


Labo Beeldverwerking - Inleiding beeldverwerking in Matlab

Kristof Teelen

Labo beeldverwerking  2008-09

In het eerste labo Multimedia wordt een inleiding tot Matlab gegeven. De focus ligt op de voorstelling van digitale beelden binnen deze programmeeromgeving en op enkele basisbewerkingen. Vooral de image processing toolbox van Matlab is een handig hulpmiddel. Ook de Matlab Help is zeer uitgebreid en kan voor elke functie opgeroepen worden als **help functienaam** in de Matlab workspace of bekeken worden onder het tabblad Matlab Help.



Figuur 1: cameraman en lena.color

1 Beeldverwerking in Matlab

Bovenstaande beelden tonen de cameraman en Lena, respectievelijk een grijswaarden- en een kleurenbeeld; sla ze op in de lokale Matlab\work directory. Een beeld kan geladen worden met `imread(filenaam)` en eventueel toegekend worden aan een variabele. Informatie over de file kan gevraagd worden met `imfinfo filenaam`. Een beeld wordt in Matlab voorgesteld als een matrix van pixelwaarden, met de oorsprong linksboven. Beelden die opgeslagen zijn in een variabele kunnen getoond worden met `imshow(variabele)`; deze functie werkt ook met als argument rechtstreeks een filenaam. `help imshow` toont u alle mogelijke opties. De Matlab Help is zeer uitgebreid en kan altijd opgeroepen worden als **help functienaam** in de Matlab workspace of bekeken worden onder het tabblad Matlab Help.

1.1 Conversie tussen verschillende types beelden

Wanneer een grijswaardenbeeld ingelezen wordt, wordt dat voorgesteld als een 2D-matrix met pixelwaarden van het type UINT8 (unsigned integer byte), waarmee voor elke pixel een intensiteitswaarde tussen 0 en 255 voorgesteld kan worden. De representatie van kleurenbeelden verschilt van grijswaardenbeelden, het zijn 3D-matrices waarin 3 kleurlagen voorkomen: Rood, Groen en Blauw, en worden daarom ook RGB-beelden genoemd. De kenmerken van variabelen in Matlab kan u bekijken met `whos`. Binnen het datatype UINT8 geven rekenkundige bewerkingen niet noodzakelijk het verwachte resultaat. Waarom niet? De functie `im2double` biedt hier een oplossing. Soortgelijke functies zijn: `im2uint8`, `im2uint16`, `mat2gray`, `im2double`, `im2bw`. Beelden kunnen terug als een file opgeslagen worden met `imwrite`. Voor bepaalde filetypes kan de compressiefactor als een parameter meegegeven worden. Probeer *lena_color.jpg* op te slaan als een grijswaardenbeeld (`rgb2gray`) in gecomprimeerd jpg-formaat.

```
im = imread('cameraman.jpg');
info = imfinfo('cameraman.jpg')
im(1:20,1:20)
% deze indexering in de matrix van het beeld geeft een deel van het
% beeld weer zoals het in matlab gebruikt wordt: een matrix van
% pixelintensiteiten op elke (x,y)-positie
figure, % dit opent een nieuw venster in windows
imshow(im)
im2 = imread('lena_color.jpg');
info = imfinfo('lena_color.jpg')
im2(1:20,1:20,:) % een kleurenbeeld is een 3D matrix met 3 lagen van pixelintensiteiten die respectievelijk
figure, imshow(im2)
whos % dit toont alle variabelen in de workspace, hun grootte en type
im3 = rgb2gray(im2);
imwrite(im3,'lena_gray.jpg','Quality',75)
```

1.2 Algemene Matlab functies

Bestudeer de uitleg bij volgende functies door middel van de help functie, en test ze eventueel uit.

```
figure    close    plot    input    ginput
size      ndims    ones    zeros    magic    rand
max       min      transpose
```

Matlab kent ook structuren zoals selectie en iteratie, en is daarmee een volwaardige programmeertaal:

```
if else elseif for
while break continue
switch return
```

Ook voor bewerkingen op strings voorziet Matlab de nodige functies:

```
disp input str2num class strread
strcmp error lower upper
```

Voor de basis van het programmeren in Matlab kan je best even de gegeven tutorial doorlezen. Je kan ook veel bijleren door de helppagina's over de image processing toolbox te lezen.

1.3 Het programmeren van m-functies

In Matlab kunt u een sequentie van opdrachten schrijven als een `filenaam.m` file. Deze file vormt dan een nieuwe functie binnen Matlab, die aangeroepen wordt als `filenaam` in de workspace. Parameters doorgeven naar een m-functie gebeurt in de regel voor de functie-definitie: `function [outputs] = name(inputs)`. Door `edit` op te roepen in de workspace opent de Matlab Editor, waarin de nodige code kan geschreven worden.

