Brent Van Wynsberge
3e bachelor Informatica, Universiteit Gent
Modelleren en Simuleren
Stamnummer: 01201853

1 januari 2018

Ruiseliminatie met fft en dct

Project Modelleren en Simuleren

Inhoudsopgave

1. Afbeeldingen

Doorheen dit verslag zullen we een aantal afbeeldingen met 2 verschillende types ruis als voorbeelden gebruiken.

De gekozen afbeeldingen zijn 'lena.tiff' en 'rockefeller.tiff', beide hebben verschillende dimensies. Op die manier kunnen we analyseren of een hogere resolutie betere resultaten op zal leveren. We voeren de tests uit op hun RGB-versie en hun zwart-wit versie.

Figuur 1.1: lena.tiff (512x512)

Figuur 1.2: rockefeller.tiff (2048x2048)

Voor de ruis kiezen we voor Gaussische ruis met $\sigma = 0.1$ en zwarte verticale strepen doorheen de hele afbeelding.

Figuur 1.3: Gaussische ruis met $\sigma = 0.1$

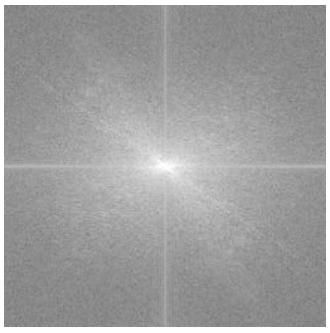
Figuur 1.4: Zwarte strepen doorheen de afbeelding

De reden waarom we deze 2 types ruis kiezen is omdat er over de geteste soorten ruis 2 soorten zijn die belangrijke verschillen hebben op de manier waarop de ruis verwijderd kan worden:

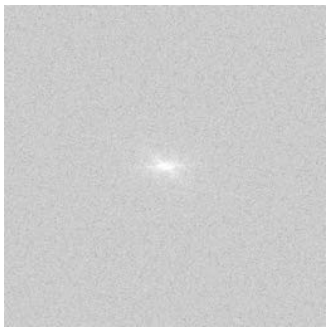
- De willekeurige ruis, waarvan de Gaussische ruis een voorbeeld is. Deze ruis kan het beste verwijderd worden door transformaties op de hele afbeelding te doen.
- De patroonruis, waarvan de strepen een voorbeeld zijn. Deze ruis treed op over een vast patroon binnen de afbeelding. Deze ruis kan het beste verwijderd door de patronen in het ruimtelijke domein te verwijderen.

De verschillen tussen de hierboven genoemde ruistypes worden nog duidelijker wanneer we de afbeelding in het ruimtelijke domein bekijken. De magnitude van het signaal berekenen we via onderstaande formule:

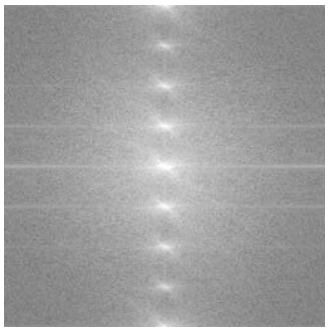
$$Magnitude(dB) = 20 * \log |Afbeelding| \tag{1.1}$$



Figuur 1.5: Origineel



Figuur 1.6: Gaussisch



Figuur 1.7: Strepen

We zien dat in het ruimtelijke domein patroonruis veel geconcentreerder zit dan willekeurige ruis. We zullen dan ook opmerken dat als we de patronen in het ruimtelijke domein kunnen wegwerken, we een groot deel van het ruispatroon ook kunnen verwijderen.

De willekeurige ruis is echter enorm verspreid in het ruimtelijke domein. In dit geval zijn er dus transformaties op de hele afbeelding nodig.

Een eigenschap die we nog opmerken is dat lagere frequenties een hogere magnitude hebben dan de hogere frequenties. Maw. de meeste informatie van de afbeeldingen zit in de lagere frequenties opgeslagen. Dit is een indicatie dat low-pass filtering veel ruis zal kunnen verwijderen zonder dat er al te veel informatie over de afbeelding verloren gaat.

2. Metrieken

Om te kunnen bepalen of de kwaliteit van de afbeelding verbeterd is na het toepassen van de filters hebben we een maatstaf nodig voor de kwaliteit van de afbeelding (tegenover zijn origineel). We hanteren onderstaande metrieken. We berekenen telkens het verschil van de gebruikte metriek voor filtering en na filtering om te meten hoeveel de afbeelding beter/slechter wordt.

