kleurenafbeelding wordt verondersteld dat deze het beste eindresultaat opleveren. Wanneer deze echter visueel vergeleken worden blijkt deze veronderstelling niet te kloppen. Een voorbeeld van dergelijke eindoplossingen zijn weergegeven in Figuur 6.20. Voor beiden wordt een  $a$ -waarde van 2 gebruikt.

**Tabel 6.14:** De beste MSE-waarden van colors.jpg in kleur en zwart-wit met veranderlijke  $a$ -waarden

Zwart-wit				RGB			
Variable	Best mse	iteration	time	Variable	Best mse	iteration	time
a=2.0	177.0871	9	0.156	a=1.0	448.6551	5	0.215
a=1.0	183.0809	6	0.099	a=2.0	461.9138	7	0.301
a=3.0	183.3368	15	0.272	a=3.0	499.049	7	0.354
a=0.0	188.6545	6	0.077	a=4.0	523.5251	7	0.346
a=4.0	199.3211	20	0.346	a=0.0	555.4579	2	0.088





**Figuur 6.20:** Eindresultaten colors.jpg in kleur (links) en zwart-wit (rechts)

Deze anomalie wijst erop dat de MSE- en SSI-waarden van een zwart-wit en kleurenafbeelding niet zomaar vergeleken kunnen worden. Een eventuele verklaring kan gevonden worden in het feit dat een kleurenafbeelding opgebouwd is uit drie verschillende lagen waardoor een gunstige wijziging op de ene, een negatief effect kan hebben op de anderen waardoor de globale MSE- of SSI-beoordelingen minder snel positief evolueren. Bijgevolg zal een kleine positieve bijdrage in een kleurenafbeelding in realiteit een groter visueel effect opleveren dan eenzelfde bijdrage in zwart-wit.

Uit een tweede visuele waarneming volgt dat het resultaat bekomen volgens optimale SSI-waarde beter oogt dan deze volgens beste MSE-waarde. Dit verschil is vooral waarneembaar indien de afbeelding in kleur verwerkt wordt. Een mogelijke verklaring kan gevonden worden in de reeds vermelde tekortkoming van het MSE-algoritme omtrent helderheid en kleur. Door het uitsmeren van ruis zullen pixels een verdonkerde kleur voorstellen. Dit fenomeen is extra duidelijk indien de pixel oorspronkelijk reeds een lichte kleur aannam. Hierdoor zullen bijvoorbeeld initieel witte oppervlakken een grijze kleur aannemen waardoor het MSE-algoritme deze pixels als afwijkend gaat beschouwen. Indien de initiële afbeelding bijgevolg opgebouwd is uit grote, licht gekleurde oppervlakken zal de SSI-waarde een betere indicator zijn.

Een laatste verklaring waarom de gebruikte kleurenafbeelding de beste visuele resultaten oplevert zit hem in het feit dat de grijswaarden van groen en cyaan (respectievelijk 170 en 177) weinig verschillen. Hierdoor is de overgang tussen beide tinten minder duidelijk en worden ze door het algoritme veeleer als eenzelfde oppervlak beschouwd. Dit resulteert op zijn beurt in uitgesmeerde randen wat leidt tot een veeleer wazig en onaantrekkelijk eindresultaat.

Vervolgens worden de gegevens uit Tabel 6.14 grondiger bestudeert. Hieruit blijkt onder meer dat per  $a$ -waarde, de iteratie waarin een minimale MSE-waarde gevonden wordt het laagst is indien de afbeelding in kleur verwerkt wordt. Deze vaststelling blijkt echter niet algemeen geldig. Om dit na te gaan werd ook een totaal verschillende afbeelding: `flor.jpg` (Figuur 6.1) zowel in kleur als zwart-wit verbeterd. Hieruit bleek dat de beste MSE-waarden in zwart-wit en kleur respectievelijk gevonden worden in iteraties vijf en zes. Houdt opnieuw rekening dat een latere iteratie niet altijd overeenkomt met een langere rekestijd. Zo blijkt uit Tabel 6.15 hoe de MSE-extrema voor zwart-wit permutaties in latere iteraties gevonden worden t.o.v. de equivalente RGB-permutatie, niettemin blijkt de rekestijd waarin een zwart-wit afbeelding een extrema vindt korter te zijn.

Wanneer de totale CPU-tijden, weergegeven in Tabel 6.16, voor het berekenen van 20 iteraties in RGB en zwart-wit vergeleken worden blijken deze voor kleuren afbeeldingen ongeveer drie keer langer te zijn dan die van de overeenkomstige zwart-wit afbeelding. Dit voldoet aan de verwachtingen aangezien een kleuraafbeelding elke iteratie ongeveer driemaal zoveel berekeningen uitvoert.

Daarnet werd reeds geconcludeerd dat de MSE- en SSI-verbeteringen tussen kleuren- en grijstinten afbeeldingen niet zomaar vergeleken kunnen worden. Om deze reden worden deze bijgevolg niet verder besproken.

**Tabel 6.15:** De beste MSE-waarden van `flor.jpg` in kleur en zwart-wit

Variable	Best mse	iteration	time
<code>rgb=0</code>	197.0126	5	8.899
<code>rgb=1</code>	239.7885	6	37.379

**Tabel 6.16:** De totale CPU-tijden (in seconden) van `colors.jpg` in kleur en zwart-wit met veranderlijke  $a$ -waarden

Variable	CPU time
<code>a=0.0 rgb=0</code>	0.2508
<code>a=1.0 rgb=0</code>	0.3237
<code>a=2.0 rgb=0</code>	0.3178
<code>a=3.0 rgb=0</code>	0.3601
<code>a=4.0 rgb=0</code>	0.3531
<code>a=0.0 rgb=1</code>	0.7689
<code>a=1.0 rgb=1</code>	1.3916
<code>a=2.0 rgb=1</code>	0.8364
<code>a=3.0 rgb=1</code>	1.0575
<code>a=4.0 rgb=1</code>	0.9968

Afgaand op de numerieke MSE-waarden in Tabel 6.14 wordt enigszins vermoed dat de optimale a-waarde zal verschillen naargelang een afbeelding in kleur- of grijstinten wordt verwerkt. Om deze veronderstelling na te gaan moet deze gelden voor een groter aantal verschillende afbeeldingen. Om deze reden worden op analoge wijze nog drie andere afbeeldingen in kleur en zwart-wit verbeterd. Bijkomend wordt ook nog een monochroom kleurenpalet onderzocht. In tegenstelling tot zwart-wit en kleurenafbeeldingen kunnen pixels in monochrome afbeeldingen slechts twee mogelijke kleuren aannemen: zwart of wit, geen grijstinten. Concreet worden “flor.jpg” (Figuur 6.1), “blockSize14.jpg” (Figuur 6.18) en “blockSize112.jpg” (Figuur 6.18) getest. De gevonden optimale a-waarden voor zowel MSE als SSI extrema in kleur, zwart-wit en monochroom worden per afbeelding weergegeven in Tabel 6.17.

**Tabel 6.17:** De optimale a-waarde volgens MSE en SSI in kleur, zwart-wit en monochroom

Afbeelding	Tint	a-waarde optimale MSE	a-waarde optimale SSI
flor.jpg	RGB	1	2
flor.jpg	zwart-wit	1	2
flor.jpg	monochroom	3	5
blockSize14.jpg	RGB	2	3
blockSize14.jpg	zwart-wit	2	3
blockSize14.jpg	monochroom	4	4
blockSize112.jpg	RGB	2	2
blockSize112.jpg	zwart-wit	2	2
blockSize112.jpg	monochroom	3	4

Uit deze resultaten blijkt, in tegenstelling tot voorgaande test, de optimale a-waarde niet te veranderen naargelang de afbeelding in kleur of grijstinten wordt verwerkt. De monochrome afbeeldingen daarentegen verkiezen duidelijk een veel grotere a-waarde voor zowel MSE als SSI. In de meeste gevallen ziet het er ook naar uit dat de iteratie waarin een extrema gevonden wordt het kleinst is voor monochrome afbeeldingen.

### 6.2.7 Wat is de invloed van de gekozen diffusie-functie op het eindresultaat?

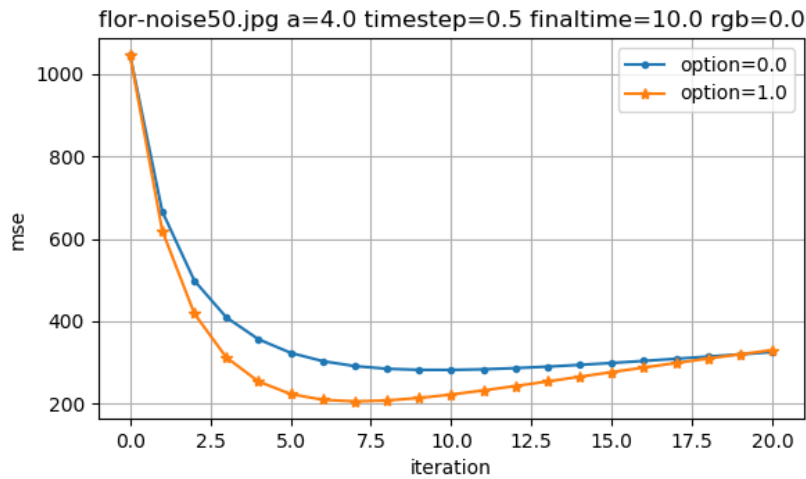
In volgende experimenten wordt de invloed van de gekozen diffusie-functie op het bekomen eindresultaat onderzocht. De implementatie biedt de keuze uit twee verschillende functies aangeduid via de *option* parameter. Deze neemt een waarde aan van 0 of 1 wat respectievelijk overeenkomt met de in (3.8) en (3.9) vermelde diffusie-functies. Opnieuw wordt gebruik gemaakt van de standaardafbeelding (Figuur 6.2) met 50% ruis. Er worden tien oplossingen gegenereerd door de *a*-waarde te laten variëren van nul tot vier en voor elke *a*-waarde de twee mogelijke diffusie-functies uit te proberen. De beste MSE- en SSI-waarden zijn terug te vinden in Tabel 6.18.

**Tabel 6.18:** De beste MSE- en SSI-waarden per diffusie functie met veranderlijke *a*-waarden

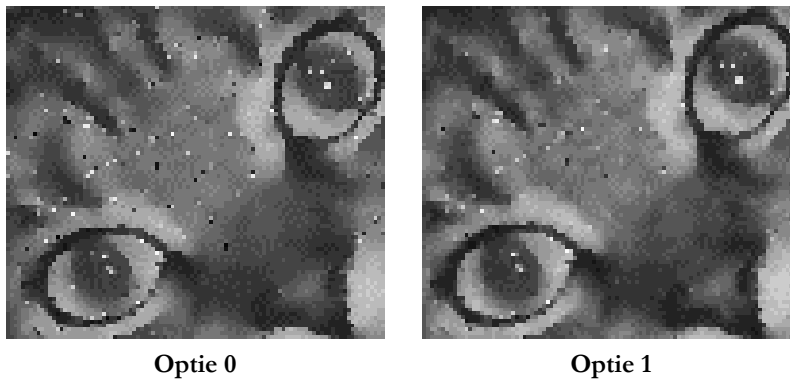
Variable	Best mse	iteration	time	Variable	Best ssi	iteration	time
a=2.0 option=1.0	184.5726	4	7.294	a=3.0 option=1.0	0.7059	7	14.423
a=1.0 option=0.0	189.5256	3	5.505	a=2.0 option=1.0	0.7021	6	11.114
a=3.0 option=1.0	189.9752	6	12.284	a=2.0 option=0.0	0.7006	7	12.933
a=1.0 option=1.0	191.9382	4	7.714	a=4.0 option=1.0	0.6972	9	18.915
a=2.0 option=0.0	197.0126	5	8.998	a=1.0 option=0.0	0.6891	5	8.954
a=0.0 option=1.0	203.7829	3	4.903	a=1.0 option=1.0	0.6884	6	11.977
a=4.0 option=1.0	205.2146	7	14.512	a=0.0 option=0.0	0.6695	5	8.281
a=0.0 option=0.0	205.2628	3	4.767	a=0.0 option=1.0	0.6694	6	10.397
a=3.0 option=0.0	240.6355	8	17.788	a=3.0 option=0.0	0.6655	10	22.313
a=4.0 option=0.0	281.651	10	22.09	a=4.0 option=0.0	0.6187	11	24.214

Als eerste wordt onderzocht of de gekozen optie een effect heeft op de iteratie waarin een extrema gevonden wordt. Wanneer de permutaties met gelijke *a*-waarde vergeleken worden blijkt optie 1 een gunstig (lees verkleinend) effect te hebben op de iteratie op voorwaarde dat de gebruikte *a*-waarde groter is dan één. Deze waarneming is geldig voor zowel MSE- als SSI-waarden.

Vervolgens wordt het effect van de gekozen optie op de MSE- en SSI-waarden zelf bestudeerd. Aangezien telkens dezelfde afbeelding gebruikt wordt kunnen deze waarden rechtstreeks vergeleken worden. Hieruit blijkt dat voor hogere *a*-waarden (2 of meer) de keuze voor optie 1 de gunstigste MSE en SSI-waarden oplevert. In Figuur 6.21 wordt dit verschijnsel geïllustreerd aan de hand van de MSE-*assessment graph* van de oplossingen met *a*=4. De uitvergroete eindresultaten van beide oplossingen zijn eveneens weergegeven in Figuur 6.22. Hierop is duidelijk te zien dat meer ruis verwijderd wordt indien gebruik gemaakt wordt van optie 1.