1.4 Nuttige tips betreffende de uitvoeringstijd in Matlab

Matlab is een programmeertaal die specifiek op het verwerken van matrices gericht is. Door hiervan gebruik te maken als het kan, kunnen we de verwerkingssnelheid verbeteren.

- Wanneer door een declaratie de uiteindelijke grootte van een variabele op voorhand kan vastgelegd worden, werkt Matlab veel sneller. Probeer: (de functies `tic` en `toc` dienen hier enkel om de uitvoeringstijd te meten)

```
clear f; tic; for x=1:10000; f(x)=sin(x/(2*pi)); end; toc
```

En vergelijk met het resultaat van de volgende regel:

```
clear f; f=zeros(1,10000); tic; for x=1:10000; f(x)=sin(x/(2*pi)); end; toc
```

De tweede bewerking doet hetzelfde als de eerste, maar is meer dan 3x sneller. Waarom? Door `f` te initialiseren, wordt er genoeg geheugenruimte voorzien om de resultaten uit te schrijven naar de variabele `f`.

- For-lussen zijn veel trager dan bewerkingen op vectoren of matrices.

```
clear f; tic; x=1:10000; f=sin(x); toc
```

Deze bewerking geeft een derde maal hetzelfde resultaat, maar wordt in vergelijking met de eerste maar liefst 12x sneller uitgevoerd. Het is dus nuttig om for-lussen, indien mogelijk, te elimineren door ze om te zetten naar een bewerking met vectoren of matrices. Voor meerdimensionale vectoren is de functie `meshgrid` handig. `help meshgrid` geeft u een duidelijk voorbeeld.

2 Opdrachten

1. Schrijf een m-functie 'change', die opgeroepen wordt als: `variabele = change('filenaam1.jpg', 'filenaam2.jpg');`. De variabele moet het verschil weergeven tussen de volgende twee beelden, geef dat verschil terug als een beeld met de pixelwaarden als `UINT8` type. Toon eerst een beeld waarin het verschil berekend wordt door de beelden gewoon van elkaar af te trekken. Hou er rekening mee dat het resultaat van een rekenkundige bewerking niet noodzakelijk binnen het 8-bit bereik valt. Hoe kan dit opgelost worden?



Figuur 2: watch1.jpg en watch2.jpg

Maak een tweede implementatie die test of pixels op overeenkomstige posities dezelfde intensiteitswaarde hebben, en een logische matrix genereerd, waarin de 1 aanduidt dat er op die positie verandering was, en de 0 dezelfde intensiteit aangeeft. Toon dit beeld.

Waarom werkt deze methode niet? Dat is het gevolg van ruis die tijdens het opnameproces in het beeld geïntroduceert wordt. Dat is onder meer een gevolg van meetfouten tijdens het opnameproces, veranderlijk lichtinval, ... Hoe kan je dit in deze methode oplossen?

2. Schrijf een m-functie `regionofinterest = selectregionofinterest('filenaam.jpg')`; die een stuk van een grijswaarden- of een kleurenbeeld selecteert, teruggeeft als variabele en toont op het scherm. Schrijf twee versies: `n` waarbij u de pixelcoördinaten van de linkerbovenhoek en de rechteronderhoek van de selectie meegeeft als variabele en `n` waarbij u deze punten kan aanduiden met 2 muisklikken in het beeld getoond door Matlab.
3. Schrijf een m-functie `puzzle = image2puzzle('filenaam.jpg')`; die van een beeld een schuifpuzzel maakt. Deel het beeld op in een aantal vierkante puzzelstukken van een gelijke grootte. Toon dan een figuur waarin er 1 willekeurig stuk ontbreekt, en de andere stukken willekeurig door elkaar geschoven zijn. Maak gebruik van de functie `rand`. Als je wil, mag je ook de nodige code schrijven om deze puzzel interactief (door muisklikken op het te verschuiven puzzelstuk) op te lossen, maar dit is zeker niet verplicht.
4. Schrijf een m-functie `logicimage = createlogicimage('filenaam.jpg')`; die voor het onderstaande beeld enkel de botten teruggeeft. *logicimage* moet een zwart-witbeeld zijn (zwart = 0, wit = 1) waarin enkel de botten als wit worden weergegeven. Beschrijf in enkele regels commentaar uw idee over hoe u automatisch de hoge intensiteiten van de lagere zou kunnen scheiden voor dit soort X-ray-beelden. Bekijk eerst het histogram voor dit beeld met behulp van (`imhist`). Het histogram toont een grafiek waarin voor elke intensiteitwaarde het aantal pixels met die intensiteit weergegeven wordt.

