

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

October 28, 2018

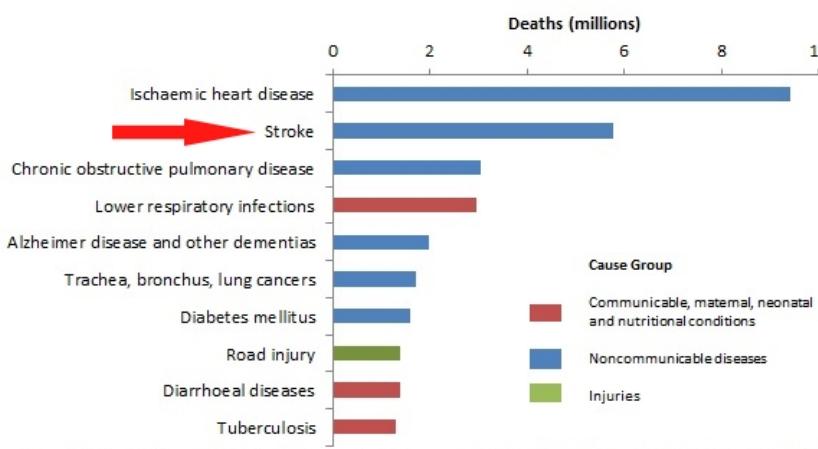
---

## Inleiding

Beroertes zijn in onze westerse samenleving na hartinfarcten en kanker de derde grootste doodsoorzaak. Op wereldvlak kapen ze zelfs de tweede plaats weg. Dit is te zien in het histogram op Figuur 1. In België komen er jaarlijks 25000 beroertes voor. In 15% van die gevallen overlijdt de patiënt. Een ischemische beroerte ontstaat nadat een bloedklontje in een hersenbloedvat blijft steken. Gewoonlijk dient men een middel toe die deze klontje oplost, maar indien dit niet gebeurt verwijdert men deze manueel. In dat laatste geval kan men de bloedklontje gebruiken voor verdere analyse in het labo.

Om het aantal slachtoffers aan beroertes te doen slinken, wordt er tegenwoordig veel onderzoek gedaan naar bloedklonters. Bij dit soort onderzoek probeert men de samenstelling van de bloedklonters te analyseren. Hiervoor worden afbeeldingen gemaakt van met indicator gekleurde bloedklonters en worden deze bijna volledig manueel geanalyseerd. Het manueel analyseren is erg tijdverdurend op grote schaal. Er is ons dan ook gevraagd om een app te ontwikkelen die de analyse van de afbeeldingen kan automatiseren. In dit verslag gaan we eerst in op wat de klant specifiek van ons verwacht en aan welke specificaties ons ontwerp moet voldoen. Hierna bespreken we ons design en lichten we het toe. Verder bespreken we ook enkele van onze voorlopige resultaten. Ten slotte wordt er nog een blik geworpen naar de vakken uit eerste drie semesters die ons hierbij geholpen hebben.

### Top 10 global causes of deaths, 2016



Figuur 1: Statistieken van de *World Health Organization* van 2016 waarin te zien is dat wereldwijd beroertes (*strokes*) de tweede meest frequente doodsoorzaak is.



## Inhoudsopgave

<b>1 Klantenvereisten</b>	<b>4</b>
<b>2 Ontwerpspecificatie</b>	<b>4</b>
<b>3 Onze oplossing</b>	<b>4</b>
3.1 Achtergrondverwijdering . . . . .	4
3.1.1 Bepalen van de beste threshold . . . . .	5
3.1.2 Lokaliseren van de bloedklonters . . . . .	7
3.1.3 Ruisfilter . . . . .	8
3.2 Lokalisatie van de indicator . . . . .	9
3.2.1 Het HSV kleurmodel . . . . .	9
3.2.2 Een algemene threshold . . . . .	9
3.2.3 Optimalisatie van de threshold . . . . .	10
3.3 De gebruiksvriendelijke applicatie . . . . .	12
<b>4 Voorlopige resultaten</b>	<b>13</b>
<b>5 Verantwoordelijkheden en taakverdeling</b>	<b>15</b>
<b>6 Integratie van vakken</b>	<b>15</b>
<b>7 Besluit</b>	<b>15</b>



## 1 Klantenvereisten

De klant verwacht een gebruiksvriendelijke app waarbij hij een afbeelding kan invoegen zodat deze automatisch verwerkt wordt. Dit houdt in dat de foto bijgesneden en de achtergrond verwijderd moet worden. Daarnaast is het de bedoeling om de samenstelling van de bloedklonter te analyseren aan de hand van de aangebrachte indicator.

## 2 Ontwerpspecificatie

De klant wil dat een afbeelding van een bloedklonter automatisch bewerkt en daarna geanalyseerd wordt. Het bewerken van de afbeelding houdt twee dingen in. Eerst en vooral moet de afbeelding zodanig bijgesneden worden dat de volledige bloedklonter erop staat. Hierbij mogen we de randen echter niet te breed nemen, aangezien er dan nuttige geheugenruimte verspild wordt. Naast het bijnissen, moet ook de achtergrond verwijderd worden. Dit betekent dat alle pixels die niet tot de bloedklonter behoren wit gekleurd worden. Indien dit niet goed gebeurt, kunnen de resultaten van de kleurenanalyse namelijk vertekend zijn.