2.1 PSNR

De Peak signal-to-noise ratio (PSNR)[1] is een methode om het wiskundige verschil (ruis) tussen 2 afbeeldingen te bepalen. We gebruiken dus een wiskundig model om te schatten hoe groot het verschil tussen 2 afbeeldingen is voor menselijke oog, dit zal niet altijd consistent zijn met wat we zelf waarnemen. Een hogere PSNR wil zeggen dat de afbeeldingen meer op elkaar lijken.

We berekenen de PSNR via onderstaande formule (of de ingebouwde matlabfunctie 'psnr'):

$$20 * \log(255) - 10 * \log(MSE) \tag{2.1}$$

Met MSE de 'mean squared error'.

2.2 SSIM

SSIM of de 'structural similarity index'[2] is een metriek die minder wiskundig is dan PSNR in de zin dat het gebruik maakt van perceptiemodellen om gelijkenissen tussen afbeeldingen te vinden. Het resultaat is een getal tussen -1 en 1. Hoe hoger dat getal, hoe meer de afbeeldingen op elkaar lijken. Deze index zal in de meeste gevallen een betere indicatie geven van hoe verschillend de afbeeldingen zijn voor het menselijke

[1] https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio
[2] https://en.wikipedia.org/wiki/Structural_similarity

oog dan PSNR. We gebruiken de ingebouwde matlabfunctie 'ssim' om de index te berekenen.

3. Algoritmen

3.1 Convolutiefilter

De eerste filter die we implementeren is er een die een gaussian smoothing[1] toepast. Dit komt neer op het toepassen van een low-pass filter, we zullen dus hogere frequenties niet toelaten. De hogere ruisfrequenties worden geblokkeerd waardoor de hoeveelheid ruis in de afbeelding afneemt. Echter bevinden er zich veel van de details over randen in het hogere frequentiedomein, dit zal als gevolg hebben dat naast de ruis ook de scherpheid van de afbeelding afneemt.

We berekenen het resultaat van deze filter door de 3 x 3 gaussische convolutiekern op te stellen en de convolutie met de originele afbeelding te berekenen. Er zijn echter 2 problemen die het berekenen van deze convolutie vermoeilijken. Een eerste probleem is dat het berekenen van de convolutie in het afbeeldingsdomein erg traag is. Het tweede probleem is dat de convolutiekern niet dezelfde dimensies heeft als de afbeelding.

Het eerste probleem kunnen we oplossen door FFT toe te passen op de afbeelding (omzetting naar het ruimtelijke domein) en op de kern. Nadien kunnen we het puntsgewijze product van de twee matrices berekenen, wat een relatief goedkope operatie is. Ten slotte passen we de inverse-FFT toe op het resultaat van het product. Deze matrix is de convolutie van de afbeelding met de kern, en is onze gefilterde afbeelding.

Het tweede probleem kunnen we oplossen door de kern te transformeren naar een nxn (dimensies van de afbeelding) matrix die equivalent is met de 3x3 kern nadat we er FFT op toepassen. We doen dit door de kern in een aantal blokken op te delen en de kern cyclisch te roteren tot het block dat het centrum van de kern linksboven heeft het eerste blok van de matrix is. Nadien kunnen we tussen elk blok nullen vullen tot de kern de juiste dimensies heeft.

[1] https://en.wikipedia.org/wiki/Gaussian_blur

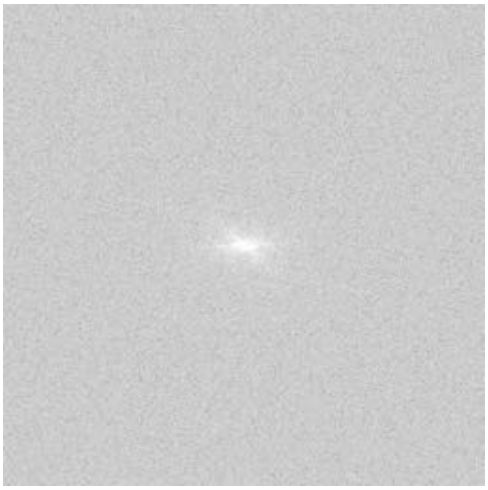
We passen de filter toe op de 2 soorten ruis.