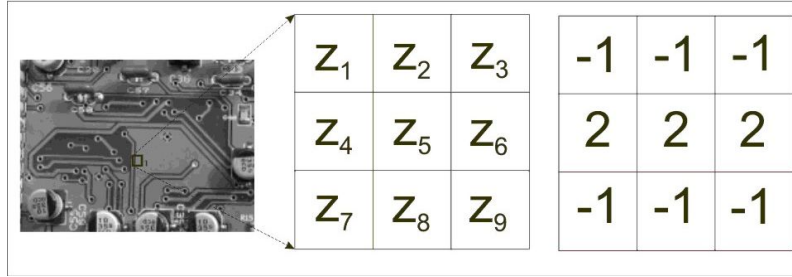


Figuur 3: Xraywrist.gif

3 Spatiale filters

In beeldverwerking gebeurt de spatiale filtering door convolutie van het beeld met een masker. Het masker glijdt over het beeld en in elke pixel die overeenkomt met het centrum van het masker wordt het resultaat van de bewerking bijgehouden. Bij lineaire filters wordt de som van de producten van de filtercoëfficiënten met de pixelwaarden berekend.

Er zijn verschillende types lineaire filters, die elk hun functie hebben. Hieronder geven we enkele voorbeelden voor een smoothing operatie (wat ook voor ruisonderdrukking zorgt). Eerst met een averaging filter :



Figuur 4: Illustratie van de berekening van de respons van een 3x3 filter voor één beeldpixel. Voor deze ene pixel (in het voorgestelde geval is dat de centrale pixel in het venster, dus die met intensiteit z_5) is de informatie van alle pixels in een 3x3 regio rond die pixel in het beeld nodig, dus $z_1 - z_9$. De filterrespons kan dan berekend worden als $\sum_{i=1}^9 z_i * w_i$, met w_i de gewichten van de filter en z_i de intensiteitswaarden (waarbij z_5 overeenkomt met de centerpixel). Het resultaat van deze sommatie wordt toegewezen aan de centerpixel. Herhaal deze operatie voor elke pixels in het beeld.

```
afilt = fspecial('average',5)
g1 = imfilter(im,afilt,'replicate');
figure, imshow(g1,[]); % De vierkante haakjes staan er om het intensiteitsbereik te mappen op het volledige beeld
```

Een ander filter is een gaussiaan (ook gebruikt voor smoothing en ruisonderdrukking), van een bepaalde grootte en standaarddeviatie (wat betekenen deze variabelen voor de gaussiaan?)

```
gfilt = fspecial('gaussian',[3,3],0.5)
g2 = imfilter(x,gfilt,'replicate');
figure, imshow(g2, [] );
```

De parameters van deze en andere types kan je bekijken in de help van **fspecial**. Er bestaan ook niet-lineaire filters, zoals de orde-filters (**ordfilt2**, die je kan gebruiken in toepassingen als mediaanfiltering tegen salt&pepper-ruis. Gebruik deze filters om zelf de door ruis gecorrumpeerde beelden te verbeteren (*lena_gaussn.jpg* en *lena_spn.jpg*).

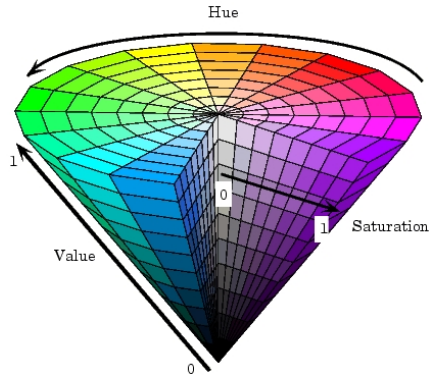
4 Bewerkingen op kleurenbeelden

Kleurenbeelden in een RGB-formaat bestaan uit drie lagen die met een bepaalde bitdiepte (meestal 8 per kleur laag) de rode, de groene en de blauwe component voorstellen. Er bestaan ook enkele andere representaties voor kleurenbeelden, waarvan er hier enkele kort zullen voorgesteld worden.