In de kleurenanalyse moet het percentage aan met indicator gekleurde pixels geteld worden. Voor dit project moeten we slechts twee soorten kleuringen analyseren. Een voorbeeld van deze is te zien in Figuur 8. In beide gevallen moet de app op een accurate manier onderscheid kunnen maken tussen de eiwitten die gedetecteerd moeten worden en de rest van de bloedklonter.

Dit alles moet samengegoten worden in een visuele en gebruiksvriendelijke app. Dit wil zeggen dat de app makkelijk te installeren en te gebruiken is. De gebruiker moet ook een overzicht van de verschillende afbeeldingen van de bloedklonter kunnen terugroepen. Dit overzicht bestaat uit de originele afbeelding, de afbeelding zonder achtergrond en de afbeelding waarbij de indicatorpixels zijn aangeduid. Zo kunnen mogelijke fouten snel gedetecteerd worden. Indien gewenst kan deze app nog heel wat extra functionaliteiten uitvoeren zoals het opslaan van deze afbeeldingen, het wegschrijven van alle data in een csv<sup>1</sup> bestand.

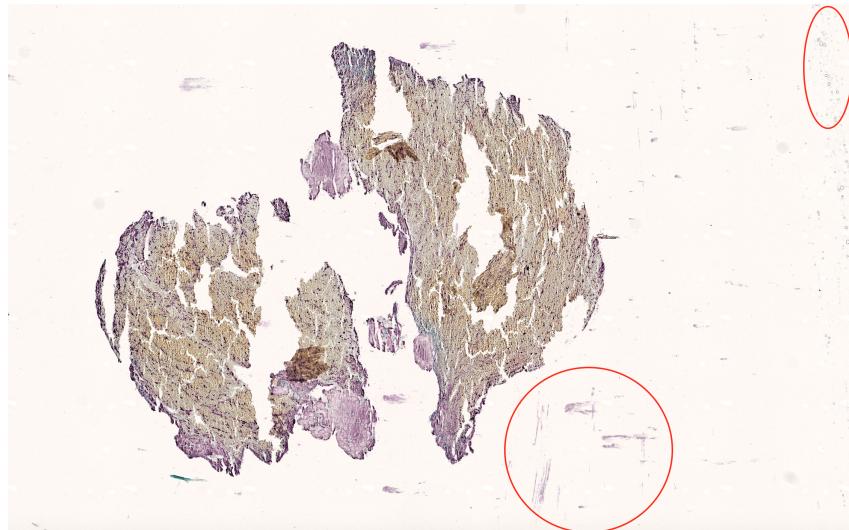
## 3 Onze oplossing

### 3.1 Achtergrondverwijdering

Het eerste deelprobleem van ons project is het verwijderen van de achtergrond van de bloedklonters. Op de afbeeldingen is er heel wat vuilheid te vinden. Voorbeelden hiervan zijn luchtbellen of kleine verkleuringen naast de bloedklonter zoals men ziet op de Figuur 2. Hieronder zullen we verschillende operaties beschrijven om de grootste kloners te lokaliseren en alles wat geen kloner is te verwijderen. Dit leidt tot een ruisvrije afbeelding.

---

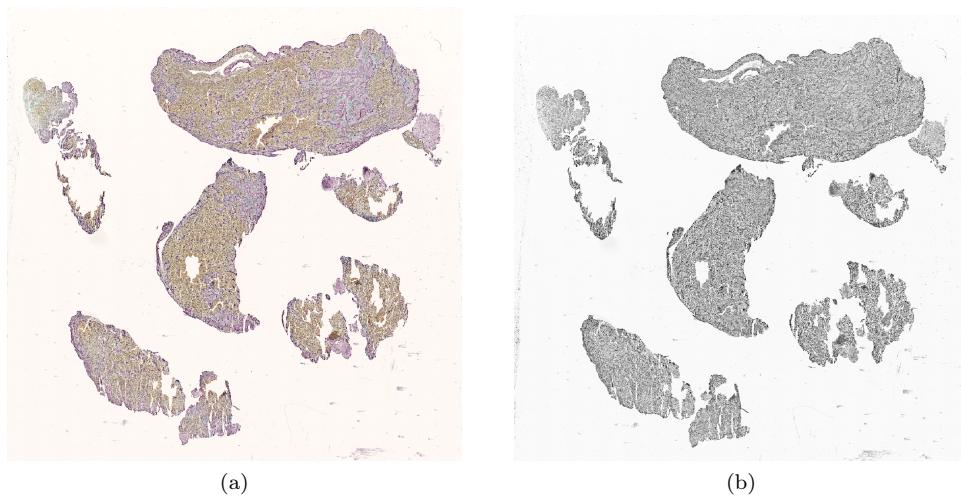
<sup>1</sup>Een csv (Comma Separated Values) bestand is een tekstbestand die als tabel ingelezen kan worden. Het kan eenvoudig in excel geopend worden voor verdere analyse.