3.1.1 Willekeurige ruis

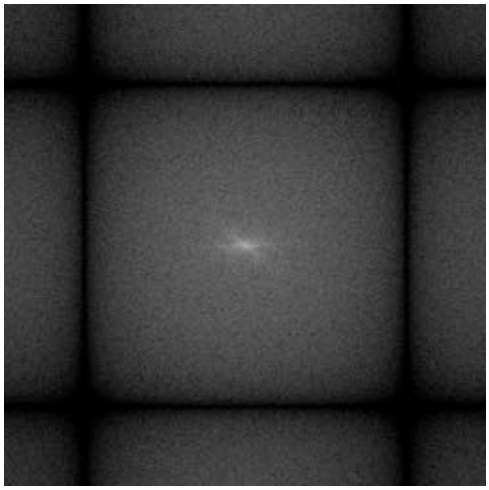
Figuur 3.1: Originele ruisafbeelding (SSIM met ruisloze versie: 0.0613)

Figuur 3.2: Gefilterde afbeelding (SSIM: 0.2697)

We merken op dat de ruis aanzienlijk afneemt. En wanneer we de SSIM-index bekijken merken we ook een verbetering van 0.2083 op.



Figuur 3.3: Ruimtelijke domein originele ruisafbeelding



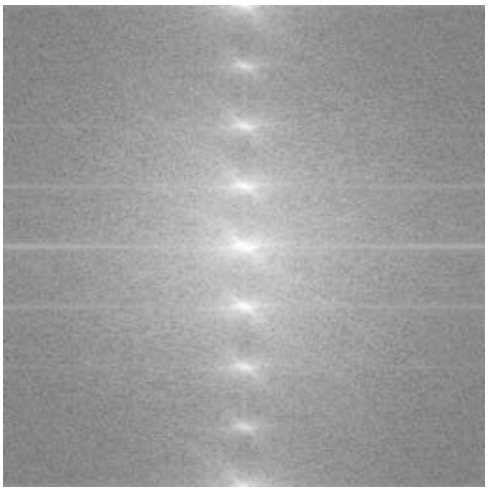
Figuur 3.4: Ruimtelijke domein gefilterde afbeelding

3.1.2 Patroonruis

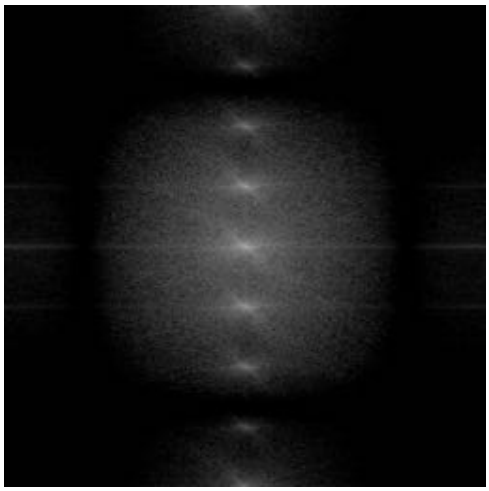
Figuur 3.5: Originele ruisafbeelding (SSIM met ruisloze versie: 0.0986)

Figuur 3.6: Gefilterde afbeelding (SSIM: 0.1171)

We merken op dat voor de patroonruis de verbetering miniem is (SSIM-verschil: 0.0185).



Figuur 3.7: Ruimtelijke domein originele ruisafbeelding



Figuur 3.8: Ruimtelijke domein gefilterde afbeelding

Wanneer we het ruimtelijke domein van de afbeelding bekijken zien we dat de filter meer van het spectrum wegfiltert, maar dat het patroon nog altijd blijft bestaan. De filter is dus niet effectief voor dit type ruis.

3.2 Spectrumfilter

Bij deze filter bekijken we het ruimtelijke spectrum van de afbeelding. We berekenen een cut-off waarde via volgende formule:

$$f_{cut-off} = \mu_{magnitude} + 1.5 * \sigma_{magnitude} \tag{3.1}$$

We zetten dus de locaties in het spectrum waar de waarde meer dan 1,5 x de standaardafwijking van het gemiddelde afwijkt op 0. Dit omdat dit vermoedelijk piekwaarden zijn. Wanneer we enkel dit zouden doen echter zou de afbeelding erg donker zijn. Dit omdat we in het centrum van de ruimtelijke voorstelling van de afbeelding pieken kunnen verwachten die de details van de afbeelding bevatten, maar deze worden op 0 gezet. Daarom bakenen we rond het centrum van het spectrum een gebied van n/10 pixels af waar we de waarden niet op 0 zetten. Dit geeft voldoende ruimte om de te verwachten pieken te laten voorkomen.