Geïndexeerde beelden hebben twee componenten: een datamatrix met integers die gebruikt worden als pointers naar de waarden van de kleur die opgeslagen zitten in een mx3 kleurenmatrix. De lengte m van de kleurenmatrix definieert het aantal kleuren van het beeld. Elke rij van de kleurenmatrix specificeert de rood-, groen- en blauwcomponent van een kleur. Door een directe mapping van de datamatrix op de kleurenmatrix wordt aan elke pixel een kleur toegewezen. In Matlab zijn enkele standaardkleurenmaps beschikbaar, kijk eens naar de functie **colormap**.

Een andere kleurenruimte is het NTSC-formaat: daarin wordt de informatie opgeslagen in drie componenten: Y (luminantie), I (hue) en Q (saturatie). De Y-component slaat alle grijswaarden-informatie op, terwijl de kleur vervat zit in I en Q. Het grote voordeel is dat dit signaal dan zowel voor monochrome als voor kleurentelevisies gebruikt kan worden. De conversie naar NTSC vanuit RGB gebeurt door een simpele matrixvermenigvuldiging met de vector van de RGB-waardes.

Voor digitale video-signalen wordt dikwijls de YCbCr kleurenruimte gebruikt. Ook in dit formaat wordt de luminantie informatie voorgesteld door n component, Y, terwijl de kleurinformatie als twee kleur-verschil componenten is opgeslagen, Cb en Cr: Cb is het verschil tussen de blauwe component en een referentiewaarde, en Cr is het verschil tussen rood en een referentiewaarde. De conversie van RGB naar YCbCr gebeurt door een matrixvermenigvuldiging gevolgd door een optelling met de referentiewaarden.



Figuur 5: HSV-kleurenruimte.

De HSV-ruimte (Hue, Saturation en Value) is een kleurensysteem waarbij kleuren gekozen worden uit een kleurenwiel of van een palet. Dit systeem staat dicht bij de menselijke ervaring en beschrijving van kleur dan het rgb-systeem. Als de hue varieert van 0 tot 1, dan variëren de kleuren van rood over geel, groen, cyaan, blauw en magenta terug naar rood, dus circulair zoals je kan zien op onderstaande figuur (er is dus rood in de buurt van zowel 0 als 1). De saturation kan ook variëren van 0 tot 1, en de corresponderende kleuren (of hue) variëren mee van ongesatureerd (wit en lichtgrijze schakeringen) tot volledig gesatureerd (geen witte component meer aanwezig). De value of 'brightness' varieert ook van 0 tot 1, en de kleuren worden lichter naarmate de value stijgt. De conversie gebeurt door mapping van cartesische rgb-coördinaten naar cilindrische hsv-coördinaten.

Cyaan, magenta en geel zijn de secundaire kleuren van het licht, of de primaire kleuren van pigment, dus de meeste toestellen, zoals kleurenprinters, die pigment op papier zetten, vereisen CMY input i.p.v. RGB (tenzij ze intern de conversie maken). De conversie tussen beiden is simpel: CMY is het complement van RGB (bv. $R = 1 - C$). Bij vierkleurendruk gebruikt men het CMYK-model, dat een vierde laag toevoegt om zwart goed voor te kunnen stellen (even grootte waarden C, M en Y levert ook zwart, maar het resultaat is niet altijd mooi zwart).

Matlab voorziet de conversie tussen RGB en voorgaande kleurenruimtes met de volgende commando's: **rgb2ntsc**, **rgb2ycbcr**, **rgb2hsv** en **imcomplement**, of vice versa. De conversiefactoren kan je zien als je bv. **edit rgb2ntsc** typt.

Een toepassing is bijvoorbeeld segmentatie op basis van kleur: implementeer nu zelf een kleurenfilter om de belangrijkste kleuren van verkeersborden te detecteren: rood, blauw, wit en zwart.

Je kan dit in de RGB-ruimte proberen, maar intuïtief kan je al aanvoelen dat dit beter zal gaan in de HSV-ruimte: blauw en rood zijn makkelijk te onderscheiden in de hue-component van deze ruimte (eventueel kan je ook op saturatiewaarden filteren, omdat die vrij hoog is voor de kleuren van een verkeersbord). Leg minimale en maximale drempelwaarden op voor de waarden van de hue. Denk er ook aan dat rood zowel rond 0 als 1 voorkomt, dus hou daarmee rekening voor de grenzen van de minimale en maximale hue-waarde. Zoek zelf uit hoe je best voor wit en zwart filtert: welke kleurenruimte, welke laag, ... Als output van de filters wil ik een black-and-white-beeld (= een logische matrix) waarin de enen duiden op de aanwezigheid van de gefilterde kleur in het beeld.



Figuur 6: trafficsign1.jpg en trafficsign2.jpg.