Objectclassificatietechnieken voor het tellen van mijten

Nils De Schepper

June 7, 2013

Voorwoord

Voor ik met deze masterproefscriptie start zou ik graag een aantal personen willen bedanken voor het mogelijk maken van deze masterproef. Beginnende met mijn promotor Kristof Van Beeck, voor de steun en het advies bij de het uitwerken van deze masterproef. Verder wil ik hem en Toon Goedemé ook nog bedanken voor het helpen de opnames van de mijten te maken. Ook wil ik Steven Puttemans bedanken voor de hulp bij het configureren van Visual Studio en voor de hulp bij het annoteren van de mijten.

Verder wil ik ook mijn familie bedanken, zij hebben mij tijdens mijn studies altijd gesteund en hebben het mogelijk gemaakt voor mij om deze studies te beginnen.

Tot slot wil ik graag mijn vriendin, Shana Van Eynde bedanken voor het nalezen van deze tekst en het mee bekijken van de duizenden foto's van de mijten.

©2013, Nils De Schepper

De auteur geeft de toelating deze tekst op papier en digitaal voor consultatie beschikbaar te stellen en delen ervan te kopiëren voor eigen gebruik. Elk ander gebruik valt onder de strikte beperkingen van het auteursrecht. In het bijzonder wordt gewezen op de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van tekstdelen.

Abstract

In deze masterproeftekst zullen we objectclassificatietechnieken bespreken met als doel het detecteren van mijten. Deze masterproef loopt onder EAVISE in opdracht van Biobest. Biobest is een bedrijf dat gespecialiseerd is in de productie en commercialisatie van mijten voor biologische en geïntegreerde gewasbescherming. Momenteel tellen zij de mijten handmatig, maar zij zouden dit graag automatiseren en hiervoor doen zij een beroep op EAVISE.

We gaan de Amblyseius californicus mijt proberen te detecteren, dit is één van de vele mijten die Biobest kweekt, verder gaan we enkele classificeerders bespreken. We bespreken de methode van Viola & Jones waarvan we de cascade structuur gaan gebruiken om onze classificeerders te trainen. Vervolgens bespreken we Local Binary Patterns (LBP), deze techniek gaan we gebruiken om onze classificeerders te trainen. We bespreken ook de grammar models van Felzenszwalb, waaruit we de Histogram of Oriented Gradients gaan gebruiken. Dit is een populaire techniek net als LBP. Initieel dachten we ook gebruik te maken van spectrale beeldvorming, meer bepaald de golflengte-scan methode en compromis methode, maar de nodige apparatuur bleek hiervoor niet voor handen.

Om de classificeerders te trainen hebben we beelden nodig. Deze zijn we bij Biobest gaan opnemen. Nu deze beelden moeten bewerkt worden om de classificeerders te trainen. We gaan in de opgenomen beelden de mijten handmatig aanduiden, deze gaan we vervolgens uitknippen. Vervolgens gaan we de mijten roteren zodat ze dezelfde oriëntatie hebben, dit om de beste resultaten te bekomen met de classificeerders.

Vervolgens gaan we de classificeerder trainen, waarbij een aantal parameters specifiëren. Deze parameters bepalen hoe goed de classificeerder werkt. Dan gaan we deze classificeerders testen op een aantal afbeeldingen. Dit levert heel veel detecties op, we gaan bij deze detecties kijken naar het aantal detecties die elkaar overlappen en gaan vervolgens dan itereren over het aantal overlappen totdat we slecht één detectie overhouden. Hierbij bemerkten we direct dat de LBP classificeerder het er veel beter vanaf brengt dan de HOG classificeerder.

We eindigen met enkele precision-recall curves waarin we zien dat het aanpassen van de trainingsparameters van de classificeerder, kan leiden tot betere resultaten. In de toekomst kunnen de classificeerders nog verbeterd worden door een groter aantal positieve en negatieve beelden in te trainen met meer variatie. Momenteel kunnen ook alleen rechtopstaande mijten gedetecteerd worden, dit kan ook uitgebreid worden.

Short Summary

In this master thesis we are going to describe the object categorization for the detection of predator mites. We are going to describe the background of this thesis first then we will study some literature on the subject. We will make our own classifier and describe how we'll do so. Finally we will conclude this thesis with a conclusion.

Background

This thesis subject was proposed by the company Biobest and is conducted under the supervision of EAVISE. EAVISE is a research group at Campus De Nayer from Thomas More. Their research goal is implementing advanced image processing applications on embedded systems. Biobest is a company that is specialized in the production and commercialization of useful insects and mites for biological and integrated crop protection. At the moment they detect and count the amount of mites they produce manually. This is a very inaccurate and time consuming process. It's because of this that Biobest appeals to the help of EAVISE to automate this process and that's where this thesis takes place. In this thesis we will use one of the many predator mites Biobest uses, namely the Amblyseius californicus.

Literature

In this section we will describe some classifiers. Among which are the LBP and HOG classifiers. We also described spectral imaging, but we didn't have the appropriate material to make spectral images.

HOG

The main idea behind the histogram of oriented gradients (HOG) is that the appearance and the shape of a local object in an image can be described by calculating the intensity gradients or the direction of the edges. This is implemented by dividing the image in small adjacent regions, called cells, and then compiling a histogram of gradient directions or edge orientations for the pixels in the cells. The combination of these histograms then represents the classifier.

LBP

With the Local Binary Pattern (LBP) it is possible to describe two dimensional textures with to complementary factors: on one hand you have the local spatial patterns and on the other you have the difference in gray scales. The LBP operator forms labels for the pixels in an image by thresholding the n × n matrix with the center pixel value and considering the result as a binary number. This can be presented with the following formula:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^p \qquad s(x) = \begin{cases} 1 & \text{,if } x \geq 0; \\ 0 & \text{,else.} \end{cases}$$

Where P is the amount of pixels, R is the size of the matrix, g_p are the values of the surrounding pixels and g_c is the center pixel, for which we are calculating the LBP binary code.

Classifier

To train the classifier we need images, we went to Biobest to obtain these images. These images need to be adapted to train the classifier. First we need to manually detect the mites, once detected we crop the image so that it only contains the mite. After this we rotate the images so they all have the same orientation, this is needed to get the best results with the classifier.

After we adapted the images we can start with the training of the classifier, where we can specify a number of parameters. These parameters determine how good our classifier will be. Once trained, we test our classifiers on a number of images. This yields a lot of detections, among these detections we are going to check how many of these detections overlap. Next we iterate the number of overlaps until we only have one detection left. With these tests we noticed that the LBP classifier scores a lot better than the HOG classifier. We also noticed that a lot of background is detected as mites. In figure 1 we see this for the HOG classifier and in figure 2 for the LBP classifier.

Evaluation

To evaluate the classifiers we produced precision-recall curves. We calculate precision and recall with following formulas:

$$Precision = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$Recall = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Where true positives are the number of correctly labeled data, false positives are the number of incorrectly labeled data and false negatives is the number that is supposed

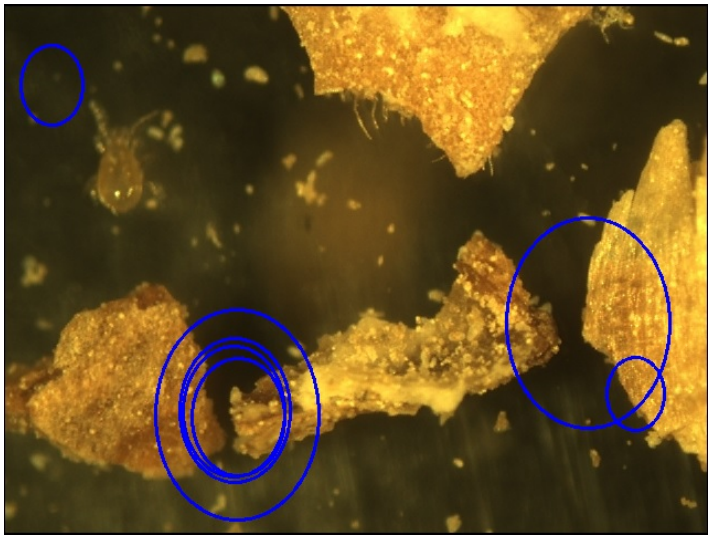


Figure 1: *Testing the HOG classifier, number of overlaps is 0.*

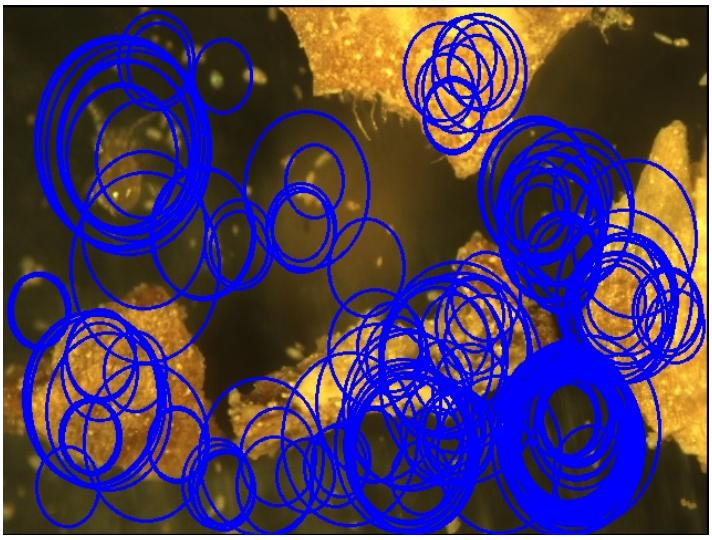


Figure 2: *Testing the LBP classifier, number of overlaps is 0.*

to be labeled but isn't. In figure 3 we see a precision-recall curve. The red curve is a standard curve and in the green we've added a larger amount of negative trainingsimages. Here we see that adding a larger amount of negative trainingsimages is very good for the classifier.

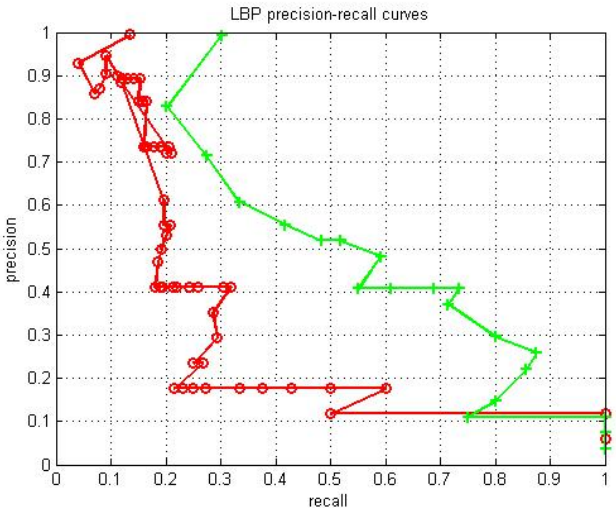


Figure 3: *Influence of changing the amount of negative trainingsimages.*

Conclusion

We conclude with some precision-recall curves where we can see that the adjustment of the training parameters of the classifier can lead to better results. In future work we can improve the classifier by using a larger number of positive and negative examples with more variation to train the classifier. At the moment we can only detect mites in the upright position, this can also be extended to full detections.

Inhoudsopgave

Lijst van figuren

Hoofdstuk 1

Inleiding

1.1 Algemene probleemstelling

Deze masterproef wordt uitgevoerd onder EAVISE in opdracht van Biobest, die in hoofdstuk 2 verder kort besproken zullen worden. In deze masterproef is het de bedoeling om de telling van roofmijten te automatiseren. Momenteel gebeurt deze telling handmatig waarbij een werknemer de roofmijten onder een microscoop moet proberen te tellen. Dit is geen eenvoudige opdracht omdat de roofmijten samen met andere mijten gekweekt worden. Deze andere mijten dienen als voedsel voor de roofmijten en mogen dus niet meegeteld worden bij het roofmijtenaantal. Nimfen, wat mijten zijn in niet volwassen stadia, moeten wel bij het roofmijtenaantal geteld worden. Dit brengt ongetwijfeld onnauwkeurigheden met zich mee, maar is vooral een zeer tijdsintensief proces.

Biobest maakt gebruik van verschillende soorten roofmijten, waaronder Phytoseiulus persimilis, de Amblyseius californicus enz. In de eerste fase van deze masterproef was het de bedoeling om de detectie van de Phytoseiulus persimilis mogelijk te maken en dan in een tweede fase het proces voor andere mijten te optimaliseren. In een latere fase gaf Biobest de voorkeur aan dat we met de Amblyseius californicus zouden werken.

1.2 Organisatie van deze tekst

Na deze inleiding gaan we verder met paragraaf 2, hierin schetsen we waar de masterproef zich situeert en gaan we de doelstellingen van deze masterproef bepalen.

In hoofdstuk 3 bespreek ik met welke mogelijke technieken ik de automatisatie van de telling zou kunnen realiseren en maak ik een afweging van wat in dit geval de beste keuze zou zijn.

Verder zal ik in paragraaf 3.4 spreken over welke programmeeromgeving ik ga gebruiken. Mijn kennis hierover heb ik in Zweden opgedaan door de cursus Advanced Programming

in C++[1] te volgen, waardoor ik ondertussen de basis en iets gevorderde topics van C++ onder de knie heb.

In hoofdstuk 4 bespreek ik kort even de twee populaire technieken, HOG en LBP, als kenmerktype om de classificeerders te trainen. Deze technieken zijn ook aangehaald in de paragraaf 3.2 die over objectdetectie gaat.

Hoofdstuk 5 bespreekt alle stappen die nodig zijn om een classificeerder te trainen.

Hierna gaan we in hoofdstuk 6 een aantal classificeerders vergelijken en bekijken welke parameters bij het trainen van deze classificeerders het meeste invloed hebben op de correctheid van de detecties.

Tot slot gaan we in het besluit de belangrijkste elementen weergeven en een antwoord geven op de vragen die in de doelstelling gesteld zijn.

[1] https://www.ida.liu.se/ TDDD38/

Hoofdstuk 2

Situering en doelstelling

2.1 Situering

2.1.1 EAVISE

EAVISE is een onderzoekscel en staat voor Embedded Artificially Intelligent Vision Engineering. Zij zijn een onderdeel van de EmSD (Embedded Systems Design) onderzoeksgroep aan Campus De Nayer van de onderwijsinstelling Thomas More. Hun visie en artificiële intelligentie onderdelen worden geassocieerd met respectievelijk VISICS en DTAI van de K.U. Leuven. Hun onderzoeksdoel is het implementeren van geavanceerde beeldverwerkingsapplicaties op embedded systemen. Dit door gebruik te maken van de aller nieuwste technieken en algoritmes, die ze zelf ontwikkeld hebben of die van hun universitaire partners komen. Door deze technieken te implementeren op embedded systemen willen ze een eerste stap zetten richting geavanceerde gebruikersapplicaties [7].

2.1.2 Biobest

Biobest [8] is een bedrijf, op internationaal niveau, dat onder andere gespecialiseerd is in de productie en commercialisatie van nuttige insecten en mijten voor biologische en geïntegreerde gewasbescherming. Voor de productie van mijten hebben zij een beroep gedaan op EAVISE om een systeem te ontwikkelen om roofmijten te tellen. Zoals in de inleiding is vermeld, gebeurt dit momenteel nog handmatig en brengt dit onnauwkeurigheden met zich mee. Verder is dit een zeer tijdsintensief proces. Biobest maakt gebruik van verschillende soorten roofmijten, waaronder Phytoseiulus persimilis, de Amblyseius californicus (die in paragraaf 3.1 kort wordt besproken) enz.

2.2 Doelstelling

Er zijn heel wat roofmijten waaronder de Phytoseiulus persimilis, de Amblyseius californicus, enz. waarbij er heel wat verschil zit in uiterlijk. De mensen van Biobest hadden reeds een poging ondernomen om de telling te automatiseren door gebruik te maken van een kleuringstechniek, maar dit is mislukt. Biobest maakt gebruik van twee soorten tellingen, natte en droge tellingen. Bij de natte telling is er heel weinig verschil tussen de draagstof en de mijt zelf, bij de droge telling is dit verschil duidelijker en daarom hebben wij geopteerd om hiervoor te kiezen bij onze detecties. Buiten het feit dat deze roofmijten samen zitten met andere mijten die als voedsel dienen, moeten ook de nimfen (mijten in niet volwassen stadia) meegeteld worden. Aan de hand van de eerste beelden die we verkregen hadden was het de bedoeling om de Phytoseiulus persimilis als eerste te detecteren, maar na overleg met de mensen van Biobest is beslist om te werken met de Amblyseius californicus. Dit zal gebeuren door gebruik te maken van bestaande detectie algoritmes die ik zal aanpassen zodat ze geschikt zijn om de roofmijten te tellen. Dit kunnen we opsplitsen in enkele onderzoeksvragen.

2.2.1 Onderzoeksvragen

Deze doelstelling kan opgesplitst worden in enkele onderzoeksvragen.

Onderzoeksvragen met betrekking tot efficiëntie:

- Met welke efficiëntie kan de detectie gebeuren?
- Welk effect heeft de draagstof vermiculiet op de telling (beperking op zichtbaarheid van de roofmijten)?

Onderzoeksvragen met betrekking tot tijd:

- Is het mogelijk om dit proces real-time te laten verlopen?
- Indien niet real-time: met welke vertraging kan de detectie dan verlopen?

Onderzoeksvragen met betrekking tot detectie:

- Welke kenmerken kunnen we best gebruiken om de roofmijt te detecteren?
- Welke detectiemethode kunnen we best gebruiken om de roofmijt te detecteren?