**Figuur 6.21:** De MSE-assessment graph voor de verschillende diffusie-functies (3.8) en (3.9)



**Figuur 6.22:** Eindresultaten flor.jpg voor beide diffusie-functies met  $a=4$

Tot slot wordt de rekestijden per  $a$ -waarde en optie met elkaar vergeleken. Hieruit volgt opnieuw de vaststelling dat voor  $a$ -waarden groter dan of gelijk aan twee de vereiste rekestijd alvorens een extrema gevonden wordt voor optie één het laagst is.

Uit voorgaande experimenten kan dus besloten worden dat optie één de voorkeur geniet bij  $a$ -waarden groter dan of gelijk aan twee.

Aangezien enkel de allerbeste eindresultaten belangrijk zijn is het interessanter om de oplossing met beste  $a$ -waarde voor optie 0 te vergelijken met de oplossing met beste  $a$ -waarde voor optie 1. Voor maximale SSI gaat dit in bovenstaande resultaten over een  $a$ -waarde van twee met optie 0 en een  $a$ -waarde van drie met optie 1. Deze berekeningen worden nu nog tweemaal herhaald voor dezelfde afbeelding met 20 en 80 procent ruis. Alle beste SSI-waarden voor opties nul en één zijn respectievelijk weergegeven in Tabel 6.19 en Tabel 6.20. Hoewel hieruit blijkt dat optie één steeds de beste resultaten oplevert, moet er opgemerkt worden dat het verschil met optie nul miniem ( $<1\%$ ) is en bijgevolg geen visueel verschil zal opleveren. Rekening houdend met het feit dat de berekeningen voor optie één een langere rekestijd vereisen lijkt optie nul toch voordeliger te zijn. Het verschil in MSE-waarden is visueel ook onmerkbaar.

In praktijk kunnen natuurlijk geen MSE- of SSI-waarden berekend worden aangezien daarvoor een onbeschadigde versie van de afbeelding nodig is. Hierdoor zal op voorhand een  $a$ -waarde geschat moeten worden naargelang bijvoorbeeld de structuur van de beschadigde afbeelding. Afhankelijk van deze schatting kan dan gekozen worden voor optie 1 indien een  $a$ -waarde groter dan 2 verwacht wordt.

Aangezien hoge  $a$ -waarden beter presteren indien optie één gebruikt wordt kan geconcludeerd worden dat de gekozen optie de optimale  $a$ -waarde beïnvloed. Dit is wiskundig gezien geen verrassing aangezien deze parameter inherent deel uitmaakt van de gebruikte diffusie-functie.

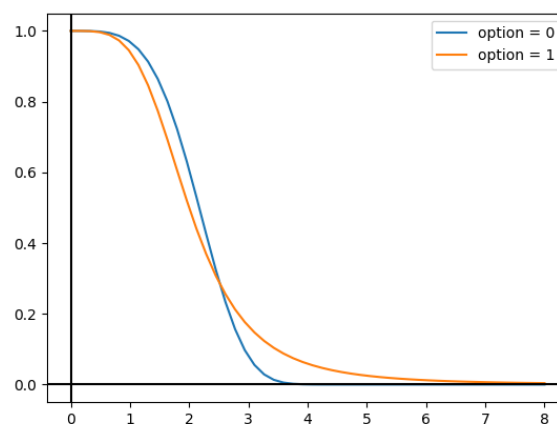
**Tabel 6.19:** De beste  $a$  en bijhorende SSI-waarden bekomen via optie 0 met veranderlijke hoeveelheden ruis

Hoeveelheid ruis	Beste $a$ -waarde	SSI	Rekentijd
20%	2	0.9073	6.9942
50%	2	0.7006	13.8573
80%	2	0.5429	20.4029

**Tabel 6.20:** De beste  $a$  en bijhorende SSI-waarden bekomen via optie 1 met veranderlijke hoeveelheden ruis

Hoeveelheid ruis	Beste $a$ -waarde	SSI	Rekentijd
20%	3	0.9086	7.4021
50%	3	0.7059	15.3198
80%	4	0.54433	25.5313

Een verklaring waarom optie 1 betere resultaten oplevert bij hoge  $a$ -waarden kan opnieuw gevonden worden in het verloop van de gebruikte diffusie-functies met  $a=4$ , weergegeven in Figuur 6.23. Hierop is te zien hoe de overgang tussen de extreme waarden het “zachtst” verloopt bij optie 1 waardoor de beschadigde pixels minder snel foutief verscherpt zullen worden en bijgevolg een beter resultaat opleveren.



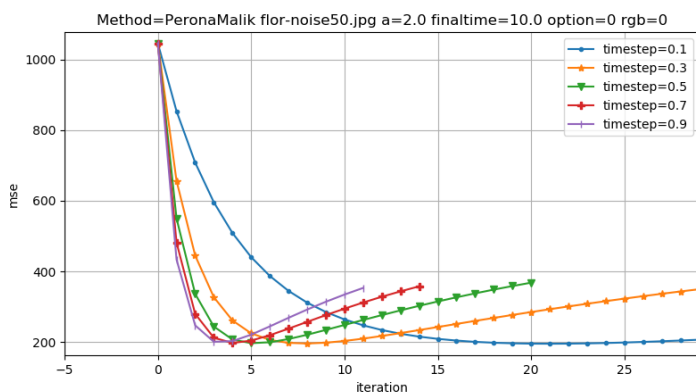
**Figuur 6.23:** Het verloop van beide diffusie-functies (3.8) en (3.9) met  $a=4$

### 6.2.8 Wat is de invloed van de gebruikte tijdstap op het eindresultaat?

In het volgende experiment worden enkele oplossingen gegenereerd met variërende tijdstap. In een eerste fase worden de tijdstappen kleiner dan één bestudeerd. Omdat nog steeds een eindtijd van tien gebruikt wordt zullen kleine tijdstappen zeer veel iteraties veroorzaken (totaal aantal iteraties = eindtijd/tijdstap).

Er wordt vertrokken vanuit 'flor.jpg' (921x518 px) met 50% ruis. De MSE-*assessment graph* van een beperkt aantal oplossingen en de bijhorende resultaten van alle gegenereerde oplossingen zijn terug te vinden in Figuur 6.24 en Tabel 6.21. Hierop is te zien hoe de kleinste tijdstap zowel de hoogste iteratie als benodigde rekentijd heeft alvorens een extrema gevonden wordt. Dit is logisch aangezien een kleinere tijdstap minder verandering per iteratie veroorzaakt waardoor meer iteraties nodig zijn voordat een optimaal resultaat bekomen wordt. Bijgevolg wordt verwacht dat een kleinere tijdstap nauwkeurigere extrema oplevert. De resultaten in Tabel 6.21 bevestigen deze veronderstelling mits een uitzondering voor tijdstappen 0.6 en 0.7. Wel moet opgemerkt worden dat alle gevonden extrema dicht bij elkaar liggen waardoor ze visueel weinig tot niet verschillen. De extreme SSI-waarden vertonen dezelfde eigenschappen.

Aangezien elke iteratie voortbouwt op de vorige is het niet zo dat een oplossing bekomen via een tijdstap van 0.1 om de twee iteraties eenzelfde resultaat oplevert als de resultaten bekomen via een oplossing met een tijdstap van 0.2.

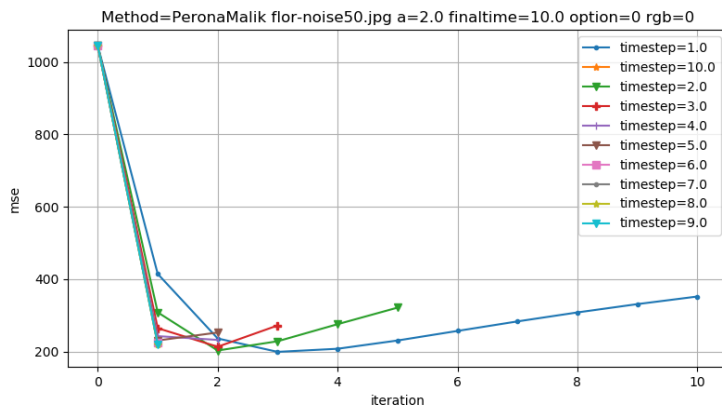


**Figuur 6.24:** De MSE-*assessment graph* met veranderlijke tijdstap < 1

**Tabel 6.21:** De beste MSE-waarden met veranderlijke tijdstap < 1

Variable	Best mse	iteration	time
timestep=0.1	195.4156	21	27.577
timestep=0.2	195.7242	11	16.03
timestep=0.3	196.0394	8	12.671
timestep=0.4	196.5455	6	9.811
timestep=0.5	197.0126	5	8.638
timestep=0.7	197.6248	4	7.604
timestep=0.6	198.6136	5	9.226
timestep=0.8	199.1238	4	7.866
timestep=0.9	200.9743	3	6.202

Nu worden de tijdstappen groter dan één getest. De resultaten hiervan zijn terug te vinden in Figuur 6.25 en Tabel 6.22. Meteen valt op hoe de beste MSE-waarde bij tijdstappen groter dan 2 merkbaar slechter worden. Bovendien blijkt een kleinere tijdstap in veel gevallen niet langer een garantie voor een lagere MSE-waarde. Een tijdstap van vier zal bijvoorbeeld een slechter resultaat opleveren dan een tijdstap van tien of negen. Uit Tabel 6.23 volgt dat een grotere tijdstap een hogere rekentijd per iteratie vereist. Om zo snel mogelijk een extrema te vinden zal dus steeds gezocht moeten worden naar een compromis tussen het aantal benodigde iteraties en de rekentijd per iteratie.



**Figuur 6.25:** De MSE-assessment graph met veranderlijke tijdstap > 1

**Tabel 6.22:** De beste MSE-waarden met veranderlijke tijdstap > 1

Variable	Best mse	iteration	time
timestep=1.0	199.3132	3	6.319
timestep=2.0	203.0429	2	6.011
timestep=3.0	213.4229	2	6.139
timestep=8.0	220.0518	1	4.614
timestep=9.0	220.3383	1	4.795
timestep=7.0	221.1867	1	4.61
timestep=10.0	221.5581	1	4.993
timestep=6.0	224.3702	1	4.355
timestep=5.0	230.7789	1	3.704
timestep=4.0	232.2791	2	6.819

**Tabel 6.23:** De beste rekentijd (in seconden) met veranderlijke tijdstap

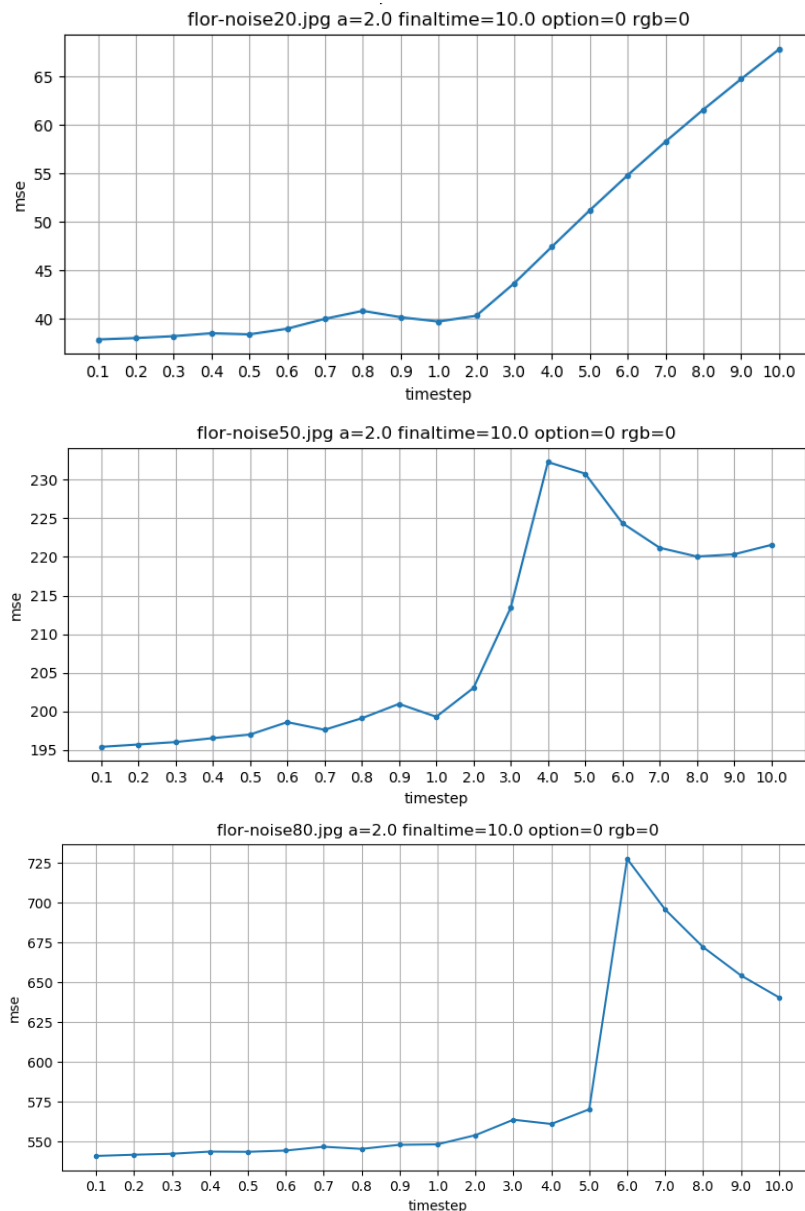
Variable	Best calculationTime
timestep=1.0	2.0508
timestep=2.0	2.6005
timestep=3.0	3.0695
timestep=4.0	3.4097
timestep=5.0	3.704
timestep=6.0	4.3554
timestep=7.0	4.6104
timestep=8.0	4.6144
timestep=9.0	4.7952
timestep=10.0	4.9931

Wanneer dit experiment herhaald wordt met de kleinere afbeelding 'flor-small.jpg' (230x129 pixels) kunnen min of meer dezelfde conclusies genomen worden. Een verschil zit hem in het feit dat reeds bij kleinere tijdstappen afgeweken wordt van de veronderstelling dat kleine tijdstappen een kleinere MSE-waarde opleveren. Concreet wordt voor 'flor-small.jpg' een minimale MSE-waarde gevonden bij een tijdstap van 0.4 en niet 0.1. Opnieuw geldt dat het verschil in MSE-waarden bij tijdstappen kleiner dan één niet of nauwelijks waarneembaar zijn waardoor deze afwijkingen praktisch gezien verwaarloosbaar zijn.