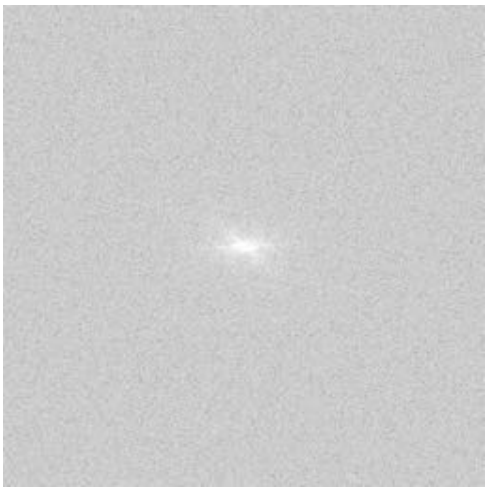
We bekijken nu de resultaten voor deze filter.

3.2.1 Willekeurige ruis

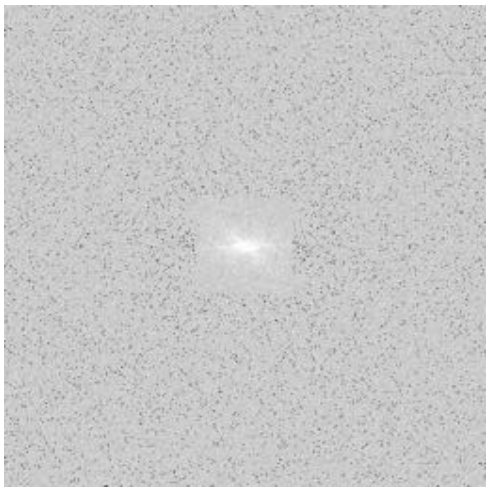
Figuur 3.9: Originele afbeelding (SSIM: 0.0613)

Figuur 3.10: Gefilterde afbeelding (SSIM: 0.0649)

Wanneer we de afbeelding bekijken is het bijna onmogelijk om het verschil te zien. Het verschil tussen beide SSIM-indexen is erg klein (0.0035).



Figuur 3.11: Ruimtelijke domein afbeelding



Figuur 3.12: Ruimtelijkde domein gefilterde afbeelding

We zien dat in het ruimtelijke domein er weinig punten op 0 gezet zijn, dit omdat het gemiddelde van deze afbeelding al relatief hoog is door de willekeurige ruis. Zoals verwacht werkt dit algoritme dus niet erg goed voor afbeeldingen met willekeurige ruis.

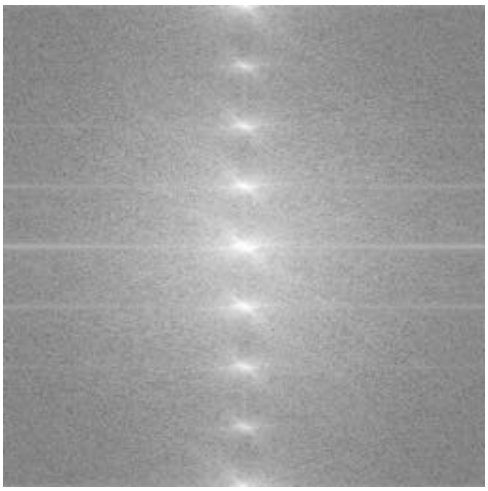
3.2.2 Patroonruis

Figuur 3.13: Originele afbeelding (SSIM: 0.0986)

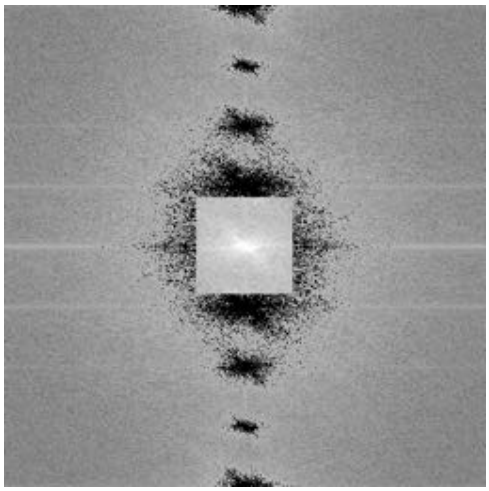
Figuur 3.14: Gefilterde afbeelding (SSIM: 0.6684)

Bij patroonruis merken we een aanzienlijke verbetering op. De lijnen zijn grotendeels vervaagd en bijna onzichtbaar. Ook volgende ze nu een stuk natuurlijke lijnen in de

afbeelding. Het verschil tussen de SSIM-indexen is 0.5698.