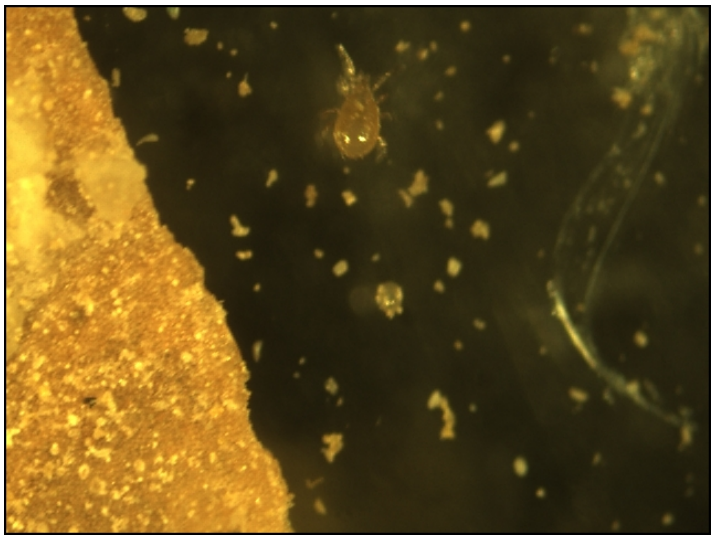
Hoofdstuk 3

Literatuurstudie

In deze literatuurstudie ga ik eerst uitleg geven over de Amblyseius californicus, die zoals aangehaald in de inleiding en de doelstelling in mijn masterproef als kernelement zal beschouwd worden. Verder zal ik objectdetectie en spectrale beeldvorming bespreken. Ten slotte zal ik de literatuurstudie beëindigen met welke techniek volgens mij het best kan gebruikt worden om het tellen van de Amblyseius californicus te automatiseren.

3.1 Amblyseius californicus

Deze roofmijt is oorspronkelijk afkomstig vanuit Californië en Florida. Ze wordt gebruikt als biologische bestrijding van spint en kunnen gebruikt worden op aardbeien, citrus en sierplanten [8]. De vrouwtjes zijn ongeveer 0.1 mm lang en ovaal en de mannetjes zijn iets kleiner. De volwassen mijt is half doorzichtig en licht oranje van kleur, zoals te zien is in figuur 3.1.



Figuur 3.1: *Een volwassen roofmijt.*

De mijt doorloopt vijf stadia in zijn leven: een wit eitje van zo'n 0.04 mm groot, het larvestadium, gevolgd door het protonimfe-, het deutonimfe- en tenslotte het adulte stadium. De hele reproductie cyclus varieert van 10 dagen voor de roofmijt en 16 dagen voor de spintmijt bij 21°C en bij 30°C duurt het 5 dagen voor de roofmijt en 7 dagen voor de spintmijt. De levenstijd van de roofmijt bedraagt ongeveer 20 dagen. Het vrouwtje legt hierbij tussen de twee en de vier eitjes per dag. De roofmijt eet het liefst de eitjes en de nimfen van de spintmijt. Tijdens een dag van 26°C kan één roofmijt ongeveer 11 á 12 eitjes en nimfen van de prooimijt opeten. Een vrouwtje kan over haar volledige levensduur ongeveer 156 mijten opeten [9]. In figuur 3.2 kan u zien hoe de roofmijt een prooimijt aanvalt.

Deze mijten hebben een lange actieve periode doordat ze kunnen overleven op alternatieve voedselbronnen, ze zijn niet kannibalistisch maar ze consumeren wel verschillende soorten van prooimijten en ze zijn resistent tegen invloed van chemicaliën.

Figuur 3.2: *De roofmijt die de spintmijt aanvalt [1].*

3.2 Objectdetectie

Objectdetectie is het detecteren van een object in een beeld. Bij de objectdetectiemethodes, waarover ik in de volgende paragrafen meer ga vertellen, maakt men gebruik van een soort venster dat over het beeld beweegt, sliding-window genaamd.

Telkens wanneer het venster zich verplaatst wordt opnieuw de objectdetectiemethode toegepast op dat deel van de foto dat zich hier binnen bevindt. Het formaat van het venster kan gekozen worden en zal vaak in verschillende groottes op één foto toegepast worden, een voorbeeld hiervan kan je zien in figuur 3.3. Wanneer er binnen het venster voldaan wordt aan de eigenschappen die door het objectdetectie algoritme getest worden, dan volgt er een detectie.

Figuur 3.3: *Sliding-window [2].*

Deze methodes maken gebruik van een zelf lerend algoritme. Waarbij we het algoritme eerst moeten trainen. Dit gebeurt door beelden te gebruiken waarvan we zeker zijn dat het object dat we willen detecteren erop staat en beelden waarvan we zeker zijn dat het object er niet op staat. Dit worden positieve en negatieve voorbeelden genoemd. Bij de positieve voorbeelden wordt dan aangeduid waar het object zich bevindt, om zodoende een classificatie te maken. In de volgende paragrafen gaan we Viola & Jones, LBP en HOG besprekn

3.2.1 Viola & Jones

De methode ontwikkeld door Paul Viola en Michael Jones [3] is een methode om objecten, oorspronkelijk voornamelijk gezichten, te detecteren. De methode is zelf lerend, kan beelden snel verwerken en behaalt een hoge detectiegraad. Deze methode maakt gebruik van 3 belangrijke kenmerken. Het eerste kenmerk is dat ze het beeld op een andere manier voorstellen, het integraal beeld. Het tweede kenmerk is om een classificeerder te maken door een klein aantal belangrijke features te selecteren met behulp van Adaboost. Het derde kenmerk is een methode om succesvol meer complexe classificeerders te combineren in een cascadestructuur dat de snelheid van de detector verhoogt door meer the focussen op regio's waar de kans groter is om een detectie te doen. Deze 3 kenmerken zal ik nu iets uitgebreider bespreken.

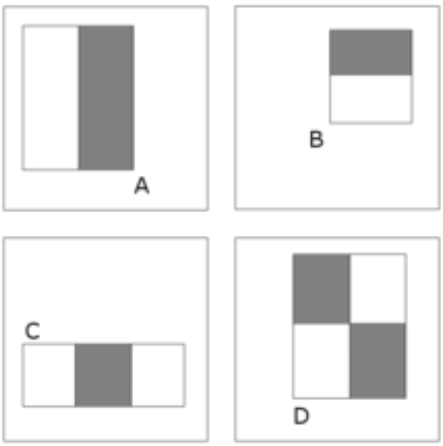
3.2.1.1 Integraal beeld

In plaats van rechtstreeks gebruik te maken van de pixels van een afbeelding, wordt er gebruik gemaakt van eenvoudige kenmerken. De redenen hiervoor zijn dat het sneller is en dat de kenmerken gebruikt kunnen worden om ad-hoc domein kennis te verkrijgen. De eenvoudige kenmerken die gebruikt worden doen denken aan Haar basis functies [10]. Meer bepaald de twee-, drie- en vier-rechthoekige kenmerken. Hoe de waarden van deze kenmerken berekend kunnen worden ziet u in figuur 3.4. Deze rechthoekige kenmerken kunnen zeer snel berekend worden door gebruik te maken van een tussenliggende voorstelling van de afbeelding, wat het integraal beeld wordt genoemd. Het integraal beeld

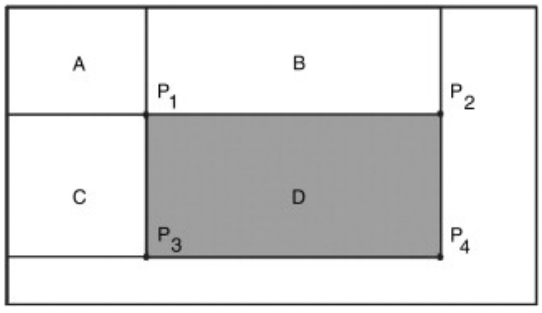
op een locatie x, y bevat de som van de pixels boven en links van x, y, met x, y inclusief:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

waarbij ii(x,y) staat voor het integraal beeld en i(x,y) voor het origineel beeld. Op afbeelding 3.5, zien we hoe dat we door gebruik te maken van het integraal beeld elke rechthoekige som kunnen berekenen met behulp van 4 punten. Dit maakt dat het twee-, drie- of vier-rechthoekig kenmerk kan berekend worden door gebruikt te maken van respectievelijk 6, 8 of 9 arrays.



Figuur 3.4: *Voorbeelden van de rechthoekige kenmerken relatief gezien t.o.v. het detectievenster. De som van de pixels in de witte rechthoeken wordt afgetrokken van de som van de pixels in de grijze rechthoeken. In (A) en (B) zien we voorbeelden van twee-rechthoekige kenmerken, in (C) drie-rechthoekige en (D) vier-rechthoekige kenmerken.*



Figuur 3.5: *De som van de pixels in de rechthoek D kunnen berekend worden door het verschil tussen de sommen van de punten op de diagonalen, dus 4+1-2+3. De waarde van het integraal beeld op locatie 1 is de som van de pixels in A. De waarde op locatie 2 is A+B, op locatie 3 A+C en op locatie 4 A+B+C+D. Daaruit volgt dat 4+1-2+3 = (A+B+C+D+A)-(A+B+A+C) = D.*

3.2.1.2 Adaboost

In dit systeem wordt een variant van Adaboost [11] gebruikt om een klein aantal kenmerken te selecteren en een classificeerder te trainen. In originele vorm wordt Adaboost gebruikt om de classificatie performantie van een eenvoudig zelf lerend algoritme te verbeteren.

Per venster zijn er meer dan 180000 rechthoekige kenmerken. Al deze kenmerken berekenen zou veel te kostelijk zijn. Daarom gaan ze een klein aantal kenmerken gebruiken die gecombineerd een goede classificeerder maken. Het moeilijkste onderdeel hiervan is deze kenmerken te vinden. Om dit doel te verwezenlijken gaat het eenvoudige zelf lerend algoritme het rechthoekig kenmerk gebruiken dat het best de positieve van de negatieve voorbeelden scheidt.

3.2.1.3 Cascade

In dit deel wordt een algoritme besproken om een cascade van classificeerders te maken, wat een verbeterde detectie en een reductie in verwerkingstijd geeft.

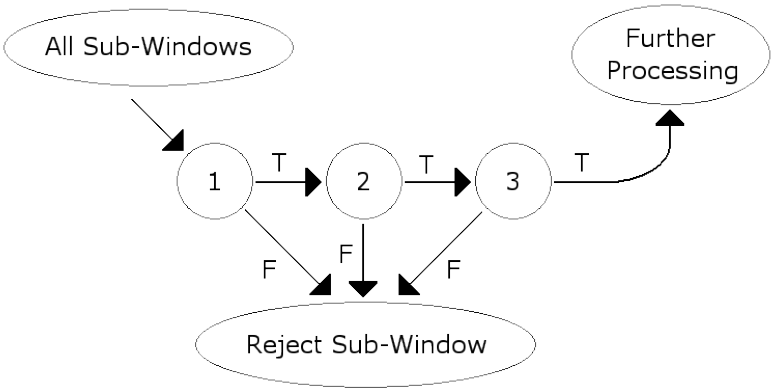
Het concept is dat kleinere, meer efficiënte en verbeterde classificeerders gevormd kunnen worden. Deze verwerpen veel negatieve vensters (geen detectie van het object) terwijl zij ondertussen bijna alle positieve vensters detecteren. Eerst worden dus eenvoudige classificeerders gebruikt om de meerderheid van de vensters te verwerpen, waarna meer complexe classificeerders gebruikt worden om lage vals positieve waardes te bekomen.

De cascade is dus een aaneenschakeling van classificeerders: wanneer de eerste classificeerder een positief resultaat geeft wordt de evaluatie van de tweede getriggerd. Een positief resultaat van de tweede activeert de derde en zo voort. Een negatief resultaat leidt onmiddellijk tot een verwerping van dat venster, zoals te zien is in figuur 3.6.

De verschillende classificeerders worden getraind met behulp van Adaboost waarbij de drempel aangepast wordt om valse negatieve te minimaliseren. In principe kan men een ideale cascade maken, waarbij het aantal classificeerders, het aantal kenmerken per classificeerder en de drempel van elke classificeerder zo gekozen zijn dat ze het verwacht aantal geëvalueerde kenmerken verkleinen. Dit is jammer genoeg een heel moeilijk probleem. In praktijk wordt dit gedaan door bij elke classificeerder de vals positieve detecties te verlagen en de detectiemogelijkheid te verkleinen. Elke classificeerder wordt getraind door kenmerken toe te voegen tot de vals positieve detecties en de detectiemogelijkheden voldoen aan de eisen. Verder worden classificeerders aan de cascade toegevoegd tot aan de algemene eisen voldaan wordt.

3.2.1.4 Resultaten

- **Snelheid:** deze zou ongeveer 15 maal sneller zijn dan de Rowley-Baluja-Kanade detector en 600 maal sneller dan de Schneiderman-Kanade detector.



Figuur 3.6: *Een schematische voorstelling van de detectie cascade.*

- **Scannen van het beeld:** dit gebeurt door gebruik te maken van meerder vensterschalen. De beste resultaten waren er met venster die een 1.25 uit elkaar lagen. Het venster verplaatst zich over de afbeelding door telkens een aantal pixels op te schuiven.
- **Samenvoegen van meerdere detecties:** doordat het detectievenster slechts kleine verschuivingen maakt, zullen rond elk gezicht meerdere detecties gebeuren. Deze nemen we dan samen tot één detectie.
- **Stemprocedure:** door drie detectoren, die op gelijke manier getraind zijn, over de afbeelding te laten lopen en vervolgens bij een detectie de meerderheid als correct te beschouwen. Dit geeft een kleine verbetering t.o.v. één detector.
- **Resultaten op de MIT+CMU test set:** zie afbeelding 3.7, hier zien we de detectiegraad van een bepaalde detector bij een bepaald aantal vals positieve detecties.

False detections / Detector	10	31	50	65	78	95	110	167	422
Viola-Jones	78.3%	85.2%	88.8%	89.8%	90.1%	90.8%	91.1%	91.8%	93.7%
Rowley-Baluja-Kanade	83.2%	86.0%	-	-	-	89.2%	-	90.1%	89.9%
Schneiderman-Kanade	-	-	-	94.4%	-	-	-	-	-
Roth-Yang-Ahuja	-	-	-	-	(94.8%)	-	-	-	-

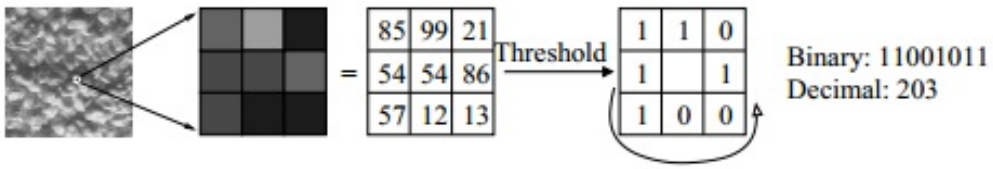
Figuur 3.7: *Testresultaten van MIT+CMU test set met 130 foto's en 507 gezichten [3].*

3.2.2 LBP

In de paper geschreven door Ahonen et al. [4] gaat men gezichtsdetecties uitvoeren door gebruik te maken van Local Binary Patterns (LBP). In hun paper geven ze een gedetailleerde analyse van de voorgestelde detectie methode, geven ze resultaten van hun detectie methode en nog verdere uitbreidingen.

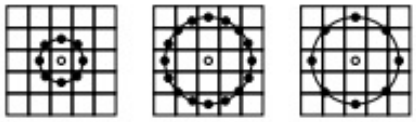
3.2.2.1 LBP methode

De LBP operator was oorspronkelijk bedoelt om texturen te beschrijven. De operator geeft een label aan elke pixel van een afbeelding door de 3×3 omringende pixels van elke pixel te vergelijken met de middelste pixel en het resultaat als een binair nummer voor te stellen. Het histogram van deze labels kan dan gebruikt worden om de textuur voor te stellen. In figuur 3.8 zie je een voorstelling van een standaard LBP operator. Wanneer het gesampelde punt niet in het middelpunt van een pixel valt wordt er gebruik gemaakt van bilineaire interpolatie. Om later dan om te kunnen gaan met labels van verschillende



Figuur 3.8: *De standaard LBP operator.*

schalen, werd de LBP operator uitgebreid om het aantal omringende pixels te laten variëren. Hierbij gaan ze de omringende pixels definiëren als een reeks van gesamplede punten die evenredig over een cirkel geplaatst worden die als middelpunt de te labelen pixel heeft. Dit maakt dat je de straal van de cirkel en het aantal gesamplede punten kan kiezen. We maken gebruik van de notatie (P, R) waarbij P het aantal gesamplede punten voorstelt en R de straal van de cirkel. Een voorbeeld hiervan is te zien in figuur 3.9. Een volgende



Figuur 3.9: *Hier zien we van links naar rechts (8, 1), (16, 2), (8,2) omgeving.*

uitbreiding is de definitie van de uniforme patronen. Een LBP wordt uniform genoemd als het binaire patroon maximaal twee bitgewijze overgangen maakt van 0 naar 1 of omgekeerd. De patronen 00000000 (0 overgangen) en 11001111 (2 overgangen) zijn uniform maar bijvoorbeeld het patroon 01010011 (6 overgangen) is niet uniform. Bij de berekening van het LBP histogram worden uniforme patronen gebruikt zodat het histogram voor elk

van deze patronen een aparte container heeft en de niet uniforme patronen worden samen in één container geplaatst. Uit experimenten bleek dat 90.6% van de patronen in een (8, 1) omgeving en 85,2% in een (8, 2) omgeving uniforme patronen zijn.