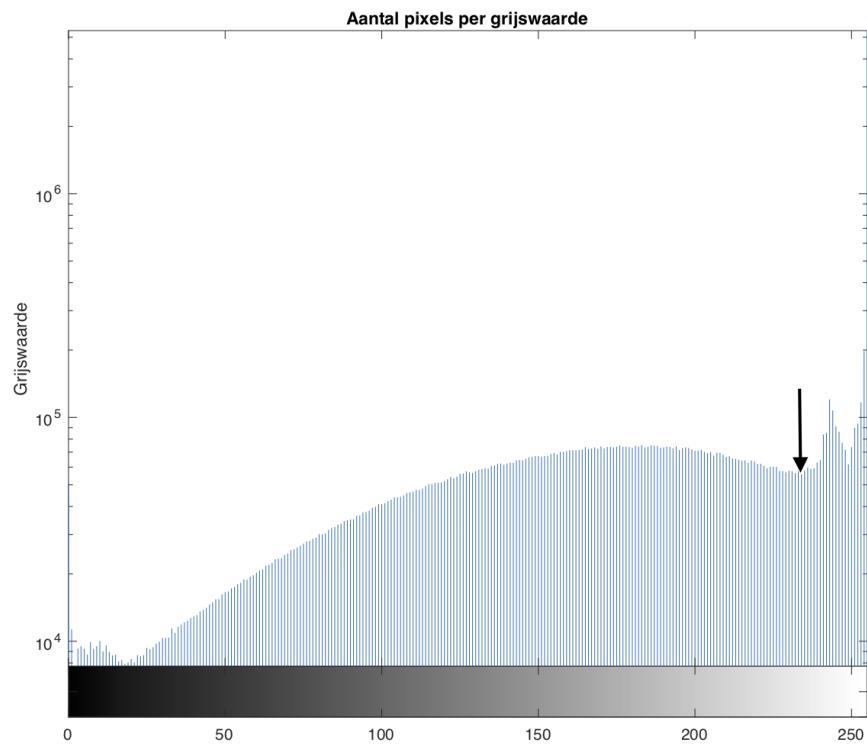
Figuur 2: In de rechterbovenhoek zien we luchtbellen en in de onderste cirkel zien we een verkleuring die geen deel uitmaakt van een bloedklonter.

### 3.1.1 Bepalen van de beste threshold

Wanneer we de grijswaarden van de afbeelding berekenen, zien we een duidelijk verschil tussen de achtergrond (eerder wit) en de bloedklonter (eerder grijs) zoals we zien op Figuur 3. Wanneer we een histogram van deze grijswaarden opstellen, vinden we ook dat er een lokaal minimum is tussen het grijs en het wit. Dit zien we duidelijk op het histogram in Figuur 4. Wanneer we dit minimum berekenen vinden we de theoretisch optimale threshold, de grijswaarde om een bloedklonterpixel van een achtergrondpixel te onderscheiden. Wanneer we die threshold toepassen en een binaire representatie zoals in Figuur 5 vormen, zien we duidelijk dat dit inderdaad een goede threshold is.



Figuur 3: Illustratie van de originele foto (a) en deze omgezet in grijswaarden (b).



Figuur 4: 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 5: Op deze binaire representatie zijn de bloedklonters duidelijk te zien.

### 3.1.2 Lokaliseren van de bloedklonters

Eenmaal deze binaire representatie gevonden is, lokaliseren we de grootste klonters om de ruis te verwijderen. Hiervoor zullen we eerst alle holtes in de klonters opvullen door alle ingesloten pixels<sup>2</sup> wit te maken. Het resultaat is te zien in Figuur 6. De klonters zijn nu nog veel duidelijker zichtbaar, waardoor we kunnen overgaan naar de effectieve ruisfilter.

---

<sup>2</sup>Een ingesloten pixel is een zwarte pixel die onmogelijk de achtergrond van de afbeelding kunnen bereiken zonder over witte pixels te gaan



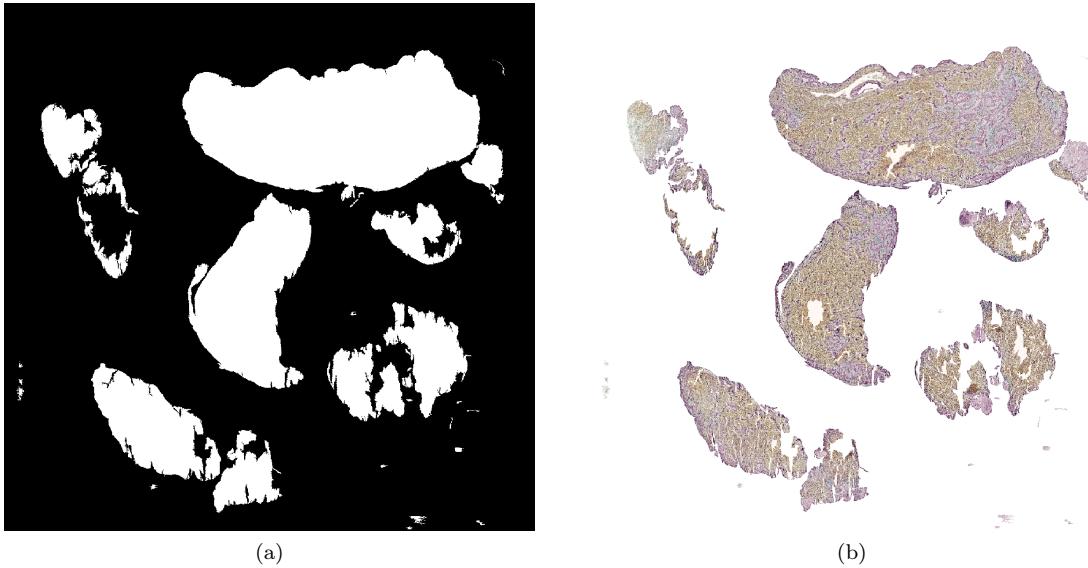
Figuur 6: Alle gaatjes zijn nu opgevuld. We onderscheiden nu enkele grote bloedklonters en nog ruis. Op basis van de grootte van de gebieden, kunnen we de ruis nu verwijderen.

