Beeldverwerking (image processing)

Irob – robotica domotica & industriële automatisering

Inleiding

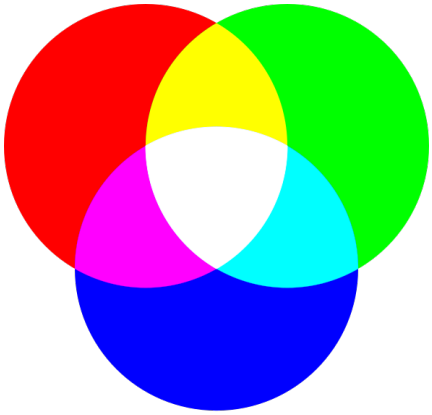
In deze les leer je hoe je afbeeldingen kunt bewerken met OpenCV [1], numpy, matplotlib, en python. OpenCV is een image library (of package) die veel gebruikt wordt voor beeldbewerking (image processing). We zullen deze les langzaam opbouwen van het inladen van een afbeelding en kleuren bewerken, tot meer geavanceerde technieken als image segmentation. Bij dit document behoren meerdere python scripts (jupyter notebook). In de scripts staan opdrachten die je moet uitvoeren.

Afbeelding inladen

Het inladen van afbeeldingen kan op veel manieren. Een veel gebruikte package is matplotlib. Wij gaan echter OpenCV gebruiken. Op de DLO vind je een bestand oefening1_afbeeldinginladen. (voor zowel jupyter notebook gebruikers als andere editors). Hierin laad je een foto. We gebruiken foto's uit de Kodak dataset van Kaggle [2]. Deze kun je ook van de DLO downloaden. Run dit python script en kijk of het werkt.

Kleuren en kleuren maken

Voor het representeren van kleuren gebruiken we de colorspace RGB. Dit staat voor Rood, Groen, en Blauw. Met deze afzonderlijke kanalen kun je alle kleuren maken die er zijn. In dit voorbeeld gebruiken we het python script oefening2_afzonderlijkekleuren. Met deze kleuren kun je kleuren maken zoals geel, magenta of elke andere kleur.



In python gebruiken we de package numpy om kleuren te representeren. Stel we hebben een afbeeldingen van 640 bij 480 pixels. Dan zal numpy zeggen dat de data de dimensie heeft van (640, 480, 3). De 3 is voor elk kleur kanaal. Kleur waarden worden gecodeerd in unint8. Dat zijn waarden van 0 tot en met 255 voor elk kanaal. In totaal kun je dus meer dan 16 miljoen kleuren maken.

Grijswaarden

Buiten rgb zijn er nog andere colorspaces zoals HSV (hue, saturation, value) die we niet behandelen. Wel gebruiken we de grijswaarden. We kunnen een kleur afbeelding converteren naar een zwart/wit afbeelding door gebruik te maken van de volgende formule.

```
Grayscale = 0.299R + 0.587G + 0.114B
```

Wat je hier ziet is een gewogen gemiddelde. Wat opvalt is dat Groen meer gewicht krijgt. Dit komt omdat ons oog meer gevoelig is voor deze kleur. Run het volgende python script om te kijken hoe de conversie plaatsvindt: oefening3_grijs

Region of interest (ROI)

Zoals je gezien hebt is een afbeelding een 2D matrix voor elk kanaal. Omdat we dit kunnen bewerken met numpy kunnen we ook een selectie maken van de matrix. Zo kunnen we een klein stukje uit de afbeelding knippen. Een specificieke uitsnijding van een afbeelding noemen we ook wel een Region of Interest (ROI). Het gebied wat we selecteren zijn we speciaal in geïntereseert (later hierover meer). Run het volgende script om te zien hoe dit werkt: oefening4_selection_ROI

Histogram

Het is handig om van de kleuren, of grijswaarden een histogram te plotten. Histograms geven een overzicht van de spreiding en de meest dominante kleurwaarden. Ook kan een histogram helpen bij het bepalen van een threshold waarden (later hierover meer). Open het script oefening6_histogram en plot het histogram voor de afzonderlijke kleurkanalen.