3.2.2.2 LBP gezichtsbeschrijving

Gebruik makende van de LBP methode beschreven in de vorige paragraaf, gaan ze in de paper [4] gezichtsbeschrijvingen doen. Ze gaan de textuurbeschrijver gebruiken om een aantal lokale beschrijvingen van het gezicht op te bouwen en dit dan combineren in een globale beschrijving. Deze methode is gekozen om twee redenen. Ten eerste omdat de methode van een op lokaal gebaseerde kenmerken voor gezichtsherkenning aan populariteit wint ten opzichte van methodes die enkel werken met een globale beschrijving en ten tweede omdat wanneer men direct een globale beschrijving zou proberen te maken van een gezicht met de textuur methode, dit niet goed zou gaan omdat deze de neiging heeft om uit te middelen over de volledige afbeelding. Het standaard histogram kan uitgebreid worden naar een ruimtelijk uitgebreid histogram, hetgeen zowel het voorkomen alsook de ruimtelijke relaties van gezichtsregio's beschrijft. Wanneer we bijvoorbeeld m gezichtsregio's hebben, dan wordt voor elk van deze regio's het histogram bepaald. Deze m histogrammen worden gecombineerd zodat een ruimtelijk uitgebreid histogram bekomen wordt. Dit histogram heeft dan een grootte van $m \times n$ waar n de lengte is van een enkel LBP histogram. In dit ruimtelijk uitgebreid histogram hebben we een gezichtsbeschrijving op drie niveaus van lokaliteit: de LBP labels geven informatie op pixelniveau, deze labels worden gesommeerd over een kleine regio om informatie te hebben op een regionaal niveau en deze regionale niveaus worden samengevoegd om een globale beschrijving van het gezicht te bekomen.

3.2.2.3 Vergelijking LBP met andere methodes

In figuur 3.10 vergelijken ze de LBP methode met enkele andere methodes. De fb, fc, dup I en de dup II kolommen tonen de herkenningswaardes voor de FERET [12] test reeks, dit is een reeks van mensen gezichten. De laatste drie kolommen tonen de gemiddelde herkenningswaarde van de permutatietest met een 95% betrouwbaarheidsinterval. We zien dat bij fb (een eenvoudige testreeks) LBP het redelijk goed doet. Bij fc (een reeks waarbij de belichting verandert is t.o.v. fb) zien we dat het een heel stuk beter is dan de andere methodes. De dub I en dub II zijn testen bij veroudering (hierbij zit er minsten 18 maand tussen het nemen van de foto's in dub I en dub II) en ook hier zien we dat LBP beter scoort dan de overige methodes.

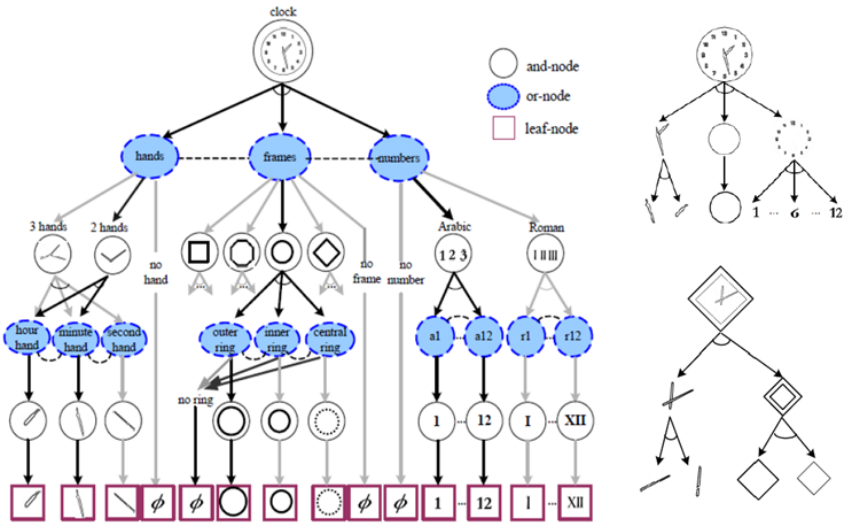
Method	fb	fc	dup I	dup II	lower	mean	upper
Difference histogram	0.87	0.12	0.39	0.25	0.58	0.63	0.68
Homogeneous texture	0.86	0.04	0.37	0.21	0.58	0.62	0.68
Texton Histogram	0.97	0.28	0.59	0.42	0.71	0.76	0.80
LBP (nonweighted)	0.93	0.51	0.61	0.50	0.71	0.76	0.81

Figuur 3.10: *Vergelijking LBP met andere methodes [4].*

3.2.3 Felzenszwalb

Dit is een uitbreinding op HOG. HOG is een zeer populair kenmerktype om classificeerder te trainen door de snelheid waarmee hij werkt en de hoge accuraatheid die hij behaald. Op dit kenmerktype ga ik in het volgende hoofdstuk dieper ingaan.

Hier gaan we objectcategorieën, zoals mensen of auto's, proberen te lokaliseren en te detecteren in een stilstaand beeld. Dit is niet zo gemakkelijk omdat de objecten binnen zo'n categorie heel sterk van uiterlijk kunnen variëren. De methode beschreven door Felzenszwalb et al. [13] gaat deze objecten detecteren door gebruik te maken van een mengeling van meerschalige vervormbare deelmodellen. Ze maken gebruik van picturale structuren. Picturale structuren gebruiken een collectie van onderdelen geordend in een vervormbare configuratie, dit om objecten voor te stellen. Hoewel vervormbare modellen significante variaties in voorkomen kunnen opvangen, gaat een enkel vervormbaar model niet genoeg zijn om een grote objectcategorie voor te stellen. Uiteindelijk is het de bedoeling om tot een '*grammar model*' te komen. Dit is een manier om een object op te bouwen uit alle mogelijke combinaties van zijn deelobjecten. In afbeelding 3.11 ziet u hoe een '*grammar model*' van een klok er zou kunnen uitzien.



Figuur 3.11: *Grammar model van een klok [5].*

3.2.3.1 Grammar modellen

Girshick et al. [14] leggen uit wat dit inhoud. Object detectie grammars stellen objecten recursief voor in termen van andere objecten. N is een set van niet terminale symbolen en T is een set van terminale symbolen. Een terminaal symbool kunnen we zien als een basis bouwsteen die gevonden kan worden in een afbeelding. De niet terminale worden gedefinieerd als onderdeel van de terminale symbolen.

Laat Ω een set van mogelijke locaties van symbolen in een afbeelding zijn. Een geplaatst symbool, $Y(\omega)$, specifieert de plaatsing van $Y \in N \cup T$ op een locatie $\omega \in \Omega$.

De structuur van een grammar model wordt gedefinieerd door een set, R, van gewogen producten van de vorm

$$X(\omega_0) \xrightarrow{s} Y_1(\omega_1), \ldots, Y_n(\omega_n),$$

met X $\in$ N, $Y_i \in N \cup T, \omega_i \in \Omega$ en s $\in \mathbb{R}$ is een score. We schrijven de score van r $\in$ R als s(r).

Een uitbreiding van X(ω) leidt tot een afleidingsboom B die begint in X(ω). De uiteinden van B worden gelabeld als geplaatste terminale symbolen. De inwendige knopen als geplaatste niet terminale symbolen en als de producten die deze symbolen moeten vervangen.

We definiëren een verschijningsmodel voor de terminale symbolen gebruik makende van een functiescore, score(A, ω), dat de score voor de plaatsing van terminaal symbool A op plaats ω berekent. Deze score hangt impliciet af van de afbeeldingsdata. We definiëren de score van de afleidingsboom B als de som van de scores van de producten, gebruikt om B te genereren, en de score om de terminale symbolen, geassocieerd met de uiteinden van B, op hun respectievelijke plaats te zetten.

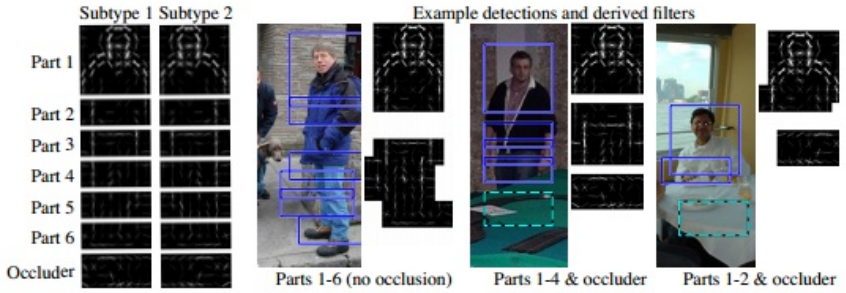
$$score(B) = \sum_{r \in interneknopen(B)} s(r) + \sum_{A(\omega) \in uiteinden(B)} score(A, \omega)$$

3.2.3.2 Grammar model om mensen te detecteren

Verder gaan Girshick et al. [14] dit grammar model dus gebruiken om mensen te detecteren. Ze beginnen met het beschrijven van een oppervlakkig model waar alle filters dezelfde resolutie krijgen in de kenmerkenkaartpiramide, dit is te zien in afbeelding 3.12.

Dit model bevat 6 delen van de mens en een 'verstopper', welke in twee types voorkomen. Een detectie plaatst een subtype van elk zichtbaar onderdeel op een locatie en schaal in de afbeelding. Wanneer bij de detectie niet alle delen kunnen geplaatst worden, moet de 'verstopper' geplaatst worden. Als een persoon niet volledig zichtbaar is, dan moet hier een oorzaak van zijn. Dit kan ofwel de rand van de afbeelding zijn of een voorwerp waarachter een deel van de persoon verstopt is. Er wordt een niet triviaal model gebruikt om de voorwerpen (verstoppers) die ervoor zorgen dat de persoon niet volledig zichtbaar is te beschrijven.

We zien ook duidelijk een verschil in zichtbaarheid in figuur 3.12, van op de meest linkse foto een man die volledig zichtbaar is tot de rechtse foto een man die slechts vanaf de schouders zichtbaar is.



Figuur 3.12: *Oppervlakkig grammar model.*

Vervolgens wordt er over een grondiger model gesproken. Hierbij breiden ze het oppervlakkig model uit door er onderdelen met de dubbele resolutie aan toe te voegen. Wanneer er dan grote objecten gedetecteerd moeten worden gebruiken ze de hoge resolutie onderdelen en voor kleine objecten de lage resolutie onderdelen.

3.2.3.3 Resultaten

Om tot deze resultaten te komen werd het standaard PASCAL VOC comp3 test protocol gebruikt. Dit meet detectie performantie door gemiddelde precisie (AP = average precision) over verschillende niveaus. In figuur 3.13 ziet u de resultaten van de PASCAL VOC 2010 in comp3, waar de UoC-TTI methode won in de personencatergorie [14].

	Grammar	+bbox	+context	UoC-TTI [9]	+bbox	+context	Poselets [9]
AP	47.5	47.6	**49.5**	44.4	45.2	47.5	48.5

Figuur 3.13: *PASCAL VOC 2010 resultaten. Grammar en UoC-TTI namen deel in comp3 en Poselets in comp4.*

3.3 Spectrale beeldvorming

Spectrale beeldvorming combineert spectroscopie met beeldvorming. Deze combinatie is niet triviaal omdat het vereist dat er een driedimensionale dataset gecreëerd wordt. Deze set bestaat uit vele afbeeldingen van hetzelfde object, die elk bij een andere golflengte gemeten worden. Dit houdt in dat de verwervingstijd van deze set lang is en er dus moet gezocht worden naar een compromis om deze tijd in te korten.

Om de karakteristieken van spectrale beeldvorming te bespreken gaan we eerst wat dieper in op zijn bouwstenen, spectroscopie en beeldvorming.

3.3.1 Beeldvorming

Beeldvorming is de wetenschap en technologie voor het verkrijgen van ruimtelijke en tijdelijke data informatie van objecten met als doel informatie te verzamelen. Op dit moment is digitale beeldvorming, waarbij gebruik wordt gemaakt van een digitale camera zoals een CCD (Charged Coupled Device), de meest geavanceerde en gebruikte methode [15].

De kwaliteit van een afbeelding determineert de hoeveelheid informatie die uit deze afbeelding gehaald kan worden. In de volgende lijst zien we de parameters die verkregen afbeeldingen karakteriseren.

- Ruimtelijke resolutie bepaald de dichtst mogelijke onderscheidbare kenmerken in het object. Dit hang af van de golflengte (λ), de numerieke lensopening (NA), de vergroting en de pixelgrootte van de arraydetector, meestal een CCD.
- Het laagst detecteerbaar signaal hangt af van de kwantum efficiëntie (hoger = beter), het ruisniveau (lager = beter), de NA en de kwaliteit van de optiek (hoger = beter).
- Het dynamisch bereik van de verkregen data bepaalt het aantal verschillende intensiteit niveaus die gedetecteerd kunnen worden in een afbeelding.
- FOV (Field Of View) bepaalt het maximaal bereik dat bekeken kan worden.
- Andere parameters zijn de belichtingstijd en het weggooien van CCD pixels om de sensitiviteit te verhogen.

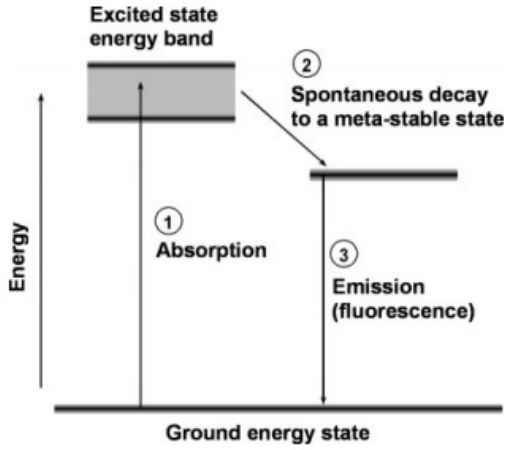
Bij metingen zijn er vele imperfecties die de kwaliteit van een foto naar beneden halen, zoals autofluorescentie, niet specifieke vlekken, verbleking enz., deze moeten onderscheiden worden van de fysieke limitaties.

3.3.2 Spectroscopie

Spectroscopie is de wetenschap van het verkrijgen en verklaren van de spectrale karakteristieken van materie en kan op vele gebieden toegepast worden, maar we limiteren ons hier tot optische spectroscopie, meer bepaald het zichtbaar licht. Een spectrum is een

collectie van lichtintensiteiten bij verschillende golflengtes.

De structuur van atomen en moleculen is direct gerelateerd met spectroscopie. Het spectrum is een meting van de energieniveaus van een gedetecteerde structuur. Moleculen hebben een specifieke energiebandstructuur, welke in figuur 3.14 te zien is. Deze energieniveaus zijn intrinsieke eigenschappen van de molecule en het spectrum. Ze worden dus gezien als een vingerafdruk van de moleculen.



Figuur 3.14: *We zien hier een vereenvoudigt energieniveau diagram van een fluorescente molecule. (1) Elektronen worden geëxciteerd naar de geëxciteerde band door absorptie van een foton. (2) De elektronen vallen snel terug naar een metastabiele toestand. (3) Elektronen vallen terug naar hun grondtoestand door een foton uit te zenden.*

Hoewel we ons richten op het spectraal bereik van zichtbaar licht wordt het infrarood spectrum ook zeer veel gebruikt. Dit zorgt voor vibraties in de moleculen wat specifiek is voor inter-atomaire banden. Dat maakt van infrarode absorptie een uitstekende vingerafdruk van de structuur van een molecule.

Het is belangrijk om de fluorescentie- en absorptieprocessen te onderscheiden. Bij fluorescentie is het dan weer belangrijk om een onderscheid te maken tussen het zwak uitgestoten licht en het sterk excitatielicht. Hiervoor wordt gebruik gemaakt van kleurenfilters, excitatie barrière filter, enz. Wanneer er gebruik wordt gemaakt van een externe lichtbron, dan moet de spectrale informatie van dat licht in rekening worden gebracht om het gewenste spectrum er exact uit te halen.

Om het spectrum te meten gaat het licht verdeeld worden in zijn verschillende golflengtes (kleuren) en wordt de intensiteit hiervan gemeten. Enkele belangrijke karakteristieken van een spectrum zijn:

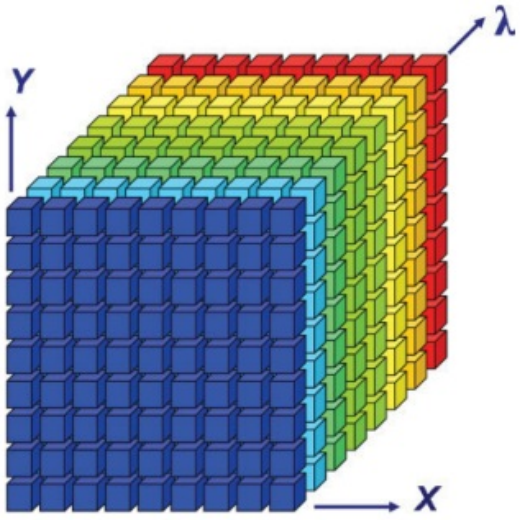
- De spectrale resolutie determineert de kortste golflengte dat onderscheiden kan worden.
- Het spectraal bereik waar spectra kunnen gemeten worden.

- Het laagst detecteerbaar signaal en dynamisch bereik, wat het kleinst meetbare signaal en het aantal onderscheidbare niveaus in een gegeven meting detecteert.

3.3.3 Spectrale beeldvorming

Zoals eerder aangehaald combineert spectrale beeldvorming deze twee methodes, die hierboven beschreven staan. Waar beeldvorming de intensiteit van elke pixel van de afbeelding bepaald, I(x, y), en een typische spectrometer een enkel spectrum bepaald, I(λ), zo biedt een spectraal beeld een spectrum bij elke pixel I(x, y, λ). Dit is een 3D dataset en kan gezien worden als een kubus van informatie, zoals in figuur 3.15.

Een spectraal beeld bevat meestal honderdduizenden spectra, één voor elke pixel. Dit maakt dat de databestanden te groot en complex zijn om visueel te interpreteren, waardoor we dus een uitgebreide set van tools nodig hebben om de databestanden te verwerken en de resultaten te presenteren. Hiervoor zijn de meeste moderne beeldverwerkingstechnieken voldoende.