Er werd reeds vastgesteld hoe bij tijdstappen groter dan twee de gevonden extrema drastisch verslechteren. In een laatste experiment wordt nagegaan of deze bovengrens afhankelijk is van de hoeveelheid initiële ruis. De MSE *extremum graphs* van drie afbeeldingen met een initiële ruis van 20, 50 en 80 procent zijn weergegeven in Figuur 6.26. Op eerste gezicht lijkt de bovengrens te stijgen naargelang de hoeveelheid ruis. Zo zal voor 20% ruis een tijdstap van drie of hoger plots slechte resultaten opleveren terwijl er voor 80% ruis pas drastische verslechtering plaatsvindt voor



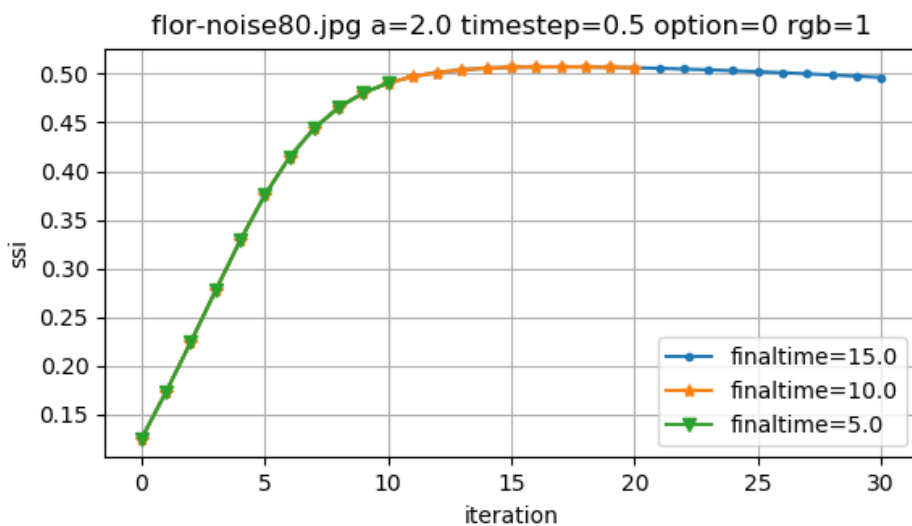
tijdstappen groter dan vijf. Hieruit zou men kunnen concluderen dat bij een hoge initiële ruis de tijdstap minder voorzichtig gekozen moet worden. Dit is echter een incorrecte veronderstelling. Wanneer de minimale MSE-waarde voor elke tijdstap vergeleken wordt met de globaal minimale MSE-waarde, voor alle drie de gevallen bekomen bij een tijdstap van 0.1, blijken deze bij een lage hoeveelheid ruis veel minder af te wijken dan bij hogere hoeveelheden. Indien twee resultaten met een MSE-verschil van maximaal tien fouten als visueel identiek beschouwd worden dan zal bij 20, 50 en 80 procent ruis respectievelijk een maximale tijdstap van vier, twee en één geen slechter resultaat opleveren als wanneer een kleinere tijdstap gebruikt zou worden. De keuze voor de gebruikte tijdstap is met andere woorden minder strikt voor afbeeldingen met een lage hoeveelheid initiële ruis. Bij twijfel wordt aangeraden een tijdstap tussen 0.5 en 1 te kiezen aangezien deze ten alle tijden een goed resultaat zullen opleveren.



**Figuur 6.26:** De MSE *extremum graph* met veranderlijke tijdstap van een afbeelding met 20% (boven), 50% (midden) en 80% (onder) initiële ruis

### 6.2.9 Wat is de invloed van de eindtijd op het eindresultaat?

De eindtijd bepaalt samen met de tijdstap hoeveel iteraties uitgevoerd zullen worden. Ter illustratie worden enkele oplossingen gegenereerd in kleur van “flor-noise80.jpg” met verschillende eindtijden. De SSI-*assessment graph* van deze oplossingen is weergegeven in Figuur 6.27. Hierop is te zien hoe de waarde van de gekozen eindtijd de wiskundige berekeningen niet beïnvloedt en enkel dienstdoet als stop voorwaarde. Bijgevolg moet deze waarde enkel groot genoeg gekozen worden zodat ten alle tijden een extrema gevonden wordt. Zo kan bijvoorbeeld uit Figuur 6.27 afgeleid worden dat een eindtijd van vijf, wat met een tijdstap van 0.5 resulteert in tien iteraties, niet voldoende is aangezien het SSI extrema pas in iteratie zeventien gevonden wordt. De keuze voor de eindtijd zal bijgevolg afhankelijk zijn van alle factoren die de iteratie waarin een extrema gevonden wordt beïnvloeden. Uit voorgaande onderzoeksvragen werd reeds besloten dat de hoeveelheid ruis, de grootte, de structuur (randen, blokgruote) en kleur van een afbeelding hierop een invloed kunnen hebben. Ook werd doorheen het onderzoek vastgesteld dat het SSI-extrema van een afbeelding zich telkens in een latere iteratie bevinden dan het MSE-externum. Er zijn met andere woorden een heleboel factoren die de waarde van de eindtijd zouden kunnen beïnvloeden. In de praktijk bevinden de meeste extrema zich gelukkig in de eerste 20 iteraties waardoor een eindtijd van 10 en tijdstap van 0.5 meer dan voldoende zijn om zowel een MSE-als SSI-extrema te vinden.



**Figuur 6.27:** De SSI-*assessment graph* met veranderlijke eindtijd (finaltime)

## 6.3 Experimenteren: geregulariseerde Perona-Malik

### 6.3.1 Wat is de invloed van de gekozen sigma op het eindresultaat?

De implementatie van de geregulariseerde Perona-Malik vergelijking maakt gebruik van alle reeds besproken parameters uit de normale Perona-Malik implementatie aangevuld met een nieuwe parameter: sigma. Om de invloed van deze parameter op het eindresultaat te onderzoeken wordt de afbeelding ‘flor.jpg’ met 50% ruis verbeterd via de geregulariseerde Perona-Malik vergelijking met sigma-waarden tussen nul en twee. De beste MSE en SSI-waarden voor elke oplossing zijn weergegeven in Tabel 6.24. Deze tabellen zijn aangevuld met een oplossing die gebruik maakt van de gewone Perona-Malik vergelijking met dezelfde parameterwaarden. Hieruit wordt afgeleid dat een sigma-waarde van nul dezelfde resultaten oplevert als die van de gewone Perona-Malik vergelijking doch een langere rekentijd vereist. Een sigma-waarde van nul is dus inefficiënt en wordt best ten alle tijden vermeden.

Verder valt op hoe de iteratie waarin een extrema gevonden wordt voor elke sigma-waarde, met uitzondering van nul, lager is dan bij de gewone Perona-Malik vergelijking. Bovendien zijn ook alle MSE-waarden beter indien het geregulariseerde model gebruikt wordt, zeker voor een sigma-waarde gelijk aan 0.5 is het resultaat veelbelovend. De SSI-waarden daarentegen zijn beter voor de gewone Perona-Malik vergelijking hoewel het verschil indien een sigma-waarde van 0.5 gebruikt wordt verwaarloosbaar klein is.

In een volgende stap worden enkele andere afbeeldingen op dezelfde manier getest om af te leiden of bovenstaande waarnemingen algemeen geldig zijn. Concreet worden de afbeeldingen met scherpe en vloeiende randen (Figuur 6.15), de grote en kleine variant van ‘flor.jpg’, de afbeeldingen met variërende blokgroottes (Figuur 6.18) en een nieuwe afbeelding: ‘frul.jpg’ (Figuur 6.28) getest. Voor acht van de negen afbeeldingen zal een sigma-waarde van 0.5 nog steeds het beste MSE-resultaat opleveren. Enkel voor de afbeelding met vloeiende randen daalt de MSE-waarde met stijgende sigma-waarden. Het verschil tussen deze MSE-waarden is echter zo klein ( $< 0.1$  fouten) dat deze deviante waarneming geen significante betekenis heeft. Verder moet vermeld worden dat in sommige gevallen de geregulariseerde Perona-Malik vergelijking met een sigma-waarde van twee slechter presteert dan de gewone Perona-Malik vergelijking. Dit is echter geen probleem aangezien een sigma-waarde van één of kleiner in alle geteste gevallen wel beter presteert waardoor hogere sigma-waarden nooit overwogen zullen worden.

**Tabel 6.24:** De beste MSE- en SSI-waarden met veranderlijke sigma-waarden

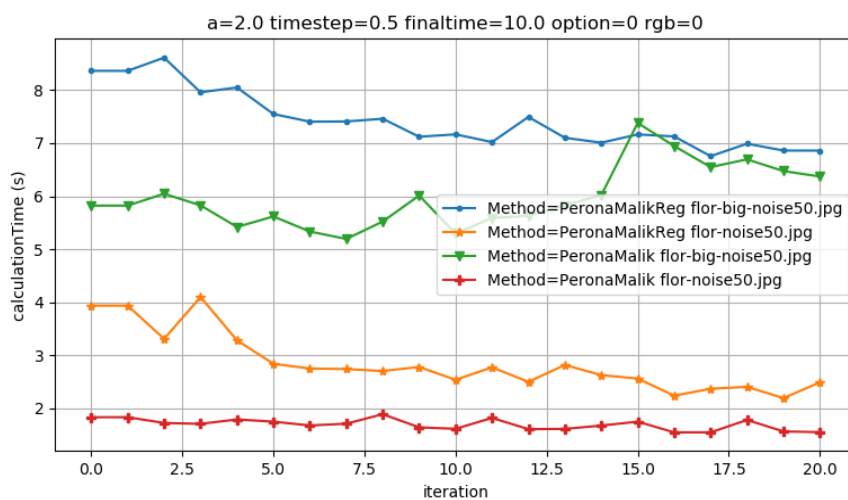
Variable	Best mse	iteration	time	Variable	Best ssi	iteration	time
Method=PeronaMalikReg sigma=0.5	186.331	3	9.61	Method=PeronaMalikReg sigma=0.0	0.7006	7	15.061
Method=PeronaMalikReg sigma=1.0	191.2266	2	6.127	Method=PeronaMalik	0.7006	7	13.017
Method=PeronaMalikReg sigma=1.5	193.581	2	6.327	Method=PeronaMalikReg sigma=0.5	0.6939	4	12.422
Method=PeronaMalikReg sigma=2.0	195.7622	2	6.276	Method=PeronaMalikReg sigma=1.0	0.686	4	12.066
Method=PeronaMalikReg sigma=0.0	197.0126	5	10.858	Method=PeronaMalikReg sigma=1.5	0.6814	4	12.326
Method=PeronaMalik	197.0126	5	9.255	Method=PeronaMalikReg sigma=2.0	0.6786	4	12.548



**Figuur 6.28:** De afbeelding frul.jpg, gebruikt om de invloed van de gekozen sigma-waarde na te gaan

Uit alle geteste afbeeldingen is geen duidelijk verband af te leiden tussen de CPU-tijd en gebruikte sigma-waarde. In de meeste gevallen zal een hogere sigma-waarde leiden tot een iets hogere verwerkingstijd hoewel dit niet steeds het geval is. Zelfs indien de sigma-waarde een invloed uitoefent zal de grootte van de afbeelding en het al dan niet in kleur verwerken ervan een veel grotere impact hebben op totale CPU-tijd. Wel is het zo dat in alle geteste afbeeldingen het aantal benodigde iteraties alvorens een extrema gevonden wordt daalt (of constant blijft) bij stijgende sigma-waarde.

