

Subfaculteit wetenschappen



Probleemoplossen en ontwerpen 3

# Biologische Data Analyse App

**Marthe Böting  
Robin Bruneel  
Toon Ingelaere**

Titularis : Koen Van Den Abeele

Begeleider : Senna Staessens

Academiejaar 2018 - 2019



## BDA App

December 15, 2018

Marthe Böting, Robin Bruneel en Toon Ingelaere

## Inleiding

### Inhoudsopgave

<b>1 Klantenvereisten</b>	<b>2</b>
<b>2 Ontwerpspecificatie</b>	<b>2</b>
<b>3 Onze oplossing</b>	<b>2</b>
3.1 Achtergrondverwijdering . . . . .	2
3.1.1 Bepalen van de beste threshold . . . . .	3
3.1.2 Ruisfilter . . . . .	5
3.1.3 Uitzonderingen . . . . .	6
3.2 Lokalisatie van de indicator . . . . .	7
3.2.1 Het HSV kleurmodel . . . . .	9
3.2.2 Een algemene threshold . . . . .	10
3.2.3 Optimalisatie van de threshold . . . . .	11
3.2.4 Problemen . . . . .	12

## 1 Klantenvereisten

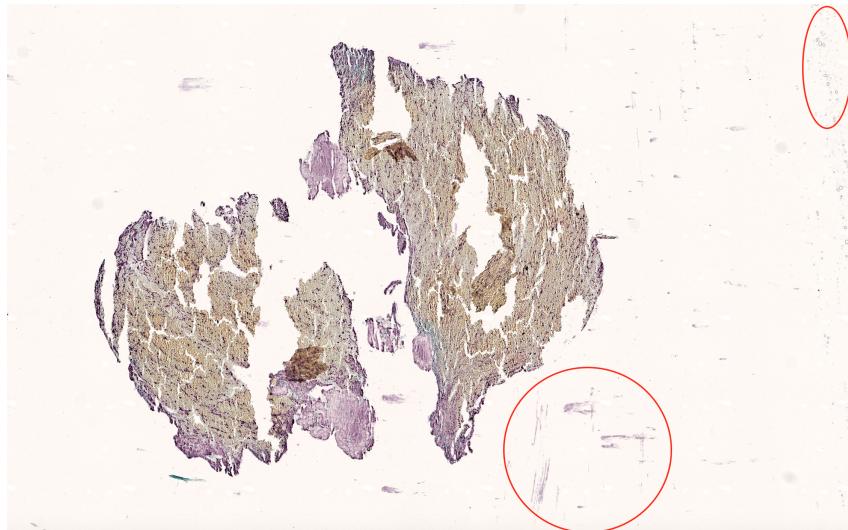
## 2 Ontwerpspecificatie

## 3 Onze oplossing

In dit hoofdstuk trachten we een oplossing te vinden voor ons probleem. We willen namelijk het percentage indicatorpixels in een willekeurige afbeelding kunnen berekenen. Het hoofdstuk is opgedeeld in verschillende deelproblemen, namelijk het verwijderen van de achtergrond, het lokaliseren van de indicator en de gebruiksvriendelijke app. Alle resultaten zijn bekomen met behulp van de programmeertaal MATLAB.

### 3.1 Achtergrondverwijdering

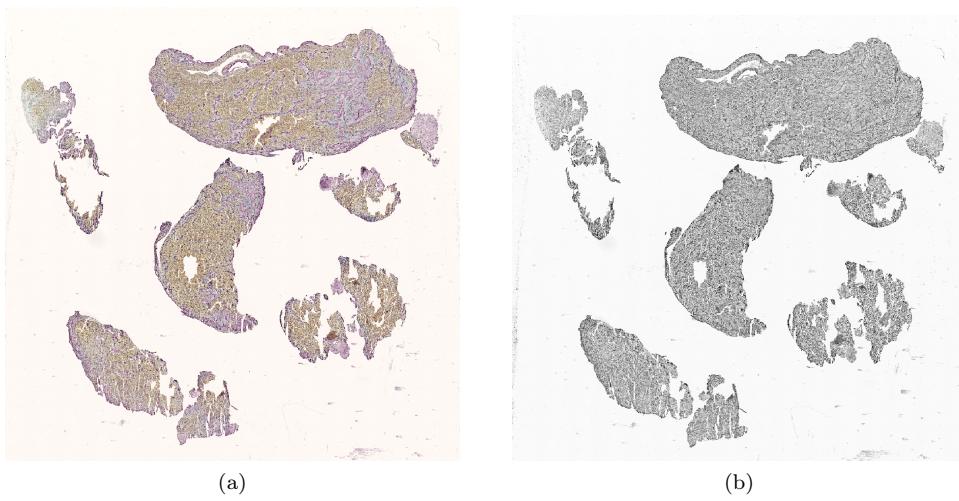
Het eerste deelprobleem is het verwijderen van de achtergrond. Op de afbeeldingen is er heel wat vuilheid te vinden. Voorbeelden hiervan zijn luchtbellen of kleine verkleuringen in de achtergrond zoals men ziet op de Figuur 1. Hieronder beschrijven we verschillende operaties om de grootste kloners te lokaliseren en alles wat geen klonter is uit de afbeelding te verwijderen. Dit leidt in feite tot een ruisvrije afbeelding.



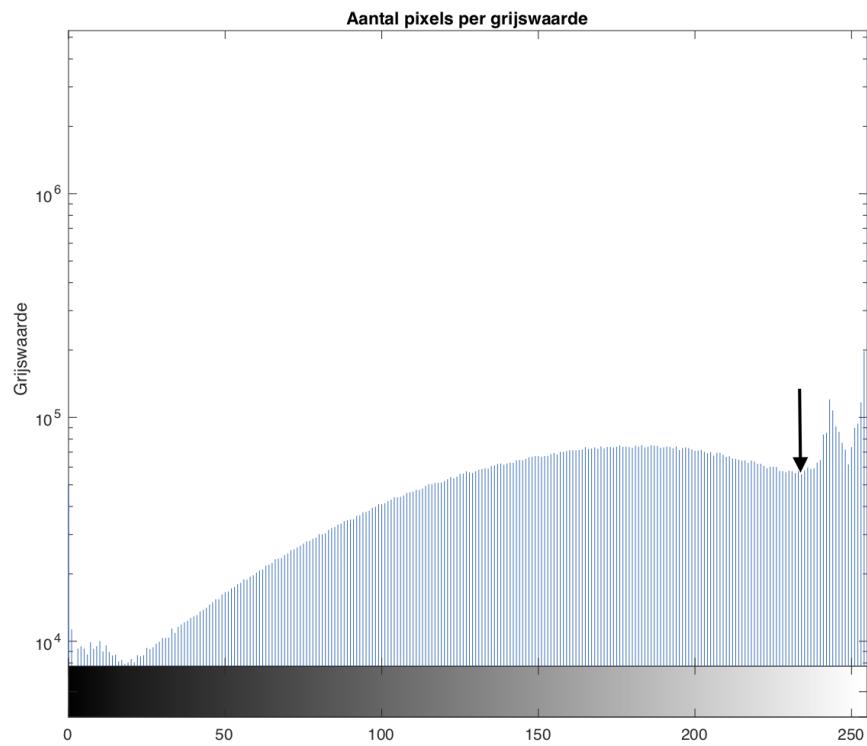
Figuur 1: In de rechterbovenhoek zien we luchtbellen en in de onderste cirkel zien we een verkleuring die geen deel uitmaakt van een bloedklonter.

### 3.1.1 Bepalen van de beste threshold

Het onderscheid tussen de achtergrond (eerder wit) en de bloedklonter (eerder grijs) is makkelijker te zien op de grijswaarden van de afbeelding, zie Figuur 2. Eén grijswaarde is nu voldoende om het onderscheid te maken. Een histogram van deze grijswaarden leert ons dat er een duidelijk dipje in de frequenties tussen het wit en het grijs te zien is, zie Figuur 3. Dit is dan ook de theoretisch optimale threshold, de grijswaarde om een bloedklonterpixel van een achtergrondpixel te onderscheiden. Deze bepalen we simpelweg als het minimum in de tweede helft van de histogram. Wanneer we deze threshold toepassen, bekomen we de binaire Figuur 4. Dit lijkt de klonters in de afbeelding nagenoeg goed te detecteren.



Figuur 2: Illustratie van de originele foto (a) en deze omgezet in grijswaarden (b).



Figuur 3: Histogram van het aantal pixels gegroepeerd per grijswaarde. We zien duidelijk een lokaal minimum rond de waarde 230. Opmerking: we maken hier gebruik van een logaritmische y-as.

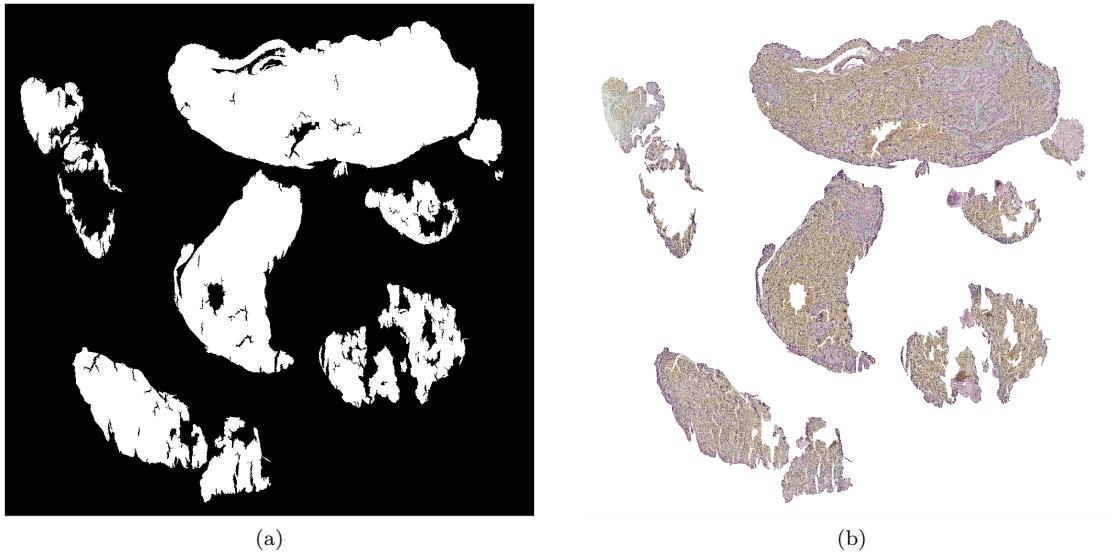


Figuur 4: Op deze binaire representatie zijn de bloedklonters duidelijk te zien.

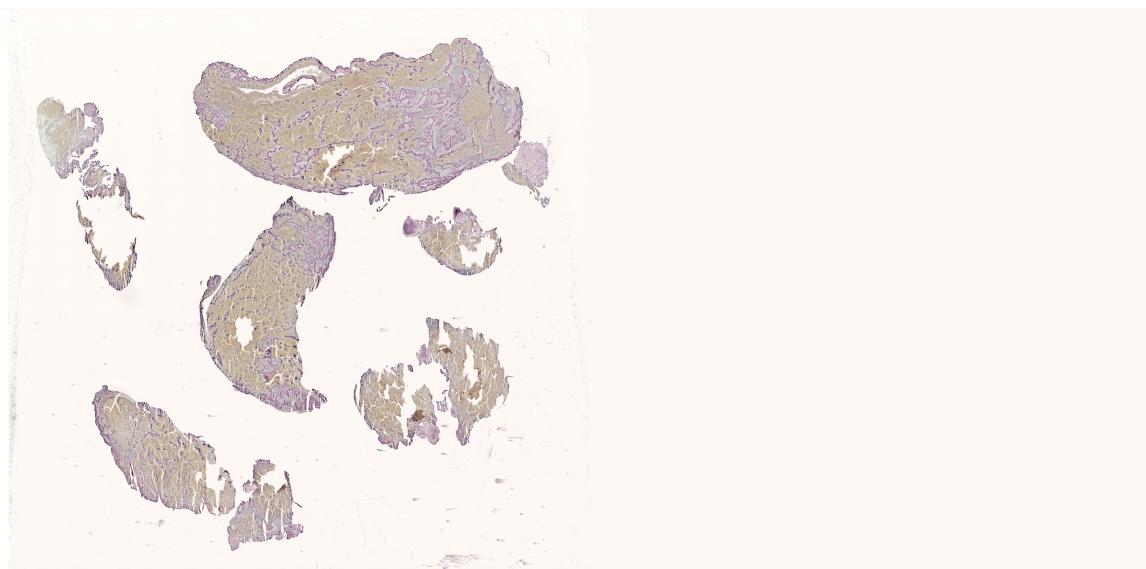
### 3.1.2 Ruisfilter

Deze binaire representatie bevat heel wat holtes en ruis. In dit onderdeel proberen we dit probleem op te lossen. Hiervoor hebben we een simpele methode bedacht door enerzijds alle kleine groepen witte pixels te vervangen door zwarte pixels, zodat alleenstaande pixels (voornamelijk ruis) verdwijnen. Daarnaast vervangen we ook alle kleine groepen zwarte pixels door witte pixels, hierdoor worden daarenboven alle holtes in de bloedklonters opgevuld. Het resultaat van deze operatie is te zien op Figuur 5 (a). We bekomen nu een zogenaamde 'mask' die toegepast kan worden op de originele afbeelding. Hierbij worden alle zwarte pixels op de mask vervangen door witte pixels op de originele afbeelding. Alle witte pixels op de mask behouden hun kleur op de originele afbeelding. Daarnaast wordt deze afbeelding nog bijgesneden om kostbare geheugenruimte<sup>1</sup> te sparen, de afbeelding is te zien in Figuur 5 (b). In dit deelprobleem werd echter met een reeds bijgesneden bloedklonter gewerkt om de afbeeldingen duidelijk te houden. Een volledig onbewerkte afbeelding is echter te zien in Figuur 6

<sup>1</sup>Eén afbeelding neemt al snel 50MB geheugen in beslag aangezien de resolutie kan oplopen tot 8000x8000 pixels.



Figuur 5: Alle ruis is verwijderd, dit is een foto die verder verwerkt kan worden.



Figuur 6: Dit is een volledig onbewerkte afbeelding die we in 5 (b) hebben bijgesneden en waarvan de achtergrond verwijderd is.

### 3.1.3 Uitzonderingen

Naast de gewone bloedklonters zijn er ook nog enkele uitzonderingen. Sommige klonters hebben namelijk geen rode bloedcellen waardoor ze zeer wit zijn. Dit zien we ook op Figuur 7. Het probleem is dat we hier geen optimale threshold kunnen berekenen zoals eerder uitgelegd. Er is namelijk een zeer kleine afstand tussen de kleur van de bloedklonter en de achtergrond. Vandaar dat we in de app een slider verwerkt hebben om desnoods zelf die threshold te kunnen aanduiden. Wanneer de optimalisaties daarna op deze threshold toegepast worden, bekomen we Figuur 8. Het is dus duidelijk te zien dat dit algoritme zeer krachtig is.



Figuur 7: Een afbeelding van een bloedklonten zonder rode bloedcellen.



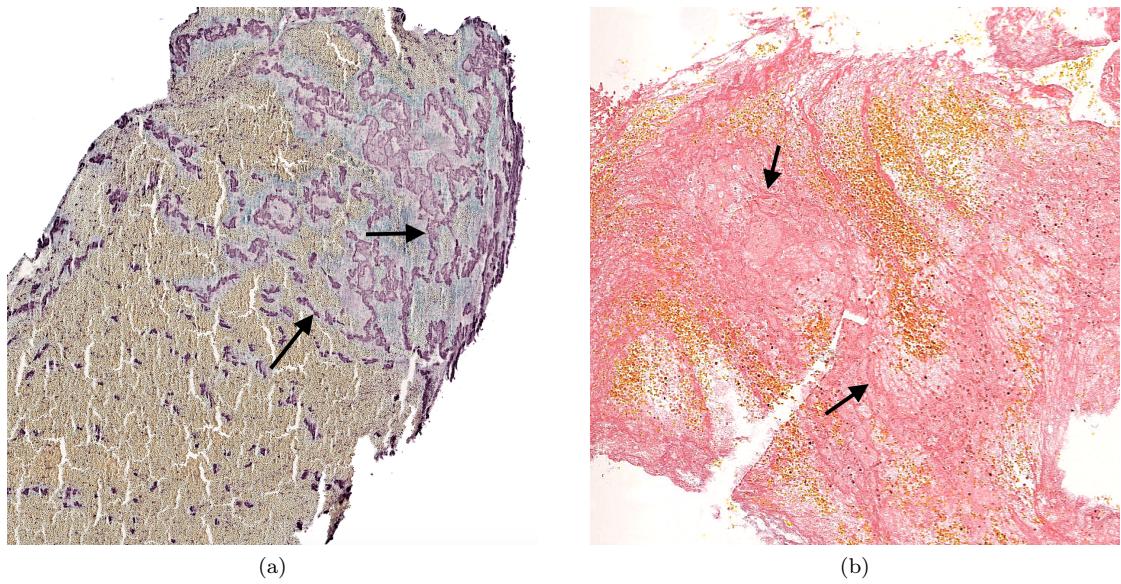
Figuur 8: Dit is de binaire mask die we bekomen na het toepassen van de threshold en de optimalisatie van deze. De klonters worden nu met zekere precisie aangeduid.

### 3.2 Lokalisatie van de indicator

Het tweede deel van ons project is de indicators lokaliseren en kwantificeren. Ons project moet in staat zijn twee specifieke soorten van indicatoren te onderscheiden. Van deze is een voorbeeld op Figuur 9 te zien. Omdat het kleurverschil tussen wat aangeduid worden (indicator) en wat niet (achtergrond) niet

altijd even groot is, vormen we het oorspronkelijke *RGB* kleurmodel<sup>2</sup> om naar het zogenaamde *HSV* kleurmodel. Dit model is een alternatieve voorstelling waarbij men alle kleuren op een cirkel voorstelt, de hoek die dit kleur dan maakt, noemt men de *Hue*. Naast deze waarde heeft *HSV* nog twee andere parameters namelijk *Saturation* en *Value*. *Saturation* kan simpel beschouwd worden als een aanduiding van de hoeveelheid witte kleur en *Value* een aanduiding van de zwarte kleur. Een grafische voorstelling is te zien op Figuur 10.

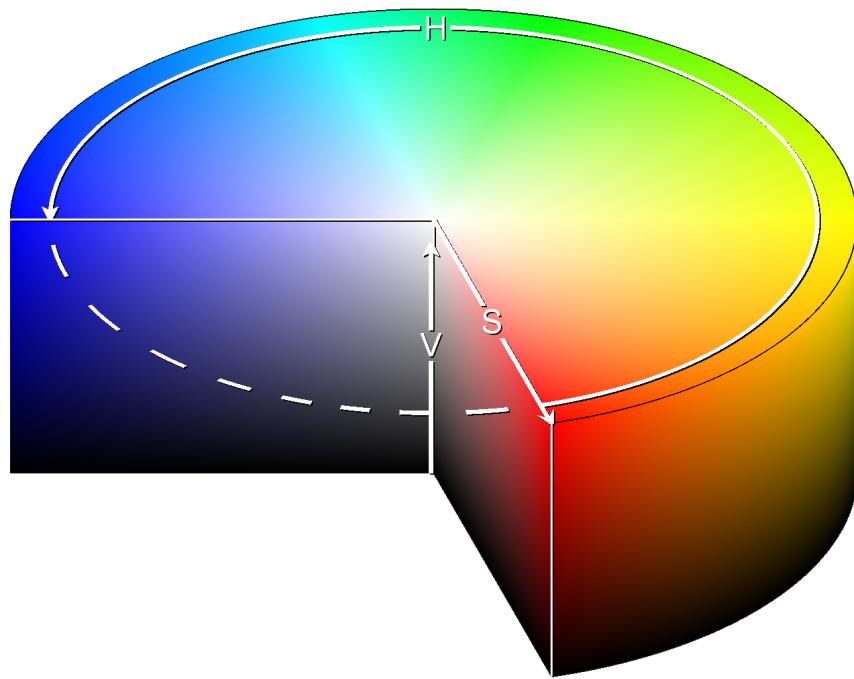
Het voordeel van deze transformatie is dat het heel wat eenvoudiger is om een onderscheid tussen dichtbijgelegen kleuren te vinden. Een andere mogelijke transformatie is die naar het *LAB* of het *CMYK* kleurmodel die gelijkaardige eigenschappen heeft en desnoods ook gebruikt kan worden. Eenmaal we een duidelijk onderscheid tussen de indicator en de achtergrond gemaakt hebben, berekenen we eenvoudig het percentage indicator als de verhouden van het aantal bloedklonten- en indicatorpixels. Ons stappenplan wordt in de volgende hoofdstukken besproken en wordt toegepast op de afbeelding uit Figuur 9 (a). Maar kan evenwel gebruikt worden op de andere kleuring, met andere beginwaarden.



Figuur 9: Illustratie van de twee soorten indicator (respectievelijk paars en donkerroze) die gedetecteerd moeten worden. We zien duidelijk dat het detecteren op afbeelding (a) eenvoudiger zal zijn dan op afbeelding (b).

---

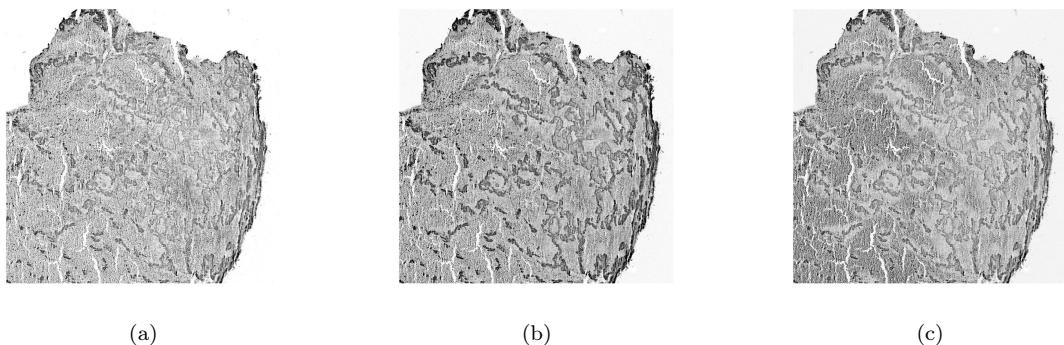
<sup>2</sup>Het *RGB* kleurmodel is een voorstelling waarbij ieder kleur voorgesteld wordt door een waarde van de drie basiskleuren (rood, groen en blauw)



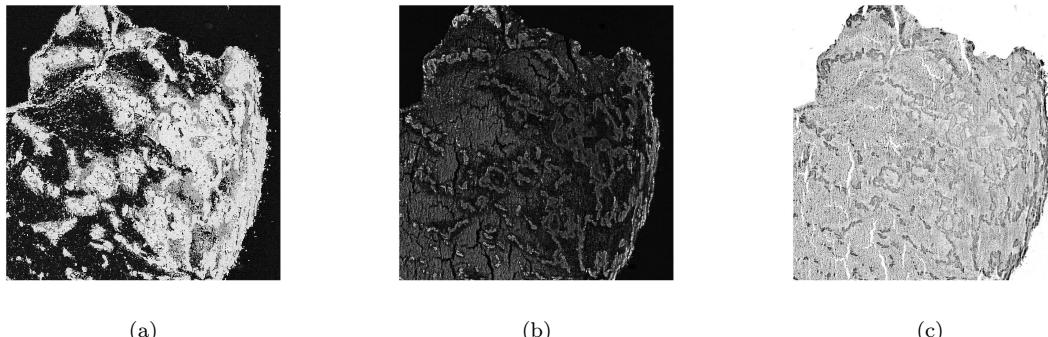
Figuur 10: Grafische voorstelling van het *HSV* (*Hue, Saturation, Value*) kleurmodel

### 3.2.1 Het HSV kleurmodel

Zoals reeds vermeld, beginnen we met een transformatie naar het *HSV* kleurmodel. Bijgevolg kunnen we de twee voorstellingen vergelijken. We hebben het model telkens ontbonden in de drie kleurwaarden, waarbij zwart de laagste waarde voor dat kleur voorstelt en wit de hoogste. De resultaten zijn te zien in de Figuren 11 en 12. Het verschil tussen de twee kleurmodellen is duidelijk te zien. Afbeelding (a) en (b) uit Figuur 12 lijken namelijk een iets agressiever onderscheid te maken.



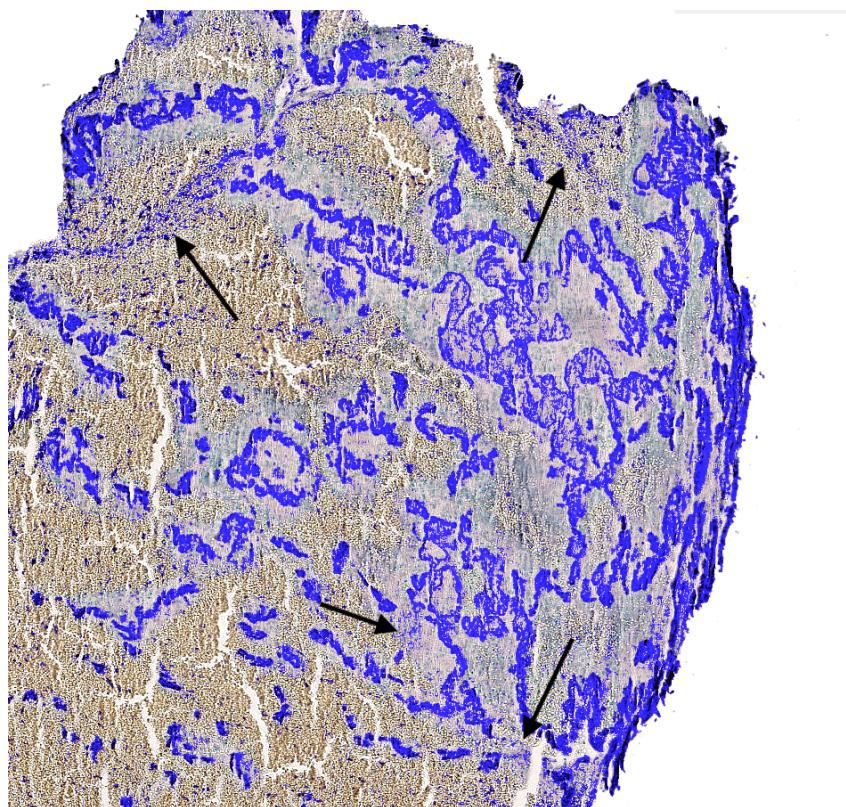
Figuur 11: Illustratie van respectievelijk de rode, groene en blauwe kleurwaarden (*RGB*). Hierbij komt overeen met de maximumwaarde en zwart met de minimumwaarde van die kleur.



Figuur 12: Illustratie van respectievelijk de *hue*, *saturation* en *value* kleurwaarden (*HSV*). Hierbij komt wit overeen met de maximumwaarde en zwart met de minimumwaarde van die kleurwaarde.

### 3.2.2 Een algemene threshold

De volgende stap is een filter bepalen voor de indicatorpixels. Het probleem is echter dat een goede filter voor de ene afbeelding niet altijd een goede filter voor de andere is. Daarom hebben we voor iedere afbeelding afzonderlijk manueel een 'threshold' bepaald en deze achteraf met elkaar vergeleken. Het resultaat is bijgevolg een vrij algemene threshold die alle indicatorpixels met zekerheid aanduidt, zoals te zien is op Figuur 13. Het enigste probleem is dat bepaalde pixels verkeerd aangeduid worden. Het aantal is weliswaar niet zo groot, maar aangezien ze in iedere afbeelding voorkomen zullen we dit trachten te omzeilen.



Figuur 13: We hebben alle indicatoren met een blauwe kleur aangeduid. De algemene threshold selecteert alle indicatorpixels, maar jammer genoeg ook enkele verkeerde, deze zijn aangeduid met een pijl.

### 3.2.3 Optimalisatie van de threshold

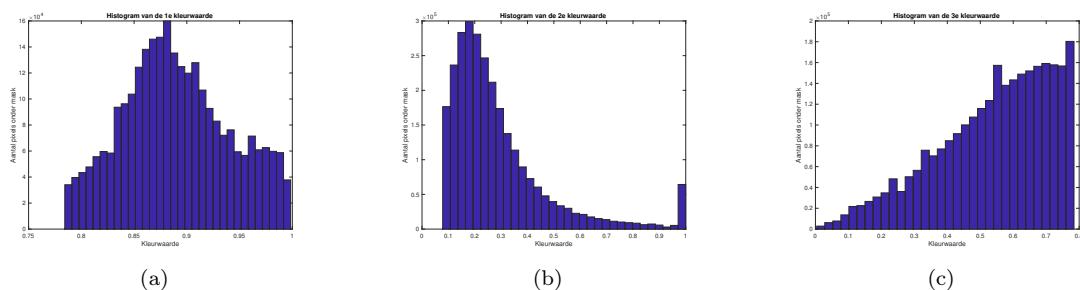
In dit hoofdstuk willen we het aantal verkeerde pixels verminderen zonder de juiste te beïnvloeden. We hebben echter al een vrij correcte threshold waardoor we een statistische analyse van wat onder de bijhorende mask ligt, kunnen uitvoeren. Hiervoor stellen we een histogram van de *HSV* kleurwaarden onder de mask op. Het levert ons een interessant resultaat dat te zien is in Figuur 14. We zien namelijk duidelijke verschillen in de frequenties van bepaalde kleurwaarden.

Het idee is om nu de verkeerde pixels via deze diagrammen eruit te filteren. Hiervoor doen we de aanname dat de verkeerde pixels essentieel in kleur verschillen van indicatorpixels. We weten ook uit Figuur 13 dat verkeerde pixels minder vaak voorkomen dan de juiste. De huidige threshold zou dus in feite manueel bijgestuurd kunnen worden, maar dit heeft geen zin. De grafieken van verschillende afbeelding hebben namelijk eenzelfde vorm, maar hun pieken liggen soms meer dan 5% verschoven. Daarom passen we op iedere afbeelding afzonderlijk automatisch een filter toe. In principe kunnen alle pixels met een frequentie onder een bepaalde grenswaarde geschrapt worden. Maar een betere benadering is misschien om het punt te vinden, waar de frequentie van de pixels enorm begint toe te nemen. Indien we deze thresholdwaarde naar de pieken toeschuiven, zijn we eigenlijk grote 'indicatoraders' aan het verwijderen. Dit zien we op Figuur 15. Dit punt kan theoretisch benaderd worden als het maximum van de tweede afgeleide naar de kleurwaarde.

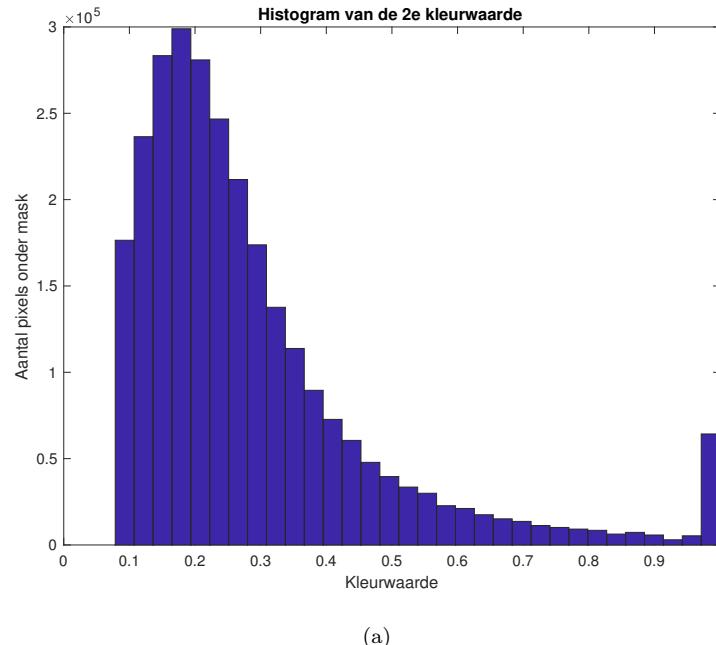
Wanneer we deze filter toepassen, bekomen we het resultaat dat te zien is op Figuur 16 (a). De zones die eerder verkeerd werden aangeduid zijn nu bijna volledig verdwenen.

Een extra stap die we tenslotte nog toepassen is het morfologisch sluiten van de pixels. Dit doen we door indicatorpixels die niet ver van elkaar gelegen zijn met elkaar te verbinden. Het is namelijk zo dat wanneer een pixel omringt is door indicatorpixels, de kans hoog is dat deze pixel ook een indicatorpixel is. Dit is te zien op 16 (b).