### 3.1.3 Ruisfilter

Voor de ruisfilter verwijderen we elke groep witte pixels met een oppervlakte kleiner dan een constante waarde. Indien dit zo is, worden de witte pixels door zwarte vervangen. Dit simpel algoritme verwijdert alle ruis zoals te zien is in Figuur 7. Deze zogenaamde 'mask'<sup>3</sup> kunnen we toepassen op de originele afbeelding om definitief de ruis te verwijderen.

---

<sup>3</sup>Bij het toepassen van een mask op een afbeelding, worden alle pixels in de afbeelding die zwart zijn in de mask, wit gemaakt. De andere pixels behouden hun kleur.



Figuur 7: Alle ruis is verwijderd, dit is een foto waarmee we kunnen werken.

### 3.2 Lokalisatie van de indicator

Het tweede deel van ons project is de indicators lokaliseren en kwantificeren. We hebben de opdracht gekregen twee soorten te kunnen onderscheiden. Van elk van deze twee is een voorbeeld te zien op Figuur 8 te zien. Omdat het kleurverschil tussen indicator en achtergrond niet altijd even groot is, vormen we het oorspronkelijke *RGB* kleurmodel<sup>4</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 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 9.

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* kleurmodel die gelijkaardige eigenschappen heeft en desnoods ook gebruikt kan worden. Eenmaal we een duidelijk onderscheid tussen de indicator en de achtergrond maken, kunnen we eenvoudig het percentage indicator berekenen door het aantal bloedklonter- en indicatorpixels op te tellen. Ons stappenplan wordt in volgende hoofdstukken besproken en wordt toegepast op de afbeelding uit Figuur 8 (a).

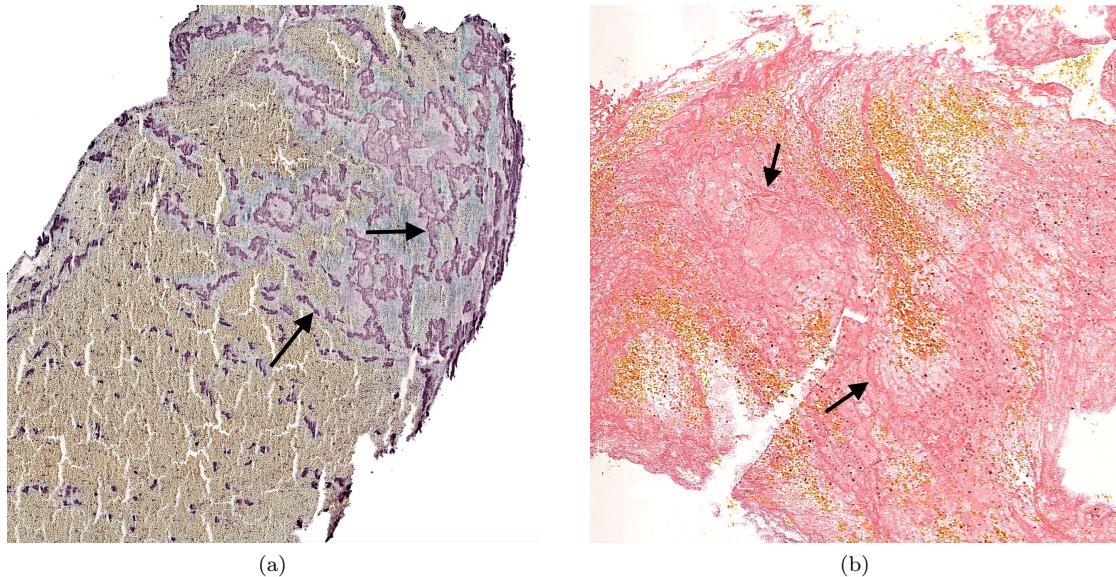
#### 3.2.1 Het HSV kleurmodel

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 is en wit de hoogste. De resultaten zijn te zien in de Figuren 10 en 11. Het verschil tussen de twee kleurmodellen is duidelijk te zien. Afbeelding (a) en (b) uit Figuur 11 lijken namelijk een iets agressiever onderscheid tussen de kleuren te maken.