In Figuur 6.29 worden de rekestijden per iteratie tussen de gewone en geregulariseerde Perona-Malik vergelijking vergeleken. Aangezien de geregulariseerde methode eigenlijk een uitbreiding is van de standaard Perona-Malik vergelijking verbaast het niet dat deze gemiddeld meer CPU-tijd vraagt. Hierboven werd reeds vastgesteld hoe de geregulariseerde versie minder iteraties vereist om een extrema te bekomen. Hierdoor kan, desondanks de langere rekestijd per iteratie, het toch voorkomen dat de geregulariseerde versie sneller een optimale MSE of SSI-waarde bekomt.



**Figuur 6.29:** De rekestijden per iteratie (in seconden) voor de gewone en geregulariseerde PM-vergelijking

### 6.3.2 Gedraagt de geregulariseerde Perona-Malik vergelijking zich anders dan de standaard Perona-Malik vergelijking?

Zoals reeds aangekaart is de geregulariseerde Perona-Malik vergelijking een uitbreiding op de standaardmethode. Zo beschikken deze namelijk, op de parameter  $\sigma$  na, over dezelfde parameters. In deze sectie wordt onderzocht of de genomen conclusies voor de klassieke Perona-Malik vergelijking ook gelden voor de geregulariseerde variant.

Concreet wordt het effect van de grootte v.d. initiële afbeelding, de hoeveelheid initiële ruis, het al dan niet in kleur verwerken en de gekozen optie bestudeert. In Figuur 6.30 bevinden zich links de *MSE-assessment graphs* van de experimenten uitgevoerd met de geregulariseerde vergelijking, rechts bevinden zich ter vergelijking de equivalente grafieken voor de standaardmethode. Wanneer deze resultaten vergeleken worden blijkt hoe in de eerste drie experimenten de geregulariseerde Perona-Malik vergelijking zich gelijkaardig gedraagt als de gewone Perona-Malik vergelijking. In het laatste experiment daarentegen, waar de invloed van de optie parameter op verschillende  $a$ -waarden bestudeerd wordt, zijn beide grafieken niet langer gelijkvormig. Hieronder wordt deze afwijking verder onderzocht.

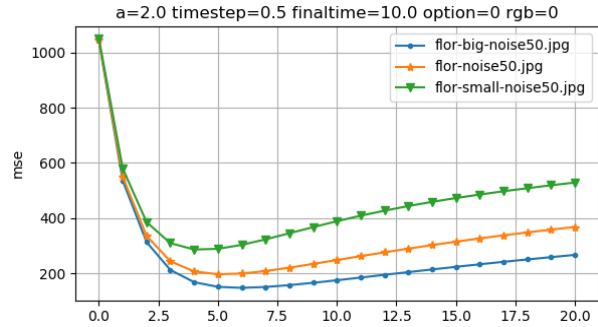
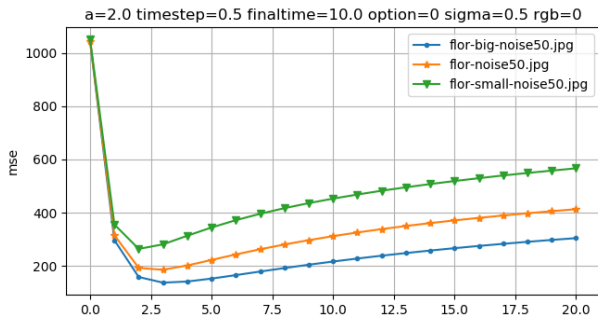
In Tabel 6.25 bevinden zich de beste MSE- en SSI-waarden voor de geregulariseerde en standaard Perona-Malik vergelijking. Indien de gewone vergelijking gebruikt wordt zal de keuze voor optie één het beste resultaat opleveren indien de  $a$ -waarde relatief groot is (veronderstel voor deze afbeelding groter dan of gelijk aan twee). Een kleinere  $a$ -waarde daarentegen levert een visueel nagenoeg equivalent resultaat op ongeacht de gebruikte optie. Indien de resultaten voor de geregulariseerde methode bekeken worden valt op hoe optie 1 steeds betere resultaten oplevert ongeacht de gebruikte  $a$ -waarde. Het verschil tussen beide opties is in alle gevallen echter vrij klein waardoor deze visueel amper verschillen. Bijgevolg zal de  $a$ -waarde en gekozen diffusie-functie amper invloed uitoefenen op het bekomen eindresultaat indien de geregulariseerde Perona-Malik vergelijking gebruikt wordt.

Er kan besloten worden dat de geregulariseerde methode minder fluctuerende eindresultaten zal opleveren. Hiermee wordt bedoeld dat het verschil tussen de beste en slechtste MSE of SSI-waarde voor alle geteste  $a$ -waarden veel kleiner zal zijn voor de geregulariseerde Perona-Malik vergelijking dan voor de standaardmethode. Deze MSE-verschillen zijn ter illustratie voor beide opties terug te vinden in Tabel 6.26. Samenvattend kan gesteld worden dat indien een niet optimale  $a$ -waarde gebruikt wordt in de geregulariseerde methode het eindresultaat visueel niet veel zal verschillen dan wanneer de beste  $a$ -waarde gebruikt zou worden.

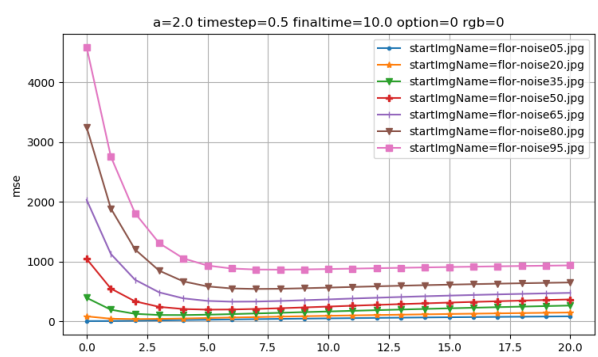
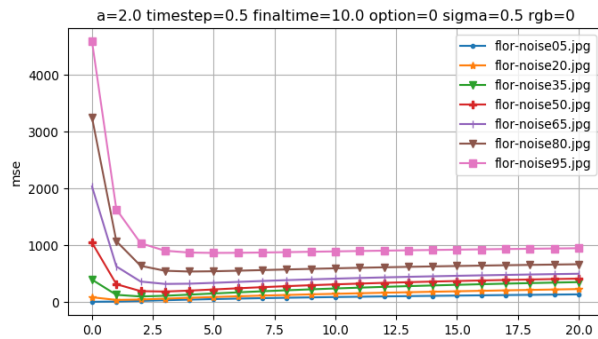
**Geregulariseerde Perona-Malik**

**Standaard Perona-Malik**

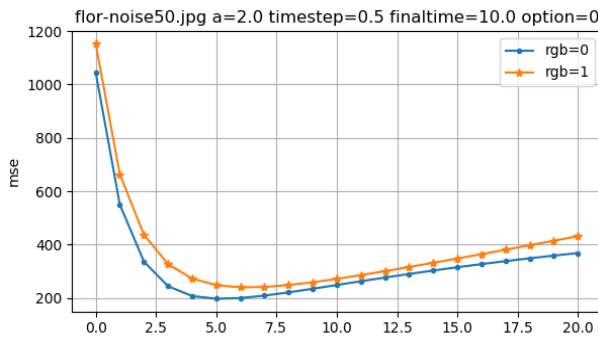
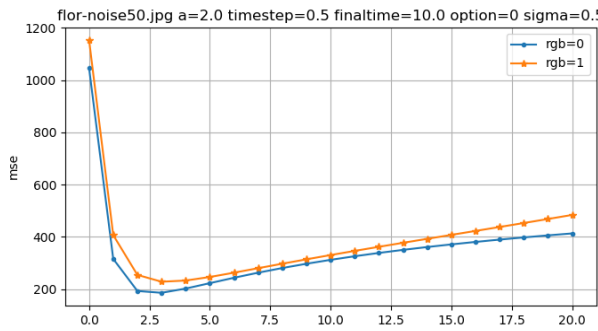
**Invloed afmeting initiële afbeelding**



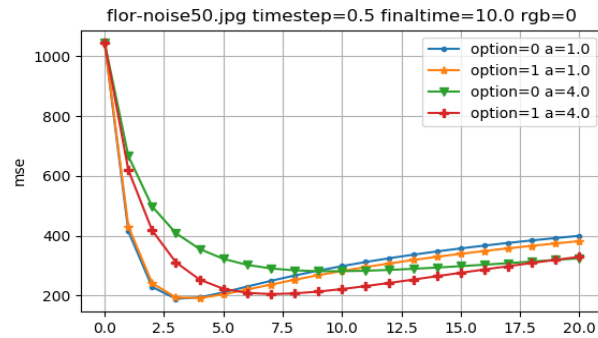
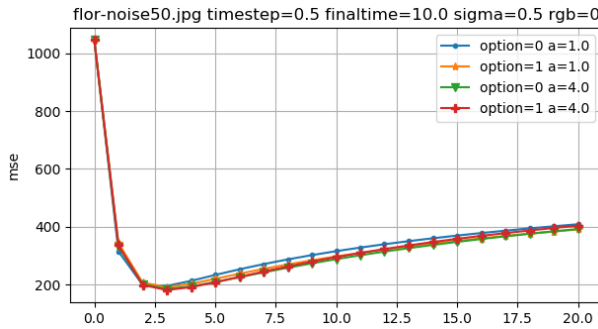
**Invloed hoeveelheid ruis in initiële afbeelding**



**Invloed van kleur of grijs tinten**



**Invloed van de gekozen diffusie-functie**



**Figuur 6.30:** Een vergelijking tussen de geregulariseerde (links) en de gewone PM-vergelijking (rechts)

**Tabel 6.25:** De beste MSE- en SSI- waarden met veranderlijke diffusie-functie en a-waarden voor de geregulariseerde (links) en gewone PM-vergelijking (rechts)

Geregulariseerde Perona-Malik				Standaard Perona-Malik			
Variable	Best mse	iteration	time	Variable	Best mse	iteration	time
option=1 a=4.0	180.6847	3	8.277	option=1 a=2.0	184.5726	4	6.918
option=1 a=5.0	181.3552	3	8.36	option=0 a=1.0	189.5256	3	7.051
option=1 a=3.0	181.3772	3	8.317	option=1 a=3.0	189.9752	6	12.146
option=0 a=3.0	183.6576	3	9.927	option=1 a=1.0	191.9382	4	6.883
option=0 a=4.0	184.4418	3	9.415	option=0 a=2.0	197.0126	5	8.899
option=1 a=2.0	184.4797	3	7.923	option=1 a=4.0	205.2146	7	12.722
option=0 a=5.0	185.9507	3	10.56	option=1 a=5.0	226.5543	9	16.536
option=0 a=2.0	186.331	3	11.188	option=0 a=3.0	240.6355	8	15.007
option=1 a=1.0	190.8783	3	7.489	option=0 a=4.0	281.651	10	20.476
option=0 a=1.0	194.8376	3	9.363	option=0 a=5.0	311.6055	11	22.169

Variable	Best ssi	iteration	time	Variable	Best ssi	iteration	time
option=0 a=1.0	0.6847	4	12.482	option=0 a=5.0	0.5842	12	23.989
option=1 a=1.0	0.6874	5	12.525	option=0 a=4.0	0.6187	11	22.903
option=0 a=2.0	0.6939	4	15.287	option=0 a=3.0	0.6655	10	18.861
option=1 a=2.0	0.6945	4	10.437	option=1 a=5.0	0.6791	11	19.996
option=1 a=3.0	0.6996	4	11.09	option=1 a=1.0	0.6884	6	10.38
option=0 a=3.0	0.6999	4	12.881	option=0 a=1.0	0.6891	5	11.329
option=1 a=4.0	0.7024	4	10.889	option=1 a=4.0	0.6972	9	16.328
option=0 a=4.0	0.703	4	12.754	option=0 a=2.0	0.7006	7	12.33
option=1 a=5.0	0.7041	4	10.996	option=1 a=2.0	0.7021	6	10.103
option=0 a=5.0	0.7045	4	13.286	option=1 a=3.0	0.7059	7	13.856

**Tabel 6.26:** Het MSE-verschil tussen het beste en slechtste resultaat met veranderlijke a-waarden per diffusie functie

	MSE-verschil beste en slechtste resultaat: geregulariseerde PM	MSE-verschil beste en slechtste resultaat: standaard PM
Optie 0	11.18	122.0799
Optie 1	10.1936	41.9817

## 6.4 Experimenteren: Fractionele Diffusie

### 6.4.1 Wat is de invloed van de grootte van het masker op het eindresultaat?

De implementatie laat de gebruiker toe de grootte van het gewenste masker in te stellen. In deze paragraaf zal onderzocht worden hoe deze parameter het eindresultaat beïnvloed. In onderdeel 3.3 werd reeds vermeld dat het masker steeds een vierkante vorm aanneemt waarbij alle zijden een oneven aantal pixels lang zijn zodat het ten allen tijden één centraal middelpunt heeft. Concreet wordt in dit experiment het standaard masker met een grootte tussen drie en vijftien pixels breed gebruikt. De afbeelding wordt verbeterd totdat zowel de optimale MSE- en SSI-waarden bekomen zijn. Als afbeelding wordt opnieuw gebruik gemaakt van “flor-noise50.jpg” (Figuur 6.2).