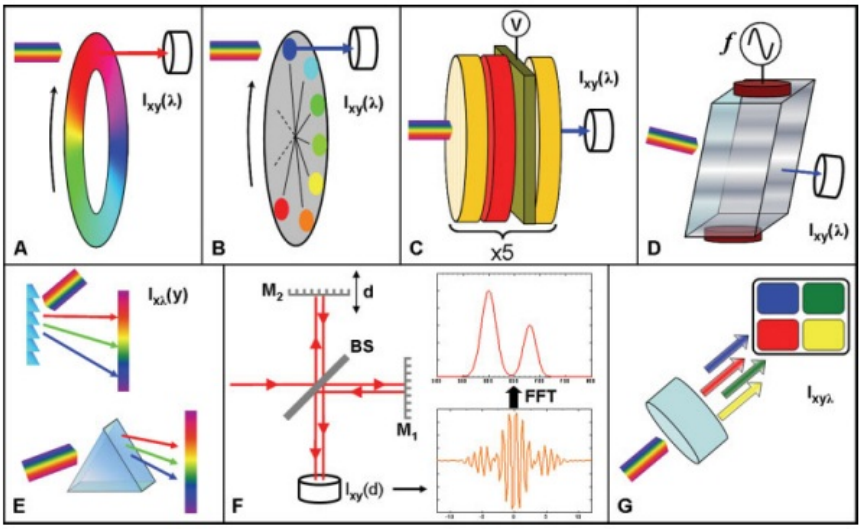


Figuur 3.15: *Een spectraal beeld dataset*

3.3.4 Realisatie van spectrale beelden

Spectrale beeldvorming vereist de combinatie van een dispersief element om de spectrale informatie te kunnen verkrijgen met een beeldvormingssysteem. Zoals eerder aangehaald, is de data die verzameld wordt per pixel de intensiteit van iedere golflengte, I(x, y, λ) (Fig. 3.15). Wanneer een 2D array detector gebruikt wordt, kan je het spectrale beeld niet in één keer verkrijgen. Wanneer een nog lagere dimensie-detector gebruikt wordt zal er nog meer tijd nodig zijn om een spectraal beeld te verkrijgen. Dit heeft geleid tot de ontwikkeling van verschillende methodes. Deze kunnen onderverdeeld worden in volgend methodes:

- **Golflengte-scan methodes:** deze meten het beeld met één golflengte per keer (fig. 3.16 A-D).
- **Ruimtelijke-scan methodes:** zij meten het volledige spectrum van een deel van de foto per keer en scannen het beeld (fig. 3.16 E).
- **Tijd-scan methodes** deze meten een set van beelden waar elk van hen een superpositie is van spectrale of ruimtelijke beeldinformatie. Op het einde wordt de data getransformeerd naar het eigenlijke spectrale beeld (door bv. Fourier methodes) (fig. 3.16 F).
- **Compromis methodes:** deze meten het hele spectrale beeld simultaan op, maar bepalen dan minder punten in het spectrum, FOV of de ruimtelijke resolutie (fig. 3.16 G).



Figuur 3.16: *Verschillende beeldvormingsmethodes: (A-D) golflengte-scan methodes, E ruimtelijke-scan methode, F tijd-scan methode, G compromis methode*

Ik denk dat in het kader van mijn masterproef de golflengte-scan en compromis methodes de meest geschikte methodes zijn, maar we beschikken enkel over het materiaal om de ruimtelijke-scan methode te gebruiken. Hier zal ik in de volgende paragrafen iets dieper op in gaan. Voor meer informatie refereer ik graag naar '*Spectral Imaging: Principles and Applications*' [15].

3.3.4.1 Golflengte-scan methode

Een eenvoudige methode voor het meten van een spectraal beeld is om een set van kleuren filters te gebruiken (fig. 3.16 B), elk van deze laat een kleine golflengteband (bandbreedte van 10 nm) door. Dit is alleen bruikbaar wanneer een klein aantal golflengtes nodig zijn.

Een meer geschikte manier is om gebruik te maken van een variabele filter. Drie veel voorkomende variabele filters zijn CVF (circular-variable filter) (fig. 3.16 A), LCTF (liquid-crystal tunable filter) (fig. 3.16 C) en AOTF (acousto-optical tunable filter) (fig. 3.16

D). CVF zendt een smalle band licht uit naar gelang van de positie van de straal op het oppervlak. AOTF en LCTF zijn elektro-optische componenten met geen bewegende delen.

Deze types van filters leggen een volledig spectraal beeld vast door het meten van één beeld per keer met een verschillende golflengte. Ze hebben het voordeel dat de gebruiker een aantal golflengtes kan kiezen waarmee beelden gemaakt moeten worden. De spectrale resolutie is meestal hardware afhankelijk, wat geen wijzigingen toelaat.

3.3.4.2 Ruimtelijke-scan methode

Bij deze methode wordt de spreiding van het licht bereikt door het licht door een rooster of een prisma te sturen (fig. 3.16 E). Er kan slechts één lijn per keer van een object gemeten worden, hierdoor zal je het object onder de camera moeten verplaatsen om een volledig beeld te verkrijgen (lijn per lijn inlezen). Dit zorgt voor een hogere verwervingstijd dan CCD-gebaseerde manieren. We hebben deze methode toegepast, maar de beelden waren niet bruikbaar.

3.3.4.3 Compromis methode

In deze benadering wordt een compromis gezocht voor het simultaan opmeten van spectrale en ruimtelijke parameters. Dit is bruikbaar voor spectrale beeldvorming die met een hoge snelheid moet gebeuren. De gemakkelijkste manier is om een kleiner deel van het FOV te nemen en dit meerder keren te projecteren op eenzelfde CCD, waarbij ze elk door een andere filter gestuurd worden (fig. 3.16 G). In de meeste gevallen wordt ruimtelijke resolutie en FOV afgewogen tegen spectrale informatie.

3.4 OpenCV

OpenCV staat voor Open Source Computer Vision Library. Het is dus een vrij te gebruiken bibliotheek waar verschillende honderden computervisie algoritmes in verwerkt zijn. De detectietechnieken besproken in paragraaf 3.2, hebben al heel wat algoritmes die in OpenCV geïmplementeerd zijn. Hierdoor is dit een goede bibliotheek om te gebruiken bij het programmeren van deze detectietechnieken. De huidige versie OpenCV 2.4 API is in essentie een C++ API tegenover de oudere (1.x) versies die C gebaseerd waren [16]. C++ is een zeer ruime programmeertaal en dus kan de implementatie van de spectrale beeldvorming ook hierin geschreven worden.

OpenCV heeft een modulaire structuur, waarvan de volgende beschikbaar zijn:

- **Core:** een compacte module die de basis data structuren definieert, die door alle andere modules gebruikt worden.
- **Imgproc:** een beeldverwerkingsmodule dat lineaire en niet lineaire filters, geometrische transformaties, histogrammen en nog veel meer bevat.

- **Video:** een videoanalyse module dat bewegingsbenadering, achtergrondsubstractie en object detectie algoritmes bevat.
- **Objdetect:** detecties van objecten van voorgedefinieerde instanties, zoals gezichten, ogen, mensen, enz.
- **Highgui:** een gebruiksvriendelijke interface voor video opnames, beeld en video codecs, alsook eenvoudige UI mogelijkheden.
- ... nog een aantal andere modules.

Zoals eerder aangehaald is dit een C++ API. Moderne C++ kan beschouwd onderverdeeld worden in drie onderdelen [17]:

- De low-level programmeertaal, waarvan veel van C is overgenomen.
- De meer geavanceerde functies, zoals de mogelijkheid om eigen types te definiëren en grootschalige programma's te organiseren.
- De standaard bibliotheek, hetgeen de geavanceerde functies gebruikt om nuttige datastructuren en algoritmes te voorzien.

3.5 Conclusie

We gaan de Amblyseius californicus mijt detecteren. Hiervoor heb ik in deze literatuurstudie kort een aantal papers besproken. Het beste lijkt mij om gebruik te maken van de LBP of de HOG methode. Dit zijn twee populaire methodes doordat ze heel snel zijn en een hoge accuraatheid behalen. Daarom ga ik in het volgende hoofdstuk deze 2 methodes kort bespreken en verder in paragraaf 6 gaan we deze twee methodes met elkaar vergelijken. Om de algoritmes te trainen gaan we gebruik maken van een cascade structuur zoals besproken in paragraaf 3.2.1. De spectrale beeldvorming hebben we niet kunnen gebruiken om de mijten te detecteren omdat we niet echt over het juiste materiaal beschikte. We hebben wel met een zelfgemaakte spectrograaf beelden opgenomen maar deze waren niet bruikbaar.

Hoofdstuk 4

Detectie van mijten

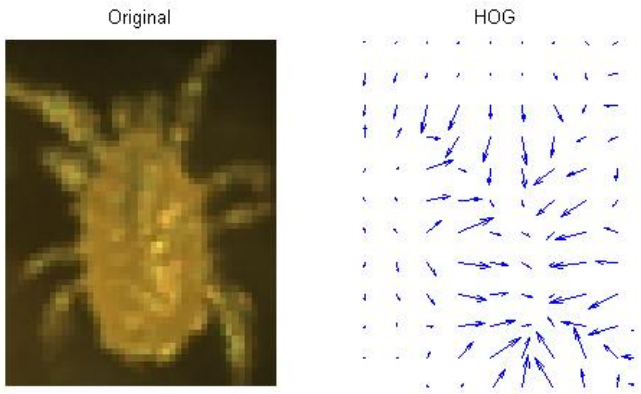
Om de classificeerders te trainen heb ik de keuze gemaakt om voor HOG en LBP kenmerktypes te kiezen. Dit zijn twee populaire methodes die erg snel werken, waardoor real-time toepassingen mogelijk zijn. We hebben deze kenmerktypes al kort aangehaald in het vorige hoofdstuk maar we gaan deze nu iets gedetaileerder bekijken.

4.1 HOG

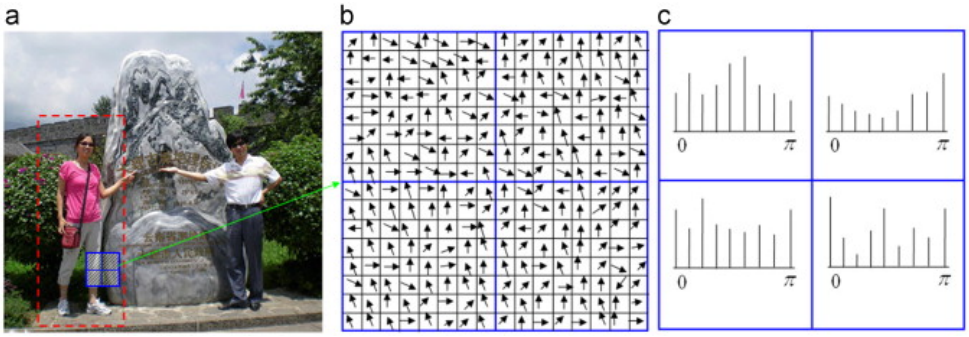
Het Histogram of Oriented Gradients beschrijft kenmerken in een afbeelding met als doel een object in die afbeelding te detecteren. De methode telt het aantal voorkomens van gradiënt oriëntaties in gelokaliseerde delen van een afbeelding. Deze methode bevat een aantal voordelen [18]. Doordat de methode gebruik maakt van gelokaliseerde cellen, hebben geometrische en fotometrische transformaties geen invloed op de detecties, alleen de oriëntatie van het object speelt een rol. Nu is dit wel belangrijk bij de detecties van de mijten omdat een mijt niet steeds in dezelfde richting staat. Dit wordt verder nog besproken in paragraaf 5.1.2.3. In figuur 4.1 zien we de mijt samen met zijn HOG beschrijving.

4.1.1 Werking

De essentie van HOG [18] is dat het uiterlijk en de vorm van een lokaal object in een afbeelding kan beschreven worden door het berekenen van intensiteit gradiënten of de richting van de randen. Dit wordt geïmplementeerd door de afbeelding in kleine aaneenliggende regio's, cellen genaamd, te verdelen en dan voor elke cel het histogram van de gradiëntrichtingen of de richting van de randen voor elke pixel in de cel samen te stellen. De combinatie van deze histogrammen stelt dan de beschrijving voor. Hiervan zien we een voorbeeld in figuur 4.2, waar we van links naar rechts eerst een foto, dan een beeld van de gradiënten en vervolgens de bijhorende histogrammen zien. Een toepassing van HOG kan je terugvinden in mijn literatuurstudie onder paragraaf 3.2.3.



Figuur 4.1: *Links de mijt en rechts zijn HOG beschrijving. De pijlen geven de groote en richting van de gradiënt voor elke cel.*



Figuur 4.2: *Hier zien we: (a) De afbeelding, (b) de gradiënten en (c) de histogrammen [6].*

4.2 LBP

Het Local Binary Pattern is een eenvoudig maar zeer efficiënte textuur operator die de pixels van een afbeelding labelt door gebruik te maken van de omliggende pixels en dat resultaat als een binair nummer beschouwt. Het is een populaire operator geworden door het goed onderscheidend vermogen en zijn rekenkundige eenvoud. Door zijn rekenkundige eenvoud is het mogelijk om beelden real-time te analyseren.

4.2.1 Werking

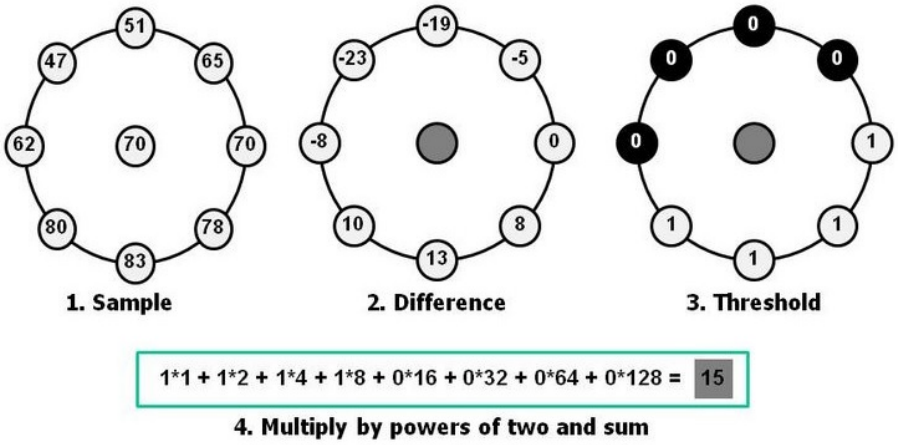
Het idee achter LBP [19] is dat tweedimensionale texturen kunnen beschreven worden door twee complementaire factoren: enerzijds heb je de lokale ruimtelijke patronen en anderzijds het verschil in grijswaarden. De LBP operator vormt dan labels voor de pixels van de afbeelding door de drempelwaarde van de n×n matrix, met als centerpunt de pixel waarop de berekeningen worden uitgevoerd, te berekenen en dit dan als een binair getal voor te stellen, dit kan je zien in figuur 4.3. Zoals we in figuur 4.3 zien maken ze hier gebruik van een 3×3 matrix. Eerst bepalen we de waardes van alle pixels, dan nemen we het verschil met de middelste pixel (dit is de pixel waarvoor we de LBP waarde gaan bepalen). Tot slot vergelijken we de waarde van de omliggende pixels met nul, indien kleiner dan nul dan is de waarde voor die pixel nul en anders is de waarde één. Vervolgens vermenigvuldigen we deze waardes met de machten van twee. Dit kan voorgesteld worden met de formule [19]:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^p \qquad s(x) = \begin{cases} 1 & \text{,als } x \geq 0; \\ 0 & \text{,anders.} \end{cases}$$

Hier stelt P het aantal pixels voor, R is de grootte van de matrix, g_p zijn de waardes van de omliggende pixels van de te berekenen pixel en g_c is de middelste pixel, waarvoor we de LBP code aan het bepalen zijn.

4.3 Cascade

Deze twee classificeerders gebruiken we in een cascadestructuur. Een cascadestructuur is een aaneenschakeling van kleinere classificeerders. We beginnen hierbij met een eenvoudige classificeerder, deze verwerpt veel negatieve vensters (geen detectie van het object) en behoudt bijna alle positieve vensters. Hierna worden telkens complexere classificeerders aan de cascade toegevoegd. Zo worden er steeds meer negatieve vensters verworpen tot er slecht positieve vensters overblijven. Dit komt overeen met de cascade beschreven in paragraaf 3.2.1.3.



Figuur 4.3: *Hier zien we de stappen van een LBP code berekening.*

4.4 Conclusie

In dit hoofdstuk heb ik kort de twee classificatiemethodes beschreven die ik ga gebruiken. Beide methodes zijn populaire methodes die gebruikt worden bij gezichtsherkenning. In hoofdstuk 6 zal ik de twee classificatiemethodes experimenteel met elkaar vergelijken.

Hoofdstuk 5

Uitwerking

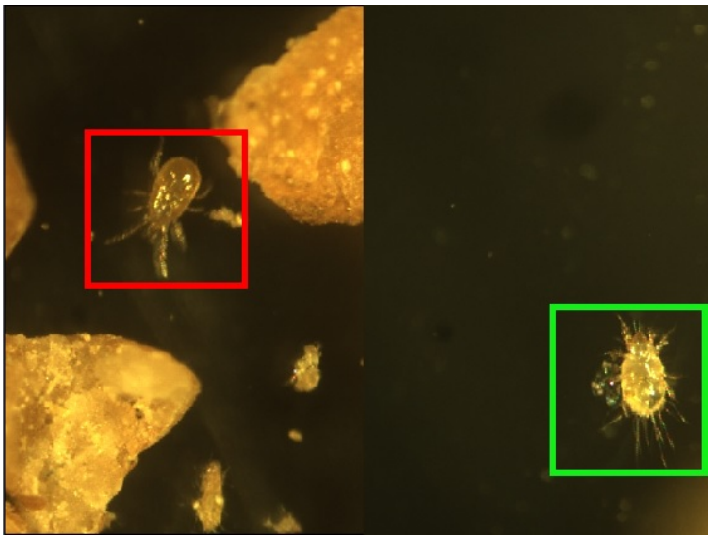
In dit deel ga ik de uitwerking van mijn masterproef uitleggen. We kunnen dit opdelen in drie grote delen. Het eerste deel noem ik het *'Voorbereiden van de beelden'*, hierin leg ik wat meer uit over het opnemen en het bewerken van de beelden. Waarna ik in het tweede deel uitleg hoe ik deze beelden gebruikt heb om het *'cascade'* gebaseerd algoritme te trainen. In het laatste deel beschrijf ik dan hoe ik dit algoritme toegepast heb op een nieuwe afbeelding en een aantal parameters afgesteld heb.