#### 3.2.2 Een algemene threshold

De volgende stap is nu een filter bepalen voor alle indicatorpixels. Het probleem is echter dat een goede filter voor de ene foto niet altijd een goede filter voor de andere foto is. Daarom hebben we voor iedere foto manueel een 'threshold' bepaald en deze achteraf met elkaar vergeleken. Het resultaat is een vrij algemene threshold die alle indicatorpixels met zekerheid aanduidt. Af en toe worden jammer genoeg

<sup>4</sup>Het *RGB* kleurmodel is een voorstelling waarbij ieder kleur voorgesteld wordt door een waarde van de drie basiskleuren (rood, groen en blauw)



Figuur 8: 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).

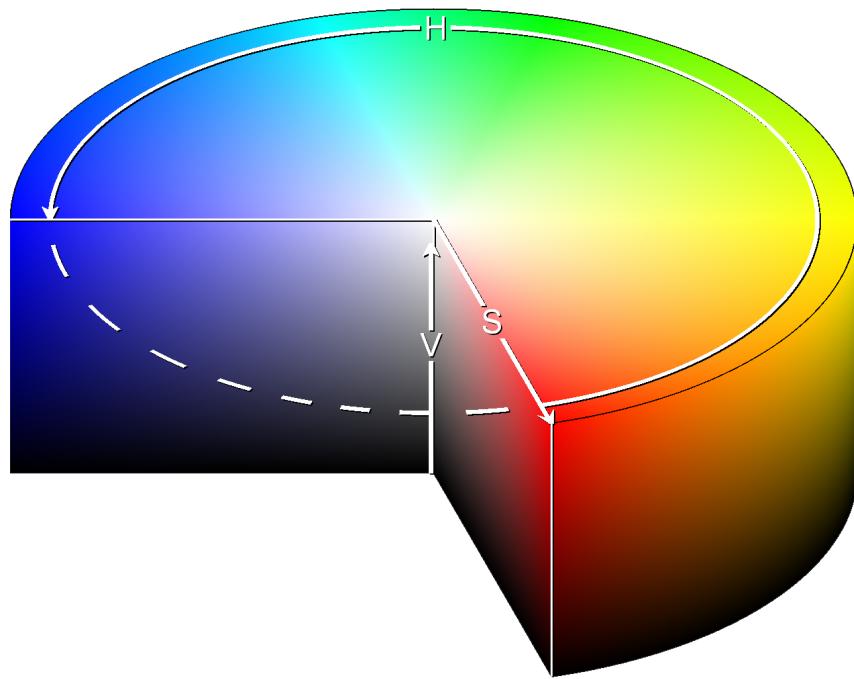
verkeerde pixels aangeduid. Het aantal is weliswaar niet zo groot, maar dit zullen we trachten te omzeilen in het volgende hoofdstuk. Een voorbeeld van deze algemene threshold is te zien op de Figuur 12.

### 3.2.3 Optimalisatie van de threshold

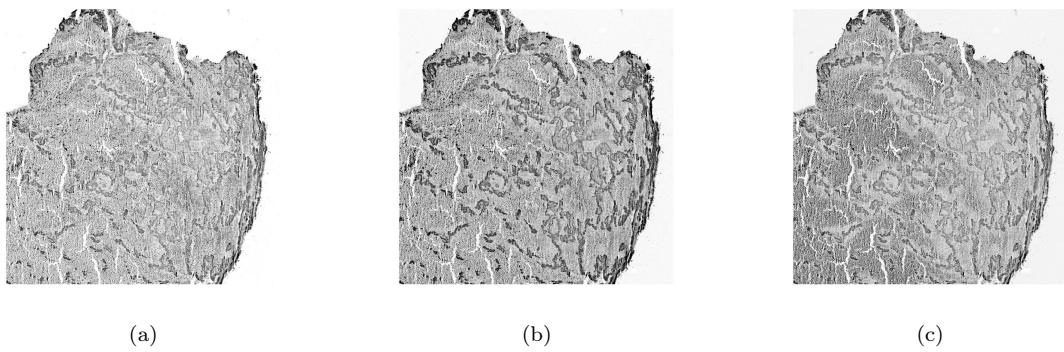
In dit hoofdstuk willen we het aantal verkeerde pixels verminderen zonder de juiste te beïnvloeden. We hebben echter al een vrij goede threshold waardoor we een statistische analyse van wat onder deze mask ligt, kunnen uitvoeren. We stellen hiervoor een frequentiediagram van de *HSV* kleurwaarden onder de mask op. Het levert ons namelijk een interessant resultaat dat te zien is in Figuur 13. We zien namelijk duidelijke verschillen in de frequenties van de kleurwaarden.

Het idee is om nu de verkeerde pixels via deze diagrammen eruit te filteren. Indien we veronderstellen dat de frequenties van deze specifieke pixels laag zijn en dat de verkeerde pixels in kleur verschillen met de indicatorpixels. Dan kunnen we in principe alle pixels met een frequentie onder een bepaalde grenswaarde schrappen. Een betere benadering is misschien om het punt te vinden, waar de frequentie van de pixels enorm begint toe te nemen of we met andere woorden grote 'indicatoraders' aan het verwijderen zijn. Dit punt kan theoretisch benaderd worden als het maximum van de tweede afgeleide naar de kleurwaarde. Jammer genoeg hebben we de tijd nog niet gehad om dit idee te testen.