Figuur 3.15: Ruimtelijke domein afbeelding



Figuur 3.16: Ruimtelijkde domein gefilterde afbeelding

Wanneer we het ruimtelijke domein bekijken zien we dat de patronen weggefilterd worden. We kunnen nu ook het afbakeningsgebied in het centrum zien.

Dit algoritme werkt het beste om patroonruis weg te krijgen.

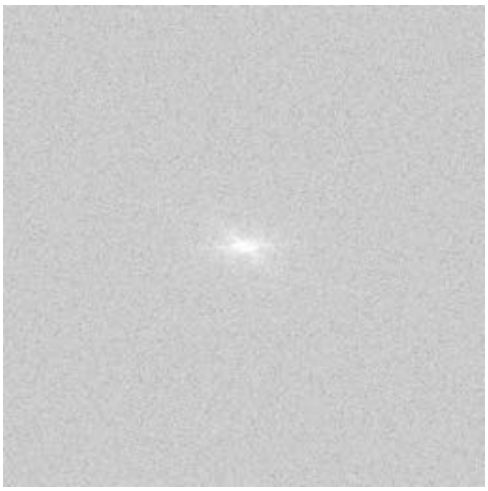
3.3 Gecombineerde filter

Bij dit algoritme passen we de 2 bovenstaande filters na elkaar toe om zowel patroon- als willekeurige ruis weg te proberen werken. De volgorde waarin we we de filters toepassen heeft een (verwaarloosbaar) kleine invloed op het resultaat. Hier passen we eerst de spectrumfilter en nadien de convolutiefilter toe.

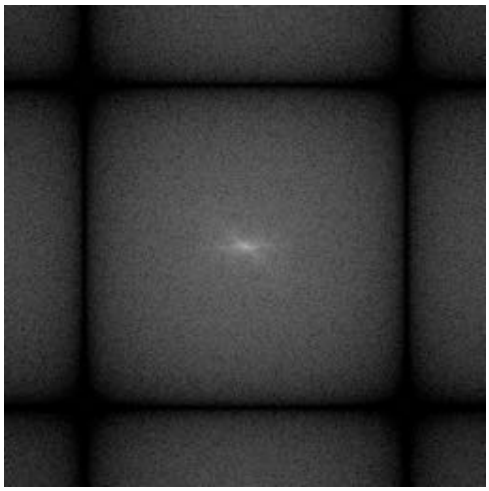
3.3.1 Willekuerige ruis

Figuur 3.17: Originele afbeelding (SSIM: 0.0613)

Figuur 3.18: Gefilterde afbeelding (SSIM: 0.2817)



Figuur 3.19: Ruimtelijke domein afbeelding



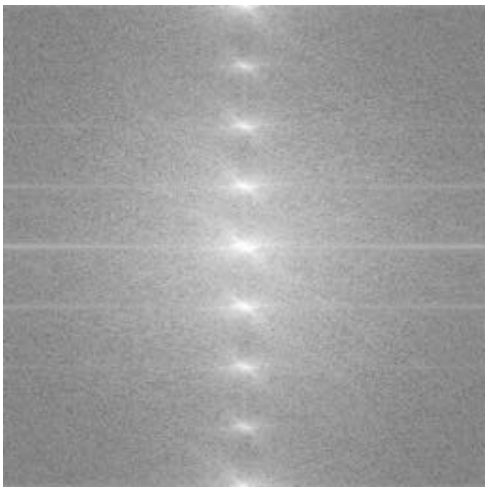
Figuur 3.20: Ruimtelijkde domein gefilterde afbeelding

De afbeeldingen hebben een SSIM-verschil van 0.2204.

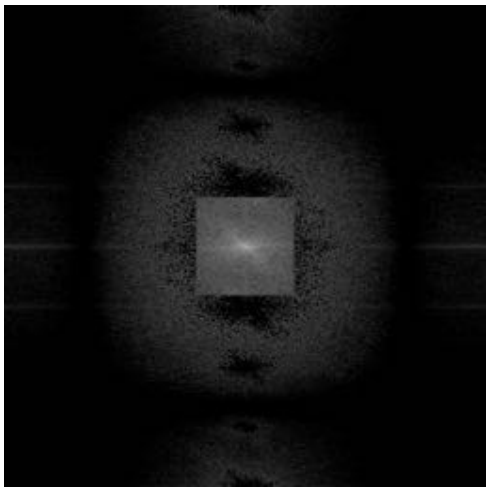
3.3.2 Patroonruis

Figuur 3.21: Originele afbeelding (SSIM: 0.0986)

Figuur 3.22: Gefilterde afbeelding (SSIM: 0.6715)



Figuur 3.23: Ruimtelijke domein afbeelding



Figuur 3.24: Ruimtelijkde domein gefilterde afbeelding

Deze afbeeldingen hebben een SSIM-verschil van 0.5729.