Thresholding

Thresholding (drempelwaarde) is een veelgebruikte techniek binnen image processing. Het wordt gebruikt om bepaalde kleur gebieden te selecteren of het detecteren van bewegende objecten. Thresholding is heel eenvoudig. Je gaat van elke pixel waarden in een afbeelding

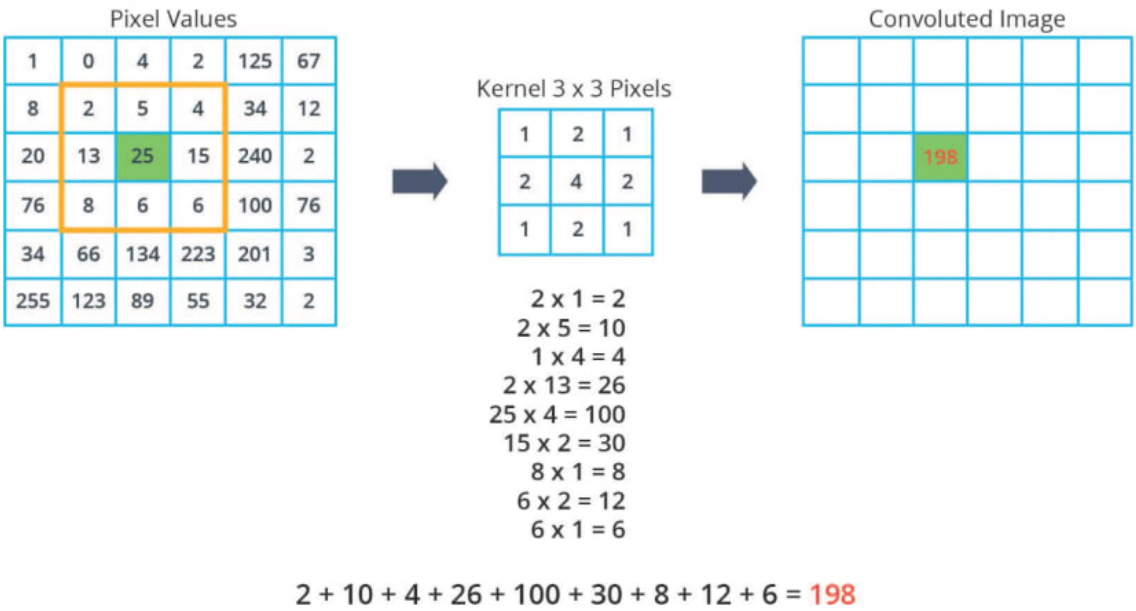
kijken of deze boven een bepaalde drempel waarde komt of niet. Bijvoorbeeld bij een zwart/wit afbeelding heb je een kanaal (grijs) met waarde van van 0 tot 255. De drempelwaarde is bijvoorbeeld 150. Als een pixel groter is dan deze waarde geeft Python een **TRUE** terug. Je converteert zo een afbeelding naar een binaire (TRUEs en FALSEs) afbeelding die je later bijvoorbeeld kunt gebruiken als mask. (oefening7_threshold)

Masking

Je kent het wel, een feestmasker. Misschien heb je zoiets ooit gedragen tijdens carnaval. Een masker maskeert een gebied wat daardoor niet zichtbaar is. Het gedeelte wat niet gemaskeert is blijft zichtbaar. Dit principe kun je ook toepassen bij afbeeldingen. Je kunt in een afbeelding specifieke gebieden maskeren die je niet wilt zien.
Een masker is een binaire afbeeldingen (pixelwaarden van 0 of 1, of TRUE en FALSE). Het masker leg je over de afbeelding (afbeeldingen moeten dezelfde dimensies hebben). Op de plaatsen waar het masker een 0 is wordt in de afbeelding de kleur op zwart gezet (0). Waar het masker 1 is blijft de oorspronkelijke kleur behouden. Probeer dit zelf met het volgende python script: oefening8_mask

Filters

Na alle basis operaties gaan we nu filters behandelen. In programma's als (Adobe) Photoshop ken je vast wel de functie waarbij je een afbeelding kun vervagen (blurren). Maar hoe werkt dat nou eigenlijk?
Dit doen we door een zogenaamde kernel te bewegen over een afbeelding.



Een kernel is meestal een 3 bij 3 matrix met getallen. Deze leg je over een afbeelding heen. De waarden in de kernel worden vermenigvuldigd met de waarden in de afbeelding op die positie. Alle producten worden gesommeerd en creëren een nieuwe waarde voor de nieuwe afbeelding. Dit is schematisch weergegeven in bovenstaande figuur. De kernel wordt over de hele afbeelding geschoven van links naar recht en van boven naar onder. Dit schuiven over een afbeelding noemen we **convolutie**.