5.1 Voorbereiden van de beelden

Hier ga ik wat meer vertellen over hoe ik de beelden ben gaan opnemen en vervolgens zal ik de stappen uitleggen die ik heb uitgevoerd om mijn beelden zo ideaal mogelijk te krijgen om in te voeren in het trainingsalgoritme, dat in het volgende deel besproken zal worden.

5.1.1 Opnemen van de beelden

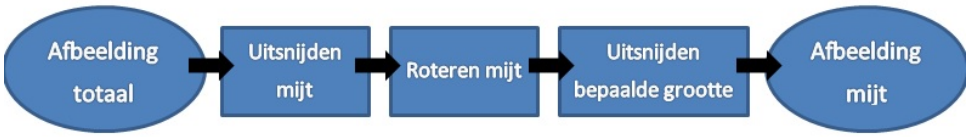
Voor het opnemen van de beelden konden we terecht bij het bedrijf Biobest. Hier hebben we een groot aantal trainingsbeelden opgenomen. We hebben daar zowel beelden met een gewone camera als met een spectrograaf gemaakt.We hebben hier beelden gemaakt van de Amblyseius californicus. Bij deze mijtensoort is het verschil tussen de roof- en de prooimijt niet zo duidelijk, zoals te zien is in figuur 5.1. Hier zien we dat het grootste verschil zit in de fijne haartjes die de prooimijt heeft en de roofmijt niet. Nu deze foto werd gemaakt met een Guppy camera door CETI microscoop. We hebben een 13000-tal foto's genomen. Deze foto's moesten dan bewerkt worden om het algoritme te kunnen trainen, hier ga ik in de volgende paragraaf dieper op in gaan.



Figuur 5.1: *In het rode vierkant de Amblyseius californicus (roofmijt) en in het groene vierkant de prooimijt.*

5.1.2 Bewerken van de beelden

In deze paragraaf ga ik uitleggen welke bewerkingen allemaal uitgevoerd moesten worden op de beelden om het algoritme te kunnen trainen. We kunnen dit in een aantal delen opsplitsen: het handmatig detecteren van de mijten, deze mijten dan uit de foto knippen, ze roteren en ze opnieuw uit de geroteerde foto knippen op een formaat dat je zelf kiest zodat dit voor alle foto's hetzelfde is. Dit kan je ook zien in het schema in figuur 5.2. In de volgende paragraven zal ik deze stappen grondiger toelichten.



Figuur 5.2: *Schema van de stappen die doorlopen zijn om de beelden voor het trainingsalgoritme te bekomen.*

5.1.2.1 Handmatige detectie

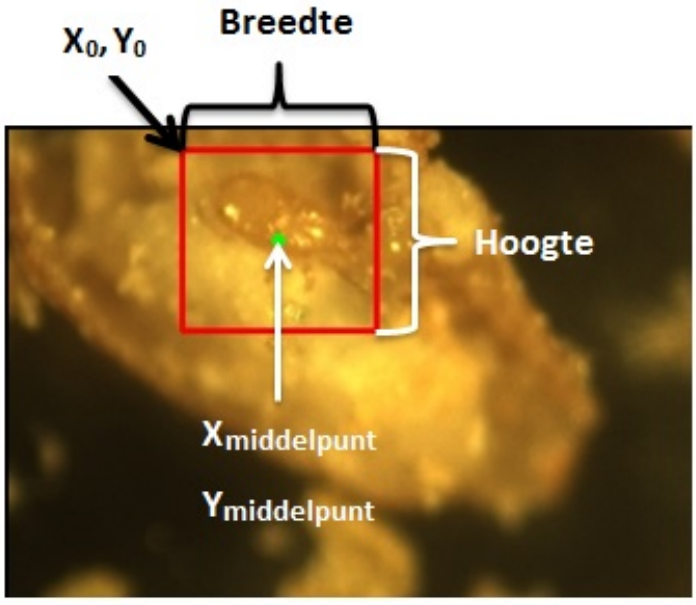
Ik heb met behulp van een annotatietool die ik verkregen heb vanuit EAVISE, deze handmatige detectie uitgevoerd. Hierbij moet je de folder waarin de foto's staan opgeven en specifiëren wat je allemaal in je output file wilt hebben, zoals te zien is in figuur 5.4. Vervolgens ga je door al de foto's en duid je de mijten aan die je in die foto ziet, waarna je naar de volgende foto gaat. Deze aanduidingen worden weggeschreven naar de output file. De output file is een txt-bestand waarvan de opbouw er als het volgt uitziet:

Padnaam Aantal detecties X_0 Y_0 Breedte Hoogte $X_{middelpunt}$ $Y_{middelpunt}$

Waarbij

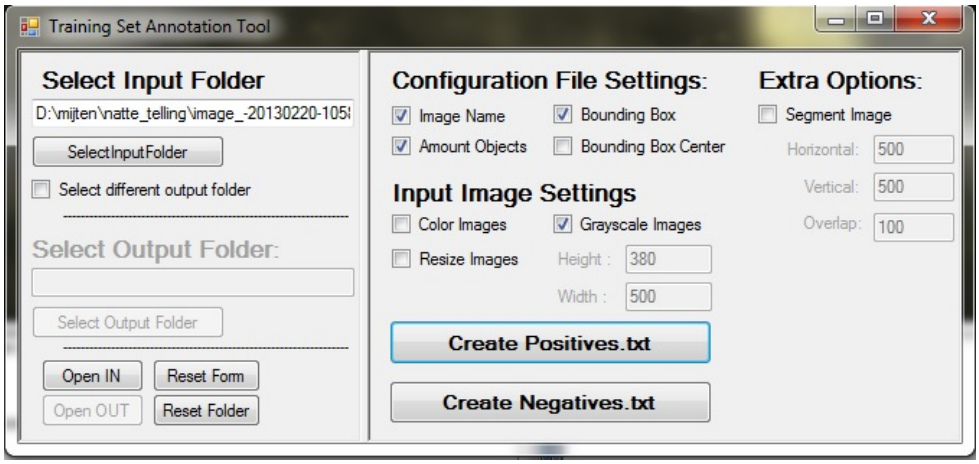
- **Padnaam:** Dit is de locatie waar de afbeelding is opgeslagen op de computer.
- **Aantal detecties:** Het aantal mijten dat in de afbeelding gedetecteerd is.
- X_0**:** De X coördinaat van het beginpunt (de linker bovenhoek).
- Y_0**:** De Y coördinaat van het beginpunt (de linker bovenhoek).
- **Breedte:** De breedte van het detectiegebied.
- **Hoogte:** De hoogte van het detectiegebied.
- $X_{middelpunt}$**:** De X coördinaat van het middelpunt (midden van het detectiegebied)
- $Y_{middelpunt}$**:** De Y coördinaat van het middelpunt (midden van het detectiegebied)
- **Voorbeeld:**

```
D:\mijten\droge_telling\image_-20130220-112811-230
\positives\0000000672.ppm 1 90 12 99 92 139 58
```



Figuur 5.3: *Annotatie afbeelding.*

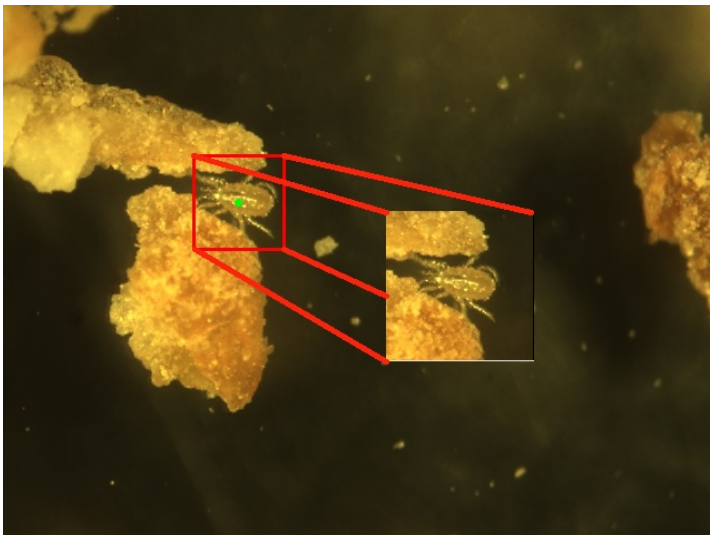
Deze output file zegt dus waar elke aangeduide mijt zich bevindt in de foto. Dit gaan we gebruiken om de verder fotobewerkingen te kunnen uitvoeren.



Figuur 5.4: *Annotatie tool.*

5.1.2.2 Uitknippen van de mijt uit de totale foto

We kunnen nu gebruik makende van de output file besproken in de vorige paragraaf, de mijten gaan uitsnijden uit de volledige foto's. Dit hebben we geautomatiseerd met het programma dat je in bijlage kan vinden onder Bijlage A. Hierbij gebruiken we de informatie uit de output file van de annotatietool om de mijten uit de foto te knippen. Deze waardes vergroten we met een bepaalde waarde (bijvoorbeeld 25 pixels langs iedere zijde) omdat wanneer we deze afbeeldingen roteren we toch voldoende informatie zouden blijven behouden. Dit is te zien in figuur 5.5, waarbij we de originele foto zien met de uitgeknipte versie samen en we zien dus dat de uitgeknipte versie groter is dan de aanduiding in de originele foto.



Figuur 5.5: *Hier zien we de originele foto en de uitgeknipte mijt.*

5.1.2.3 Alle mijten dezelfde oriëntatie geven

De mijten in de beelden die we aan het trainingsalgoritme geven moeten allemaal dezelfde oriëntatie hebben. De beelden moeten ook dezelfde grootte hebben. Dit is omdat het algoritme uniform getraind moet worden op deze mijten, als we bijvoorbeeld 50% van de mijten 180° zouden roteren tegenover de normale input dan gaat het algoritme niet de juiste oriëntatie weten en gaan we slechtere resultaten verkrijgen. Hiervoor heb ik een tweede programma geschreven in matlab. Een deel code heb ik gekregen vanuit EAVISE en ik heb deze verder aangepast. Dit programma kan je in bijlage vinden onder Bijlage B.

In dit programma bepalen we de gradiënten op elk punt zowel in x als in y-richting. Vervolgens bepalen we de grootte van deze gradiënten. Hierna gaan we met behulp van de boogtangens de hoek van de gradiënten bepalen in het juiste kwadrant.

Dan gaan we bepalen per stappen van hoeveel graden er gedraaid kan worden. We gaan dan de gradiënten verdelen over de (360° gedeeld door de grootte van stap) onderdelen. Dit doen we door de hoek van de gradiënt te delen door de grootte van de stap, zodat we de gradiënt van 0° tot 360° in het juiste onderdeel kunnen plaatsen. Dan tellen we de groottes van de gradiënten binnen een zelfde onderdeel op.

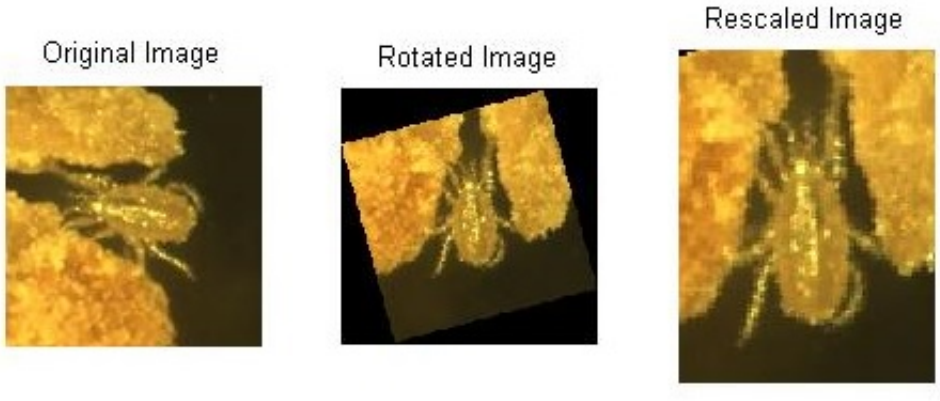
Nu hebben we een bereik tussen 0° en 360°, maar we gaan dit nu terug brengen tot een bereik tussen 0° en 180°, want bijvoorbeeld 45° en 225° zijn tegengestelde maar ze bepalen wel dezelfde richting. We kunnen alleen niet bepalen wat de boven of onderkant van de mijt is, maar dit is eenvoudig op te lossen door bij de rotatie eventueel 180° bij te tellen. We maken het eerste en het laatste onderdeel nul, omdat hier voornamelijk ruis in zit.

Hierna gaan we uit de overige onderdelen het maximum bepalen en de index hiervan gaan we vermenigvuldigen met de grootte van de stap om de juiste richting te verkrijgen, we tellen hier nog de helft van deze grootte bij op om het midden te verkrijgen. Zoals eerder vermeld tellen we hierbij nog 180° bij en dan krijgen we de afbeelding met de correcte oriëntatie zoals te zien is in figuur 5.6.

Hierna zorgen we er nog voor dat elke foto dezelfde dimensies krijgt. We gaan de mijten hierbij uitsnijden uit de geroteerde foto aan de hand van de originele breedte en hoogte van de mijt, zoals te zien in figuur 5.6. Vervolgens slagen we deze foto op en gaan verder met de volgende foto.

5.2 Trainen van het *Cascade* gebaseerd algoritme

Wanneer we nu alle mijten uit de foto's geknipt hebben, ze dezelfde oriëntatie en grootte gegeven hebben, op de manier zoals in de vorige paragraven beschreven is, dan kunnen we het algoritme beginnen te trainen.



Figuur 5.6: *Hier zien we de opeenvolgende stappen die het programma in Bijlage B overloopt.*

5.2.1 Vector file maken

Hiervoor moeten we eerst een vector file aanmaken, waarvoor een standaard programma in OpenCV zit. Ik heb dit in Visual Studio gecompileerd om tot het create_vector.exe te komen. Hierbij moet je een aantal parameters meegeven om deze vector te bekomen, zoals te zien is in de code hieronder.

```
D:\opencv\masterproef_exe>create_vector.exe
Usage: create_vector.exe
[-info <collection_file_name>]
[-img <image_file_name>]
[-vec <vec_file_name>]
[-bg <background_file_name>]
[-num <number_of_samples = 1000>]
[-bgcolor <background_color = 0>]
[-inv] [-randinv] [-bgthres
<background_color_threshold = 80>]
[-maxidev <max_intensity_deviation = 40>]
[-maxxangle <max_x_rotation_angle = 1.100000>]
[-maxyangle <max_y_rotation_angle = 1.100000>]
[-maxzangle <max_z_rotation_angle = 0.500000>]
[-show [<scale = 4.000000>]]
[-w <sample_width = 24>]
[-h <sample_height = 24>]
```

De voornaamste parameters hierbij zijn:

- **-info:** Hierbij geven we een tekst file mee waarin de locatie van de foto's staat, het aantal mijten in de foto, X_0, Y_0, breedte en hoogte, bijvoorbeeld:

  ```
  Final\mitesFinal_0_1.ppm 1 0 0 85 110
  ```

- **-vec:** Dit is de naam die we aan de vector file gaan geven.
- **-num:** Het aantal foto's dat we gaan gebruiken.
- **-w:** De breedte dat we aan de foto's geven.
- **-h:** De hoogte dat we aan de foto's geven.

5.2.2 Classificeerder trainen

Eenmaal we deze vector file hebben kunnen we het algoritme beginnen te trainen. Zoals te zien is in de code hieronder:

```
D:\opencv\masterproef_exe>train_cascade_model.exe
Usage of the cascade training algorithm:
-data <location of where the classifier needs to be stored>
-vec <vector file positive samples>
-bg <file negative samples>
-numPos <number of positive samples>
-numNeg <number of negative samples>
-numStages <number_of_stages - default 20>
[-baseFormatSave <only add if HAAR-like
old format is required>]
-featureType <HAAR - LBP - HOG>
-w <same width value as during sample creation>
-h <same height value as during sample creation>
-minHitRate < suggested value = 0.995>
-maxFalseAlarmRate <suggested value = 0.5>
-maxDepth <1 creates stumps in binary trees>
-maxWeakCount <suggested amount of weak classifiers per stage
   >
[-mode <only in case of HAAR - BASIC is upright -ALL is
   upright
and 45 degrees rotated>]
```

Ook hier zijn weer een aantal parameters noodzakelijk om het programma correct uit te voeren:

- **-data:** De plaats waar de classificeerder moet opgeslagen worden, relatief ten opzichte van waar het train_cascade_model.exe staat.
- **-vec:** De vector file die ik in het vorige puntje besproken heb, dient hier meegegeven te worden.
- **-bg:** Dit is een tekst bestandje waarin de locaties van achtergrondafbeeldingen staan.
- **-numPos:** Het aantal positieve beelden dat je wilt gebruiken om de classificeerder te trainen.

- **-numNeg:** Het aantal achtergrond beelden dat je gebruikt.
- **-numStages:** Het aantal stages dat je wilt doorlopen.
- **-featureType:** Het type van classificeerder dat je wenst te bekomen.
- **-w:** De breedte, deze moet dezelfde zijn als bij het maken van de vector file.
- **-h:** De hoogte, ook deze moet dezelfde zijn als bij het maken van de vector file.
- **-minHitRate:** Minimale gewenste succesratio per stage.
- **-maxFalseAlarmRate:** Maximale gewenste valse alarm waarde voor iedere stage.
- **-maxDepth:** De diepte van de binaire bomen, meestal 1 en dan hebben we stompen.
- **-maxWeakCount:** Het aantal zwakke classificeerders per stage.

Een voorbeeld van deze parameters is:

```
train_cascade_model.exe -data Data -vec positive.vec
-bg negatives.txt -numPos 100 -numNeg 300
-numStages 20 -featureType LBP -w 43 -h 55
-minHitRate 0.995 -maxFalseAlarmRate 0.5 -maxDepth 1
-maxWeakCount 2
```

Dan gaat het programma het trainingsalgoritme starten, dit kan tot een aantal uren duren en wanneer dit dan klaar is hebben we onze classificeerder getraind en kunnen we beginnen met detecties proberen uit te voeren op enkele test afbeeldingen.

5.3 Kwalitatieve resultaten: HOG & LBP

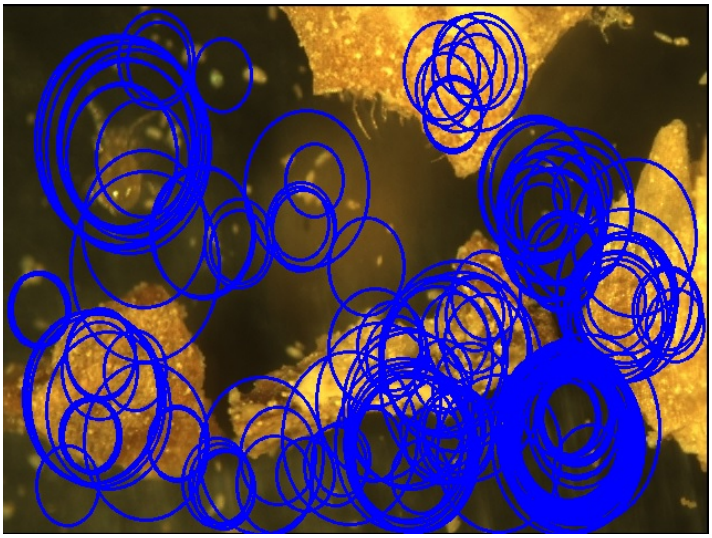
In deze paragraaf geef ik een aantal resultaten weer van de detector. Bij het trainen van de detector kan je een aantal parameters instellen die een invloed gaan hebben op hoe goed de detector gaat werken. Twee belangrijke parameters bij het trainen van de detector zijn het aantal positieve beelden (numPos) en het aantal achtergrond beelden (numNeg) dat je gebruikt. Bij het programma dat we gebruiken om de detector te testen kan je het aantal overlappen instellen. Wanneer je deze parameter op 0 zet dan ga je zeer veel detecties krijgen, waarbij de meeste onjuist zijn. Wanneer je dan het aantal overlappen verhoogt ga je steeds detecties krijgen, totdat je slechts één detectie overhoudt. De trainingsparameters moet je goed instellen, in de volgende paragrafen zie je van welke parameters vertrokken zijn en hoe we deze hebben bijgesteld tot we betere detecties bekwamen.

5.3.1 Initiële resultaten

We zijn in het begin gestart de detector te trainen met volgende parameters:

```
train_cascade_model.exe -data Data -vec positives.vec -bg negatives.txt
-numPos 50 -numNeg 150 -numStages 15 -featureType [LBP - HOG] -w 43 -h 55 -minHitRa
-maxFalseAlarmRate 0.5 -maxDepth 1 -maxWeakCount 5
```

Hierbij was de featureType parameter LBP in het geval van de local binary patterns en HOG in geval van de histogram of oriented gradients. Wanneer je het aantal overlappen op 0 zet dan ga je zeer veel detecties krijgen, waarbij de meeste onjuist zijn, dit kan je zien in figuur 5.7 voor de LBP classificeerder en in figuur 5.8 voor de HOG classificeerder. We zien hier dat HOG de mijt vanaf stap één niet kan detecteren, bij LBP lukt dit wel.



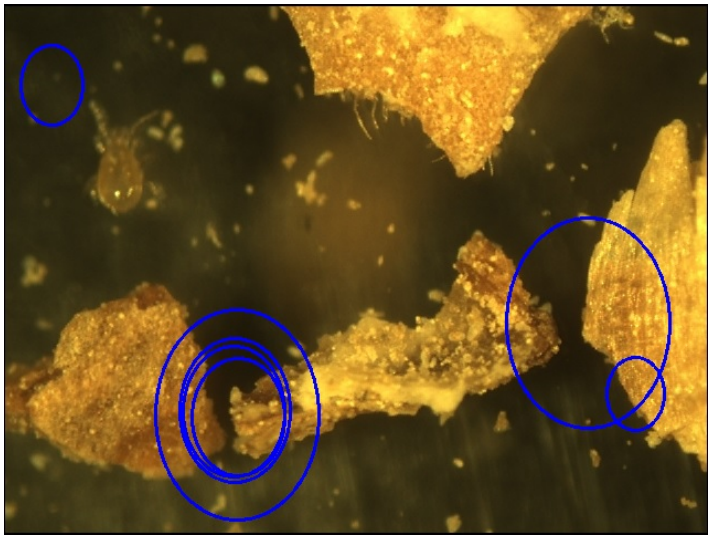
Figuur 5.7: *Testen van de LBP detector, aantal overlappen is 0.*

Wanneer we het aantal overlappen verhogen gaan we steeds minder detecties verkrijgen. Dit kunnen we zien in figuur 5.9 voor de LBP classificeerder en in figuur 5.10 voor de HOG classificeerder. We zien hier dat bij LBP naast de mijt nog drie andere detecties gebeurd zijn. Bij HOG was de mijt in de eerste fase met 0 overlappen al niet gedetecteerd dus kan ze hier zeker niet gedetecteerd zijn. Aangezien er achtergrond nog aanzien wordt als een te detecteren object, zijn we meer negatieve beelden (achtergrond beelden) gaan intrainen. Hierbij hebben we gezorgd dat deze negatieve beelden zo goed mogelijk overeen kwamen met de foute detecties die gebeurd zijn, resultaten hiervan staan in de volgende paragraaf.

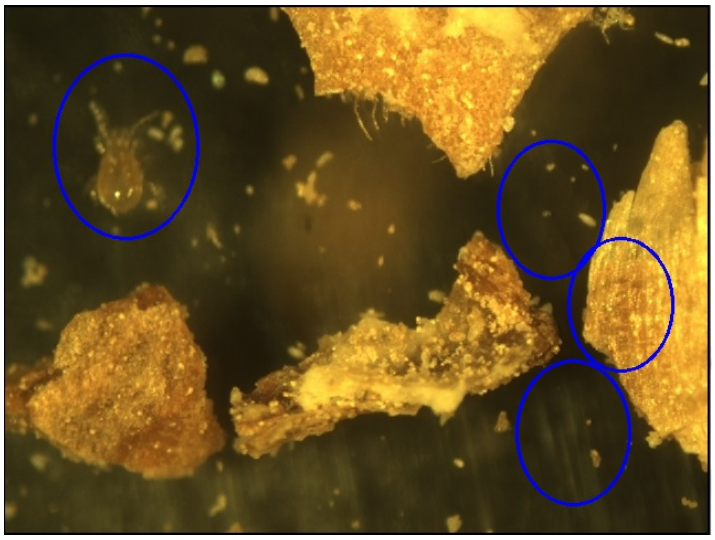
5.3.2 Optimaliseren van de detector

Hier hebben we de classificeerder getraind met meer negatieve beelden, de parameters die gebruikt werden waren de volgende:

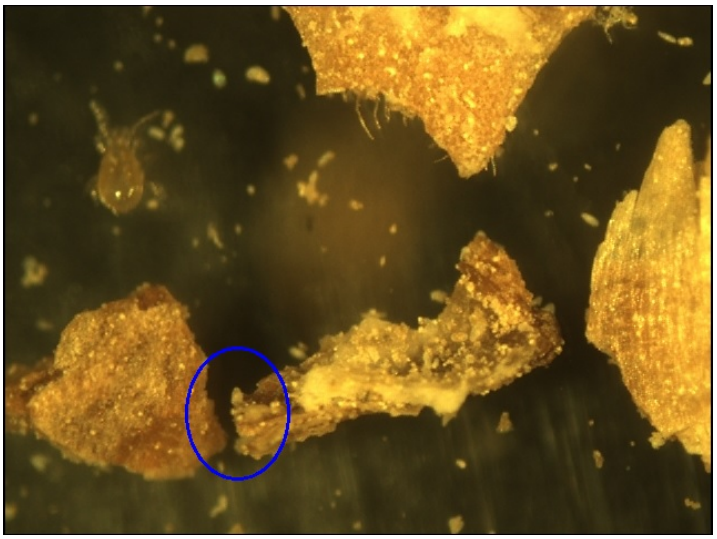
```
train_cascade_model.exe -data Data -vec positives.vec -bg negatives.txt
```



Figuur 5.8: *Testen van de HOG detector, aantal overlappen is 0.*



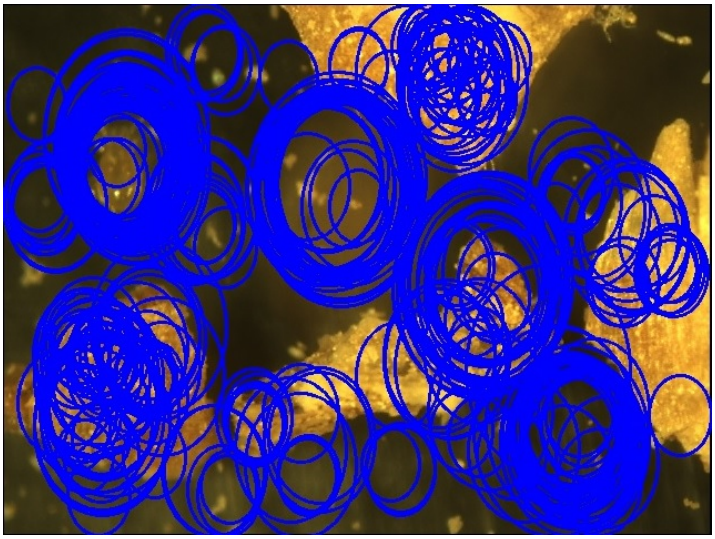
Figuur 5.9: *Testen van de LBP detector, aantal overlappen is 8.*



Figuur 5.10: *Testen van de HOG detector, aantal overlappen is 2.*

```
-numPos 128 -numNeg 500 -numStages 15 -featureType [LBP - HOG] -w 43 -h 55
-minHitRate 0.995 -maxFalseAlarmRate 0.5 -maxDepth 1 -maxWeakCount 10
```

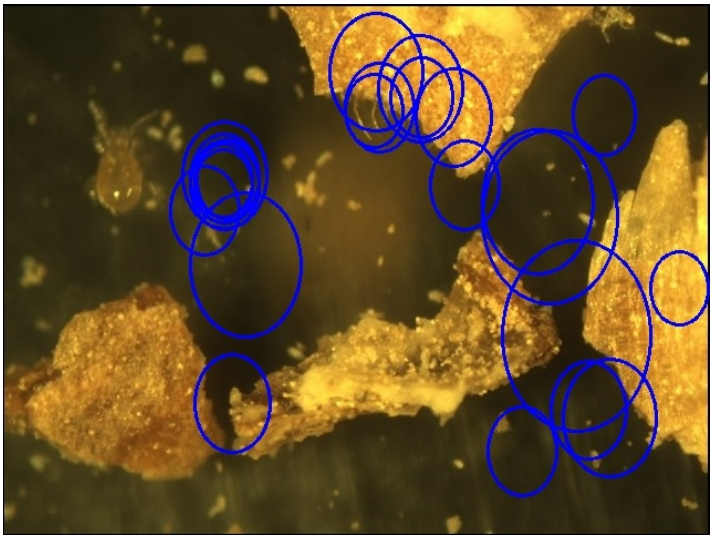
Hier starten we ook met 0 overlappen, dit kan je zien in figuur 5.11 voor de LBP classificeerder en in figuur 5.12 voor de HOG classificeerder. We verhogen het aantal overlappen



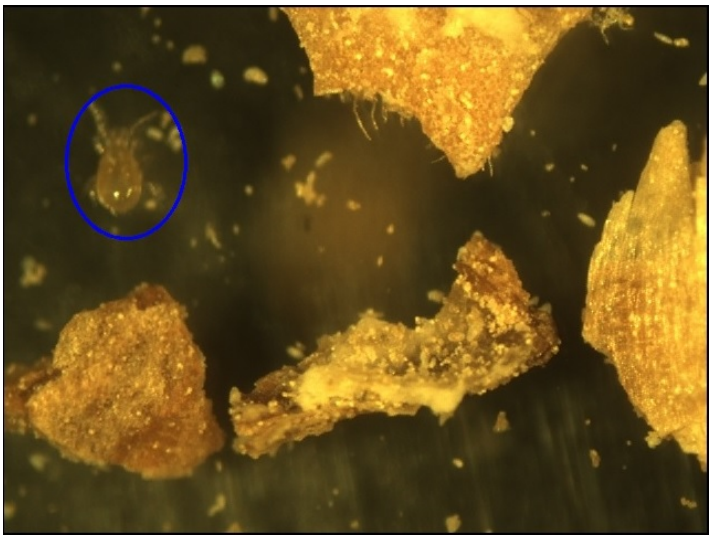
Figuur 5.11: *Testen van de LBP detector, aantal overlappen is 0.*

en zien dat de LBP classificeerder bij 40 overlappen enkel nog de mijt overhoudt. Bij de HOG classificeerder werd de mijt alweer niet gedetecteerd in de eerste fase met 0 overlappen. In figuur 5.13 zien we de LBP classificeerder en in figuur 5.14 de HOG classificeerder.

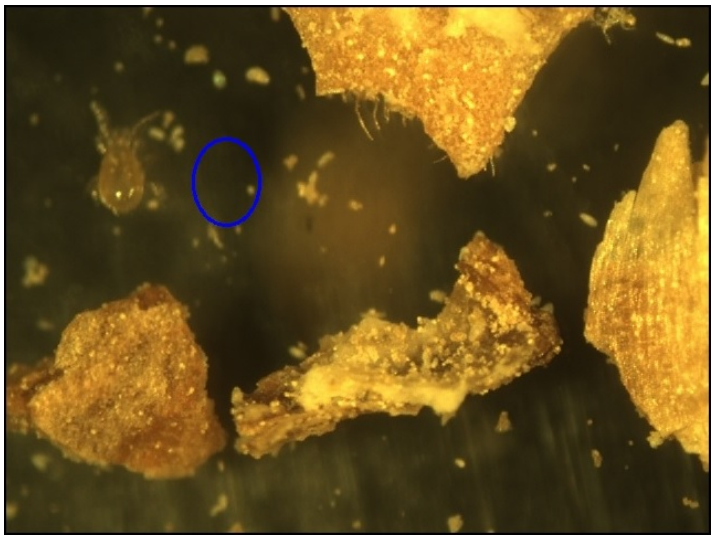
In het volgende hoofdstuk zullen we het aantal overlappen laten variëren voor een aantal foto's om zo meer data te bekomen en de correctheid van de detector op meerder foto's controleren. Dit zullen we doen door gebruik te maken van precision-recall curves, welke in paragraaf 6.2 besproken zullen worden.



Figuur 5.12: *Testen van de HOG detector, aantal overlappen is 0.*



Figuur 5.13: *Testen van de LBP detector, aantal overlappen is 40.*



Figuur 5.14: *Testen van de HOG detector, aantal overlappen is 3.*

5.4 Conclusie

We zien hier dat de LBP classificeerder een veel beter resultaat oplevert ten opzichte van de HOG classificeerder. We merken ook op dat de achtergrond vaak gedetecteerd wordt als een mijt. Dit kunnen we oplossen door meer negatieve beelden in te trainen. Deze negatieve beelden moet je kiezen aan de hand van welke delen in de achtergrond als mijt worden gedetecteerd.

Hoofdstuk 6

Evaluatie

In dit hoofdstuk gaan we de performantie van het algoritme testen. Verder gaan we de trainingsparameters van de classificeerder optimaliseren, dit werd al aangehaald in paragraaf 5.3.2, maar daar gaan we nu dieper op in. We gaan hier ook zien wat de relatie is tussen het optimaliseren van de parameters en de tijd die nodig is om de classificeerder te trainen. Waarna dit hoofdstuk eindigt met een paragraaf waarin ik enkele precision-recall curves vergelijk.

6.1 Toepassing algoritme

Nu gaan we de classificeerders testen, hiervoor heb ik een programma geschreven dat je kan vinden onder Bijlage C. In de code heb ik een deel geschreven om met user input te werken en een deel dat automatisch de testen uitvoert op een reeks foto's met een bepaald aantal classificeerders. Het programma gaat eerst een foto en een classificeerder inlezen. Waarna we het programma de detecties gaat uitvoeren. We moeten hier ook een aantal argumenten meegeven:

- **Afbeelding:** De afbeelding waarop we de detectie gaan uitvoeren.
- **Rechthoeken:** Een vector van rechthoeken waarin elke rechthoek een te detecteren object zou bevatten.
- **Herschaling:** Geeft aan hoeveel de afbeelding herschaald wordt bij elke schaal.
- **Aantal overlappen:** Het aantal detecties dat moeten overlappen om een detectie te bekomen.
- **Vlaggen:** Wordt enkel gebruikt voor de oude HAAR detecties en niet meer bij de nieuwe classificeerders.
- **Minimum grootte:** De minimale grootte van het te detecteren object, kleinere detecties worden genegeerd.
- **Maximum grootte:** De maximale grootte van het te detecteren object, grotere detecties worden genegeerd.

Hierna schrijven we de bekomen informatie naar een tekst file. Deze informatie gaan we gebruiken om precision-recall curves op te stellen. Bij handmatige input is dit één foto en gaan we de detecties weergeven op de foto. Wanneer we automatisch over een aantal afbeeldingen gaan met een aantal classificeerders dan laten we het weergeven van de foto achterwegen en gaan we itereren over het aantal overlappen en schrijven we voor elke classificeerder en foto deze informatie naar een andere tekst file.