De laatste stap die nu nog rest is het optellen van alle aangeduide pixels en alle bloedklonterpixels. Wanneer we dan de verhouding nemen, bekomen we het percentage indicator, wat net gevraagd werd.

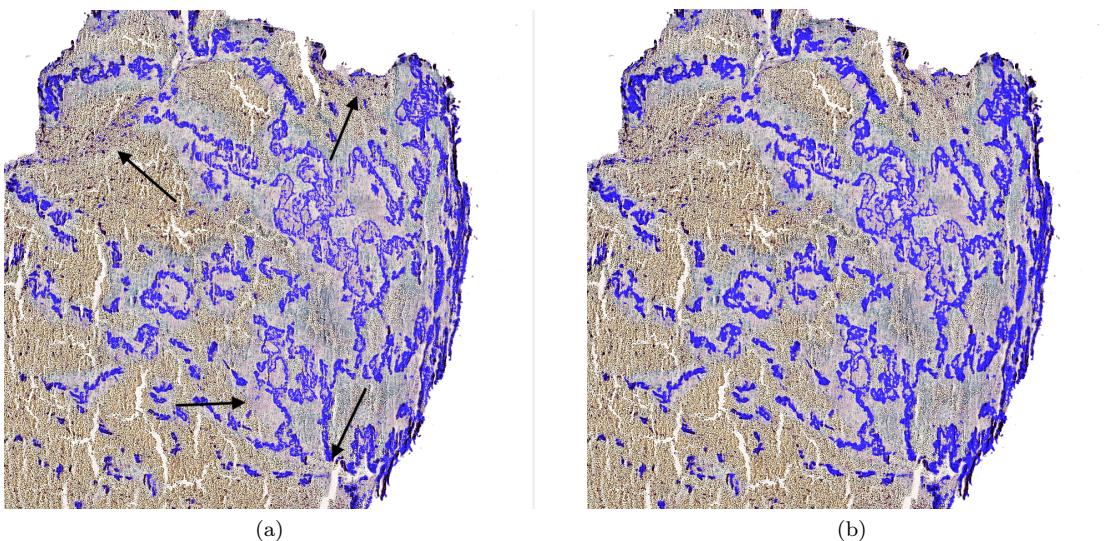


Figuur 14: Histogrammen van de *HSV* kleurwaarden. We zien duidelijk het verschil in frequenties.



(a)

Figuur 15: Op de figuur is hebben we een punt op de curve aangeduid. Wanneer we dit punt naar links verschuiven, verwijderen we telkens grotere stukken indicator van onze afbeelding. Dit willen we net vermijden.



Figuur 16: Illustratie waarbij we in (a) een algoritmische filter hebben toegepast om mogelijke ruis te verwijderen. In afbeelding (b) hebben we deze mask nog eens morfologisch gesloten, waardoor we een duidelijke aanduiding van de indicatorpixels bekomen.

### 3.2.4 Problemen

We kunnen nu wel met redelijke precisie de indicatoren in de afbeelding aanduiden, maar dit is niet even simpel voor iedere afbeelding. Het probleem met deze kleuringen is namelijk dat er heel wat variatie in de afbeeldingen zit. De effectieve kleuring in het labo is afhankelijk van verschillende factoren. Hierdoor kan deze kleuring van dag tot dag verschillen. Daarenboven zijn wij als student ingenieurswetenschappen



ook niet getraind om rode bloedcellen te detecteren. Bijgevolg hebben we in onze app ervoor gezorgd dat onze filter met behulp van verschillende sliders altijd bijgestuurd kan worden door de gebruiker en dus toch het gewenste resultaat bekomen kan worden.

	Week 1							Week 2							Week 3							Week 4							Week 5							Week 6						
	Sep							Oct							Nov																											
	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	S	S	M	T	W	F	S	S	M	T	W	T	F	S	S	S									
24	25	26	27	28	29	30	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	01	02	03	04	

1            2            3

*T.T. verslag*

Week 1		Week 2		Week 3		Week 4		Week 5		Week 6		Week 7	
		Nov						Dec					
05	06	07	08	09	10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27	28	29	30	01	02
M	T	W	T	F	S	S	M	T	W	F	S	S	M
T	W	T	F	S	S	M	T	W	F	S	S	M	T
W	T	F	S	S	M	T	W	F	S	S	M	T	W
F	S	S	M	T	W	F	S	S	M	T	W	F	S
S													S

3

4

*Endvertrag*

*Endpres.*

Taaknummer	Taakomschrijving
<b>1</b>	Achtergrond verwijderen en afbeelding bijsnijden
<b>2</b>	Kleuren detecteren en kwantificeren
<b>3</b>	Ontwerpen van een gebruiksvriendelijke app
<b>4</b>	Optimalisatie van het algoritme

Table 1: Omschrijvingen van de taken in de gantt-chart