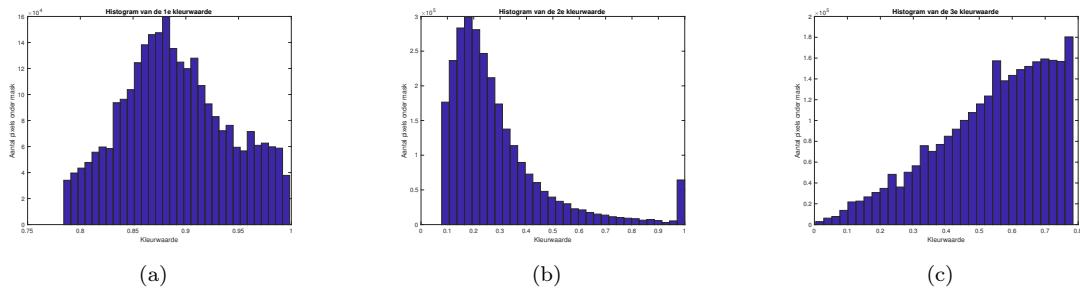
Een mogelijk extra stap is om de indicatorpixels morfologisch te sluiten met een algoritme dat reeds in MATLAB verwerkt is. Indien we met onze filtering enkele holtes maken, kunnen deze dan zonder problemen worden opgevuld. Ter illustratie geven we het standaard voorbeeld van MATLAB in Figuur 14 weer.



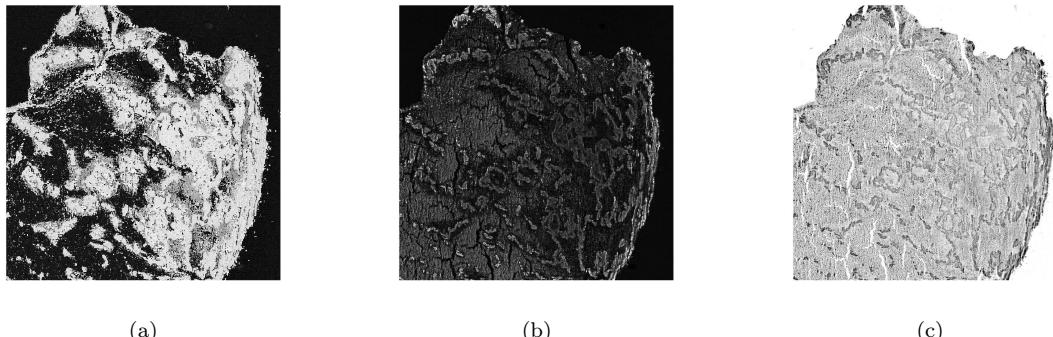
Figuur 9: Grafische voorstelling van het *HSV* (*Hue, Saturation, Value*) kleurmodel



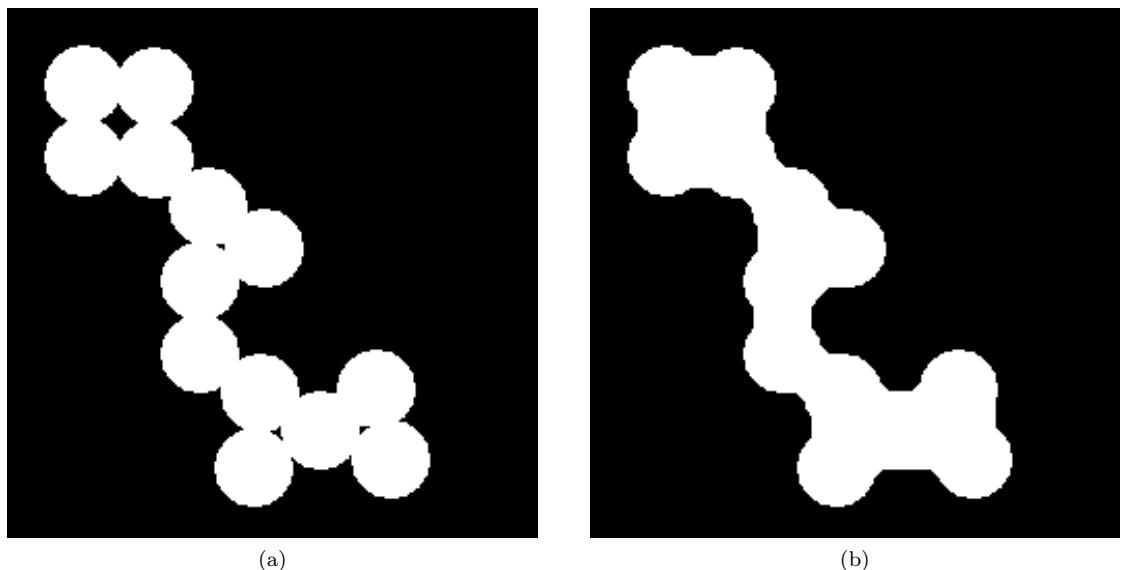
Figuur 10: Illustratie van respectievelijk de rode, groene en blauwe kleurwaarden (*RGB*). Hierbij komt overeen met de maximumwaarde en zwart met de minimumwaarde van die kleur.



Figuur 13: Histogrammen van de HSV kleurwaarden. We zien duidelijk het verschil in frequenties.



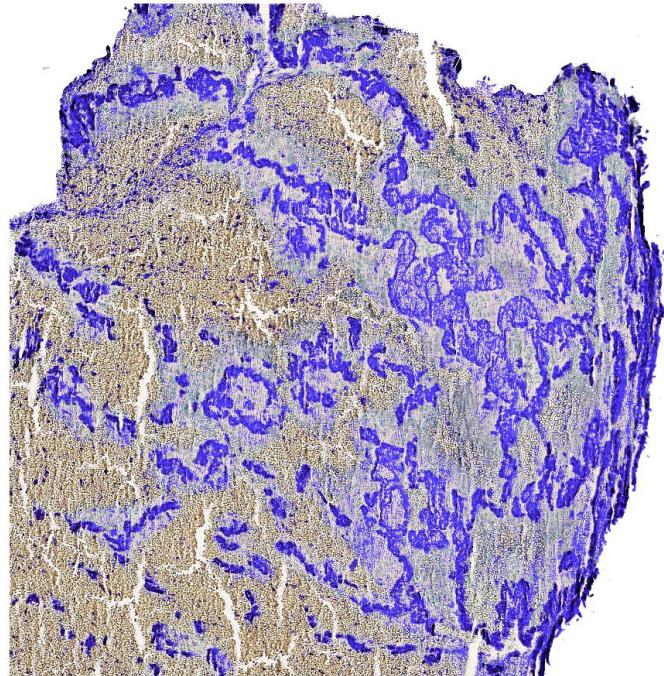
Figuur 11: Illustratie van respectievelijk de *hue*, *saturation* en *value* kleurwaarden (*HSV*). Hierbij komt wit overeen met de maximumwaarde en zwart met de minimumwaarde van die kleurwaarde.



Figuur 14: Illustratie waarbij de witte cirkels uit afbeelding (a) morfologisch gesloten worden in afbeelding (b)

### 3.3 De gebruiksvriendelijke applicatie

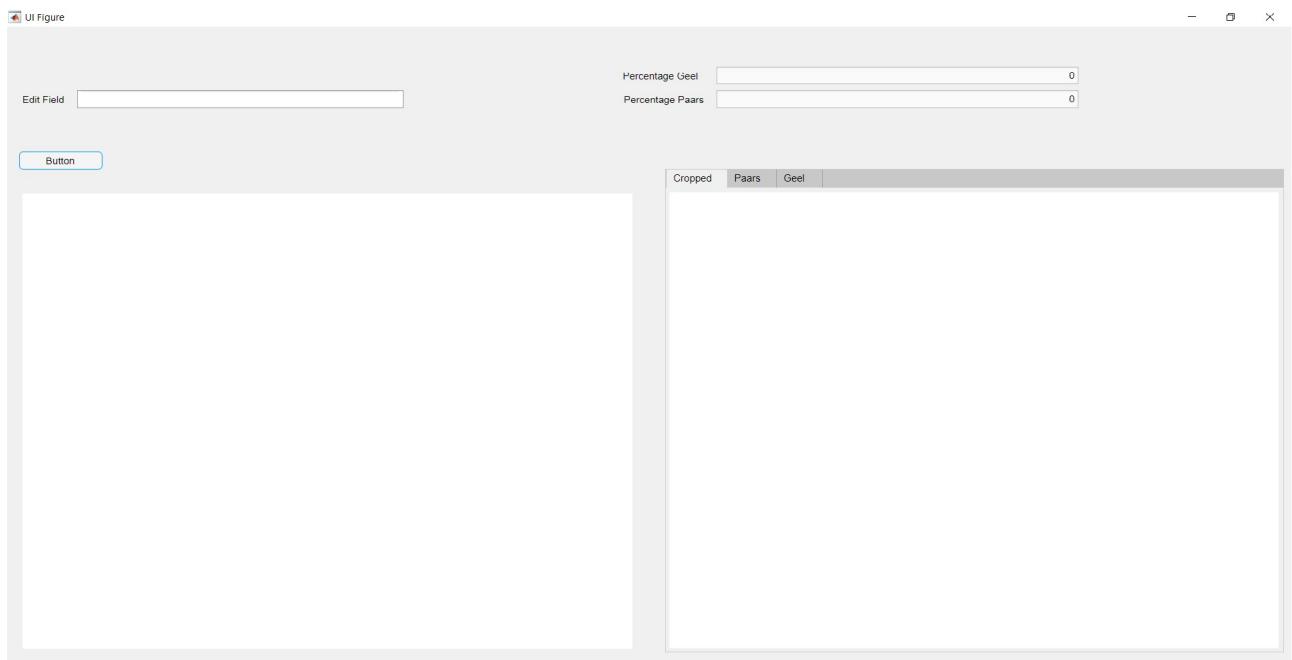
Momenteel hebben we nu nog enkel afzonderlijke programma's waarbij we steeds zelf de bestandslocatie van de aan te passen foto in de code moeten schrijven. Naar de klant toe is dit geen manier van werken en dienen deze programma's in een app samengebracht te worden. De klant zal dan zelf kunnen ingeven welke foto hij wil laten verwerken. Eénmaal de foto gekozen is, zal in de eerste fase de achtergrond van de foto wit worden gemaakt en zal de foto bijgesneden worden zodanig dat enkel de bloedklonter zichtbaar is. Als dit gebeurd is, zal er al een eerste resultaat getoond worden. Hierdoor heeft de klant een beeld waarop de verdere analyse zal gebeuren en kan die eventueel ingrijpen bij fouten. Vervolgens zal de kleurenanalyse gebeuren. Ook hiervoor zal de foto getoond worden waarop de klant kan zien welke delen de indicator bevatten en zal het percentage gegeven worden. De basis lay-out van de app hebben we al; zoals te zien is in Figuur 15.