6.2 Definitie precision-recall curves

Bij precision-recall curves gaan we de detecties onderverdelen in relevante en niet relevante detecties. Hiermee kunnen we dan precision en recall berekenen. Precision wordt gedefinieerd als het aantal relevante detecties gedeeld door het totaal aantal detecties. Recall kan dan weer gedefinieerd worden als het aantal correcte detecties gedeeld door alle relevante detecties. Bij classificaties spreken we van true positives, false positives en false negatives, waarbij:

- **True positives:** Het aantal correct gelabelde detecties, die tot de mijten behoren.
- **False positives:** Het aantal incorrect gelabelde detecties, die niet tot de mijten behoren.
- **False negatives:** Het aantal dat niet gelabeld is als mijt, maar dat wel als mijt gelabeld zou moeten zijn.

Hiermee kunnen we dan precision en recall berekenen.

$$Precision = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$Recall = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

6.3 Programma

Om de precision-recall curves te bekomen heb ik een programma geschreven in matlab, dit is te vinden onder Bijlage D. Hier gaan we eerst de coördinaten van een mijt uit de test foto's inlezen. Daarna gaan we per overlap (detecties van nul overlappen tot detecties bij 119 overlappen) de coördinaten van de mijten uit de test foto's vergelijken met de coördinaten van alle detecties die bij dat aantal overlappen zijn gevonden. We gaan bij deze vergelijking kijken of de detecties het middelpunt van de coördinaten van de mijt uit de test foto's bevat en dus ook zeker de mijt bevat. Aan de hand van deze vergelijking gaan we dan bepalen of de detectie een true positive, een false negative of een false positive is. Dit doen we voor alle verschillende overlappen en verschillende foto's. We slagen deze waardes op in de globale true positives, false negatives en false positives. Deze lopen van

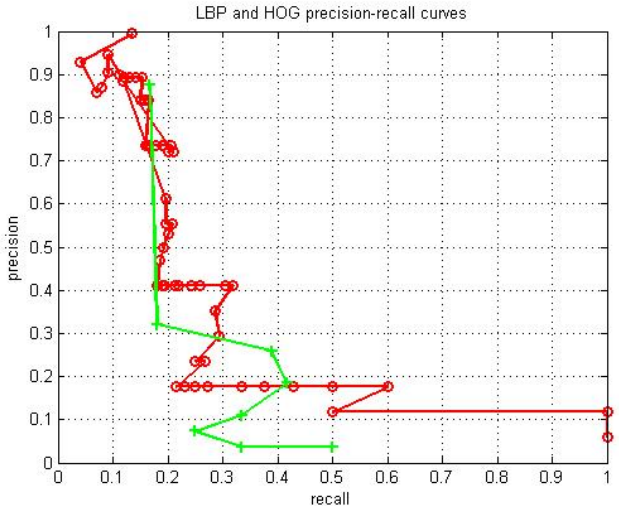
nul tot 119 overlappen en bevatten de waardes van alle test foto's. Hierna berekenen we de precision en recall voor de verschillende overlappen. Als laatste gaan we deze waardes plotten naar een grafiek om zo onze precision-recall curves te bekomen.

6.4 Curves

Als belangrijkste resultaten ga ik het hier hebben over een aantal precision-recall curves, waarbij we voor elke curve de trainingsparameters van de classificeerders licht gewijzigd hebben. Deze curves laten ons dan duidelijk zien welke parameters relevant zijn bij het proberen te detecteren van mijten en zorgt dus voor bruikbare informatie voor latere toepassingen.

6.5 Vergelijking LBP en HOG

We zijn begonnen met enkele HOG en LBP classificeerders te trainen om te kijken welke van deze twee we het beste konden gebruiken om de mijten te detecteren. In figuur 6.1 zie je in het rood de LBP classificeerder en in het groen de HOG classificeerder. De



Figuur 6.1: *Precision-recall curve van de HOG en de LBP classificeerder.*

classificeerders zijn getraind met de volgende parameters.

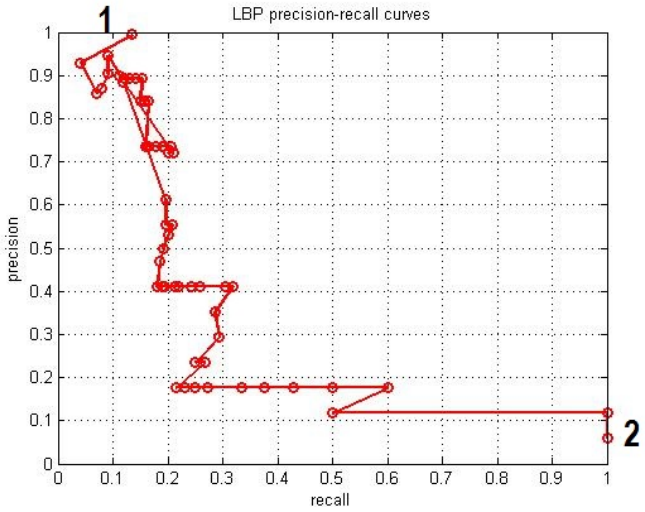
```
train_cascade_model.exe -data Data -vec positives.vec -bg negatives.txt
-numPos 128 -numNeg 500 -numStages 15 -featureType [LBP - HOG] -w 43 -h 55
-minHitRate 0.995 -maxFalseAlarmRate 0.5 -maxDepth 1 -maxWeakCount 10
```

We zien ook hier dat de LBP classificeerder een beter resultaat geeft dan de HOG classificeerder, dit komt overeen met de Kwalitatieve resultaten besproken in paragraaf 5.3.

Hierna ben ik overgestapt om enkel nog te werken met de LBP classificeerder.

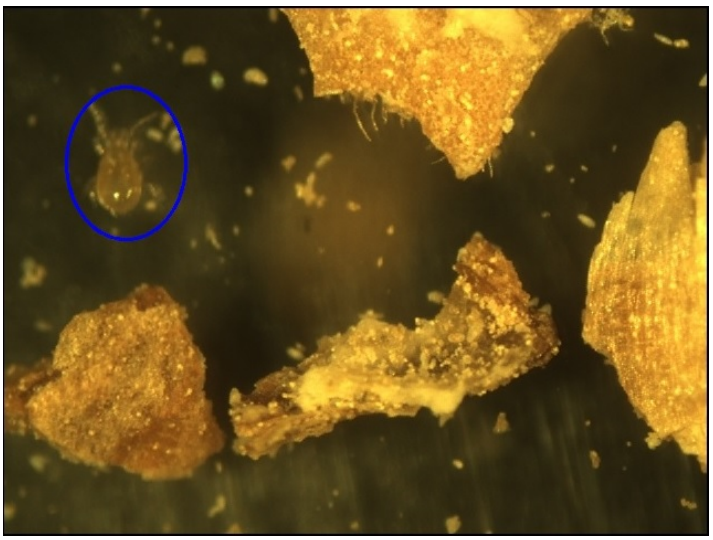
6.6 Wijzigen parameters LBP

Verder kan je nu gaan spelen met de parameters wanneer je een classificeerder traint. We gaan de classificeerder met de trainingsparameters vanuit paragraaf 6.5 gebruiken om de andere classificeerders met te vergelijken. De precision-recall curve hiervan is te zien in figuur 6.2. Bij punt één in deze figuur hebben we zeer veel overlappen dit kan



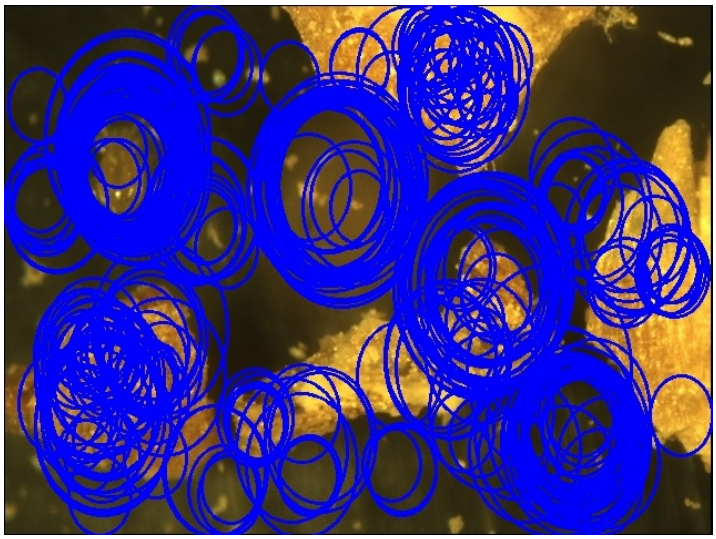
Figuur 6.2: *Precision-recall curve van de LBP classificeerder.*

je zien in figuur 6.3. Bij punt twee in deze figuur hebben we zeer weinig overlappen



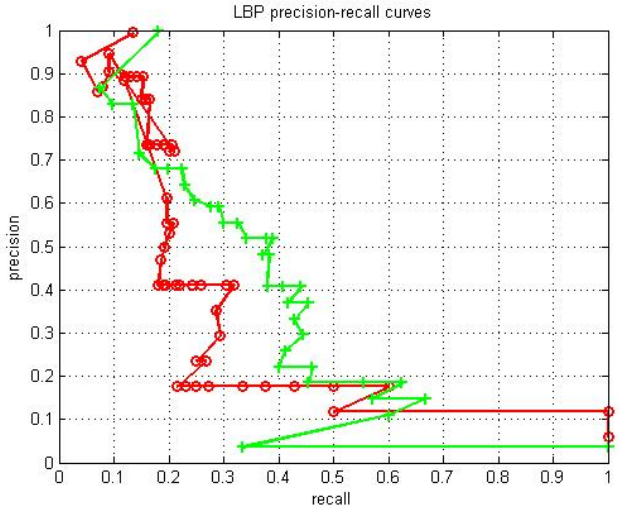
Figuur 6.3: *Deze foto laat zien wat er op plaats 1 gebeurt in figuur 6.2.*

dit kan je zien in figuur 6.4. In de volgende figuren zien we telkens in het rood de standaard classificeerder waar we met gaan vergelijken en in het groen de aangepaste



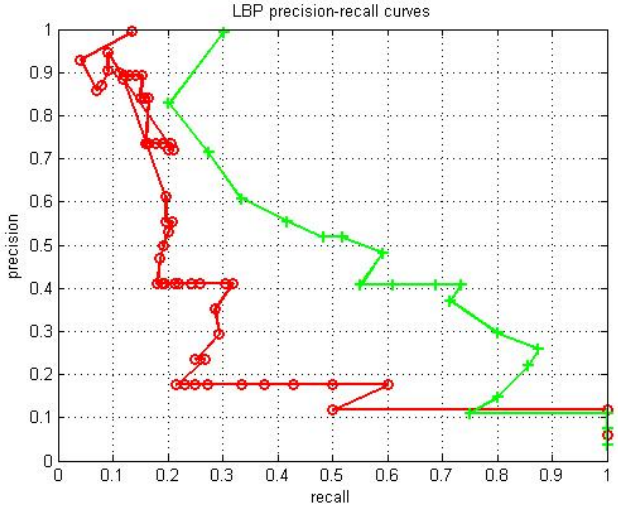
Figuur 6.4: *Deze foto laat zien wat er op plaats 2 gebeurt in figuur 6.2.*

classificeerder. In een eerste poging gaan we het aantal stages in de cascadestructuur van 15 naar 20 verhogen, we zien in figuur 6.5 dat dit al een kleine verbetering oplevert. Wanneer je het aantal stages blijft verhogen gaat de intraining op een bepaalde moment stoppen omdat aan de voorwaarden van de andere parameters is voldaan. Nu deze kleine verbetering levert een trainingstijd die ongeveer 2.5 × groter is dan die van de standaard classificeerder. De trainingstijd van de standaard classificeerder bedraagt 36 minuten en 41 seconden. De grootste verbetering is te zien in figuur 6.6, hierbij hebben we het aantal



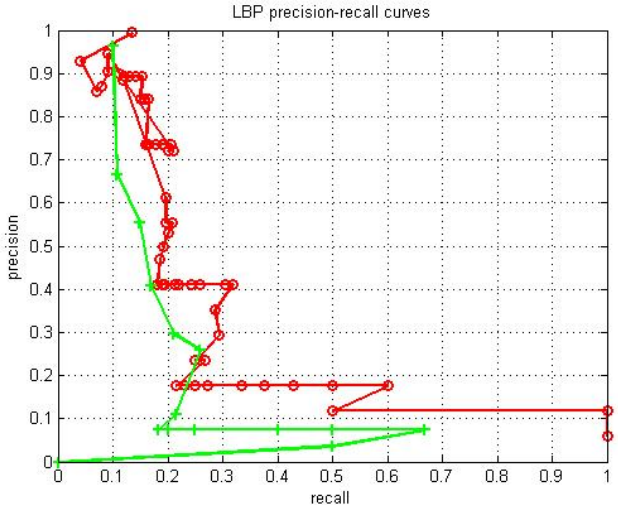
Figuur 6.5: *Invloed van het aantal te doorlopen stages.*

negatieve beelden van 500 naar 3000 verhoogd. Dit heb ik gedaan omdat wanneer ik de detecties van de standaard classificeerder op de afbeeldingen ging weergeven, ik merkte dat de classificeerder zeer veel achtergrond als mijt ging detecteren (zie paragraaf 5.3.1 en 5.3.2). Door dan het aantal negatieve beelden te verhogen gaan deze valse detecties verminderen. Dit levert een trainingstijd die ongeveer 34 × groter is dan die van de standaard classificeerder. Ik heb ook het aantal positieve beelden verminderd van 128



Figuur 6.6: *Invloed van het veranderen van het aantal negatieve beelden.*

naar 50 om de invloed hiervan te bekijken. Het resultaat hiervan kan je zien in figuur 6.7. Je kan duidelijk zien in de grafiek dat dit een nadelige invloed heeft op de detecties.



Figuur 6.7: *Invloed van het veranderen van het aantal positieve beelden.*

6.7 Conclusie

We hebben in dit hoofdstuk gezien, zoals we in paragraaf 5.3 reeds hadden aangehaald, dat de LBP classificeerder beter gebruikt kan worden om mijten te detecteren dan de HOG classificeerder. Verder hebben we dan ook gezien dat je een aantal parameters kan aanpassen waardoor de classificeerder beter of slechter de mijten gaat detecteren. In de meeste gevallen werd een groot aantal valse detecties gevonden, maar dit kunnen we verbeteren door een groter aantal negatieve beelden mee in te trainen. Deze negatieve

beelden worden gekozen na een eerste trainingsstap, waarbij we dan kijken welke achtergronden als mijt werden gedetecteerd. Je moet hierbij wel rekening houden dat hoe meer beelden je in traint hoe langer de training zal duren. We kunnen ook het aantal stages in de cascadestructuur verhogen, dit geeft ook een verbetering. We kunnen het aantal stages wel niet oneindig verhogen want de training stopt na een aantal stages vanzelf als aan de voorwaarden van de andere parameters is voldaan.

Hoofdstuk 7

Besluit

We zijn deze masterproef begonnen met het idee dat we de Phytoseiulus persimilis moesten proberen te detecteren, maar dit werd na de tussentijdse presentatie door Biobest veranderd naar de Amblyseius californicus. Dit maakte het moeilijker omdat er hierbij minder verschil is tussen de roof- en de prooimijt.

We hebben dan in de literatuurstudie een aantal classificeerders besproken die in aanmerking zouden komen om de detecties uit te voeren. We hebben dan hier de keuze gemaakt om te kiezen voor de HOG en de LBP classificeerders en om deze in een cascadestructuur te trainen. In eerste instantie dachten we er ook aan om spectrale beeldvorming te gebruiken, maar dit gaf geen bruikbaar resultaat.

In paragraaf 5.3 hebben we dan in een eerste instantie gezien dat de LBP classificeerder het er een stuk beter vanaf bracht dan de HOG classificeerder. Dit werd ook in paragraaf 6.5 bevestigd, waardoor we dus voor de LBP classificeerder gekozen hebben.

Bij het trainen van deze LBP classificeerder kan je een aantal parameters laten variëren. We hebben de variatie van deze parameters op de detectieresultaten onderzocht. We zien hier dat een voldoende aantal positieve en negatieve beelden noodzakelijk is om correcte detecties te bekomen. We hebben ook gezien dat het aantal stages dat de classificeerder doorloopt een invloed heeft om betere detecties te bekomen. Eens de classificeerder getraind is kan de detectie in real-time of met een kleine vertraging gebeuren.

In de toekomst kan deze classificeerder nog verbeterd worden door een groter aantal positieve en negatieve beelden in te trainen met meer variatie. Momenteel kunnen enkel de rechtopstaande mijten gedetecteerd worden, dit moet nog aangepast worden zodat ofwel de classificeerder getraind wordt met mijten in alle richtingen of door de classificeerder te draaien en opnieuw over de foto te laten bewegen. Het probleem is ook dat de mijten die volledig of gedeeltelijk onder de draagstof zitten niet gedetecteerd kunnen worden. Verder moet de classificeerder nog getraind worden op de andere mijten soorten.

