Labo 2 Beeldverwerking 2014
Filtering

David Van Hamme

12 februari 2014

1 Filtering

In dit labo bekijken we enkele courante filteroperaties. In beeldverwerking wordt filtering vooral gebruikt om:

- beelden te vervagen (blurren);
- beelden te verscherpen;
- ruis te verwijderen;
- randen te extraheren.

Filtering komt erop neer dat een venster of *kernel* over de afbeelding geschoven wordt en telkens een pixel vervangen wordt door een waarde berekend uit het volledige venster. Bij lineaire filters is de kernel een matrix van wegingsfactoren waarmee de omliggende pixels vermenigvuldigd worden alvorens ze te sommeren in de centrale pixel.

2 Opgave 2

Schrijf een programma dat een PNG afbeelding inleest (bestandsnaam via commandolijn) en deze vervaagt met een Gaussiaans filter. Dit filter vervangt elke pixel door een gewogen gemiddelde van de omliggende pixels. De wegingsfactoren zijn bepaald door een 2D normale verdeling rondom de centrale pixel, dus nabijgelegen pixels hebben meer invloed dan iets verder gelegen pixels. Dit soort filter wordt vaak gebruikt om *witte ruis* uit het beeld te verwijderen. Witte ruis is een vorm van ruis waarbij elke pixel een willekeurige afwijking ondergaan is van zijn originele waarde. Test je programma op **whitenoise.png**. Toon de originele afbeelding en de vervaagde versie naast elkaar in twee aparte vensters. Dien je bronbestand in op minerva als **voornaam_achternaam_opgave2.cpp**. Nieuwe functies die je nodig hebt: **GaussianBlur**.

3 Opgave 3

Schrijf een programma dat een PNG afbeelding inleest (bestandsnaam via commandolijn) en hierop *unsharp masking* toepast. Dit is een techniek om een afbeelding te verscherpen en houdt het volgende in:

- vervaag de afbeelding;
- bepaal het (absolute) verschil tussen de originele en de vervaagde afbeelding;
- tel dit verschil op bij de originele afbeelding.

Let erop dat je geen overflow of saturatie krijg in je datatype! Gebruik gepaste schalingsfactoren. Test je programma op **unsharp.png**. Toon het originele beeld en de verscherpte versie naast elkaar. Dien je bronbestand in op minerva als **voornaam_achternaam_opgave3.cpp**.

4 Opgave 4

Schrijf een programma dat een PNG afbeelding inleest (bestandsnaam via commandolijn) en hieruit de *salt and pepper noise* verwijdert met een mediaanfilter. Salt and pepper noise is een vorm van ruis waarbij sommige pixels zwart of wit gekleurd word en. Het mediaanfilter vervangt elke pixel door de mediaan van de omliggende. Hierdoor hebben de uitschieters geen invloed meer, in tegenstelling tot een Gaussiaans filter, waar de invloed enkel verminderd wordt. Test je programma op **saltandpeppernoise.png**. Toon het originele beeld en de gefilterde versie naast elkaar. Dien je bronbestand in als **voornaam_achternaam_opgave4.cpp**. Nieuwe functies die je nodig hebt: **medianBlur**.

5 Opgave 5

Schrijf een programma dat een PNG afbeelding inleest (bestandsnaam via commandolijn), omzet in grijswaarden en hiervan de horizontale eerste afgeleide berekent. Dit doe je door te filteren met een horizontaal Sobel kernel:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Dit geeft je een maat voor de horizontale verandering in een afbeelding. Test je programma op **building.png**. Dien je bronbestand in zoals gewoonlijk. Nieuwe functies: **Sobel**.

6 Opgave 6

Schrijf een programma dat een PNG afbeelding inleest (bestandsnaam via commandolijn) en deze filtert met een 15x15 kernel waarvan de eerste 7 diagonaalelementen de waarde 1/7 hebben, alle andere zijn nul.

$$1/7 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Zorg dat het ankerpunt van de kernel (de pixel waarin de berekende waarde terecht komt) rechts onderaan ligt in de kernel in plaats van in het midden zoals gebruikelijk. Wat verwacht je dat dit filter zal doen? Test je programma op **blots.png**. Toon het origineel en het gefilterde beeld naast elkaar. Dien je bronbestand in. Nieuwe functies: **filter2D**.