Samenvatting Beeldbewerken

Lucas Riedstra

25 mei 2019

Inhoudsopgave

1 Beelden

1.1 Beelden vormen, definiëren, en discretiseren

Het menselijk oog werkt (gesimplificeerd) als een *pinhole camera* (zie H8). Het feit dat een 3D-wereld geprojecteerd wordt op een 2D-retina kan voor optische illusies zorgen. We kunnen het retina zien als een (x, y)-vlak, en het retina kan de elektromagnetische energie *rond* elk punt meten, dit zien we als een functie $f(x, y)$. We zeggen specifiek *rond* omdat energie in één punt meten onmogelijk is.

Een *beeld* is een functie $f\colon E \to V$. Vaak nemen we $E = \mathbb{R}^d$ voor zekere d (meestal $d = 2$ voor een 'gewone' 2D-image). Een voorbeeld: de functie die aan elk punt zijn RGB-waarde toekent schrijven we als $f\colon \mathbb{R}^2 \to \mathbb{R}^3\colon \mathbf{x} \mapsto (r(\mathbf{x}), g(\mathbf{x}), b(\mathbf{x}))$.

Een functie met als bereik $\{0, 1\}$ noemen we een *indicator*-afbeelding. Een functie met eindig bereik noemen we een *label*-afbeelding.

Het discretiseren van een afbeeldingsfunctie $f\colon \mathbb{R}^d \to \mathbb{R}$ (met meestal $d = 2$) bestaat uit discretisatie van het domein (*sampling*) en van het beeld (*kwantisering*).

We nemen altijd aan dat beelden correct gekwantiseerd zijn, dus dat het bereik van f een discrete verzameling is. Sampling doen we in een eindig aantal punten, dit induceert (bij een 2D-beeld) een functie

$$F\colon \mathbb{Z}^2 \to \mathbb{R}\colon F(i, j) = f(i \cdot \Delta x, j \cdot \Delta y),$$

waarbij we dus *samplen* in alle veelvouden van Δx en Δy.

Definitie 1.1. Een *pixel* is een sample van een punt in 2D, en een afbeelding is een regulier grid van pixels. 3D-samples noemen we *voxels*.

1.2 Interpolatie en representatie

Het probleem: we hebben een onbekende functie $f\colon \mathbb{R}^d \to \mathbb{R}$ en een bekende sample-functie F zoals in de vorige paragraaf. Gegeven $\mathbf{x} \in \mathbb{R}^d$ willen we nu een schatting van $f(\mathbf{x})$. Er zijn meerdere interpolatiemethodes. We doen eerst 1D, dus het domein van f (notatie $\mathcal{D}(f)$) is $\mathbb{R}$;

1. De domste is *nearest neighbour*: we stellen $\hat{f}(x) = F(\lfloor x + \frac{1}{2} \rfloor)$, wat nette notatie is voor het dichtsbijzijnde interpolatiepunt nemen.
2. Iets slimmer is *lineaire interpolatie*: tussen elk van de interpolatiepunten (x_i, x_{i+1}) trek je een lijntje, om zo een stuksgewijs lineaire $\hat{f}$ te verkrijgen.
3. Nog iets slimmer is *kubische interpolatie*: voor een punt x bepaal je de i zodanig dat $x \in (x_i, x_{i+1})$, en vind je een derdegraadsfunctie die door $x_{i-1}, x_i, x_{i+1}, x_{i+2}$ heen gaat.
4. Er zijn veel betere interpolatiemethodes, bijvoorbeeld *splines* (die een gladde benadering geven). Een gevaar is *overfitting* van de data, wat voor vreemde fluctuaties kan zorgen.

Er zijn verschillende manieren om 1D-interpolatiemethodes vervolgens te generaliseren naar 2D.

Helaas worden voor het representeren van beelden verschillende coördinatensystemen gebruikt. In de assen is er een verschil tussen een *linkshandig* en *rechtshandig* assensysteem (afhankelijk van welke kant je de x-as op moet draaien om de y-as te krijgen).

Het standaard wiskundige assenstelsel is rechtshandig, het standaard assenstelsel voor weergave van arrays (en dus ook plotten van plaatjes) is linkshandig.

1.3 Histogrammen

Zij $f\colon E \to \mathbb{R}$, en $e_0, \ldots, e_n$ de gewenste randen van de bins van het histogram. Dan wordt het histogram van f gedefinieerd door

$$h_f(i) = \#\{x \in E \mid e_i \leq f(x) < e_{i+1}\},$$

met de notatie $\#S$ voor het aantal elementen in een bepaalde verzameling.

Als $f\colon E \to \mathbb{R}^d$ voor zekere $d > 1$, dan wordt het histogram *multivariaat*. De functie wordt dan iets van

$$h_f(i_1, \ldots, i_d) = \#\{x \in E \mid (e_{1,i_1} \leq f_1(x) < e_{1,i_1+1}) \wedge \cdots \wedge (e_{d,i_d} \leq f_d(x) < e_{d,i_d+1})\}.$$

In 2D kunnen we het histogram als plaatje visualiseren, voor drie of meer dimensies wordt het snel lastig.

2 Puntoperatoren

Definitie 2.1. Zij $f\colon \mathcal{D} \to \mathcal{R}$. Dan noemen we $g\colon \mathcal{D} \to \mathcal{R}'$ een *puntoperator* als er een $\psi\colon \mathcal{R} \to \mathcal{R}'$ bestaat zodanig dat $g(\mathbf{x}) = \psi(f(\mathbf{x}))$. In mensentaal zegt dit dat de waarde van g in $\mathbf{x}$ alleen afhangt van de waarde van f in datzelfde punt $\mathbf{x}$.

Dit concept kan ook gegeneraliseerd worden naar meerdere beelden. Voorbeelden zijn het puntsgewijs optellen, aftrekken, vermenigvuldigen of delen van twee beelden.

Opmerking. In de lecture notes wordt heel netjes het optellen/aftrekken/vermenigvuldigen/delen van beelden gedefinieerd (met verheffingen van aritmetische operaties), dit lijkt me onnodig.

We bekijken voorbeelden van veelvoorkomende puntoperaties.

α-blending Als we twee beelden f en g hebben, en we willen f subtiel 'omvormen' tot g, dan kunnen we voor verschilende $\alpha \in [0, 1]$ het beeld $h_\alpha = (1 - \alpha)f + \alpha g$ gebruiken.

Masking Als f een beeld is en g een *geblurde* versie van f, dan kunnen we het beeld $h_\beta = f + \beta(f - g)$ bekijken. Dit beeld zal lijken op een verscherpte versie van f.

Thresholding Gegeven een vaste waarde t kunnen we het beeld $h_t(\mathbf{x}) = \mathbb{1}_{f(\mathbf{x})>t}$ bekijken (1 als $f(\mathbf{x}) > t$, 0 anders). We noteren dit beeld ook simpelweg als $f > t$.

Contrast stretching Stel dat alle waardes van $\mathbf{f}$ tussen twee waardes $\min f$ en $\max f$ liggen, maar we willen dat de waardes netjes uitgesmeerd op het interval $[0, 1]$ liggen. Dan definiëren we

$$g = \frac{f - \min f}{\max f - \min f},$$

met $\max f$ en $\min f$ de maximale en minimale waardes die f aanneemt.

Histogramequalisatie Neem h_f het histogram van f. Nu willen we een puntoperatie g definiëren zodat g een constant histogram heeft, maar nog steeds lijkt op f in de zin dat g *orde bewaart*: als $f(\mathbf{x}_1) \leq f(\mathbf{x}_2)$, willen we ook dat $g(\mathbf{x}_1) \leq g(\mathbf{x}_2)$.

De definitie van een puntoperatie dicteert dat er een ψ moet bestaan zodanig dat $g(\mathbf{x}) = \psi(f(\mathbf{x}))$. Uit de ordebewaringseis volgt, met H_f en H_g de *genormaliseerde cumulatieve* histogrammen van f en g, dat

$$H_f(v) = H_g(\psi(v)),$$

want alle waardes die in f kleiner zijn dan $f(v)$ moeten in g ook kleiner zijn dan $\psi(f(v))$.

Nemen we aan dat ons bereik $[0, 1]$ is hebben we $H_g(\psi(v)) = \psi(v)$ (want we willen dat g een constant histogram heeft), en we vinden $\psi(v) = H_f(v)$. De histogramequalisatie is dus $g = H_f(f)$.

Dit is nuttig omdat beelden met verschillende intensiteiten vaak een gelijkaardig geëqualiseerd histogram zullen hebben.

Threshold finding Zij $f\colon \mathcal{D} \to [0,1]$ een beeld, en definieer voor alle $t \in [0,1]$ de waardes $m_L(t)$ en $m_H(t)$ als het aantal pixels wiens waarde resp. onder en boven t ligt. We zoeken nu een *threshold*, dat wil zeggen een $\hat{t}$ zodanig dat de helft van de punten onder en de helft van de punten boven t ligt.

$$\hat{t} = \frac{m_L(\hat{t}) + m_H(\hat{t})}{2},$$

Deze waarde kan gevonden worden met een simpele dekpuntiteratie.

3 Kleur

'Kleur' is elektromagnetische straling, met een bepaald *spectrum* wat bij elke golflengte de bijbehorende *intensiteit* geeft. Een spectrum is dus een functie, en de verzameling spectra is oneindigdimensionaal. We noteren zo'n spectrum met $t(\lambda)$ voor golflengte λ.

Neem f een functie van λ, dan samplen we deze op punten $\lambda_1, \ldots, \lambda_k$ met de samplingfunctie $F(k) = f(\lambda_k)$. Om het extra verwarrend te maken interpreteren we onze samples als vectoren *en noteren we deze vector ook met de letter* $\mathbf{f}$. Dus $f_k = F(k) = f(\lambda_k)$.

In onze ogen hebben we *staafjes* en *kegels*. Staafjes zien goed in het donker, en zien geen kleuren. We hebben drie soorten kegels, genoteerd met S, M, L, voor 'kleine' golflengtes (blauw), 'medium' (groen), en 'grote' (rood). De respons van zo'n kegel bij een golflengte λ noteren we met $s_i(\lambda)$ voor $i = S, M, L$. De kleur die uiteindelijk wordt meegegeven is nu

$$c_i = \int s_i(\lambda) t(\lambda) \mathrm{d}\lambda.$$

Noteren we samples nu met dezelfde letters, dus samples $\mathbf{s}_i$ van golflengtes $\mathbf{t}$, dan krijgen we de geschatte kleur $c_i = \mathbf{s}_i^T \mathbf{t} = \langle \mathbf{s}, \mathbf{t} \rangle$. De verschillende c_i schrijven we in één *responsvector*

$$\mathbf{c} = \begin{pmatrix} c_S \\ c_M \\ c_L \end{pmatrix} = \begin{pmatrix} \mathbf{s}_S^T \mathbf{t} \\ \mathbf{s}_M^T \mathbf{t} \\ \mathbf{s}_L^T \mathbf{t} \end{pmatrix} = \begin{pmatrix} \mathbf{s}_S^T \\ \mathbf{s}_M^T \\ \mathbf{s}_L^T \end{pmatrix} \mathbf{t} = \begin{pmatrix} \mathbf{s}_S & \mathbf{s}_M & \mathbf{s}_L \end{pmatrix}^T \mathbf{t} =: S^T \mathbf{t}.$$

We noemen S de *sensitiviteitsmatrix*, en de vector $\mathbf{c}$ is de 'input' die het brein bij ontvangt. Voor het brein geldt dus dat twee spectra $\mathbf{t}_1, \mathbf{t}_2$ 'hetzelfde' zijn als $S^T \mathbf{t}_1 = S^T \mathbf{t}_2$.

Kleurrespons is lineair, dus als we twee stralen met spectra $\mathbf{t}_1, \mathbf{t}_2$ zien, dan heeft de gecombineerde straal spectrum $\mathbf{t}_1 + \mathbf{t}_2$ en is de ontvangen kleur dus $\mathbf{c}_1 + \mathbf{c}_2$. Daarbij verandert een spectrum niet bij verandering van intensiteit, dus voor alle $a \neq 0$ geldt dat $\mathbf{t}$ en $a\mathbf{t}$ dezelfde kleur opleveren. Dit tweede feit betekent dat we maar twee dimensies nodig hebben om kleur te representeren, door bijvoorbeeld het vlak $c_1 + c_2 + c_3 = 1$ te kiezen. Dit geeft de zgn. *chromaticiteitscoördinaten*

$$c_X = \frac{c_1}{c_1 + c_2 + c_3}, \quad c_Y = \frac{c_2}{c_1 + c_2 + c_3}, \quad c_Z = 1 - c_Y - c_X.$$

3.1 Kleurmatching

We willen bepalen wanneer twee kleuren als hetzelfde worden gezien, zonder a priori de sensitiviteitsmatrix S te kennen. Merk hiervoor op dat voor elke inverteerbare $A \in \mathbb{R}^{3\times 3}$ geldt $S^T \mathbf{t}_1 = S^T \mathbf{t}_2 \iff (SA)^T \mathbf{t}_1 = (SA)^T \mathbf{t}_2$. Elke matrix SA noemen we een *kleurmatchmatrix*.

Het kleurmatchexperiment werkt nu als volgt: We nemen een lichtbron met spectrum $\mathbf{t}$, en drie *primaries* $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$, die lineair onafhankelijk zijn. De observeerder kan de intensiteit van de primaries aanpassen om zo een kleur

$$\mathbf{u} = a_1 \mathbf{p}_1 + a_2 \mathbf{p}_2 + a_3 \mathbf{p}_3$$

te verkrijgen, en er wordt aan de observeerde gevraagd om a_1, a_2, a_3 zo te tweaken dat hij/zij het verschil tussen $\mathbf{u}$ en $\mathbf{t}$ niet meer ziet. Schrijven we $P = \begin{bmatrix}\mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3\end{bmatrix}$ en $\mathbf{a} = (a_1, a_2, a_3)$, dan krijgen we

$$S^T\mathbf{t} = S^T(a_1\mathbf{p}_1 + a_2\mathbf{p}_2 + a_3\mathbf{p}_3) = S^TP\mathbf{a}.$$

We nemen aan dat S^TP inverteerbaar is (de kans dat het niet zo is, is nul), en krijgen

$$\mathbf{a} = (S^TP)^{-1}S^T\mathbf{t}.$$

Als we dus de matrix $C^T := (S^TP)^{-1}S^T$ kennen, kunnen we bij elk spectrum $\mathbf{t}$ de bijbehorende set intensiteiten a_1, a_2, a_3 vinden om $\mathbf{t}$ te construeren uit $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$. Om die matrix te vinden, merken we op dat als we kiezen $\mathbf{t} = (0, \ldots, 0, 1, 0, \ldots, 0) = \Delta_i$ (met de 1 op plek i), we de eerste kolom van C krijgen bij vermenigvuldiging met $\mathbf{t}$, dus door het experiment te herhalen voor elke Δ_i kunnen we C verkrijgen.

Dit heeft geleid tot de bekende matrix C_{rgb}. Deze heeft ook negatieve entries bij de roodwaardes, die bij het experiment verkregen werden door roodwaardes *op te tellen* bij $\mathbf{t}$, en die er daarna weer vanaf te halen bij de berekening.

Later is de matrix C_{xyz} opgesteld, met als eigenschappen dat deze enkel positieve getallen bevat, en dat de tweede coördinaat gelijk staat aan de helderheid van de kleur. Gegeven een spectrum $\mathbf{t}$, wordt de bijbehorende kleur in het XYZ-kleursysteem (chromaticiteitscoördinaten) gegeven door

$$(X, Y, Z) = C_{xyz}^T\mathbf{t}.$$

Aangezien enkel twee van de drie coördinaten nodig zijn, wordt in het algemeen het XY-chromaticiteitsdiagram getekend. Er zijn ook andere kleursystemen, zoals HSV.

Als we alle zichtbare combinaties van monochromatisch licht in het XY-chromaticiteitsdiagram tekenen krijgen we de bekende *horse shoe*. Een monitor werkt door lineaire combinaties te maken van drie primaire lichtbronnen. Uit de vorm van de *horse shoe* is duidelijk dat geen enkele combinatie primaire lichtbronnen *alle kleuren* kan geven, maar alleen kleuren binnen de driehoek met die lichtbronnen als hoekpunten. Deze driehoek heet het *gamut* van de monitor. Verschillende monitoren hebben verschillende gamuts, waardoor ook errors kunnen ontstaan omdat een bepaalde kleur simpelweg niet representeerbaar is.

4 Image Warping

4.1 Forward en backward transform

Definitie 4.1. Een *geometrische operator* verplaats punten van één beeld naar een ander beeld.

$$g(\varphi(\mathbf{x})) = f(\mathbf{x}).$$

Een simpele algoritme (*forward transform*): voor elke x in het oorspronkelijke beeld $\mathcal{D}_f$: zet $g(\varphi(\mathbf{x})) = f(\mathbf{x})$.

Een probleem hier is dat $\varphi(\mathbf{x})$ niet per se op een pixel terechtkomt: wélke pixel in g geven we de waarde $\varphi(\mathbf{x})$?

Nóg een probleem is dat meerdere pixels naar dezelfde pixel zullen verwijzen, en dat sommige pixels in het beeld dus zullen 'missen'. Als het beeld groter is dan het origineel treedt dit probleem met zekerheid op.

Conclusie: forward transform werkt niet. Oplossing: *backward transform*: we loopen over de pixels in het *beeld* $\mathcal{D}_g$, en daarvoor zetten we $g(\mathbf{x}') = f(\varphi^{-1}(\mathbf{x}'))$. Weer een probleem is dat $\varphi^{-1}(\mathbf{x}')$ niet per se in $\mathcal{D}_f$ zit. We zullen dus moeten interpoleren met een of andere geïnterpoleerde $\hat{f}$ en krijgen $g(\mathbf{x}') = \hat{f}(\varphi^{-1}(\mathbf{x}'))$.

Er zijn natuurlijk meerdere interpolatiemogelijkheden, waaronder *nearest neighbour* en *bilinear interpolation*. Bij *spline interpolation* kan gekozen worden tot welke orde de afgeleides continu moeten

zijn (bij simpele lineaire interpolatie is de afgeleide in het algemeen niet existent in de interpolatiepunten).

4.2 Homogene coördinaten en transformaties

Definitie 4.2. Gegeven een vector $\mathbf{x} = (x, y) \in \mathbb{R}^2$ geven we het punt de *homogene coördinaten* $(\lambda x, \lambda y, \lambda)$ voor elke $\lambda \in \mathbb{R} \setminus \{0\}$. We noteren de homogene representatie met $\tilde{\mathbf{x}}$, en gelijkheid van homogene vectoren noteren we met $\tilde{\mathbf{x}} \sim \tilde{\mathbf{y}}$. In het algemeen kiezen we $\lambda = 1$ voor de representatie $(x, y, 1)$.

Homogene coördinaten zijn nodig om translaties te kunnen opschrijven in een matrix-vector-product. De translatie $(x, y) \mapsto (x + t_1, y + t_2)$, rotatie tegen de klok in over een hoek φ en schaling met s_1 en s_2 worden gegeven door

$$\begin{pmatrix} 1 & 0 & t_1 \\ 0 & 1 & t_2 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Merk op dat de laatste rij van deze matrices gegeven wordt door $(0, 0, 1)$, en de homogene coördinaten dus behouden blijven.

In het algemeen noemen we transformaties waarvan de laatste rij $(0, 0, 1)$ is *affien*. Een belangrijke eigenschap van affiene matrices is dat ze paralleliteit van lijnen behouden. Ook worden affiene transformaties uniek gekarakteriseerd door het beeld van *drie punten*: drie punten is genoeg om de bijbehorende transformatie te bepalen en voor elke drie punten bestaat een affiene transformatie.

Een algemene (3×3)-matrixtransformatie A waarvan we enkel eisen dat $A_{33} = 1$ noemen we een *projectieve transformatie*. Projectieve transformaties behouden *geen* paralleliteit, maar wel *collineariteit*: als punten in het origineel op één lijn liggen, dan liggen ze in het beeld ook op één lijn.

Hoe bepalen we zo'n affiene transformatie gegeven drie punten? Uitschrijven geeft dat het simpelweg een stelsel van vergelijkingen is, dus kunnen we het omschrijven als $A\mathbf{x} = \mathbf{b}$, en dit kan (numeriek) worden opgelost. Een affiene transformatie wordt (als we de homogeniteit vergeten) gegeven door

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix},$$

wat we kunnen herschrijven tot

$$\begin{pmatrix} x'_i \\ y'_i \end{pmatrix} = \begin{pmatrix} x_i & y_i & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_i & y_i & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix}.$$

Voor 3 puntcorrespondenties krijgen we dus een 6×6-matrix die in het algemeen inverteerbaar zal zijn.

Voor meer puntcorrespondenties bestaat er meestal niet één transformatie die alle punten perfect op elkaar afstuurt. We kunnen de parameters dus enkel *schatten* met de least-squares-methode:

$$\mathbf{a} = (M^T M)^{-1} M^T \mathbf{b},$$

met als doel om de afstand $\|M\mathbf{a} - \mathbf{b}\|^2$ te minimaliseren. Hierbij geldt $\mathbf{a} = (a, b, c, d, e, f)^\top$ en $\mathbf{b} = (x'_1, y'_1, \cdots)^\top$.

4.3 Projectieve transformaties

We hebben nu puntcorrespondenties $(\mathbf{x}_i, \mathbf{x}'_i)$ en zoeken een projectieve matrix P zodanig dat $\tilde{\mathbf{x}'_\mathbf{i}} \sim P\tilde{\mathbf{x}}_i$, met $P \in \mathbb{R}^{3\times 3}$. Door de schalingsfactor kunnen we dit niet zomaar oplossen met standaard LinAlg-methodes.

Opmerking. Ik heb in de aantekeningen per ongeluk de letter i als index gebruikt, maar ook als entry in de matrix P. Oeps.

Schrijf

$$P = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}.$$

Door de schalingsfactor weg te delen krijgen we

$$\begin{pmatrix} x'_i \\ y'_i \\ 1 \end{pmatrix} = \begin{pmatrix} ax_i + by_i + c \\ dx_i + ey_i + f \\ gx_i + hy_i + i \end{pmatrix} \Big/ (gx_i + hy_i + i),$$

wat we natuurlijk kunnen herschrijven tot

$$(gx_i + hy_i + i) \begin{pmatrix} x'_i \\ y'_i \\ 1 \end{pmatrix} = \begin{pmatrix} ax_i + by_i + c \\ dx_i + ey_i + f \\ gx_i + hy_i + i \end{pmatrix},$$

en dat geeft een stelsel vergelijkingen

$$\begin{aligned} ax_i \quad +by_i + c \quad - gx_ix'_i \quad -hy_ix'_i - ix'_i \quad &= 0 \\ dx_i + ey_i \quad +f - gx_iy'_i \quad -hy_iy'_i - iy'_i \quad &= 0, \end{aligned}$$

en dít is nu een vergelijking in de onbekenden, en geeft een (2×9)-matrix C met

$$\begin{pmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x'_ix_i & -x'_iy_i & -x'_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y'_ix_i & -y'_iy_i & -y'_i \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Om dit in het algemeen te kunnen oplossen doen we er nog drie punten bij en krijgen een (8×9)-matrix M in plaats van C (of in het algemeen een $(2n \times 9)$-matrix met n het aantal punten). Ons probleem $M\mathbf{p} = \mathbf{0}$ zal natuurlijk $\mathbf{p} = 0$ als oplossing hebben, maar dat willen we niet – dus we stellen de eisen dat $\|M\mathbf{p}\|$ minimaal is én dat $\|\mathbf{p}\| = 1$.

Om dit op te lossen nemen we de SVD van M, ter herinnering:

$$M = UDV^T$$

met $U \in \mathbb{R}^{2n\times 2n}, V \in \mathbb{R}^{9\times 9}, D \in \mathbb{R}^{2n\times 9}$, waarbij D 'diagonaal' is (een diagonaalblok en wat nullen eronder).

Oplossen geeft

$$\min_{\mathbf{p}} \|UDV^T\mathbf{p}\| = \|DV^T\mathbf{p}\|,$$

en stellen we nu $\mathbf{q} = V^T\mathbf{p}$, dan geldt $\|\mathbf{q}\| = \|\mathbf{p}\| = 1$ volgens onze eis, en nu zoeken we $\min_{\mathbf{q}} \|D\mathbf{q}\|$. Dit minimum wordt bereikt door de vector $\mathbf{q}$ wiens *laatste* entry een 1 is (en de rest nullen), want D heeft de singuliere waardes *gesorteerd op grootte*. Hebben we onze optimale $\mathbf{q}$, dan hebben we ook $\mathbf{p} = V\mathbf{q}$, en dat is simpelweg de laatste kolom van V (of de laatste rij van V^T).

5 Lokale operatoren

5.1 Definities

We hebben *puntoperatoren* (puntsgewijs toegepast) en *geometrische operatoren* (verplaatsen van punten) gedefinieerd. Nu definiëren we lokale operatoren.

Definitie 5.1. Lokale operatoren zijn operatoren T waarbij $(Tf)(\mathbf{x})$ niet alleen afhangt van $f(\mathbf{x})$ maar ook van de waardes van f in een zekere *omgeving* van $\mathbf{y}$, notatie $\mathcal{N}(\mathbf{x})$.

Voorbeelden zijn het gemiddelde, de mediaan, en de standaarddeviatie van de punten in $\mathcal{N}(\mathbf{x})$.

Definitie 5.2. We noemen een lokale operator O *translatie-invariant* als de keuze van omgeving $\mathcal{N}(\mathbf{x})$ niet afhangt van $\mathbf{x}$ (dat wil zeggen, de omgevingen van elk punt zijn getransleerde versies van elkaar).

Als we definiëren $(T_{\mathbf{t}}f)(\mathbf{x}) = f(\mathbf{x} - \mathbf{t})$, dan komt dit neer op de eis dat $O(T_t f) = T_t(Of)$.

Als de oorsprong in het domein zit kunnen we dit ook schrijven als $\mathcal{N}(\mathbf{x}) = \mathcal{N}(\mathbf{0}) + \mathbf{x}$.

Definitie 5.3. Een *lineaire operator* is een operator L zodanig dat $L(f+g) = Lf+Lg$ en $L(cf) = cL(f)$ voor f, g in het domein en constantes c.

Merk op dat hiervoor vermenigvuldiging met constantes en optelling gedefinieerd moeten zijn in het domein en bereik.

5.2 Convoluties

Neem L een lineaire en translatie-invariante operator, en f een puntoperator. We hebben nu

$$f(\mathbf{x}) = \sum_{\mathbf{y}} f(\mathbf{y})\Delta(\mathbf{x} - \mathbf{y}),$$

met

$$\Delta(\mathbf{x}) = \begin{cases} 1 & \text{als } \mathbf{x} = \mathbf{0}; \\ 0 & \text{anders.} \end{cases}$$

Hier hebben we nog niets gedaan behalve dingen moeilijk opschrijven. We noemen Δ de *impuls*. We kunnen ook schrijven

$$f(\mathbf{x}) = \sum_{\mathbf{y}} f(\mathbf{y})(T_{\mathbf{y}}\Delta)(\mathbf{x}), \quad \text{en dus } f = \sum_{\mathbf{y}} f(\mathbf{y})(T_y\Delta),$$

met $T_{\mathbf{y}}$ de translatie $\mathbf{x} \mapsto \mathbf{x} - \mathbf{y}$.

Laten we nu L los, dan krijgen we

$$Lf = L\left(\sum_{\mathbf{y}} f(\mathbf{y})(T_{\mathbf{y}}\Delta)\right) = \sum_{\mathbf{y}} f(\mathbf{y}) \cdot L(T_{\mathbf{y}}\Delta) = \sum_{\mathbf{y}} f(\mathbf{y}) \cdot T_{\mathbf{y}}(L\Delta).$$

We noemen $W \coloneqq \Delta$ de *impulsrespons*. De laatste stap was enkel mogelijk omdat we hadden aangenomen dat L translatie-invariant was, dus het is onbelangrijk of we eerst L en dan $T_{\mathbf{y}}$ toepassen of andersom.

Nu invullen in een punt geeft

$$Lf(\mathbf{x}) = \sum_{\mathbf{y}} f(\mathbf{y}) \cdot T_{\mathbf{y}}(L\Delta)(\mathbf{x}) = \sum_{\mathbf{y}} f(\mathbf{y})(L\Delta)(\mathbf{x} - \mathbf{y}) = \sum_{\mathbf{y}} f(\mathbf{x} - \mathbf{y})(L\Delta)(\mathbf{x}),$$

wat neerkomt op $Lf = f * L\Delta = f * W$, de *convolutie*. De definitie van convolutie is dus

$$(f * W)(\mathbf{x}) = \sum_{\mathbf{y}} f(\mathbf{x} - \mathbf{y})W(\mathbf{y})$$

We noemen W een *kernel*.

Met deze afleiding zien we dat elke lineaire translatie-invariante operator te schrijven is als convolutie. Andersom geldt ook dat convolueren een lineaire en translatie-invariante operatie is.

Vervangen we het minteken door een plusteken dan noemen we dit de *correlatie*, dus

$$f *_{cr} W = \sum_{\mathbf{y}} f(\mathbf{x}+\mathbf{y})W(\mathbf{x}).$$

Voor continue functies definiëren we de convolutie als $(f*g)(\mathbf{x}) = \int f(\mathbf{x}-\mathbf{y})g(\mathbf{y})\,\mathrm{d}\mathbf{y}$.

5.3 Convolutie-eigenschappen

De convolutieformule is

$$(f*w)(\mathbf{x}) = \sum_{\mathbf{y}} f(\mathbf{x}-\mathbf{y})w(\mathbf{y}) \overset{\star}{=} \sum_{\mathbf{z}} f(\mathbf{z})w(\mathbf{x}-\mathbf{z}) = (w*f)(\mathbf{x}),$$

waarbij $\star$ de substitutie $\mathbf{z} = \mathbf{x}-\mathbf{y}$ voorstelt. Zowel $\mathbf{y}$ als $\mathbf{z}$ lopen over het hele domein (zij het in andere volgorde). Convolutie is dus commutatief áls het domein oneindig is (anders werkt het verschuiven van de sommatie niet).

Verder herschrijven geeft

$$\sum_{\mathbf{z}} f(\mathbf{z})w(\mathbf{x}-\mathbf{z}) = \sum_{\mathbf{z}} f(\mathbf{z})w^*(\mathbf{z}-\mathbf{x}) = \sum_{\mathbf{z}} f(\mathbf{z})(T_{\mathbf{x}}w^*)(\mathbf{z}).$$

Het stappenplan voor convolutie $f*W$ is dus:

1. Spiegel W in de oorsprong, noteer W^*;
2. Transleer de oorsprong van W^* naar $\mathbf{x}$;
3. Optelling van waardes van f vermenigvuldigd met bijbehorende waardes in de getransleerde W^*;
4. Doe dit voor alle $\mathbf{x}$.

Schrijven we f en W als kernels (zie notes) – dan moeten we dus W^* "op f" neerleggen op plek $\mathbf{x}$, en dan alle producten uitrekenen en sommeren.

Het opsplitsen van een kernel in kleinere kernels zorgt i.h.a. voor minder floating-point-vermenigvuldigingen.

6 Lokale structuur

6.1 Afgeleides met Gaussische blur

Schrijf $f\colon \mathbb{R}^2 \to \mathbb{R}$. We willen de partiële afgeleides van f, bv. $\partial f/\partial x$ schrijven met behulp van convoluties. Hiervoor moeten we twee dingen controleren:

1. Is het nemen van de afgeleide een lineaire operatie? Ja: $(af+bg)' = af' + bg'$.
2. Is het nemen van de afgeleide translatie-invariant? Ja: uitschrijven geeft $(T_{\mathbf{z}}f)' = T_{\mathbf{z}}f'$.

De vraag is nu natuurlijk met *welke* kernel we moeten convolueren om de afgeleide-operatie mee te maken, bijvoorbeeld $\partial f/\partial x = f * D_x$.

Hiervoor moeten we kijken wat de afgeleide betekent. Voorbeeld:

$$\frac{\partial f}{\partial x}(x,y) = \lim_{dx\to 0} \frac{f(x+dx,y)-f(x,y)}{dx}.$$

In een beeld kunnen we dx niet 'zo klein mogelijk' maken, maar we kunnen ons best doen met de gesamplede functie F. Dit geeft

$$F'_R[n] \approx \frac{F[n+1] - F[n]}{1} = F[n+1] - F[n].$$

We kunnen echter ook van de andere kant komen om te krijgen

$$F'_L[n] \approx F[n] - F[n-1].$$

Of we kunnen van links én van rechts komen om te krijgen

$$F'_C[n] \approx \frac{1}{2}(F[n+1] - F[n-1])$$

Deze drie versies noemen we de *rechter-, linker-, en center-afgeleide.*

We proberen de rechterafgeleide F'_R te bepalen met een kernel. Het gespiegelde kernel wordt [-1 1], dus het kernel zelf is [1 -1] (waarbij we de oorsprong onderstrepen). Voor F'_L krijgen we [1 -1] en voor F'_C krijgen we $\frac{1}{2}[-101]$. De kernels noemen we resp. D_R, D_L, D_C.

Voor de *tweede afgeleide* hebben we dus negen keuzes afhankelijk van welke afgeleide we kiezen te nemen. Voorbeeld: $F''_{RR} = (F * D_R) * D_R = F * (D_R * D_R)$. We rekenen $D_R * D_R$ uit. Uitschrijven geeft [1 -2 1], wat betekent dat we alleen de punten rechts nodig hebben. Maar hier verschijnt een vreeemd fenomeen, want dit kernel verwachten we eigenlijk om de tweede afgeleide één punt rechts van de oorsprong te berekenen. Wat gaat er mis? Het probleem is dat F_L eigenlijk de afgeleide in $\mathbf{x} + \frac{1}{2}$ probeert te berekenen, dus F_{LL} berekent de afgeleide één punt rechts.

F_{CC} wordt ook een beetje vreemd, we krijgen [1 0 -2 0 1], dus de punten direct rechts en links worden niet gebruikt. Wat een beter idee is, is om $D_L * D_R$ of $D_R * D_L$ uit te rekenen, dit geeft [1 -2 1], wat we ongeveer verwachten.

Een andere manier om afgeleides te berekenen is door de afgeleide van een vergladde versie van f te gebruiken.

Definitie 6.1. De *vergladde* (*smoothed*) versie van f is $f * G^s$, met

$$G^s(x, y) = \frac{1}{2\pi s^2} \exp\left(-\frac{x^2 + y^2}{s^2}\right),$$

Willen we nu de afgeleide naar x van $f * G^s$ berekenen, dan is dat hetzelfde als $f * \frac{\partial}{\partial x} G^s$. We hebben dan geen aparte afgeleide-kernel nodig. Het voordeel hiervan is dat we niet de afgeleide van een discrete functie hoeven te berekenen, enkel een continue functie te discretiseren.

We definiëren de *continue convolutie* als

$$(f * w)(\mathbf{x}) = \int_{-\infty}^{\infty} f(x - y) w(y) \, \mathrm{d}y.$$

Er geldt $\lim_{s \to 0} f * G^s = f$ als we continu convolueren, en dus ook $\lim_{s \to 0} f * \frac{\partial}{\partial x} G^s = \frac{\partial}{\partial x} f$.

Een groot probleem is *ruis*: als we bijvoorbeeld de ruis $f + r$ hebben, en we leiden af, dat we dan ruis meekrijgen in de afgeleides. Daar kan de Gaussian smoothing ook bij helpen.

Meer algemeen krijgen we

$$\frac{\partial^{n+m}}{\partial x^m \partial y^n} f * G^S = f * \left(\frac{\partial^{n+m}}{\partial x^m \partial y^n}\right) G^s. \tag{1}$$

Deze convolutie kan zowel continu berekend worden (als f analytisch en bekend is), als gediscretiseerd (als f enkel gesampled is – dan worden er natuurlijk fouten gemaakt).

Merk op dat

$$G^s(x,y) = G^s(x) \cdot G^s(y) = \frac{1}{\sqrt{2\pi s^2}} \exp(\frac{-x^2}{2s^2}) \cdot \frac{1}{\sqrt{2\pi s^2}} \exp(\frac{-y^2}{2s^2}).$$

Dit noemen we de separabiliteitseigenschap van G^s. En dus geldt

$$f * G^s(x,y) = (f *_i G^s(x)) *_j G^s(y),$$

met $*_i$ convolutie in de x-richting (langs de kolommen) en $*_j$ convolutie in de y-richting.

Als we dit op gediscretiseerde wijze doen (d.w.z. we samplen G^s), dan is het wel zo netjes om zodanig te normaliseren dat de som van de *gesamplede* waardes 1 is zodat het daadwerkelijk een gemiddeldefilter is.

Het is tijd om (1) te gebruiken. De afgeleides van G zijn ook separeerbaar, want

$$\frac{\partial^{n+m}}{\partial x^m \partial y^n}(G^s(x) * G^s(y)) = \frac{\partial^m}{\partial x^m} G^s(x) * \frac{\partial^n}{\partial y^n} G^s(y)$$

Dus we hoeven alleen de 1D-versie $G^s(x)$ te kunnen afleiden. Bijvoorbeeld

$$(G^s(x))' = \frac{-x}{s^2} G^s(x),$$
$$(G^s(x))'' = \frac{x^2 - s^2}{s^4} G^s(x).$$

Voor nu zijn we enkel geïnteresseerd in nulde, eerste en tweede afgeleides.

6.2 Canny Edge

Ons doel is het vinden van edges.

Definitie 6.2. Zij $f\colon \mathbb{R} \to \mathbb{R}$ een 1D-functie. Dan definiëren we een *edge* als de plek waar $|f'|$ groot is, en f'' ongeveer 0.

We noteren de partiële afgeleide naar x met f_x en naar y met f_y. Definieer nu de *gradiënt*

$$\nabla f(x,y) = \begin{pmatrix} f_x(x,y) \\ f_y(x,y) \end{pmatrix}.$$

De gradiëntvector wijst in de richting van maximale toename. De gradiëntvector hangt *niet* van keuze van basis. De gradiëntvector staat loodrecht op de niveaukrommes van f (in 2D).

Nu definiëren we op dezelfde plek *edges* in 2D (of hogere dimensies), behalve dat we moeten specificeren *in welke richting* we de afgeleide nemen. Het is geen verrassing dat we de afgeleide in de richting van de gradiënt nemen. Deze richting noteren we standaard met f_w, en dit geeft als definitie dat $|f_w|$ groot moet zijn en f_{ww} ongeveer 0.

De conditie $f_{ww} \approx 0$ kan problemen geven, en daarom wordt in de praktijk de conditie $f_w^2 f_{ww} \approx 0$ gebruikt. Dit kan nog steeds problemen geven omdat we een gediscretiseerd beeld gebruiken – in dit geval checken we niet of het beeld ergens 0 of klein is, maar of er een *zero crossing* is. Dit kan natuurlijk op verschillende manieren.

7 Image Stitching met SIFT

Zie hiervoor ook rapport.

7.1 RANSAC

De *RANSAC*-algoritme is bedacht om een specifiek probleem op te lossen. Gegeven een lijst puntcorrespondenties $(\mathbf{x}_i, \mathbf{x}'_i)$ en een model M, willen we een model opstellen dat de correspondenties schat met een projectieve matrix. Echter weten we ook dat een aantal van de puntcorrespondenties *fout* zijn en dat het beginmodel M dus fout is. We willen dus de outliers verwijderen, zonder te weten wat de outliers zijn. Dit is een 'kip en ei'-probleem dat we iteratief oplossen:

1. Kies willekeurig n punten;
2. Construeer het bijbehorende model;
3. Bepaal de *inliers* die met maximaal een of andere afstand δ van het model M afliggen; Voor de afstand is het belangrijk rekening te houden met projectieve coördinaten, dus eerst de projectieve coördinaten omzetten naar Euclidisch en dan iets van $\|x - y\|$ uitrekenen voor de fout.
4. Herhaal de vorige stappen een paar keer, en kies daarvan de beste (*beste* is ambigu, kan aantal inliers zijn);
5. Maak een nieuw model met alle inliers door de SVD-truc te gebruiken.

We hebben minstens vier puntcorrespondenties nodig om dit te laten werken, maar meer punten is beter!

7.2 SIFT

SIFT is een algoritme om twee plaatjes van dezelfde scène aan elkaar te plakken, terwijl ze onder verschillende hoeken en met verschillende belichtingscondities genomen kunnen zijn.

Globaal werkt de algoritme als volgt:

1. Bepaal in beide beelden *keypoints* punten in het overlappingsgebied van de plaatjes. Bij alle keypoints wordt ook een *schaal* s bepaald waarin ze het best zichbaar zijn (d.w.z. in het plaatje $f * G^s$); elk keypoint is dus een locatie (x, y) en een schaal s.
2. Bepaal van alle keypoints de *oriëntatie*, dan hebben we dus een keypoint (x, y, s, ϑ);
3. Bepaal van de keypoints de *descriptor*, die het histogram van de intensiteiten in een omgeving van het keypoint geeft (een grotere omgeving dan de schaal waarin het keypoint gemeten is). Nu is elk keypoint een set (x, y, s, ϑ, d);
4. Match in de twee plaatjes alle keypoints en descriptors, eerst door een of ander model op te stellen dat waarschijnlijk slecht is (met een brute-force matcher bijvoorbeeld), en daarna met RANSAC.
5. Stel de projectieve matrix op die de gematchte keypoints naar elkaar stuurt.

Voor een exacte beschrijving van alle stappen: zie de opdracht van week 4.

8 Pinhole camera(calibratie)

8.1 De interne matrix

Het simpelste model voor een camera (en voor hoe het oog werkt) is de *pinhole camera*.

We kiezen ons coördinatensysteem zodanig dat de oorsprong O in het gat ligt. De afstand tussen het gat en het projectievlak noemen we f, dus het vlak waarop geprojecteerd wordt is $Z = -f$.

Heeft een punt nu wereldcoördinaten (X, Y, Z), dan wordt deze geprojecteerd op

$$-\frac{f}{Z}(X, Y) = \left(-\frac{f}{X}, -\frac{f}{Y}\right)$$

in het $(Z = -f)$-vlak.

Het minteken hier is vervelend (komt door de spiegeling), dus we doen alsof we projecteren op het $(Z = f)$-vlak, en krijgen

$$(x, y) = \frac{f}{Z}(X, Y).$$

Om dit met projectieve meetkunde weer te geven willen we de deling door Z wegkrijgen, en dit kunnen we doen door ervoor te zorgen dat de derde coördinaat Z is. Dit geeft de projectieve transformatie

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix}$$

Nu voeren we nog drie correcties uit:

- Om over te gaan van wereldafstanden naar pixelafstanden schalen we nog met factoren s_x en s_y. De geschaalde $s_x f$ en $s_y f$ noemen we ook wel f_x en f_y;
- Om de oorprong te verplaatsen transleren we met zekere (u_0, v_0);
- We moeten ook nog een *skew factor* α toevoegen, mocht het projectievlak niet precies loodrecht op het retina staan.

Alles bij elkaar geeft dat de *interne matrix*

$$K \coloneqq \begin{pmatrix} f_x & \alpha & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

8.2 De externe matrix

Om dit te kunnen gebruiken moeten we eerst objecten van wereldcoördinaten naar cameracoördinaten omzetten. We weten dat er een rotatie R en een translatie $\mathbf{t}$ bestaat die het camerastelsel omzet naar het wereldstelsel. In homogene coördinaten geeft dit de matrix

$$\begin{pmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix},$$

die camera- naar wereldcoördinaten verplaatst.

Om een punt nu om te zetten van wereld- naar cameracoördinaten hebben we de inverse van deze matrix nodig, die gegeven wordt door

$$\begin{pmatrix} R^\top & -R^\top \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}.$$

Op een punt in wereldcoördinaten passen we dus eerst deze matrix toe en dan K, en dat geeft uiteindelijk de nieuwe coördinaten. Merk op dat

$$K \begin{pmatrix} R^\top & -R^\top \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} = \begin{pmatrix} f_x & \alpha & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R^\top & -R^\top \mathbf{t} \end{pmatrix}.$$

De rechtermatrix $\begin{pmatrix} R^\top & -R^\top \mathbf{t} \end{pmatrix}$ noemen we de *externe matrix*.

Opmerking. Het is dus *niet* direct mogelijk om de interne en externe matrix met elkaar te veermenigvuldigen: hiervoor moeten we de nulkolom van de externe matrix weghalen.

8.3 Cameracalibratie

De interne matrix K projecteert een 3D-punt $\tilde{\mathbf{X}}$ naar een 2D-punt $\tilde{\mathbf{x}}$. Hebben we al een set puntcorrespondenties, dan kunnen we op analoge wijze als in sectie 4.3 een projectieve matrix opstellen die de puntcorrespondenties zo goed mogelijk schat. Dit geeft dan een calibratiematrix.

Merk op dat deze matrix *niet* de fouten minimaliseert (de zgn. *reprojection errors*) – dit is een niet-lineair probleem.

8.4 Meer stitching

Er zijn twee gevallen waarin beelden zeer makkelijk aan elkaar te plakken zijn:

1. Als ze allebei van exact hetzelfde vlak gemaakt zijn;
2. Als de twee plaatjes door dezelfde camera gemaakt zijn, die enkel geroteerd heeft.

Als ze allebei van hetzelfde vlak gemaakt zijn, kunnen we zonder verlies van algemeenheid stellen dat ze van het $(Z = 0)$-vlak gemaakt zijn. Van de (3×4)-projectieve matrix hebben we dan dus maar drie kolommen nodig, waardoor deze matrix inverteerbaar wordt. Het is duidelijk dat we dan punten in elkaar kunnen omzetten.

Als de camera enkel geroteerd heeft, mogen we zonder verlies van algemeenheid stellen dat de camera- en wereldcoördinaten hetzelfde zijn, dus de externe matrix is $\begin{pmatrix} I & 0 \end{pmatrix}$. In het tweede plaatje is de camera enkel geroteerd en vinden we de externe matrix $\begin{pmatrix} R^\top & 0 \end{pmatrix}$.

Als we aannemen dat de camera eerst interne matrix K en dan interne matrix K' heeft (de interne matrix kan veranderen door bv. te zoomen), vinden we dus $\mathbf{x} = K\mathbf{X}$ en $\mathbf{x}' = K'R^\top\mathbf{X}$ voor het eerste en tweede plaatje. We kunnen dus punten omzetten van frame 1 naar frame 2 met de matrix $K'R^\top K^{-1}$.

Opmerking. We nemen hierbij aan dat K inverteerbaar is, dus dat we de nulkolom hebben weggehaald.

9 Beweging

Een *beweging* kunnen we simpelweg weergeven als een functie $f(x, y, t)$: voor elk tijdstip t is er een beeld. Voor elk vast tijdstip t_0 noemen we het beeld $f(x, y, t_0)$ een *frame*.

Er zijn twee soorten beweging:

1. Een beweging ten opzichte van de observeerder, waarbij alles beweegt. Hierbij spreken we van *optic flow*.
2. Een beweging van één object die we proberen te *tracken*, dan spreken we van *motion tracking*.

9.1 Optic Flow

Bij *optic flow*-algoritmes proberen we uit een beweging de snelheidsvector bij elk punt te schatten. We maken ten eerste een belangrijke aanname: als $\mathbf{x}$ in een plaatje verplaatst naar $\mathbf{x} + \mathrm{d}\mathbf{x}$, van tijd t naar tijd $t + \mathrm{d}t$, dan verandert de belichting niet, oftewel

$$f(\mathbf{x} + \mathrm{d}\mathbf{x}, t + \mathrm{d}t) = f(\mathbf{x}, t).$$

De snelheidsvector wordt gegeven door $\mathbf{v} = \mathrm{d}\mathbf{x}/\mathrm{d}t$, dus $\mathrm{d}\mathbf{x} = \mathbf{v}\mathrm{d}t$. Dit substitueren geeft

$$f(\mathbf{x} + \mathbf{v}\mathrm{d}t, t + \mathrm{d}t) = f(\mathbf{x}, t).$$

Een eerste-orde (tweedimensionale) Taylorbenadering van de linkerterm geeft

$$f(\mathbf{x} + \mathbf{v}\mathrm{d}t, t + \mathrm{d}t) \approx f(\mathbf{x}, \mathbf{t}) + (\nabla f)(\mathbf{x}, t)^\top \mathbf{v}\mathrm{d}t + f_t(\mathbf{x}, t) = f(\mathbf{x}, t),$$

met ∇f de afgeleide naar $\mathbf{x}$.

Nu links en rechts $f(\mathbf{x}, t)$ aftrekken geeft

$$(\nabla f)(\mathbf{x}, t)^\top \mathbf{v} \mathrm{d}t + f_t(\mathbf{x}, t) = 0.$$

Dit is één vergelijking in twee onbekenden, namelijk v_1 en v_2. Het is dus onmogelijk om de *optic flow* uit te rekenen zonder nog een aanname.

De Lucas-Kanada-methode maakt de volgende aanname: punten rond $\mathbf{x}$ hebben dezelfde snelheidsvector als $\mathbf{x}$. Met deze aanname kunnen we de vergelijking uitbreiden met arbitrair veel punten rond $\mathbf{x}$, zeg $\mathbf{x}_1, \ldots, \mathbf{x}_n$ (met $\mathbf{x}_1 = \mathbf{x}$). We krijgen nu n vergelijkingen in 2 onbekenden, die we kunnen oplossen met een least-squares-procedure.

Belangrijke keuzes om te maken zijn:

1. Welke punten moeten we kiezen in de omgeving?
2. Als we de fout gaan berekenen, welke punten moeten we dan het zwaarst meewegen?

9.2 Correlation Tracking

Stel dat we een object willen tracken dat zich over een scène beweegt, waarbij we alleen in het eerste frame mogen aanwijzen wat we aan het tracken zijn (een *mask* $\mathcal{M}$ met M elementen).

Wat *correlation tracking* doet is voor elk punt in het beeld bepalen hoe goed het masker $\mathcal{M}$ daar past, en dan de beste 'fit' nemen. Dit lijkt computationeel rampzalig, maar valt bijna volledig uit te drukken in convoluties.

Hoe goed het masker over één punt heen past valt uit te drukken in de *sum of squared distances*,

$$\mathrm{ssd}(\mathbf{x}) = \frac{1}{M} \sum_{\mathbf{y}} (f(\mathbf{x} + \mathbf{y}) - m(\mathbf{y}))^2.$$

Het probleem is dat belichting deze waarde nogal kan veranderen, dus we gebruiken niet f en m maar $\hat{f}$ en $\hat{m}$, waarbij $\hat{f} = \frac{f - \bar{f}}{s_f}$ en $\hat{m} = \frac{m - \bar{m}}{s_m}$ (en $\bar{f}$, $\bar{m}$ de gemiddeldes in een omgeving, s_f en s_m de standaarddeviaties). Merk op dat $\bar{f}$ en s_f afhangen van het punt $\mathbf{x}$ waar we kijken. Dit geeft

$$\mathrm{nssd}(\mathbf{x}) = \frac{1}{M} \sum_{\mathbf{y}} (\hat{f}(\mathbf{x} + \mathbf{y}) - \hat{m}(\mathbf{y}))^2.$$

Met wat herschrijven (zie lecture notes) kunnen we bewijzen dat dit neerkomt op het maximaliseren van iets anders, waarbij dat 'iets anders' volledig met convoluties kan worden uitgerekend.

10 Neurale netwerken