Bijlage A

Image Cropper

```
clear all
close all;
clc;

%-----------------------------------------------------------------------
% Image cropper
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% File codes:
% 20130220-112237-715 (prooi)
% 20130220-112348-537 (roof) (mites)
% 20130220-112708-094 (roof) (mites1)
% 20130220-112754-952 (roof) (mites2)
% 20130220-112811-230 (roof) (mites3)
% 20130220-112855-308 (roof) (mites4)
% 20130220-112910-112 (roof) (mites5) (leeg)
% 20130220-112925-285 (roof) (mites6)
%-----------------------------------------------------------------------


% Open file, couple a file handler
fid = fopen('D:\mijten\droge_telling\image_-20130220-112925-285\
positives-image_-20130220-112925-285.txt');

% Get a first line from file
readLine = fgets(fid);

% Process line, continue until there is no more line to fetch
filenameReference = '';

% Create a vector to store the square containing the mites
```

```
rectangle = [];

% Set count
count = 0;

while ischar(readLine)
    % Split char array based on tab
    elements = regexp(readLine, ' ', 'split');
    filename = elements{1,1};

    % Read image containing mites
    image = imread(filename, 'ppm');

    % Count the amount of detections on the line
    amountDetections = uint8(str2double(elements{1,2}));

    % Add the width and height of each detection to the array
    addValue = 25;
    i=1;
    for i=1:amountDetections
    % Put [Xmin Ymin Width Height] in rectangle
    % (if centerpoints specified correct else change *6 in *4)
        rectangle = [(uint16(str2double(elements{1,3+((i-1)*6)})))-addValue
            (uint16(str2double(elements{1,4+((i-1)*6)})))-addValue
            (uint16(str2double(elements{1,5+((i-1)*6)})))+2*addValue
            (uint16(str2double(elements{1,6+((i-1)*6)})))+2*addValue];
    % Crop the mites in the image
        newImage = imcrop(image, rectangle);
    % Write the image back to a new file
        folder = 'D:\mijten\Cropped\Roofmijt\Before_Rotate';
        newFilename = sprintf('mites6_%d_%d.ppm', count , i);
        fullFilename = fullfile(folder, newFilename);
        imwrite(newImage, fullFilename, 'ppm', 'Encoding', 'ASCII');
    end
    % increment count for next image
    count = count +1;
    % Get the next line that needs to be processed
    readLine = fgets(fid);
end
% Close the handler to the file
fclose(fid);
```

Bijlage B

Dominant orientation

```
clear all;
close all;
clc;

%-------------------------------------------------------------------------
% Acquiring dominant orientation
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
% File codes:
% 20130220-112237-715 (prooi)
% 20130220-112348-537 (roof) (mites)
% 20130220-112708-094 (roof) (mites1)
% 20130220-112754-952 (roof) (mites2)
% 20130220-112811-230 (roof) (mites3)
% 20130220-112855-308 (roof) (mites4)
% 20130220-112910-112 (roof) (mites5) (leeg)
% 20130220-112925-285 (roof) (mites6)
%-------------------------------------------------------------------------

% Open file, couple a file handler
fid = fopen('D:\mijten\droge_telling\image_-20130220-112348-537\
positives-image_-20130220-112348-537.txt');

% Get a first line from file
readLine = fgets(fid);

% Set count
count = 0;

% Set folder containing images
folder = 'D:\mijten\Cropped\Roofmijt\Before_Rotate';
```

```
while ischar(readLine)
    % Split char array based on tab
    elements = regexp(readLine, ' ', 'split');
    amountDetections = uint8(str2double(elements{1,2}));

    % Read image containing mites
    for det=1:amountDetections
        newFilename = sprintf('mites_%d_%d.ppm', count , det);
        fullFilename = fullfile(folder, newFilename);
        img = imread(fullFilename, 'ppm');
        img1 = im2double(img);
        subplot(1,4,1);
        imshow(img);title('Original Image');

        kernel = fspecial('gaussian',[10 10],5);

        smoothed = imfilter(img1,kernel,'replicate','full');
        subplot(1,4,2);
        imshow(smoothed);title('Smoothed Image');

        % Creation of gradients plot of the input image to determine
        % the dominant orientation.
        % This orientation will always be perpendicular towards the
        % largest gradient component.
        [Gx Gy] = gradient(smoothed);

        % First calculate the size of the vectors and the angles that
        % they are under using the arctangens function.
        arrowLength = sqrt(Gx.^2 + Gy.^2);
        % Angle will be dependant of which quadrant will be available
        % and can be between 0 and 360 degrees.
        % - Angle 0 degrees and 90 degrees -> arctan(y/x) will give
        % the angle of the arrow
        % - Angle 90 degrees and 180 degrees ->
        % 90 degrees + arctan(-x/y); ABS needed!
        % - Angle 180 degrees and 270 degrees ->
        % 180 degrees + arctan(-y/-x);
        % - Angle 270 degrees and 360 degrees ->
        % 270 degrees + arctan(x/-y); ABS needed!
        % However negative values here show that the actual value
        % will be negative since values can range from -255 to 255
        % for each gradient component
        % Put arrowAngle full of zeros (not necessary?)
        arrowAngle = zeros(size(arrowLength,1),size(arrowLength,2));
        for i=1:size(arrowLength,1)
```

```
    for j=1:size(arrowLength,2)
        % First quadrant
        if(Gx(i,j) > 0 && Gy(i,j) > 0)
            arrowAngle(i,j) = atand(Gy(i,j)/Gx(i,j));
        end
        % Second quadrant
        if(Gx(i,j) < 0 && Gy(i,j) > 0)
            arrowAngle(i,j) = 90 + abs(atand(Gx(i,j)/Gy(i,j)));
        end
        % Third quadrant
        if(Gx(i,j) < 0 && Gy(i,j) < 0)
            arrowAngle(i,j) = 180 + atand(Gy(i,j)/Gx(i,j));
        end
        % Fourth quadrant
        if(Gx(i,j) > 0 && Gy(i,j) < 0)
            arrowAngle(i,j) = 270 + abs(atand(Gx(i,j)/Gy(i,j)));
        end
    end
end

% Create histogram bins
% Angle step size needs to be defined
angleStep = 10;
bins = 0:angleStep:360;
amountBins = size(bins,2) - 1;

% Go through all arrow values, add to the corresponding bin
% to define the largest/dominant oriÃńntation
assignedBin = floor(arrowAngle ./ angleStep);
% Make sure that a number 0 doesnt exist,
% because this is the first bin!
% This is to make the values mean more
assignedBin = assignedBin + 1;

valuesBin = zeros(1,size(bins,2)-1);

for i = 1:size(arrowLength,1)
    for j = 1:size(arrowLength,2)
        index = assignedBin(i,j);
        valuesBin(1,index) = valuesBin(1,index)
                             + arrowLength(i,j);
    end
end

% Since we now got ranges of 0 - 360 degrees but we want
% counterpart edges to be placed together.
```

```
% For example, +45 degrees and +225 degrees,
% are contradicting arrows, BUT they show the same direction.
% Therefore, we will reduce the range to 0-180 degrees which can
% contain each possible orientation of an object not
% taking into account top or bottom of the object.

% However due to the stepSize, this arent always the
% first 180 elements, this should be written in function
% of this value.

firstPartDegrees = valuesBin(1,1:amountBins/2);
secondPartDegrees = valuesBin(1,(amountBins/2)+1:amountBins);
totalValues = firstPartDegrees + secondPartDegrees;

% We should be sure to demp the first bin,
% because all no value gradients go there,
% and this can never be a maximum gradient,
% therefore replace the value by an arbitrarily 0.
% Same goes for the last value.
totalValues(1,1) = 0;
totalValues(1,size(totalValues,2)) = 0;

% Look for the largest direction value
[value index] = max(totalValues);

% Use the index to retrieve the direction
direction = (index * angleStep) + (angleStep / 2);

% Now rotate the image around it's centerpoint,
% based on the most dominating orientation.
% This means that the most dominant orientation
% needs to be around the vertical or horizontal
% axes depending on the application.
% --> in order to rotate towards vertical axes apply direction
% --> in order to rotate towards horizontal axes
% apply -(90 degrees - direction)
rotatedImage = imrotate(img, direction+180);
subplot(1,4,3);
imshow(rotatedImage);title('Rotated Image');

% Cropping after rotation ( need larger rectangle )
widthImage = (uint16(str2double(elements{1,5+((det-1)*6)})));
lengthImage = (uint16(str2double(elements{1,6+((det-1)*6)})));
if lengthImage < widthImage
    change = lengthImage;
    lengthImage = widthImage;
```

```
            widthImage = change;
        end
        addValue = 25;
        rectangle = [3*addValue/2 addValue widthImage+addValue/2
                             lengthImage+addValue];
        newImage = imcrop(rotatedImage, rectangle);
        newImageScale = imresize(newImage, 'OutputSize', [110 85]);

        subplot(1,4,4);
        imshow(newImageScale);title('Rescaled Image');

        % Write the image back to a new file
        folderRot = 'D:\mijten\Cropped\Roofmijt\Final';
        newFilenameRot = sprintf('mitesFinal6_%d_%d.ppm', count , det);
        fullFilenameRot = fullfile(folderRot, newFilenameRot)
        imwrite(newImageScale, fullFilenameRot,
                 'ppm', 'Encoding', 'ASCII');

    end
    count = count +1;

    % Get the next line to see number of detections
    readLine = fgets(fid);
end

% Close the handler to the file
fclose(fid);
```

Bijlage C

Detection test program

```
/**
 * Detect mites using user input/or
 */
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
#include <fstream>
#include <stdio.h>

using namespace std;
using namespace cv;

// Function Headers
void detectAndDisplay( Mat frame );

// Global variables
int casnum, imnum;
string mites_cascade_name;
string image_name;
string image_number;
string cascade_number;
CascadeClassifier mites_cascade;
ofstream file;
string window_name = "Capture - Mites detected = ";

RNG rng(12345);

/**
 * @function main
 */
```

```
int main( int argc, char* argv[] )
{
  /* With user input */
  if( argc == 1 )
   {
      cout << "Usage: " << argv[0] << endl;
      cout << "  -casc <cascade_dir_name>" << endl;
  cout << "  -img <image_name>" << endl;
  return 0;
   }
  for( int i = 1; i < argc; i++ )
    {
        if( !strcmp( argv[i], "-casc" ) )
        {
            mites_cascade_name = argv[++i];
        }
if( !strcmp( argv[i], "-img" ) )
        {
image_name = argv[++i];
        }
    }
  std::stringstream fileName;
  unsigned imStart = image_name.find("img");
  unsigned imStop = image_name.find(".jpg");
  unsigned image_count = imStop-imStart;
  unsigned caStart = mites_cascade_name.find("cascade");
  unsigned caStop = mites_cascade_name.find(".xml");
  unsigned cascade_count = caStop-caStart;
  image_number = image_name.substr(imStart,image_count);
  cascade_number = mites_cascade_name.substr(caStart,cascade_count);
  fileName << "detecties_" << image_number << "_"
           << cascade_number << ".txt";
  file.open(fileName.str(), std::fstream::app);
  Mat frame;

/* Without user input */
/*for(casnum=1;casnum<=8;casnum++)
{
for(imnum=1;imnum<=26;imnum++)
{
std::stringstream fileName,casname,imgname;
fileName << "detecties_img" << imnum << "_cascade_test"
         << casnum << ".txt";
file.open(fileName.str(), std::fstream::app);
casname << "C:\\test\\cascade_test" << casnum << ".xml";
imgname << "C:\\test\\img" << imnum << ".jpg";
```

```
mites_cascade_name = casname.str();
image_name = imgname.str();*/


  //-- 1. Load the cascade

  if( !mites_cascade.load( mites_cascade_name ) )
    { printf("--(!)Error loading\n"); return -1; };

  frame = imread(image_name,CV_LOAD_IMAGE_COLOR);

  //-- 3. Apply the classifier to the frame
  if( !frame.empty() )
   { detectAndDisplay( frame ); }
  else
   { printf(" --(!) No captured frame -- Break!"); }
  file.close();

// }
// }
  return 0;

}

/**
 * @function detectAndDisplay
 */
void detectAndDisplay( Mat frame )
{
   std::vector<Rect> mites;
   Mat frame_gray;

   cvtColor( frame, frame_gray, COLOR_BGR2GRAY );
   equalizeHist( frame_gray, frame_gray );

   //-- Detect Mites
   for(int overlap = 0; overlap < 1; overlap++)
   {
mites_cascade.detectMultiScale( frame_gray, mites, 1.2, 30, 0,
 Size(50,50), Size(200,200));
file << "img" << imnum << " " << "cascade_test"
 << casnum << " " << mites.size() ;

for( size_t i = 0; i < mites.size(); i++ )
{
file << " " << mites[i].x << " " << mites[i].y << " "
```

```
<< mites[i].width << " " << mites[i].height;

//-- Draw the mites
 Point center( mites[i].x + mites[i].width/2,
 mites[i].y + mites[i].height/2 );
 ellipse( frame, center, Size( mites[i].width/2, mites[i].height/2),
  0, 0, 360, Scalar( 255, 0, 0 ), 2, 8, 0 );
}

file << endl;
//-- Show what you got
string full_name = window_name+to_string(mites.size());
imshow( full_name, frame ); waitKey(0);
}
}
```

Bijlage D

Precission-recall curves

```
clear all
close all;
clc;

cascade_test = 2;
img = 1;

% Open file cotaining mite coordinates, couple a file handler
mite = fopen('D:\mijten\Test\positives.txt');

% Get a first line from file
readMite = fgets(mite);

% Set the total counts to zero
tpfull = zeros(1,120);
fpfull = zeros(1,120);
fnfull = zeros(1,120);


while ischar(readMite)
    % Split char array based on tab
    elementMite = regexp(readMite, ' ', 'split');

    % Create a vector to store the square containing the mites that
    % should be detected
    rectMite = [];

    % Count the amount of detections on the line
    amountMite = uint8(str2double(elementMite{1,2}));

    for j=1:amountMite
```

```
% Put [Xmin Ymin Width Height] in rectMite
 rectMite = [(uint16(str2double(elementMite{1,3+((j-1)*4)})))
             (uint16(str2double(elementMite{1,4+((j-1)*4)})))
             (uint16(str2double(elementMite{1,5+((j-1)*4)})))
             (uint16(str2double(elementMite{1,6+((j-1)*4)})))];

% Open file, couple a file handler
folder = 'D:\opencv\masterproef_exe\detecties\';
folderElement = sprintf('detecties_img%d_cascade_test
                              %d.txt',img, cascade_test);
fullFile = fullfile(folder, folderElement);
fid = fopen(fullFile);

% Get a first line from file
readLine = fgets(fid);

% Process line, continue until there is no more line to fetch
filenameReference = '';

% Create a vector to store the squares containing the mites
rectangle = [];

% Set count
count = 1;

while ischar(readLine)
    % Split char array based on tab
    elements = regexp(readLine, ' ', 'split');
    imageName = elements{1,1};
    testNumber = elements{1,2};

    % Count the amount of detections on the line
    amountDetections = uint32(str2double(elements{1,3}));

    %Set tp, fp, fn
    tp = 0;
    fp = 0;
    fn = 0;

    % if amountDetections == 0, we quite
    if amountDetections ~= 0
        for i=1:amountDetections
        % Put [Xmin Ymin Width Height] in rectangle
        rectangle = [(uint32(str2double(elements{1,4+((i-1)*4)})))
                    (uint32(str2double(elements{1,5+((i-1)*4)})))
                    (uint32(str2double(elements{1,6+((i-1)*4)})))
```

```
                            (uint32(str2double(elements{1,7+((i-1)*4)})))];
                % if centerpoint of rectMite is in rectangle --> detection
                if (rectangle(1)<(rectMite(1)+(rectMite(3)/2)) &&
           (rectangle(1)+rectangle(3))>(rectMite(1)+(rectMite(3)/2)) &&
            rectangle(2)<(rectMite(2)+(rectMite(4)/2)) &&
           (rectangle(2)+rectangle(4))>(rectMite(2)+(rectMite(4)/2)))
                        tp = tp+1;
                    else
                        fn = fn+1;
                    end
                end
                if(tp < amountMite && amountDetections ~= 0)
                    fp = fp + amountMite-tp;
                end
                tpfull(count) = tpfull(count) + tp;
                fpfull(count) = fpfull(count) + fp;
                fnfull(count) = fnfull(count) + fn;

                % Calculate precision & recall
                precision(count) =
                    tpfull(count)/(tpfull(count)+fpfull(count));
                recall(count) =
                    tpfull(count)/(tpfull(count)+fnfull(count));

                % Add count
                count = count + 1;

                % Get the next line that needs to be processed
                readLine = fgets(fid);
 else
      % Go to next mite/image
                break
end
        end
    end
% Close the handler to the file
fclose(fid);
img = img +1;
readMite = fgets(mite);
end
fclose(mite);
plot(recall,precision,'-ro','LineWidth',2);
title('LBP precision-recall curves');
grid on
set(gcf, 'color', [1 1 1]);
axis([0 1 0 1])
```

```
xlabel('recall');
ylabel('precision');
```

Bibliografie

[1] [Online]. Available: http://www.biocolor-tec.es/en/products/naturalenemies/amblyseiuscalifornicus-calicolor/

[2] [Online]. Available: http://www.codeproject.com/Articles/441226/Haar-feature-Object-Detection-in-Csharp

[3] P. Viola and M. Jones, "Rapid object detection using boosted cascade of simple features," 2001.

[4] A. H. Timo Ahonen and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," 2006.

[5] [Online]. Available: http://www.sciencedirect.com/science/article/pii/S00313203080046031

[6] [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165168410003476

[7] [Online]. Available: www.eavise.be

[8] [Online]. Available: www.biobest.be

[9] [Online]. Available: http://www.biotech-system.com.ua/en/production/entomophages-and-acariphages/amblyseius-californicus/

[10] M. O. C. Papageorgiou and T. Poggio, "A general framework for object detection," 1998.

[11] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," 1996.

[12] [Online]. Available: http://www.itl.nist.gov/iad/humanid/feret/feret_master.html

[13] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[14] R. B. Girshick, P. F. Felzenszwalb, and D. McAllester, "Object detection with grammar models," *Neural Information Processing Systems (NIPS)*, 2011.

[15] Y. Garini, I. T. Young, and G. McNamara, "Spectral imaging: Principles and applications," *International Society for Analytical Cytology*, 2006.

[16] [Online]. Available: http://docs.opencv.org/modules/core/doc/intro.html

[17] B. E. M. Stanley B. Lippman, Josée Lajoie, *C++ Primer*, 5th ed. Addison-Wesley, 2012.

[18] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *International Conference on Computer Vision & Pattern Recognition*, C. Schmid, S. Soatto, and C. Tomasi, Eds., vol. 2, INRIA Rhône-Alpes, ZIRST-655, av. de l'Europe, Montbonnot-38334, June 2005, pp. 886–893. [Online]. Available: http://lear.inrialpes.fr/pubs/2005/DT05

[19] A. Hadid, "The local binary pattern approach and its applications to face analysis," 2008.