In het python script wordt de average blur behandeld. Probeer te spelen met grote kernel sizes en bekijk het effect. oefening9_blur

Edge detection

Kernels (filters) kunnen voor alles worden gebruikt. Zo bestaat er ook een kernel (Sobel kernel)[3] die gebruikt wordt om edges (randen) te herkennen. Edges worden herkent in verticale en horizontale richting gebruikmakend van de volgende 3x3 matrices

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Het resultaat van deze convolutie wordt samengevoegd met de volgende formule.

$$|G| = \sqrt{Gx^2 + Gy^2}$$

De Edge detection demo is te vinden in het volgende python script:
oefening10_edgedetection

Image segmentation

Als laatste zullen we een geavanceerde topic bespreken binnen image processing: het segmenteren van een afbeelding (image segmentation). Het doel is het vinden van losse partities of segmenten in een afbeelding. Een voorbeeld hiervan is te vinden in onderstaand figuur .

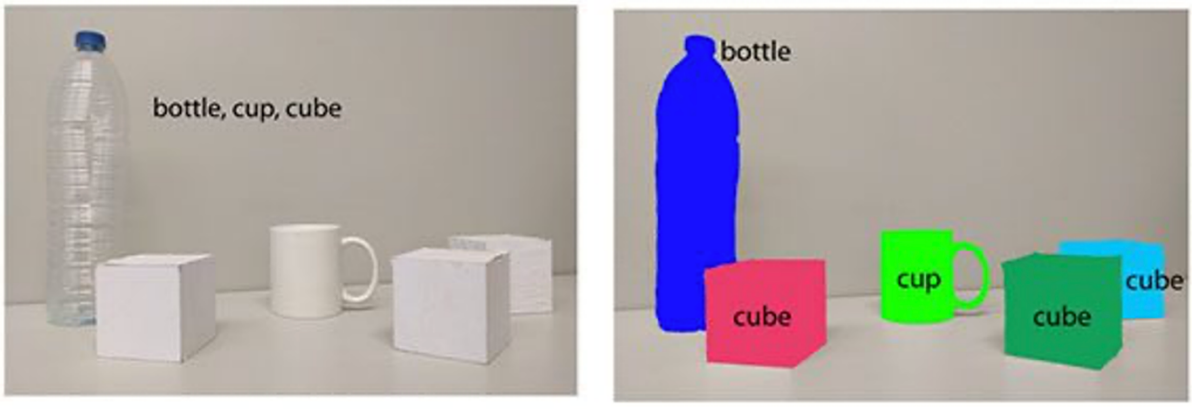
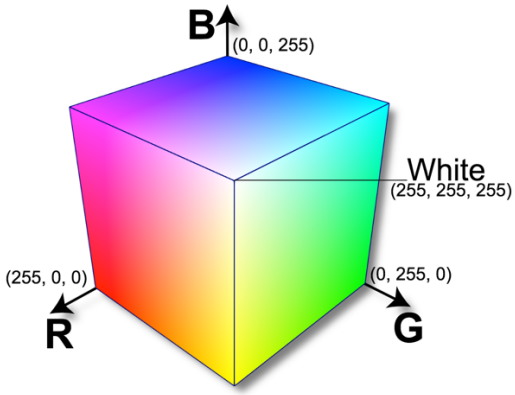


Image segmentation is erg moeilijk en de algoritmen zijn erg geavanceerd en kosten veel rekenkracht. Wij zullen hier een simpel segmentation algoritme behandelen gebaseerd op K-means clustering.



De rgb waarden van een afbeelding kun je zien als punten in een 3 dimensionale ruimte (zie figuur). Elk pixel is dan een vector en daarmee kun je dus vector algebra gebruiken. Pixels hebben bijvoorbeeld een afstand van elkaar in deze ruimte. We kunnen de afstand gebruiken om mee te clusteren. We gebruiken K-means clustering om een bepaald aantaal clusters te krijgen met de meest dominante kleurwaarden. Een voorbeeld hiervan is te zien in het python script oefening11_segmentation_Kmeans. Probeer te spelen met verschillende waarden van K en kijk wat het effect is.

[1] https://docs.opencv.org/4.x/index.html
[2] https://www.kaggle.com/datasets/sherylmehta/kodak-dataset
[3] https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm