Labo Beeldinterpretatie 2

Nummerplaatherkenning

1 Doel

Het automatisch lezen van nummerplaten van auto's biedt mogelijkheden voor heel wat toepassingen. Niet enkel de politie heeft er baat bij, zodat hardrijders en overtreders automatisch geïdentificeerd worden, maar bijvoorbeeld ook kan dit gebruikt worden bij de toegangscontrole van private ondergrondse parkings. De slagboom gaat automatisch omhoog als de nummerplaat van een toegelaten wagen wordt herkend. Maar er zijn zeker nog veel toepassingen te bedenken van Automatic Licence Plate Recognition (ALPR), zoals de techniek technisch genoemd wordt.

ALPR zoekt eerst een nummerplaat in het camera-beeld en maakt dan gebruik van OCR-technieken (Optical Character Recognition) om de letters en cijfers ervan te lezen. Helaas is dat systeem niet honderd procent waterdicht. Het komt regelmatig voor dat een letter verkeerd gelezen wordt, bijvoorbeeld een '1' die als een 'I' wordt aanzien, of een '0' als een 'O'. De oplossing die ALPR-software gebruikt voor het corrigeren van dit soort leesfouten, is ervan uit gaan dat een nummerplaat een vooraf gedefineerde structuur heeft, zoals in ons land alle nieuwe nummerplaten de structuur '1-ABC-123' hebben. Foutief gelezen karakters kunnen gedecteerd en gecorrigeerd worden door na te gaan of ze aan deze letters-cijfers-structuur voldoen.

Helaas is het tegenwoordig ook mogelijk om een gepersonaliseerde nummerplaat te verkrijgen voor je wagen. In tegenstelling tot vroeger moet deze niet meer passen in de structuur '1-ABC-123', maar mag het over elke combinatie van maximaal 7 letters en/of cijfers gaan.

Uiteraard werkt hierdoor het hierboven geschetste correctiesysteem niet meer. In dit labo realiseren we een alternatief correctiesysteem voor nummerplaatherkenning, dat werkt zonder structuur-veronderstellingen. In plaats van de num-

merplaatkarakters uit één foto te proberen achterhalen, gaan we hier de informatie uit meerdere foto's combineren. De meeste auto's worden gefotografeerd terwijl ze rijden, dus als de camera snel genoeg is kunnen er gemakkelijk meerdere foto's van dezelfde wagen genomen worden. Hieronder zie je een voorbeeld van de foto's die een ALPR-camera maakte van een wagen die een parkeergarage binnenrijdt.

In dit labo gebruiken we standaard ALPR software (het open source pakket OpenALPR) om de nummerplaat in elk van de foto's te detecteren en de karakters ervan te lezen. De clou van de aanpak is dat we de leesresultaten van verschillende foto's combineren in een *voting*-systeem om zo tot een meer betrouwbaar resultaat te komen.

Het probleem is uiteraard dat de wagen rijdt terwijl de beelden genomen werden. De nummerplaat is op elke foto op een andere plaats in beeld zichtbaar, en heeft een verschillende grootte omdat de wagen dichterbij de camera komt. Bovendien kunnen we niet rechtstreeks de gelezen karakters met elkaar beginnen vergelijken, omdat we niet weten of in alle foto's wel heel de nummerplaat gedetecteerd is. Om een betrouwbare voting te kunnen doen per karakter, moeten we eerst meten hoeveel de nummerplaat is verschoven en vergroot tussen twee beelden. Daaruit kunnen we de beeldcoördinaten van overeenkomstige karakters van verschillende beelden op elkaar mappen en zien voor elk karakter van de nummerplaat de verschillende leesresultaten vergelijken. Hieruit kunnen we op een veel betrouwbare manier beslissen wat het juiste letter of cijfer was van elk karakter van de nummerplaat.

Om de hoeveelheid verschuiving en scalering te vinden tussen opeenvolgende beelden van de ALPR-camera, zullen we gebruik maken van *local feature matching*. Hierbij worden kleine regio's (*keypoints*) in de beelden geëxtraheerd, waarna ze met behulp van hun beschrijvingsvector vergeleken worden. Een *outlier rejection*-stap op basis van RANSAC is nodig omdat er ook verkeerde overeenkomsten gevonden worden.

In dit labo zullen we volgende stappen uitwerken:

1. Inlezen van de OpenALPR-output voor elk van de camerabeelden

2. Extraheren van *keypoints* in elk van de camerabeelden. Voor deze extractie hebben we een executable ter beschikking die een ASCII file wegschrijft.

3. Inlezen van de ASCII files met de geëxtraheerde keypoints en het uitselecteren van de keypoints die in een gebied rond de nummerplaat liggen.

4. Het zoeken van matches: corresponderende beeldkenmerken tussen opeenvolgende camerabeelden. Deze matches worden ook gevisualiseerd.

5. RANSAC: robuuste outlier-rejection, welke door een homografie te schatten tussen beide beelden foutieve correspondenties wegfiltert.

6. Mappen van de coördinaten van gedeteceerde nummerplaatkarakters op elkaar.

7. Voting: via stemming bepalen welke de juiste karakters op de nummerplaat waren.

In eerste instantie kan je beginnen met één beeldenpaar (bijvoorbeeld de twee eerste foto's). Als je dit helemaal uitgewerkt hebt, kan je ook meerdere foto's combineren om nog een hogere betrouwbaarheid te bekomen van de gelezen nummerplaattekst.

2 Inlezen van de ALPR-output

De eerste taak is het inlezen van een ASCII-file `alpr_out.txt` met de resultaten van de nummerplaatherkenning. Het OpenALPR-pakket heeft in elk van de vier testbeelden een nummerplaat gevonden. De coordinaten hiervan zijn als een rechthoek opgeslagen, waarin de OCR werd uitgevoerd om de karakters (proberen) te lezen. Het bestand `alpr_out.txt` bevat de volgende data per lijn:

- De bestandsnaam van een foto die onderzocht is, gevolgd door het aantal karakters dat in de nummerplaat gevonden zijn.
- De *bounding box* van de nummerplaat die gedetecteerd is in het beeld: x- en y-coördinaat van de linker bovenhoek van deze rechthoek, en dan achtereenvolgens de breedte en de hoogte ervan.
- Voor elk van de gevonden karakters: het gelezen ASCII-symbool, en de bounding box van het karakter (ook $[x, y, w, h]$: coordinaten linker bovenhoek en breedte en hoogte)

Lees deze tekstfile in en sla het resultaat op in een passende structuur.

Gebruik de functies uit het vorige labo om de foto's in te lezen en de gedetecteerde nummerplaat en karakters er op aan te duiden als rechthoeken.

3 Keypoints extraheren

Eén van de meest gebruikte algoritmes voor lokale beeldkenmerken is SIFT, Scale Invariant Feature Transform van David Lowe [1]. Zoals in de lessen uitgelegd is de kracht van deze en andere methodes dat in elk beeld apart wordt

gezocht naar lokale beeldkenmerken, zogenaamde *keypoints* bij SIFT. De visuele inhoud van elk keypoint wordt gecodeerd door middel van een *beschrijvingsvector*, welke onafhankelijk is van belichtingsinvloeden en robuust tegen veranderingen in de kijkhoek.

Door de euclidische afstand tussen twee beschrijvingsvectoren te berekenen, kan de visuele similariteit bepaald worden tussen twee keypoints. Met andere woorden, dit geeft informatie over hoe erg de keypoints op elkaar lijken. Deze 'matching'-stap wordt in de volgende sectie 5 beschreven.

3.1 Extractie

Maar eerst moeten de features geëxtraheerd worden. Hiervoor biedt David Lowe [2] gelukkig een kant-en-klare executable aan die we kunnen gebruiken. Als je dit downloadt kan je met het commando `./sift <image.pgm >keypoints.key` voor elk beeldje `image.pgm` (in *pgm*-formaat) SIFT-features berekenen. De output is een textbestand `keypoints.key`, waarin de keypoint-extractieresulaten vastgelegd zijn.

Om voor jullie het werk een beetje te reduceren, hebben we dit voor elke foto die bij dit labo hoort al gedaan. Je zal dus voor elke `*.pgm` een `*.key` vinden met de keypoints die in dat beeldje gevonden zijn.

3.2 Inlezen keypoints-file

De in de vorige stap gegenereerde ASCII-file `keypoints.key` bevat de detectieresultaten: een lijst met keypoints, inclusief hun beschrijvingsvector. Open zo'n file om de inhoud te bekijken. Het bestandsformaat bestaat uit:

- 2 integers die het totaal aantal gevonden keypoints en de grootte van de beschrijvingsvectoren (meestal 128) aangeven
- voor elk keypoint:
  - 2 floating point getallen met de locatie van het keypoint in het beeld: subpixel rij- en kolomwaarden
  - 1 floating point met de schaal (grootte) van het keypoint
  - 1 floating point met de orientatie van het keypoint (in radialen van $-\pi$ tot π)
  - De beschrijvingsvector als een reeks integers in het bereik [0,255].

Een functie die deze gegevens inleest van die file en de gegevens van elk keypoint in een gelinkte lijst van structs opslaat is gegeven in de hulpbestanden `util2.c` en `util2.h`:

```
Keypoint ReadKeyFile(char *filename)
```

Schrijf een programma dat de keypoint-files inleest van de testfoto's.

4 Uitselecteren van de keypoints rond de nummerplaat

Omdat enkel de wagen verschuift in het beeld en de achtergrond stilstaat, moeten we natuurlijk enkel de verschuiving en vergroting van de wagen meten daarom gaan we in de geëxtraheerde keypoints een selectie doen. Enkel de keypoints die in een bounding box rond de gedetecteerde nummerplaat liggen, houden we over. De grootte van die bounding box is experimenteel vast te stellen. Die zou zo groot mogelijk zijn, zodat zo veel mogelijk visuele elementen van de voorzijde van de wagen worden meegenomen, maar niet te groot zodat er achtergrond-keypoints mee in zitten. Een voorbeeld is de bounding box drie maal zo breed is als de nummerplaat te kiezen en vijf maal zo hoog.

Voer deze selectie uit op de ingelezen keypoints, op basis van de coördinaten van de door de ALPR-software gevonden nummerplaat. Enkel de keypoints in een ruime bounding box omheen de nummerplaat moeten overgehouden worden om mee verder te werken.

5 Correspondenties vinden

Wanneer in twee verschillende beelden keypoints zijn geselecteerd, kunnen er d.m.v. de beschrijvingsvectoren correspondenties gevonden worden. We gaan dus elk keypoint uit het eerste beeld vergelijken met alle keypoints uit het tweede beeld. Een 'match' wordt gevonden wanneer de afstand minimaal is en de tweede beste afstand genoeg verschilt.

In formulevorm: Als k_i^1 een keypoint is in beeld I_1 en k_j^2 een keypoint in beeld I_2, zoeken we de beste en de tweede beste afstand:

$$d_{beste} = \min_{\forall j \in I_2} d(k_i^1, k_j^2) \tag{1}$$

$$d_{tweede} = \min_{\forall l \in I_2 | l \neq j} d(k_i^1, k_l^2) \tag{2}$$

Nu beschouwen we k_i^1 en k_j^2 als corresponderend enkel als:

$$\begin{cases} d(k_i^1, k_j^2) = d_{beste} \\ \frac{d_{beste}}{d_{tweede}} \leq T_h \end{cases} \tag{3}$$

T_h is een vaste threshold, bijvoorbeeld 0.7.

De afstand $d(\cdot, \cdot)$ in bovenstaande formules is gedefinieerd als de eulidische[1] afstand:

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_i (x_i - y_i)^2} \tag{4}$$

Implementeer bovenstaande methode om correspondenties tussen de twee beelden te vinden.

[1] Voor een optimale implementatie is het misschien niet nodig elke keer die vierkantswortel uit te rekenen. In de plaats kan je met gekwadrateerde afstanden werken. Zorg dan uiteraard wel voor een juiste implementatie van de threshold T_h.

6 Correspondenties visualiseren

Als test van het matching-algoritme, en te experimenteren met de threshold T_h, is het handig als de gevonden matches gevisualiseerd worden. Hiervoor zullen we een nieuw beeldje aanmaken waarin beide foto's boven elkaar getoond worden. De gevonden matches worden dan aangeduid door rechte lijnen te trekken tussen corresponderende keypoints. Fig. 1 toont een voorbeeld.

Figuur 1: Voorbeeld output visualisatie van correspondenties (enkel de bounding box rond de nummerplaat is hier getoond).

De hulpfile `util2.h` groepeert een aantal handige functies om deze visualisatie te verwezenlijken:

- `Image ReadPGMFile(char *filename)`
- `Image CombineImagesVertically(Image im1, Image im2)`
- `void DrawLine(Image image, int r1, int c1, int r2, int c2)`
- `void WritePGM(FILE *fp, Image image)`

Lees het commentaar bij deze functies om te weten hoe je ze moet gebruiken.

Pas dit nu toe: pas je programma aan zodat zowel de keypointfiles als de foto's zelf worden ingelezen. Combineer beide foto's vertikaal en teken er lijnen op die overeenkomstige pixels verbinden, een lijn voor elk van de in sectie 5 berekende correspondenties. Fig. 1 geeft een voorbeeld.

7 RANSAC outlier rejection

Als al het bovenstaande werkt, blijkt, zoals ook in fig. 1, dat er naast juiste correspondenties nog een relatief groot aantal foutieve matches zijn gevonden. Deze zogenaamde outliers zijn te verklaren omdat enkel op lokale schaal wordt

vergeleken. En inderdaad, sommige structuren zien er zeer gelijkaardig uit, alhoewel ze niet fysisch hetzelfde object zijn. Zeker bij nummerplaten met repetitieve karakters is dit probleem niet te vermijden.

Gelukkig kunnen we ook het *globale* aspect in rekening brengen. We weten immers dat tussen de twee foto's de geometrische transformatie enkel bestaat uit een translatie, en een schalering. Alle juiste correspondenties moeten (ongeveer) aan diezelfde transformatie voldoen.

Met dit feit in het achterhoofd kan een RANSAC-methode uitgewerkt worden om die onbekende transformatie te schatten. Deze *Random Sample Consensus* gaat als volgt:

1. Neem een willekeurig sample van 2 correspondenties
2. Bereken de geometrische transformatie van deze correspondentieset
3. Test hoeveel correspondenties van de gehele set aan deze transformatie voldoen (de *support*)
4. Herhaal vanaf stap 1 voor een andere sample
5. Selecteer de transformatie met de hoogste support, m.a.w. die waar de meeste correspondenties aan voldoen
6. Dit bepaalt de inliers en de transformatie

Stap per stap zullen we dit algoritme uitwerken in dit labo.

7.1 Theoretische afleiding van de geometrische transformatie

Beide foto's zijn genomen vanop dezelfde plaats naast de weg, maar de wagen is naar de camera toe gereden. De auto vershuift daardoor in het beeld en wordt groter. Hierdoor weten we dat er een transformatie bestaat tussen beide, die bestaat uit een schaling (met factor s) en een translatie (over $\vec{t} = (t_x, t_y)$). We kunnen dit wiskundig vatten in een vectoroperatie:

$$\vec{x'} = s \cdot \vec{x} + \vec{t} \tag{5}$$

waarin $\vec{x}$ de coördinaten zijn van een pixel in het eerste beeld, en $\vec{x'}$ die van een pixel in het tweede beeld. Als we de vectoren uitschrijven wordt dit:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = s \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{6}$$

Met behulp van homogene coördinaten kunnen we dit schrijven als een matrixvermenigvuldiging:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{7}$$

In de 3×3-matrix in vergelijking 7, de homografie H, staan dus drie onbekenden (s, t_x en t_y). Stel dat de SIFT-methode een correspondentie vond

tussen (x_1, y_1) in het eerste beeld en (x'_1, y'_1) in het tweede. Als we de matrixvergelijking nu uitschrijven voor dit correspondentiepaar krijgen we:

$$\begin{cases} x'_1 &= sx_1 + t_x \\ y'_1 &= sy_1 + t_y \\ 1 &= 1 \end{cases} \tag{8}$$

De vergelijking $1 = 1$ zegt natuurlijk niet veel, maar met de twee andere kunnen we wel verder. Dit zijn al twee vergelijkingen om de drie onbekenden te vinden. Als we dezelfde vergelijkingen voor een tweede correspondentiepaar $(x_2, y_2) \leftrightarrow (x'_2, y'_2)$ toevoegen, krijgen we een oplosbaar stelsel, waar we zelfs maar drie van de vier vergelijkingen moeten gerbuiken om de drie onbekenden te berekenen:

$$\begin{cases} x'_1 &= sx_1 + t_x \\ y'_1 &= sy_1 + t_y \\ x'_2 &= sx_2 + t_x \\ y'_2 &= sy_2 + t_y \end{cases} \tag{9}$$

Hiervan zullen we de bovenste drie vergelijkingen gebruiken voor het bepalen van de drie onbekenden (s, t_x en t_y). Deze kunnen we omzetten in matrixvorm $AX = B$:

$$\begin{bmatrix} x_1 & 1 & 0 \\ y_1 & 0 & 1 \\ x_2 & 1 & 0 \end{bmatrix} \begin{bmatrix} s \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \end{bmatrix} \tag{10}$$

Dit stelsel geeft aan hoe we in de RANSAC-methode vanuit twee correspondentieparen de parameters van de geometrische transformatie kunnen berekenen.

7.2 Geometrische transformatie berekenen

RANSAC, Random Sample Consensus, is robuust tegen *outliers*. In dit geval zijn dat de foutieve correspondenties die de SIFT-methode oplevert. De oplossing bestaat erin om herhaaldelijk een willekeurig sample te nemen van enkele correspondenties, hieruit de geometrische transformatie te berekenen en deze laatste te 'checken' bij de andere correspondenties. Zo wordt er hopelijk ooit een sample getrokken dat enkel uit juiste correspondenties bestaat, en waarvan de transformatie dan ook bij een (relatief) groot deel van de andere correspondenties klopt.

In dit geval kunnen we, zoals in de vorige sectie afgeleid, vanuit twee correspondentieparen de geometrische tranformatieparameters berekenen. Je RANSAC-implementatie zal er dus in bestaan om herhaaldelijk twee willekeurige matches uit te kiezen uit de set die de vorige stap opleverde.

Voor elke set van twee correspondentieparen kan met stelsel 10 dan een homografie H berekend worden. Het oplossen van de onbekenden (s, t_x en t_y) uit dit lineair stelsel kan eenvoudig gebeuren met de Gaussiaanse Eliminatiemethode. Om jullie te helpen staat de functie `LinearEquationsSolving()` die deze methode implementeert in de hulpfile `util2.h`. Bekijk goed de bijgesloten commentaar voor een juist gebruik.

7.3 Support berekenen

Om te weten hoe goed de net berekende homografie is, moeten we deze nakijken voor alle in de vorige stap berekende correspondenties. Het aantal correspon-

denties waarvoor de homografie klopt, is de *support* ervan. Uiteraard kiezen we als uiteindelijke oplossing de homografie met de grootste support.

Voor elke correspondentie $(x, y) \leftrightarrow (x', y')$ zou dus deze vergelijking nagekeken moeten worden:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{11}$$

Wegens afrondingsfouten en ruis, die altijd in de beelden aanwezig is, kan bovenstaande vergelijking nooit exact gelden. Daarom berekenen we de fout en kijken of deze kleiner is dan een drempelwaarde:

$$\sqrt{(\tilde{x}' - x')^2 + (\tilde{y}' - y')^2} < \epsilon \tag{12}$$

met

$$\begin{cases} \tilde{x}'_1 &= sx_1 + t_x \\ \tilde{y}'_1 &= sy_1 + t_y \end{cases} \tag{13}$$

De drempelwaarde ϵ is dus een maat voor de foutenmarge op de geometrische tranformatie, uitgedrukt in pixels. Zoek experimenteel een goede waarde hiervoor.

7.4 Itereren

De support van een berekende set homografie-parameters (s, t_x en t_y) is dus het aantal correspondenties waarvoor de fout voldoende klein is. Bij RANSAC gaan we herhaaldelijk de hoger beschreven stappen herhalen en de oplossing behouden die de beste support had.

De vraag rest nog hoeveel maal deze sample-en-berekenstap herhaald moet worden. We kunnen twee stopcriteria definiëren:

- Wanneer tijdens de berekeningen opeens een oplossing opduikt met een bijzonder goede support, kunnen we stoppen en deze oplossing houden. Hiervoor hebben we een threshold nodig op de support, bijvoorbeeld 40% van het totaal aantal correspondenties.
- We kunnen statistisch berekenen hoeveel iteraties we nodig hebben om een outlier-vrije sample te trekken ($N = \frac{\log(1-P_{\text{outlier}})}{\log(1-(1-P_{\text{outlier}})^{\text{samplesize}})}$). Bij een samplegrootte van 2 (zoals in ons geval), en als er 50% outliers zijn, hebben we bijvoorbeeld 99% kans om een sample getrokken te hebben zonder outliers na 17 trekkingen.

Implementeer beide stopcriteria.

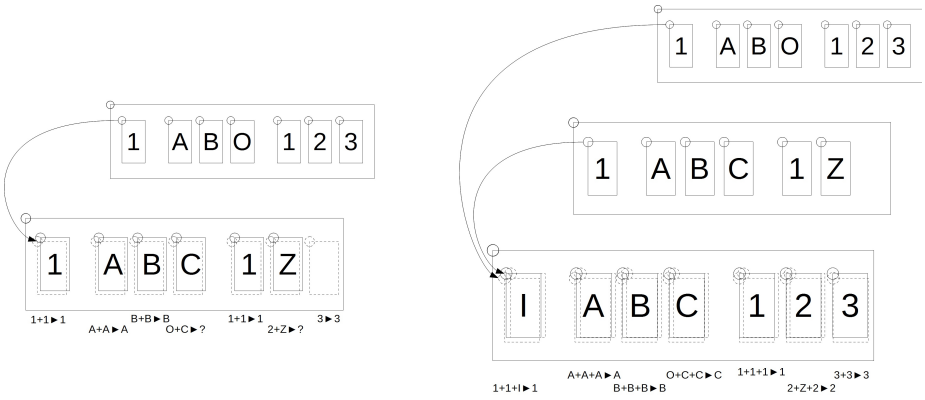
7.5 Visualisatie

Wanneer de RANSAC-lus afgelopen is, zal er een bepaalde set van homografie-parameters als oplossing naar voren komen. We hebben nu trouwens ook een zicht op welke SIFT-correspondenties correct waren, en welke foutief. Dat wordt bepaald door welke correspondenties voldoen aan vergelijking 12 voor deze homografie. Dit zijn de *inliers*.

Om het programma te debuggen en om de thresholds in te stellen laat je nu, net zoals in sectie 6, de correspondenties zien op een grafische weergave. Nu laat je uiteraard enkel de inliers zien.

8 Lezen van de nummerplaat door voting

Nu we de geometrische transformatie hebben berekend die de auto aflegde tussen twee beelden, kunnen we de verschillende door de ALPR gelezen karakters op elkaar mappen en proberen te achterhalen welke letters juist waren en welke foutief. Dit wordt geïllustreerd in onderstaande figuur.



Figuur 2: *Lezen van een nummerplaat door voting. Links: vanuit twee beelden, rechts: vanuit drie beelden.*

Eerst gaan we de coördinaten van de karakters van de nummerplaat in de eerste foto terugrekenen naar overeenkomstige coördinaten in de tweede. Daartoe kan je bijvoorbeeld gewoon formule 7 gebruiken.

Een tweede stap is dan het bepalen welke karakters met elkaar zouden moeten overeenkomen. Nu beide nummerplaten in beeldcoördinaten van de tweede foto zijn uitgedrukt, moeten we enkel op zoek gaan naar karakters van beide foto's waarvan de coördinaten samenvallen. Wegens ruis zal het weeral niet altijd volledig kloppen, zodat we ook hier een drempelwaarde op de afstand moeten toepassen, net zoals in formule 12.

Voor karakters die op overeenkomstige plaatsen zijn gevonden, gaan we dan een voting uitvoeren. Als meerdere keren hetzelfde karakter gelezen werd, dan is het waarschijnlijker dat dit het juiste was. Indien de verschillende lezingen van elkaar verschillen dan zal je moeten tellen welke het meest voorkomt[2]. Let op! Het kan ook zijn dat er een karakter niet werd gevonden in één van de foto's, iets waar je ook rekening mee dient te houden.

Referenties

[1] David G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", International Journal of Computer Vision, 2004

[2] `http://www.cs.ubc.ca/~lowe/keypoints/`

[2]Als je maar twee foto's gebruikt, dan kan je natuurlijk bij zo'n conflict geen uitsluitsel geven. Als je meerdere foto's combineert lukt dat wel.