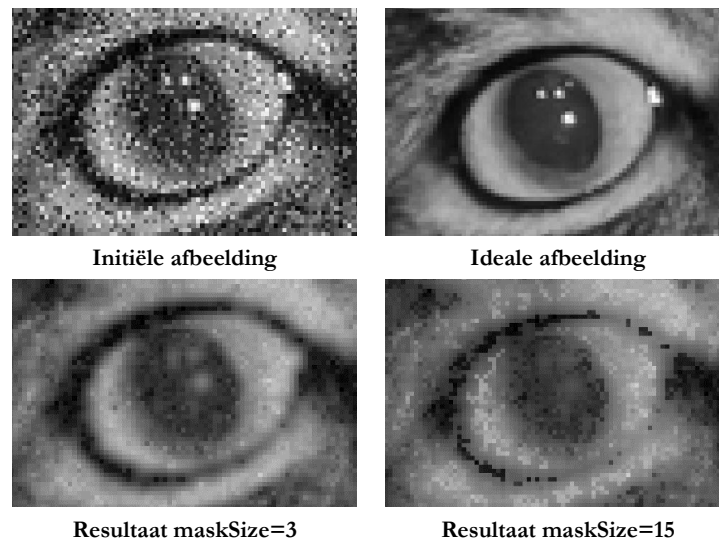
In Tabel 6.27 worden de beste MSE- en SSI-waarden per masker-grootte weergegeven alsook de benodigde iteraties en tijd. Uit deze gegevens blijkt dat voor MSE-waarden kleinere maskers merkbaar betere resultaten opleveren, voor SSI-waarden zijn de verschillen verwaarloosbaar klein. In Figuur 6.31 zijn een uitvergroot deel van de eindresultaten met optimale MSE-waarden van masker-groottes drie en vijftien weergegeven aangevuld met de initiële en ideale afbeelding ter vergelijking. Hierop is duidelijk te zien hoe een masker-grootte van drie inderdaad een beter resultaat oplevert. Verder wordt opgemerkt hoe een kleiner masker in een hogere iteratie een optimaal resultaat bekommt. Hieruit kan afgeleid worden dat een kleiner masker minder verandering per iteratie teweegbrengt.

**Tabel 6.27:** De beste MSE- en SSI-waarden met veranderlijke maskergrootte (maskSize)

Variable	Best mse	iteration	time
maskSize=3	184.4066	11	102.872
maskSize=5	189.9604	6	68.835
maskSize=7	198.1428	5	87.576
maskSize=9	204.765	4	216.9
maskSize=11	210.0147	3	192.79
maskSize=13	215.0172	3	268.026
maskSize=15	220.5804	3	367.095

Variable	Best ssi	iteration	time
maskSize=5	0.6535	11	126.543
maskSize=7	0.6501	9	174.271
maskSize=9	0.6453	7	366.657
maskSize=3	0.6451	17	158.752
maskSize=11	0.6403	6	384.733
maskSize=13	0.636	6	549.448
maskSize=15	0.6321	5	609.874

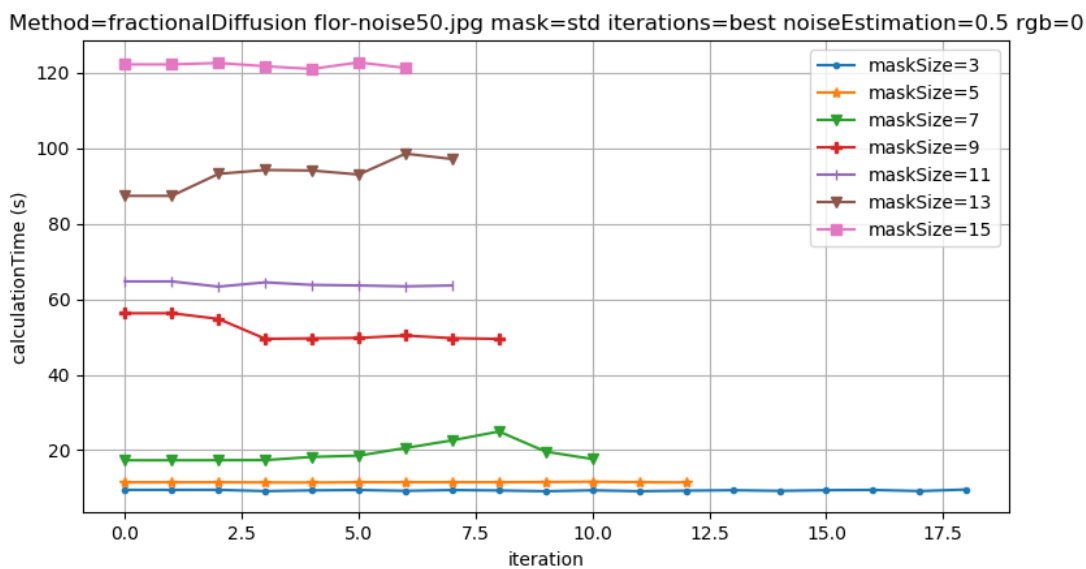




**Figuur 6.31:** Eindresultaten flor.jpg veranderlijke maskergrootte (maskSize)

Alvorens de benodigde rekestijden te vergelijken is het belangrijk te herhalen dat de implementatie ervoor zorgt dat de rekestijden van maskers groter dan zeven pixels breed niet zomaar vergeleken mogen worden met die van kleinere maskers (zie onderdeel 4.2.3).

In Figuur 6.32 worden de verschillende rekestijden per iteratie weergegeven. Hierop is te zien hoe grotere maskers een hogere rekestijd per iteratie vereisen. Dit is geen onverwachte conclusie aangezien een groter masker resulteert in meer berekeningen per convolutie. Aangezien hierboven reeds werd vastgesteld dat kleinere maskers meer iteraties nodig hebben alvorens een optimaal resultaat bekomen wordt kan niet gegarandeerd worden dat een kleiner masker ook daadwerkelijk sneller een optimaal resultaat bekommt.



**Figuur 6.32:** De rekestijd per iteratie met veranderlijke maskergrootte (maskSize)

Logischerwijs zal indien grotere maskers gebruikt worden een groter aantal nabijgelegen pixels een invloed uitoefenen op de nieuwe pixelwaarde. Vertrekkend hieruit kan men zich afvragen of grote maskers een minder slecht of zelfs beter resultaat opleveren voor afbeeldingen met grote aaneengesloten oppervlakken waarbij burens vaak dezelfde pixel waarden hebben. Dit kan getest worden door bovenstaand experiment te herhalen met de afbeeldingen “blockSize14.jpg” en “blockSize112.jpg” uit Figuur 6.18 waaraan 50% Gaussische ruis werd toegevoegd. Uit de resultaten in Tabel 6.28 volgt dat wanneer de afbeelding opgebouwd is uit grote homogene oppervlakken (grote blok-grootte), een groter masker inderdaad betere resultaten oplevert dan een klein masker.

**Tabel 6.28:** De beste MSE-waarden met veranderlijke blok-grootte en maskergrootte (maskSize)

Variable	Best mse	iteration	time
blockSize112-noise50.jpg maskSize=7	131.3231	12	170.36
blockSize112-noise50.jpg maskSize=5	137.0275	15	115.657
blockSize112-noise50.jpg maskSize=3	165.1694	21	128.797
blockSize14-noise50.jpg maskSize=3	432.3445	9	56.866
blockSize14-noise50.jpg maskSize=5	435.0476	4	33.161
blockSize14-noise50.jpg maskSize=7	443.3427	3	32.56

Aangezien het moeilijk is om op voorhand in te schatten welke masker-grootte het snelst een optimaal resultaat bekomt wordt in realiteit een compromis gezocht zodat een relatief goed resultaat bekomen wordt in een redelijke tijd.

Uit voorgaande experimenten blijkt een masker van vijf pixels breed steeds een sneller en visueel identiek resultaat op te leveren als de best mogelijke masker-grootte. Bijgevolg wordt deze masker-grootte aangeraden.

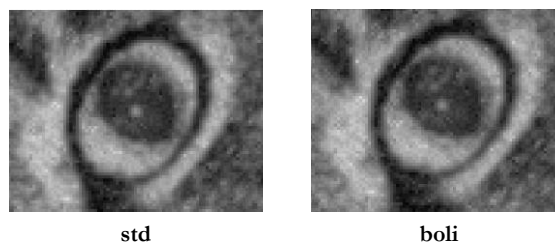
### 6.4.2 Wat is het verschil tussen de twee geïmplementeerde maskers op het eindresultaat?

Zoals in Hoofdstuk 3 werd vermeld zijn beide geïmplementeerde maskers erg gelijkaardig omdat het boli-masker een uitbreiding is van het std-masker door ook rekening te houden met de diagonale burens van elke pixel. Beide maskers werden reeds voorgesteld in Figuur 3.1 en Figuur 3.2. Om na te gaan welk masker het beste resultaat oplevert wordt de afbeelding “flor-noise50.jpg” (Figuur 6.2) door beide maskers verbeterd en vervolgens onderling vergeleken. Bijkomend worden ook verschillende masker-groottes uitgetest om na te gaan of deze zich anders gedragen afhankelijk van het gebruikte masker.

De bekomen MSE-waarden zijn weergegeven in Tabel 6.29. Hieruit blijkt dat het standaard masker, ongeacht de grootte, in theorie de slechtste resultaten oplevert. Het verschil tussen beide maskers is echter zo klein dat ze visueel niet-onderscheidbare resultaten voortbrengen (Figuur 6.33). Om deze reden zal de vereiste rekentijd doorslaggevend zijn in de keuze van het optimale masker. Indien zowel iteratie als tijd bestudeerd worden kunnen twee vaststellingen genomen worden: het boli-masker vereist meestal minder iteraties voor het vinden van een optimaal eindresultaat maar zal tegelijkertijd ook iets meer rekentijd per iteratie nodig hebben. Hierdoor zal het boli-masker met groottes van drie en zeven pixels breed het snelst zijn optimale resultaat bekomen omdat het minder iteraties vereist dan het standaard masker. Voor een masker-grootte van vijf pixels daarentegen moeten beide type maskers hetzelfde aantal iteraties uitvoeren waardoor nu het standaard masker het snelst een eindresultaat bekomt. Aangezien de rekentijd per iteratie voor beide maskers sowieso hoog oploopt wordt aangeraden zoveel mogelijk het boli-masker te gebruiken. De keuze tussen beide maskers zal met andere woorden visueel gelijke resultaten opleveren en enkel de vereiste CPU-tijd beïnvloeden.

**Tabel 6.29:** De beste MSE-waarden met veranderlijk masker en maskergrootte (maskSize)

Variable	Best mse	iteration	time
mask=std maskSize=3	184.4066	11	102.872
mask=boli maskSize=3	187.9563	9	96.895
mask=std maskSize=5	189.9604	6	68.835
mask=boli maskSize=5	196.637	6	71.559
mask=std maskSize=7	198.1428	5	87.576
mask=boli maskSize=7	204.6865	4	68.517



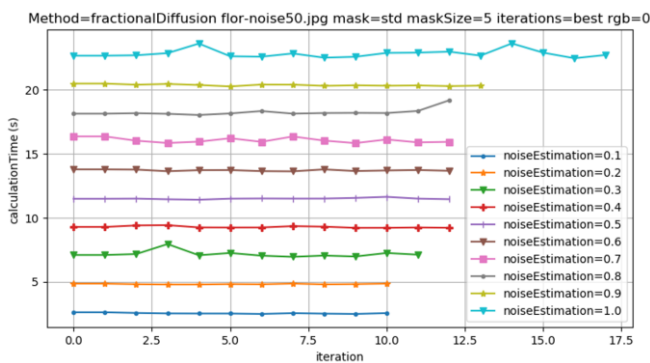
**Figuur 6.33:** Eindresultaten flor.jpg voor beide maskers

### 6.4.3 Wat is de invloed van de geschatte hoeveelheid ruis in de initiële afbeelding op het eindresultaat?

De *noiseEstimation* parameter dient als indicatie voor de hoeveelheid ruis in de beschadigde afbeelding en zal bepalen hoeveel procent van de pixels met hoogste gradiënten uiteindelijk geconvolveerd zullen worden. Een goed geschatte waarde kan voorkomen dat onbeschadigde pixels foutief aangepast worden waardoor het eindresultaat minder afwijkt van de ideale afbeelding. In het volgende experiment wordt nagegaan in welke mate een goede of slechte schatting het eindresultaat beïnvloed. Concreet zal opnieuw “flor-noise50.jpg” (Figuur 6.2) verbeterd worden met variërende hoeveelheid geschatte ruis.

Enkele numerieke resultaten van dit experiment zijn weergegeven in Tabel 6.30. Hieruit blijkt onder meer dat een schatting van 70 procent ruis het beste resultaat oplevert ook al werd aan de initiële afbeelding slechts 50 procent Gaussische ruis toegevoegd. Het verschil in MSE-waarden tussen beide schattingen is echter kleiner dan tien waardoor beide resultaten visueel niet verschillen.

Het grote voordeel van een lagere schatting zonder merkbaar verschillend eindresultaat zit hem in de mogelijke tijdswinst. Dit kan aangetoond worden via de *time* kolom in Tabel 6.30 en de grafiek weergegeven in Figuur 6.34. Hieruit volgt niet alleen dat kleinere schattingen minder rekentijd per iteratie vereisen maar ook dat ze minder iteraties vereisen alvorens een optimaal resultaat bekomen wordt. Later blijkt hoe deze laatste vaststelling niet altijd van toepassing is. De tijdswinst die met dergelijke schattingen behaald kan worden is aanzienlijk. Zo zal de keuze voor 50 procent geschatte ruis i.p.v. 70 procent geschatte ruis een tijdswinst van maar liefst 28 seconden opleveren.



**Figuur 6.34:** De rekentijd per iteratie (in seconden) met veranderlijke schatting hoeveelheid ruis (*noiseEstimation*)

**Tabel 6.30:** De beste MSE-waarden met veranderlijke schatting hoeveelheid ruis (*noiseEstimation*)

Variable	Best mse	iteration	time
<i>noiseEstimation</i> =0.7	181.2835	6	96.723
<i>noiseEstimation</i> =0.8	181.9106	5	90.57
<i>noiseEstimation</i> =0.9	183.0942	5	102.159
<i>noiseEstimation</i> =0.6	183.2902	6	82.416
<i>noiseEstimation</i> =1.0	188.4217	5	114.45
<i>noiseEstimation</i> =0.5	189.9604	6	68.835
<i>noiseEstimation</i> =0.4	206.8957	7	65.164
<i>noiseEstimation</i> =0.3	243.5633	7	50.714
<i>noiseEstimation</i> =0.2	329.6213	8	38.653
<i>noiseEstimation</i> =0.1	532.8964	9	23.037

Om het nut van deze parameter nogmaals aan te duiden wordt dit experiment herhaald met een afbeelding beschadigd door 10% *salt-and-pepper noise*. Uit de resultaten in Tabel 6.31 blijkt dat een schatting van 10% ruis nu wel degelijk het beste resultaat oplevert. Indien deze parameter niet instelbaar zou zijn, en dus alle pixels als ruis beschouwd zouden worden (*estimateNoise=1*), wordt een veel slechter eindresultaat bekomen in vijf keer zoveel tijd. De (uitvergroete) eindresultaten bekomen via een schatting van 10% of 100% ruis zijn te bezichtigen in Figuur 6.35.

**Tabel 6.31:** De beste MSE-waarden met veranderlijke schatting hoeveelheid ruis (*noiseEstimation*) voor 10% salt-and-pepper noise

Variable	Best mse	iteration	time
<i>noiseEstimation=0.1</i>	76.6831	7	23.102
<i>noiseEstimation=0.08</i>	77.7109	10	29.3
<i>noiseEstimation=0.2</i>	86.7925	4	24.489
<i>noiseEstimation=0.3</i>	92.1681	4	36.943
<i>noiseEstimation=0.4</i>	95.9179	4	51.731
<i>noiseEstimation=0.5</i>	98.7661	4	72.397
<i>noiseEstimation=1.0</i>	110.3588	4	125.991
<i>noiseEstimation=0.06</i>	110.5522	15	32.548



*noiseEstimation = 0.1*



*noiseEstimation = 1*

**Figuur 6.35:** Eindresultaten *flor.jpg* beschadigd met 10% salt-and-pepper noise met een schatting van 10% (links) en 100% hoeveelheid initiële ruis (rechts)

## 6.5 Vergelijking Perona-Malik en Fractionele Diffusie

In voorgaande experimenten werd aangetoond hoe via de (geregulariseerde) Perona-Malik vergelijking en fractionele diffusie een door ruis beschadigde afbeelding herstelt kan worden. In dit onderdeel worden beide methodes met elkaar vergeleken en wordt bestudeerd wanneer welke methode de beste resultaten oplevert. Ter vergelijking worden ook de in Hoofdstuk 2.3 vermelde basismethoden gebruikt om na te gaan of de geavanceerde wiskundige modellen daadwerkelijk zoveel beter presteren als theoretisch beweerd wordt. Opnieuw wordt vertrokken vanuit een onderzoeksvraag.

### 6.5.1 Welke methode werkt het best bij het verwijderen van Gaussische ruis?

Om na te gaan welk wiskundig model het best Gaussische ruis verwijderd worden enkele voorgaande experimenten voor alle modellen in kleur hernomen waarna de beste MSE- en SSI-waarden per methode met elkaar vergeleken worden.

In Tabel 6.32 bevinden zich de beste MSE- en SSI-waarden bekomen bij het herstellen van afbeeldingen met initieel 20, 50 en 80 procent Gaussische ruis. Rekening houdend met zowel MSE als SSI-waarden leveren de twee Perona-Malik implementaties voor alle hoeveelheden ruis consistent aanvaardbare resultaten op. Fractionele diffusie daarentegen doet het in vergelijking verrassend slecht en levert voor grote hoeveelheden ruis zelfs slechtere resultaten dan de eenvoudige methodes op basis van gemiddelden en Gaussiaans vervagen. Opvallend is ook hoe het verwijderen van ruis via medianen voor afbeeldingen met 50 en 80 procent ruis zeer goede resultaten oplevert terwijl dezelfde methode voor lagere hoeveelheden ruis duidelijk minder goed presteert. Aangezien lagere percentages ruis in de praktijk het meest voorkomen worden in Figuur 6.36 en Figuur 6.37 een uitvergroot deel van de best bekomen eindresultaten volgens SSI-waarden van de afbeelding met 20% initiële ruis weergegeven.

**Tabel 6.32:** De beste MSE- en SSI-waarden voor alle modellen met 20, 50 en 80 procent Gaussische ruis**20% Gaussische ruis**

Variable	Best mse	iteration	time
Method=PeronaMalikReg	49.2414	1	8.875
Method=PeronaMalik	49.4814	2	10.338
Method=gaussian	72.3061	1	2.018
Method=fractionalDiffusion	75.4682	3	50.369
Method=median	86.322	1	0.892
Method=mean	98.5679	1	1.746

Variable	Best ssi	iteration	time
Method=PeronaMalik	0.886	3	15.472
Method=PeronaMalikReg	0.8779	2	17.751
Method=median	0.8543	1	0.892
Method=gaussian	0.85	1	2.018
Method=mean	0.8275	1	1.746
Method=fractionalDiffusion	0.7802	7	112.869

**50% Gaussische ruis**

Variable	Best mse	iteration	time
Method=median	216.4893	2	2.055
Method=PeronaMalikReg	228.6822	3	25.454
Method=PeronaMalik	239.7885	6	37.379
Method=gaussian	259.0605	3	3.112
Method=mean	260.007	2	2.363
Method=fractionalDiffusion	277.2311	12	438.981

Variable	Best ssi	iteration	time
Method=median	0.6585	9	9.41
Method=PeronaMalik	0.6572	8	49.917
Method=PeronaMalikReg	0.644	5	43.098
Method=gaussian	0.6123	7	7.324
Method=mean	0.6118	5	5.687
Method=fractionalDiffusion	0.5348	15	553.404

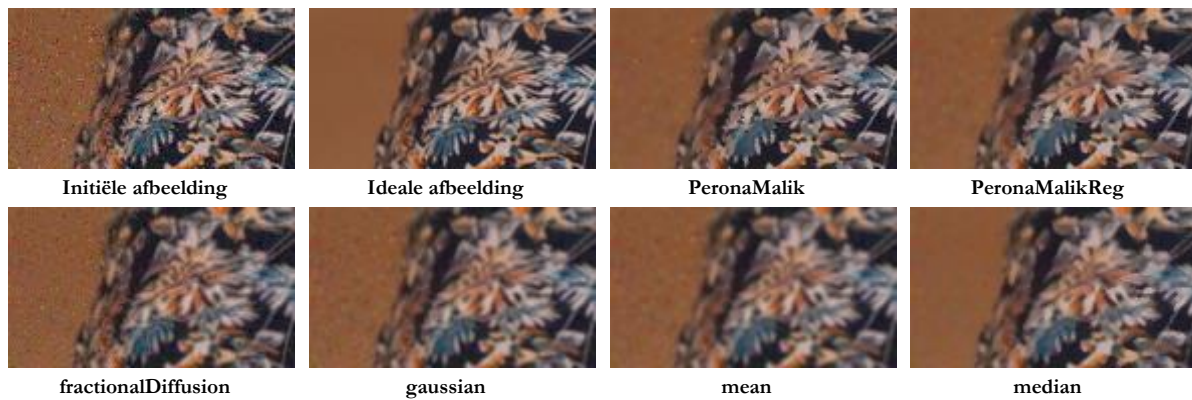
**80% Gaussische ruis**

Variable	Best mse	iteration	time
Method=median	532.6055	32	32.109
Method=PeronaMalik	618.6797	10	62.287
Method=PeronaMalikReg	631.0574	7	58.068
Method=fractionalDiffusion	678.9461	25	1386.979
Method=mean	679.0443	6	7.05
Method=gaussian	682.4983	8	8.95

Variable	Best ssi	iteration	time
Method=PeronaMalik	0.5073	17	104.278
Method=PeronaMalikReg	0.5049	16	131.456
Method=mean	0.4966	19	21.372
Method=gaussian	0.4927	21	22.804
Method=median	0.4459	42	43.418
Method=fractionalDiffusion	0.3498	31	1712.507



**Figuur 6.36:** Eindresultaten flor.jpg met 20% Gaussische ruis via alle modellen (I)



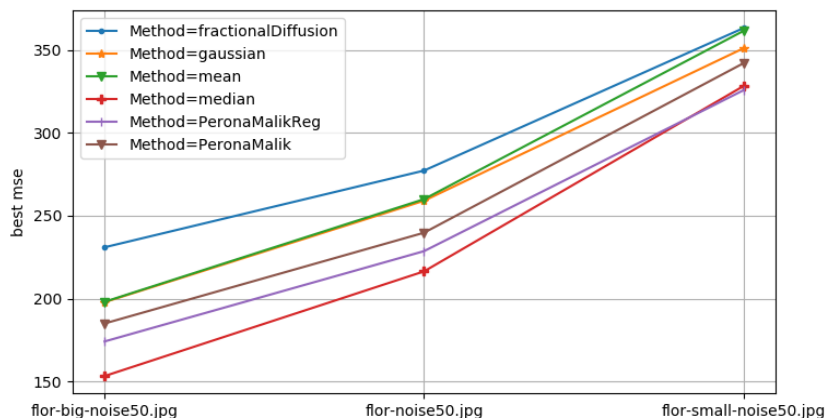
**Figuur 6.37:** Eindresultaten flor.jpg met 20% Gaussische ruis via alle modellen (II)



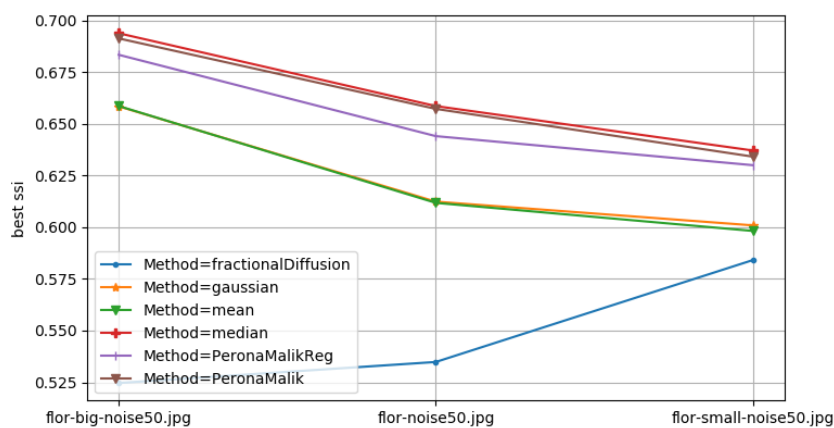
Bijkomend kan zich afgevraagd worden of het mogelijk is dat één van de geïmplementeerde methodes uitblinkt indien er een kleine of grote afbeelding gebruikt wordt? Er werd reeds vastgesteld dat met de (geregulariseerde) Perona-Malik vergelijking kleine afbeeldingen minder spectaculair hersteld kunnen worden dan de grote, meer gedetailleerde afbeeldingen. In volgend experiment wordt nagegaan of de andere methoden misschien een beter alternatief kunnen vormen.

De gebruikte afbeeldingen zijn opnieuw de kleine en grote variant van de door Gaussische ruis beschadigde “flor.jpg”. In tegenstelling tot onderdeel 6.2.3 worden deze afbeeldingen nu in kleur hersteld.

In Figuur 6.38 en Figuur 6.39 bevinden zich respectievelijk de MSE en SSI *extremum graphs*. Hieruit volgen dezelfde conclusies als hierboven. De Perona-Malik implementaties doen het beter dan fractionele diffusie en de eenvoudige methodes op basis van gemiddelden en Gaussiaans vervagen. Het gebruik van medianen blijft ook nu, ongeacht de grootte van de afbeelding, zeer goede resultaten opleveren. Een eigenaardigheid die opgemerkt wordt in de SSI *extremum graph* is hoe de resultaten bekomen via fractionele diffusie beter worden naarmate de afbeelding verkleint. Bij de andere modellen vindt het omgekeerde proces plaats; deze bereiken de globaal beste SSI-waarden bij de grootst geteste afbeelding.



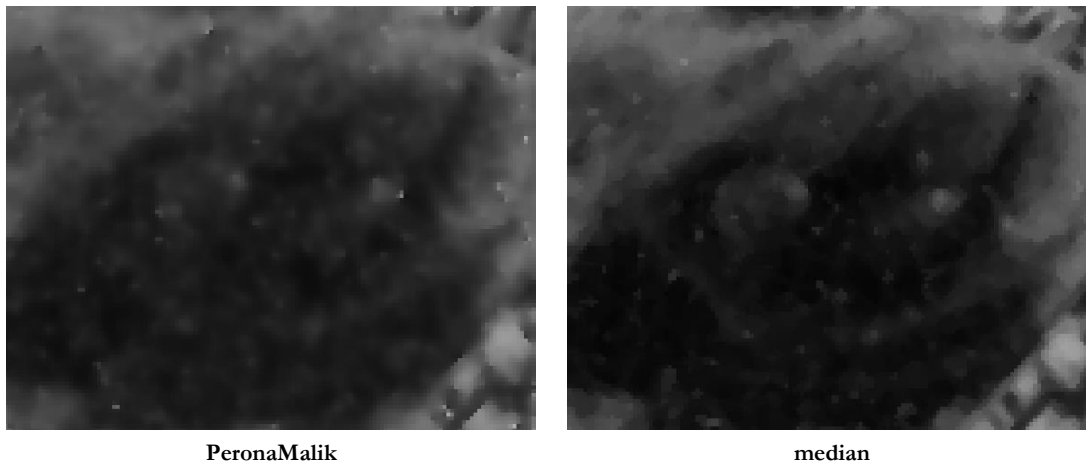
**Figuur 6.38:** De MSE-*assessment graph* van alle modellen met veranderlijke afmetingen



**Figuur 6.39:** De SSI-*assessment graph* van alle modellen met veranderlijke afmetingen

Uit voorgaande experimenten kan besloten worden dat de (geregulariseerde) Perona-Malik vergelijking superieure resultaten oplevert t.o.v. fractionele diffusie indien een afbeelding, beschadigd door Gaussische ruis, hersteld moet worden. De eenvoudige methoden op basis van gemiddelden en Gaussiaans vervagen leveren, zoals verwacht, slechtere resultaten op. Enigszins verrassend zijn de schijnbaar goede resultaten bekomen door gebruik te maken van medianen. Een verklaring voor dit fenomeen kan misschien gevonden worden in het feit dat de kleuren in de eindresultaten bekomen via de Perona-Malik implementaties visueel minder verzadigd ogen. Hiermee wordt bedoeld dat de oorspronkelijk zwarte en witte pixels in het eindresultaat een veeleer grijze kleur aannemen waardoor de afbeelding contrast verliest. Indien gebruik gemaakt wordt van medianen zal geen enkele pixel een kleur voorstellen die oorspronkelijk niet voorkwam waardoor kleuren en contrast relatief goed behouden worden. Vooral bij zwart-wit afbeeldingen kan dit fenomeen goed opgemerkt worden zoals weergegeven in Figuur 6.40.

Een tweede verklaring volgt uit een visuele waarneming van de bekomen eindresultaten en opbouw van de gebruikte afbeelding. De achtergrond van flor.jpg bestaat namelijk uit een groot, wazig oppervlak zonder details en fijne lijnen. Wanneer de bekomen eindresultaten in Figuur 6.37 vergeleken worden blijkt de methode op basis van medianen uitstekend in staat dergelijke oppervlakken te verbeteren waardoor de MSE- en SSI-waarden snel positief evolueren.

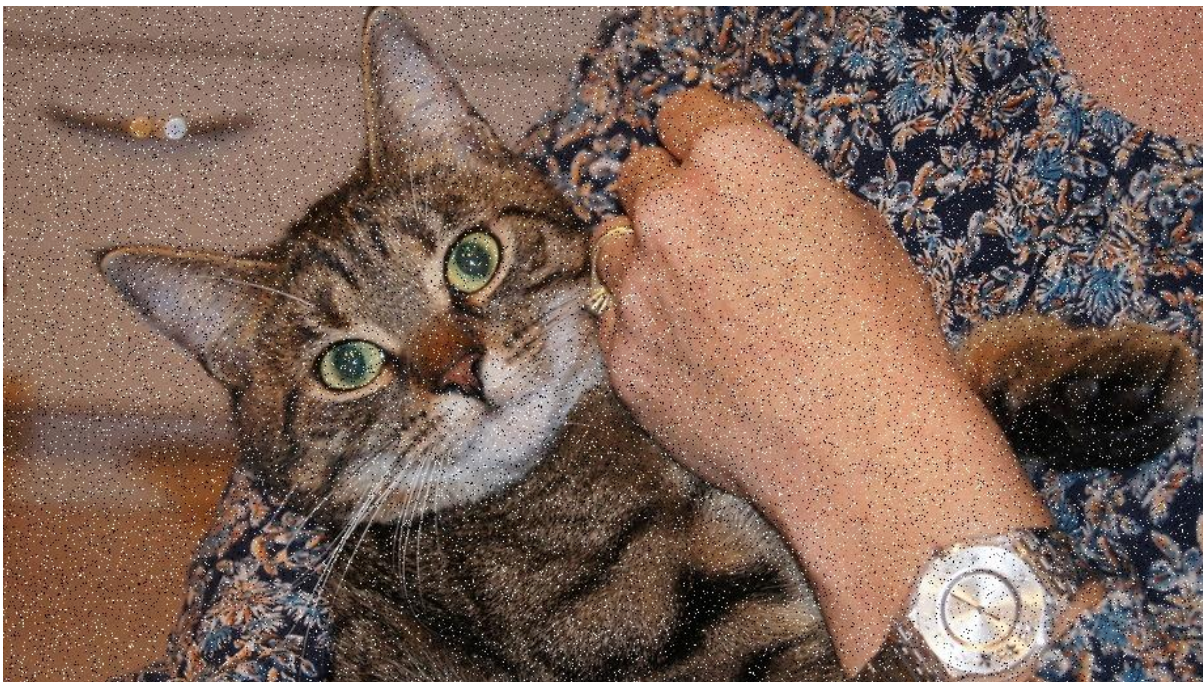


**Figuur 6.40:** Het verlies aan contrast bij Perona-Malik t.o.v. de eenvoudige methode op basis van medianen

### 6.5.2 Welke methode werkt het best bij het verwijderen van *salt-and-pepper noise*?

Tot nu toe werden in de vergelijkingen telkens afbeeldingen beschadigd door Gaussische ruis verbeterd. Hieruit bleek telkens hoe de Perona-Malik implementaties betere resultaten opleverden dan de methode op basis van fractionele diffusie. In [10] wordt echter vermeld dat de implementatie op basis van fractionele diffusie de beste resultaten oplevert bij het verwijderen van *salt-and-pepper noise*. Deze stelling wordt in onderstaande experimenten nagegaan door dergelijk beschadigde afbeeldingen te herstellen via alle geïmplementeerde modellen en vervolgens opnieuw de numerieke waarden met elkaar te vergelijken.

Allereerst wordt een aantal beschadigde afbeeldingen gegenereerd door aan de afbeelding “flor.jpg” achtereenvolgens 1, 5 en 10 procent *salt-and-pepper noise* toe te voegen. Het resultaat van deze laatste beschadiging is te bezichtigen in Figuur 6.41.



**Figuur 6.41:** flor.jpg beschadigd met 10% salt-and-pepper noise

Om optimale resultaten te bekomen gebruiken de Perona-Malik vergelijkingen een  $\alpha$ -waarde van 1 en hanteert de implementatie via fractionele diffusie een zo goed mogelijk geschatte *noiseEstimation*. De overige parameters krijgen de reeds vermelde standaard parameter waarden.

In Tabel 6.33 bevinden zich de beste MSE en SSI-waarden die met elke methode bekomen worden na het herstellen van de hierboven gegenereerde afbeeldingen. Hieruit blijkt hoe fractionele diffusie inderdaad zeer efficiënt *salt-and-pepper noise* kan verwijderen, zeker bij geringe hoeveelheid zal deze methode met voorsprong de beste resultaten opleveren. Opvallend is hoe de Perona-Malik vergelijkingen nu slechte resultaten opleveren en de eenvoudige methode op basis van medianen opnieuw goed scoort. Enkele visuele vergelijkingen tussen de bekomen eindresultaten

na het verbeteren van een afbeelding met 10% initiële ruis zijn terug te vinden in Figuur 6.42 en Figuur 6.43. Een mogelijke verklaring waarom fractionele diffusie nu wel goed presteert zit hem in het feit dat *salt-and-pepper noise* slechts een beperkt aantal pixels zal beschadigen. Zo zullen in Figuur 6.41 90% van alle pixels niet beschadigt zijn waardoor ze bijgevolg niet “verbeterd” moeten worden. Via een goed geschatte initiële hoeveelheid ruis kan de fractionele diffusie implementatie ervoor zorgen dat slechts een beperkt aantal pixels (voornamelijk ruis) daadwerkelijk aangepast worden. Aangezien de overige modellen steeds de volledige afbeelding aanpassen zullen ook de niet beschadigde pixels gewijzigd worden wat een nadelig effect heeft op de MSE en SSI-waarden.

**Tabel 6.33:** De beste MSE- en SSI-waarden voor alle modellen met 1, 5 en 10 procent salt-and-pepper noise

**1% Salt-and-pepper noise**

Variable	Best mse	iteration	time
Method=fractionalDiffusion	25.6076	7	10.593
Method=PeronaMalik	57.0926	2	10.215
Method=PeronaMalikReg	58.3319	1	7.711
Method=gaussian	70.6558	1	0.885
Method=median	73.7654	1	0.867
Method=mean	96.7725	1	0.886

Variable	Best ssi	iteration	time
Method=fractionalDiffusion	0.935	28	43.445
Method=median	0.8971	1	0.867
Method=PeronaMalik	0.8793	2	10.215
Method=PeronaMalikReg	0.876	2	15.423
Method=gaussian	0.8726	1	0.885
Method=mean	0.8458	1	0.886

**5% Salt-and-pepper noise**

Variable	Best mse	iteration	time
Method=fractionalDiffusion	53.1103	10	41.008
Method=median	80.3341	1	0.87
Method=PeronaMalikReg	125.9823	3	22.67
Method=gaussian	131.2142	1	0.883
Method=PeronaMalik	143.2664	4	21.037
Method=mean	146.0249	1	0.896

Variable	Best ssi	iteration	time
Method=median	0.8871	1	0.87
Method=fractionalDiffusion	0.8723	19	82.169
Method=PeronaMalikReg	0.7668	4	30.101
Method=PeronaMalik	0.7595	5	27.122
Method=gaussian	0.7419	2	1.767
Method=mean	0.7399	2	1.793

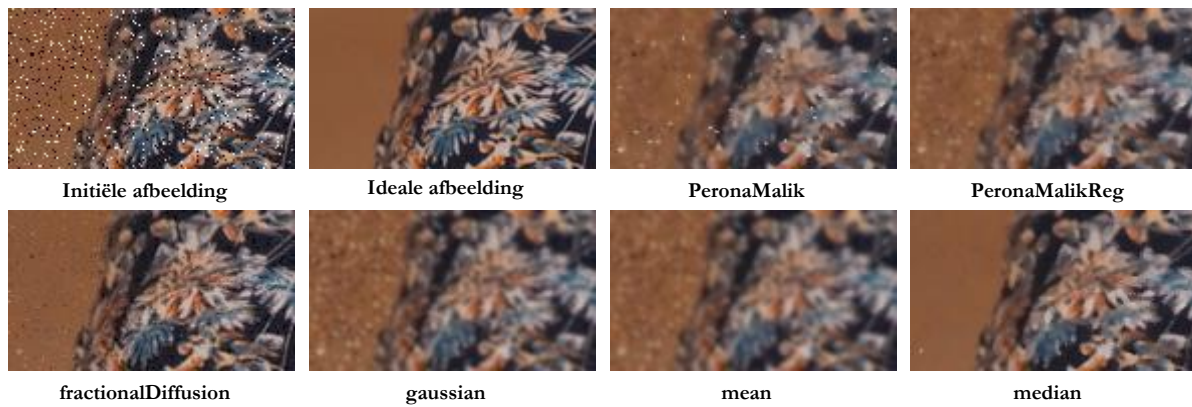
**10% Salt-and-pepper noise**

Variable	Best mse	iteration	time
Method=fractionalDiffusion	79.4844	13	89.759
Method=median	89.6748	1	0.871
Method=PeronaMalikReg	178.2793	3	23.149
Method=gaussian	189.0585	2	1.837
Method=PeronaMalik	192.3593	4	20.689
Method=mean	197.4266	2	1.745

Variable	Best ssi	iteration	time
Method=median	0.8764	1	0.871
Method=fractionalDiffusion	0.8269	21	144.305
Method=PeronaMalikReg	0.7013	4	30.627
Method=PeronaMalik	0.6991	6	31.004
Method=gaussian	0.6793	4	3.694
Method=mean	0.678	3	2.626