3.3.3 Conclusie

Niet alleen werkt deze filter goed voor zowel patroon- als willekeurige ruis, ook is er een iets grotere verbetering dan wanneer we de beste voorgaande filter voor het type ruist toepasten.

3.4 Wiener filter

We gebruiken de ingebouwde wiener filter van matlab om een maatstaf te hebben om de geïmplementeerde filters mee te vergelijken.

Figuur 3.25: Originele afbeelding (SSIM: 0.0613)

Figuur 3.26: Gefilterde afbeelding (SSIM: 0.3731)

SSIM-verschil: 0.3118

Figuur 3.27: Originele afbeelding (SSIM: 0.0986)

Figuur 3.28: Gefilterde afbeelding (SSIM: 0.1589)

SSIM-verschil: 0.0602

3.5 1D FFT

In dit onderdeel zetten we de matrix om naar een rij van de rijen van de originele afbeelding en een rij van de kolommen. Nadien passen we hier 1D-FFT op toe, daarna zetten we het terug om in een nxn matrix en passen we inverse 2D-FFT toe.

Figuur 3.29: Originele afbeelding

Figuur 3.30: 1D FFT op de rij van kolommen

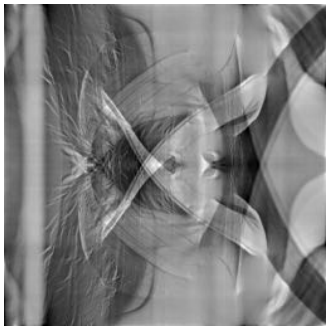
Figuur 3.31: 1D FFT op de rij van rijen

Wanneer we 2D FFT toepassen komt dit neer op een reeks van geneste sommen, of dus een samenstelling van een aantal 1D FFT's. Wanneer we de matrix omzetten naar een rij van rijen of kolommen en er slechts 1x 1D FFT op toepassen is dit niet equivalent met de 2D FFT. Er is namelijk informatie die verloren gaat wanneer er naar een nieuwe rij/kolom overgegaan wordt.

3.6 1D DCT

Nu doen we hetzelfde als hierboven maar voor de DCT.

Figuur 3.32: Originele afbeelding



Figuur 3.33: 1D DCT op de rij van kolommen

Figuur 3.34: 1D DCT op de rij van rijen

Hier merken we op dat de afbeeldingen voor de helft op de x/y-as gespiegeld worden. Dit komt door het feit dat 1D-DCT het signaal omzet in een cosinusoïde. Er zullen dus positieve en negatieve waarden zijn die elkaar opvolgen. Wanneer we de matrix opnieuw opbouwen en inverse-2D-DCT toepassen is de helft van de matrix omgewisseld van kant op de as waarvan de rij gemaakt werd (x-as: rij van rijen) door het feit dat die waarden negatief waren terwijl ze niet negatief zouden geweest zijn als de 2D-DCT toegepast zou worden.

3.7 Gekleurde afbeeldingen

We voeren nu de filters uit op elke laag van de kleurafbeeldingen en vergelijken de resultaten met de resultaten van de zwart-wit afbeeldingen.

3.7.1 Willekeurige ruis

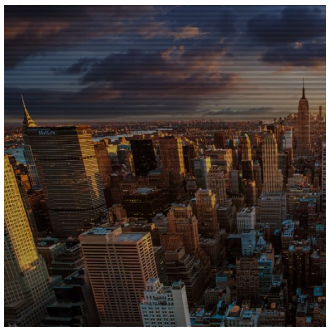
Figuur 3.35: Gecombineerde filter (SSIM-verschil-kleur: 0.2765 - SSIM-verschil-zwartwit: 0.1793)

Figuur 3.36: Convolutiefilter ((SSIM-verschil-kleur: 0.2824 - SSIM-verschil-zwartwit: 0.1912)

Figuur 3.37: Spectrumfilter (SSIM-verschil-kleur: 0.0005 - SSIM-verschil-zwartwit: -0.0048)

3.7.2 Patroonruis

Figuur 3.38: Gecombineerde filter (SSIM-verschil-kleur: 0.2867 - SSIM-verschil-zwartwit: 0.2851)



Figuur 3.39: Convolutiefilter (SSIM-verschil-kleur: 0.0353 - SSIM-verschil-zwartwit: 0.0001)

Figuur 3.40: Spectrumfilter (SSIM-verschil-kleur: 0.2918 - SSIM-verschil-zwartwit: 0.2832)

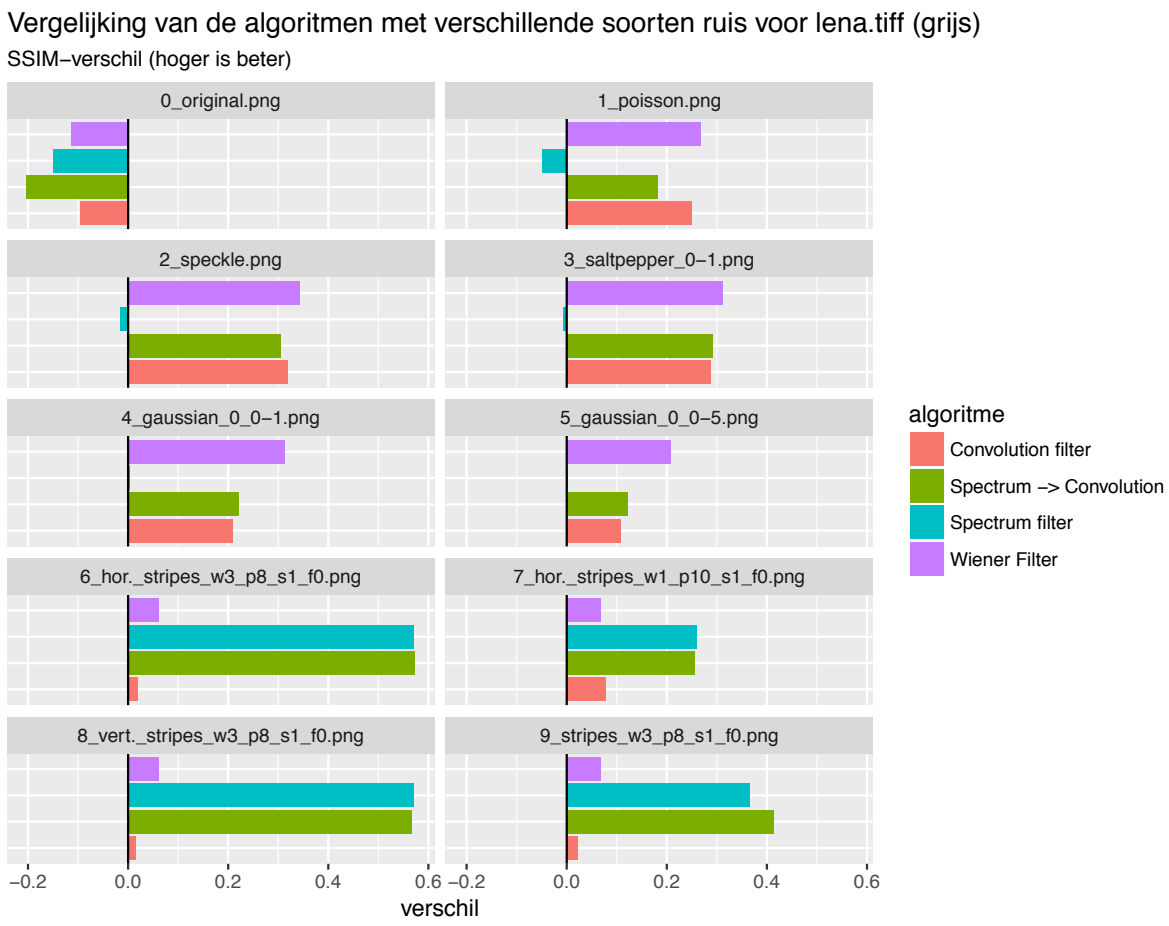
3.7.3 Conclusie

Wanneer er ruis geïntroduceerd wordt in een afbeelding met meerdere lagen (RGB) zal de ruisdata meer verspreid liggen, ook is er meer informatie over de afbeelding beschikbaar. Indien we op elke laag afzonderlijk onze filter toepassen zal het resultaat dus een kleine verbetering opleveren tegenover het filteren op 1 laag (zwartwit).