**Figuur 6.42:** Eindresultaten flor.jpg met 10% salt-and-pepper noise via alle modellen (I)

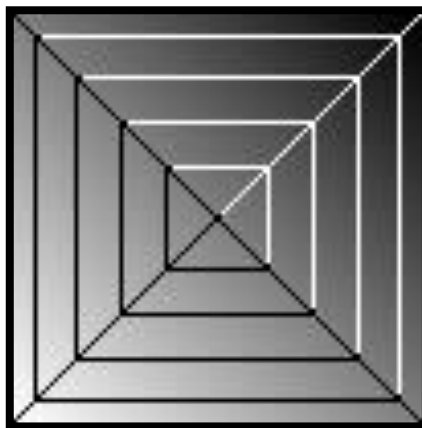


**Figuur 6.43:** Eindresultaten flor.jpg met 10% salt-and-pepper noise via alle modellen (II)

### 6.5.3 Welke methode werkt het best bij het behouden van fijne randen en details?

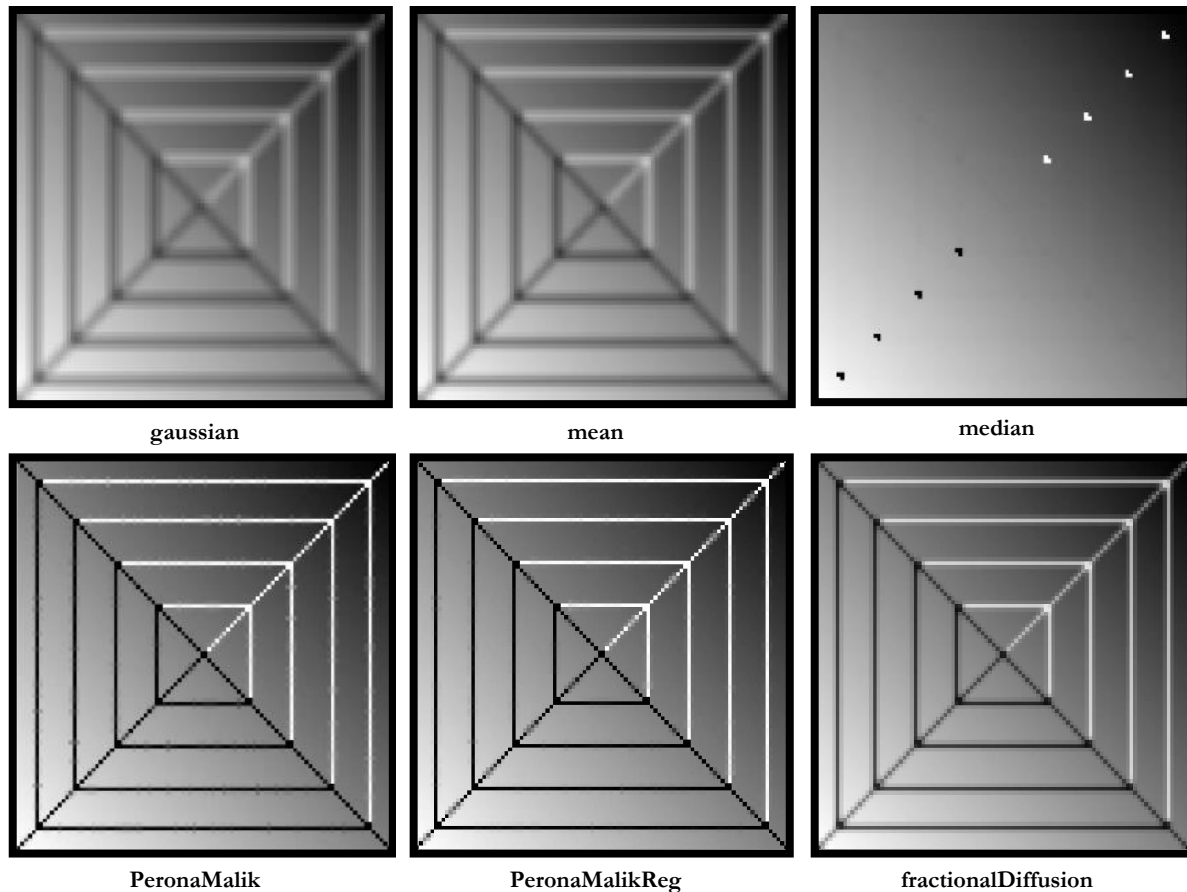
In voorgaande experimenten bleek hoe de eenvoudige methode op basis van medianen tegen de verwachtingen in goede resultaten opleverde bij het verwijderen van zowel Gaussische ruis als *salt-and-pepper noise*. Om deze reden wordt in een laatste experiment onderzocht hoe goed deze in staat is fijne randen en details te behouden ten opzichte van de meer geavanceerde methodes.

De afbeelding weergegeven in Figuur 6.44 (exclusief zwarte omkadering) werd speciaal ontworpen voor deze test en is opgebouwd uit een aantal fijne randen die samen een eenvoudig patroon voorstellen.



**Figuur 6.44:** De afbeelding lines.png gebruikt om het behoud van fijne randen en details te testen.

In tegenstelling tot alle voorgaande experimenten wordt geen ruis toegevoegd aan deze afbeelding om te vermijden dat de verschillende modellen de fijne randen foutief als ruis beschouwen. Hierdoor zullen de MSE- en SSI-waarden niet langer relevant zijn waardoor de resultaten van dit experiment louter visueel besproken zullen worden. Concreet zal elk geïmplementeerd model deze afbeelding verbeteren alsof deze beschadigd zou zijn door ruis. Aangezien dit natuurlijk niet het geval is moeten de bekomen eindresultaten zo weinig mogelijk afwijken van de oorspronkelijke afbeelding. De verschillende resultaten bekomen in de tweede iteratie volgens elke methode zijn weergegeven in Figuur 6.45.



**Figuur 6.45:** Eindresultaten lines.png na 2 iteraties via alle modellen

Uit deze resultaten blijkt meteen waarom de eenvoudige methodes in praktijk niet zo interessant zijn. Gaussiaans vervagen en uitmiddelen resulteert in wazige randen terwijl het gebruiken van medianen in sommige gevallen zelfs kan leiden tot het verwijderen ervan. Het is bijgevolg niet moeilijk om zich in te beelden dat dergelijke effecten catastrofale gevolgen kunnen hebben in medische toepassingen zoals bij het verbeteren van CT-scans.

Gelukkig worden deze fijne randen wel goed behouden bij de (geregulariseerde) Perona-Malik vergelijking en fractionele diffusie. Het moge dus duidelijk zijn dat deze gevorderde modellen, desondanks hun lange rekestijden, in praktische toepassingen waarin het behoudt van fijne randen en details een absolute noodzaak is zeker goed van pas kunnen komen.

## 7 Algemeen besluit

In een eerste fase werd een gebruiksvriendelijke console applicatie geschreven in Python. Deze stelt de eindgebruiker in staat een door ruis beschadigde afbeelding te herstellen via zes geïmplementeerde wiskundige modellen; drie eenvoudige methoden op basis van gemiddelden, Gaussiaans vervagen en medianen en drie geavanceerde technieken via de Perona-Malik vergelijking, een geregulariseerde Perona-Malik vergelijking en fractionele diffusie. Bijkomend biedt de applicatie de mogelijkheid de kwaliteit van de bekomen eindresultaten te onderzoeken aan de hand van vier objectieve beoordelingsmethoden: de *Peak signal-to-noise ratio*, de *mean squared error*, de *structural similarity index* en de vereiste rekentijd per iteratie. Via deze beoordelingsmethoden kan het programma grafieken en tabellen opstellen waarmee de bekomen eindresultaten onderling vergeleken kunnen worden. Details omtrent de implementatie en werking van het programma zijn beschreven in Hoofdstuk 4 en 5.

In een tweede fase werd de ontwikkelde applicatie gebruikt om het effect van de verschillende parameters op de bekomen eindresultaten te onderzoeken. Concreet werden een veertiental onderzoeksvragen opgesteld die zich telkens toespitsten op één veranderlijke variabele waarna een reeks experimenten werden uitgevoerd die hierop trachten een antwoord te bieden. Nadat de verschillende parameters onderzocht werden, werden de zes geïmplementeerde modellen onderling vergeleken om na te gaan welke de beste resultaten opleveren. Hieruit volgde dat de Perona-Malik vergelijkingen en de eenvoudige methode op basis van medianen de beste resultaten opleveren indien een afbeelding beschadigd door Gaussische ruis hersteld moet worden. Fractionele diffusie daarentegen levert, in tegenstelling tot de Perona-Malik vergelijkingen, dan weer goede resultaten op bij het herstellen van een door *salt-and-pepper noise* beschadigde afbeeldingen. Er kan dus gesteld worden dat de Perona-Malik vergelijkingen en de methode volgens fractionele diffusie niet zomaar vergeleken kunnen worden aangezien beide methoden gespecialiseerd zijn in het verwijderen van een ander soort ruis. Doorheen deze experimenten bleek ook hoe de eenvoudige methode op basis van medianen steeds goede resultaten opleverde. Er werd echter aangetoond dat deze niet in staat is fijne randen te behouden waardoor deze, desondanks de kortere rekentijd, niet altijd een goed alternatief vormt op de geavanceerde technieken die wel in staat zijn randen en details te behouden. Niettemin kan deze eenvoudige techniek zeker overwogen worden voor het herstellen van eenvoudige afbeeldingen waarin details minder belangrijk zijn.

In een vervolgstudie zou eventueel onderzocht kunnen worden of het mogelijk is te voorspellen in welke iteratie het beste eindresultaat bekomen wordt zonder gebruik te maken van een ideale, onbeschadigde afbeelding.



## Referenties

- [1] A. Devos, „digitale beeldverwerking met behulp van partiële differentiaalvergelijkingen,” UGent, Gent, België, 2017.
- [2] Nederlandse encyclopedie, „Digitale afbeeldingen - definitie - Encyclo,” Encyclo, 15 februari 2007. [Online]. Available: <https://www.encyclo.nl/begrip/digitale%20afbeeldingen>. [Geopend 5 mei 2019].
- [3] „Raster Data Models,” Humboldt State University, 2014. [Online]. Available: [http://gsp.humboldt.edu/olm\\_2015/Courses/GSP\\_216\\_Online/images/raster.jpg](http://gsp.humboldt.edu/olm_2015/Courses/GSP_216_Online/images/raster.jpg). [Geopend 5 mei 2019].
- [4] G. Benz, „8, 12, 14 vs 16-Bit Depth: What Do You Really Need?!” PetaPixel, 19 september 2018. [Online]. Available: <https://petapixel.com/2018/09/19/8-12-14-vs-16-bit-depth-what-do-you-really-need/>. [Geopend 6 mei 2019].
- [5] 18 juli 2008. [Online]. Available: <https://digamation.files.wordpress.com/2008/07/digamation-bit-depth1.jpg>. [Geopend 27 mei 2019].
- [6] O. Shipitko, „What does “stride” mean in image processing?,” Medium, 23 oktober 2018. [Online]. Available: <https://medium.com/@oleg.shipitko/what-does-stride-mean-in-image-processing-bba158a72bcd>. [Geopend 5 mei 2019].
- [7] P. Paalanen, „A programmer's view on digital images: the essentials,” Collabora, 16 februari 2016. [Online]. Available: <https://www.collabora.com/news-and-blog/blog/2016/02/16/a-programmers-view-on-digital-images-the-essentials/>. [Geopend 5 mei 2019].
- [8] „Image file formats,” Wikipedia the free encyclopedia, 16 april 2019. [Online]. Available: [https://en.wikipedia.org/wiki/Image\\_file\\_formats](https://en.wikipedia.org/wiki/Image_file_formats). [Geopend 6 mei 2019].
- [9] „Gaussian noise,” Wikipedia, 8 maart 2019. [Online]. Available: [https://en.wikipedia.org/wiki/Gaussian\\_noise](https://en.wikipedia.org/wiki/Gaussian_noise). [Geopend 13 mei 2019].
- [10] B. Li en W. Xie, „Image denoising and enhancement based on adaptive fractional calculus of small probability strategy,” South China University of Technology, Guangzhou 510640, China, 2015.
- [11] P. Perona en J. Malik, „Scale-space and edge detection using anisotropic diffusion,” *IEEE Transactions on pattern analysis and machine intelligence*, 1990.
- [12] M. Black, G. Sapiro, D. Marimont en D. Heeger, „Robust anisotropic diffusion,” in *IEEE Transactions on image processing*, 1998, p. 421–432.
- [13] J. Kačur, *Method of Rothe in evolution equations*, Leipzig: Teubner, 1985.
- [14] R. L. Gomes, „Is numpy really that much faster,” Rodney's Corner, 9 april 2017. [Online]. Available: <https://rlgomes.github.io/work/python/numpy/python3/2017/04/02/15.11-is-numpy-really-that-much-faster.html>. [Geopend 8 mei 2019].
- [15] „Peak signal-to-noise ratio,” Wikipedia the free encyclopedia, 14 maart 2019. [Online]. Available: [https://en.wikipedia.org/wiki/Peak\\_signal-to-noise\\_ratio](https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio). [Geopend 22 mei 2019].
- [16] A. Rosebrock, „How-To: Python Compare Two Images,” pyimagesearch, 15 september 2014. [Online]. Available: <https://www.pyimagesearch.com/2014/09/15/python-compare-two-images/>. [Geopend 18 maart 2019].
- [17] „Structural similarity,” Wikipedia the free encyclopedia, 9 mei 2019. [Online]. Available: [https://en.wikipedia.org/wiki/Structural\\_similarity](https://en.wikipedia.org/wiki/Structural_similarity). [Geopend 22 mei 2019].