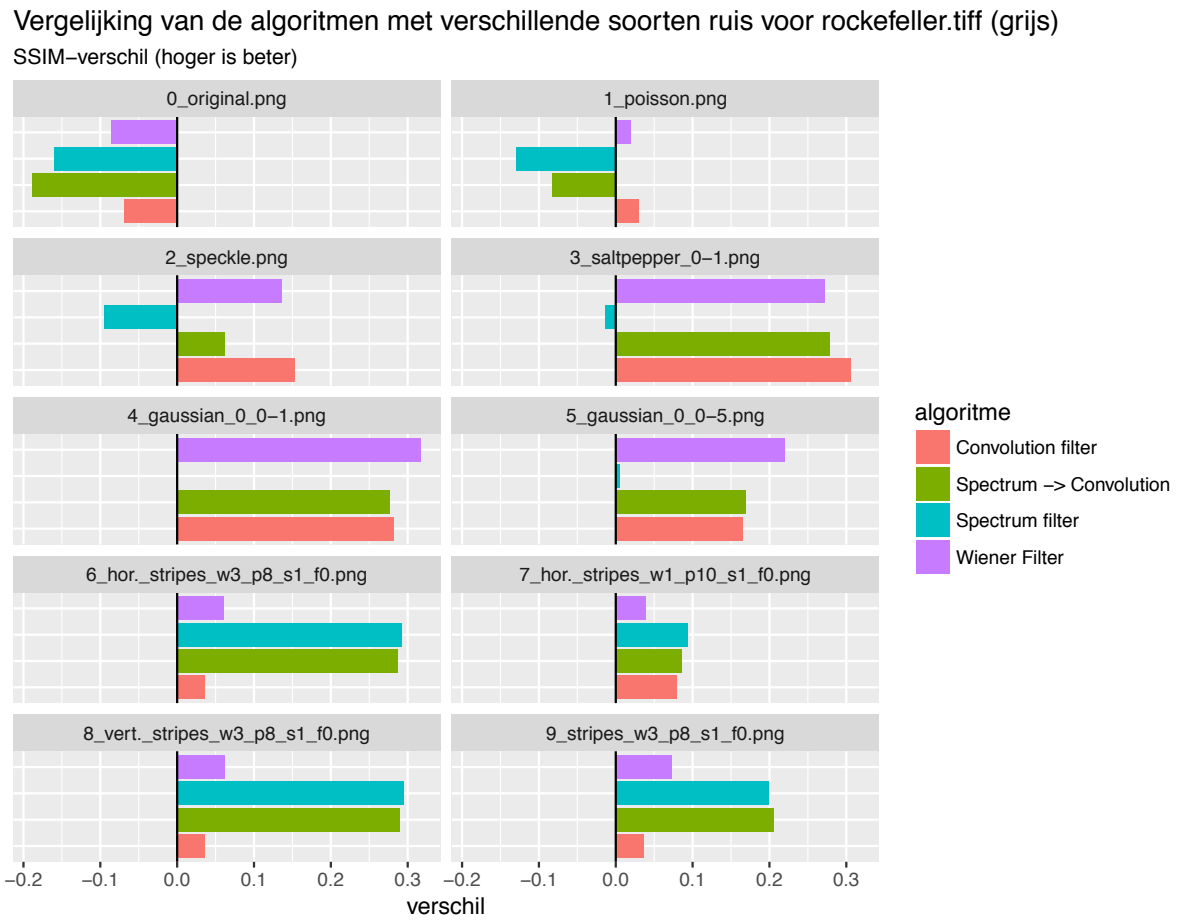
4. Vergelijking algoritmen

We maken een aantal ruis-afbeeldingen van de zwartwit versie van lena en rockefeller nadien passen we telkens elke filter toe op de afbeelding en vergelijken we de SSIM-indexen.

4.1 Lena



4.2 Rockefeller



4.3 Conclusie

In het geval waar er relatief weinig ruis in de afbeelding zit (origineel, poisson, speckle, saltpepper) scoort de convolutiefilter beter dan de gecombineerde filter. In de andere gevallen scoort de gecombineerde filter echter beter. De Wiener filter scoort beter dan de geïmplementeerde algortimen tenzij het om patroonruis gaat.

De gecombineerde filter geniet de voorkeur in gevallen waar de ruis groot genoeg is en uniform verdeeld is. In de meeste gevallen zal dit dus de beste filter zijn.