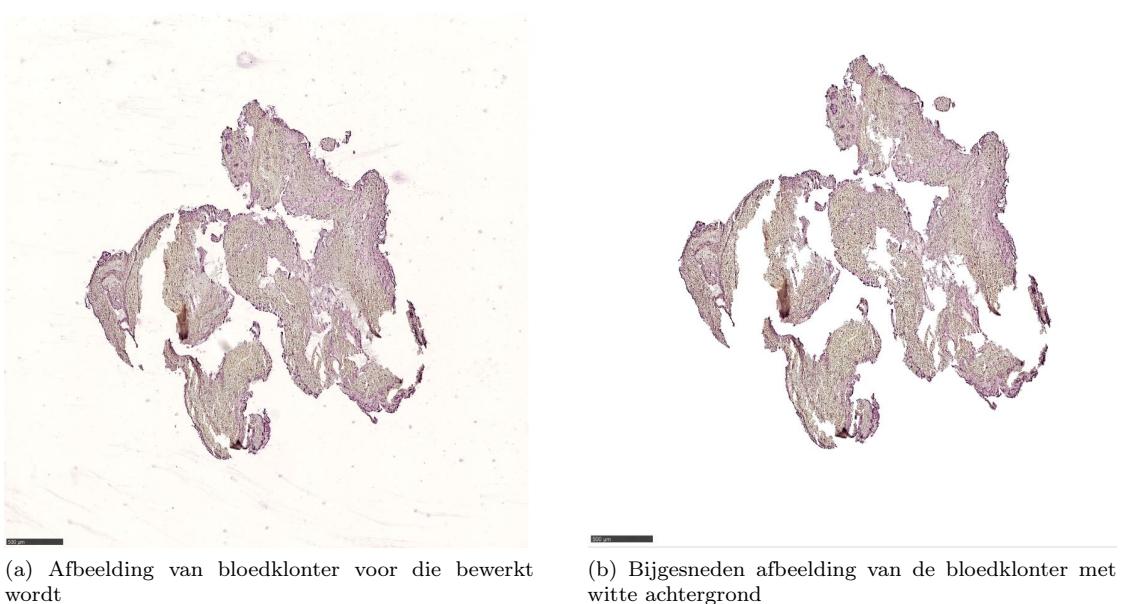
Figuur 12: We hebben alle indicatoren met een blauwe kleur aangeduid. De algemene threshold selecteert alle indicatorpixels, maar jammer genoeg ook enkele verkeerde (voornamelijk rechtsonder).

## 4 Voorlopige resultaten

Momenteel kunnen we een afbeelding gebruiksklaar maken voor de analyse ervan. Dit houdt in dat we alle pixels die niet tot de bloedklonter behoren wit kunnen maken en de afbeelding zodanig bijnijden dat er geen overbodig geheugen door de afbeelding ingenomen wordt. In Figuur 16 wordt dit geïllustreerd.



Figuur 15: Voorlopige lay-out van de app.



Figuur 16: Illustratie van hoe we de afbeelding gebruiksklaar maken voor de analyse ervan

Daarnaast zijn we bezig met de implementatie van de kleurendetectie. We hebben hier al veel vooruitgang geboekt, maar de huid staat nog niet volledig op punt. Ook hebben we al even geëxperimenteerd met de 'app designer' van MATLAB. Hier willen we namelijk onze gebruiksvriendelijke applicatie programmeren. Op basis van wat we nu al bereikt hebben, denken we dat dit project een succes kan worden en dat men het effectief zal kunnen gebruiken in het onderzoek naar de samenstelling van bloedklonten.



## 5 Verantwoordelijkheden en taakverdeling

In onze groep hebben we ervoor gekozen om Robin aan te stellen als projectleider. Hij verdeelt elke week de taken en heeft het laatste woord bij discussies. Voor de verslagen en de eindpresentatie is Marthe verantwoordelijk. Zij is de eindredactrice van alle verslagen. We schrijven deze uiteraard samen, maar het is haar taak ervoor te zorgen dat deze volledig en gestructureerd zijn en dat ze voor de deadline ingediend worden.

Voor de implementatie van de code zijn we elk verantwoordelijk voor ons eigen deel. Zo wordt het verwijderen van de ruis en het bijsnijden van de afbeelding door Toon geleid, de kleurdetectie door Robin en de implementatie van de app door Marthe.

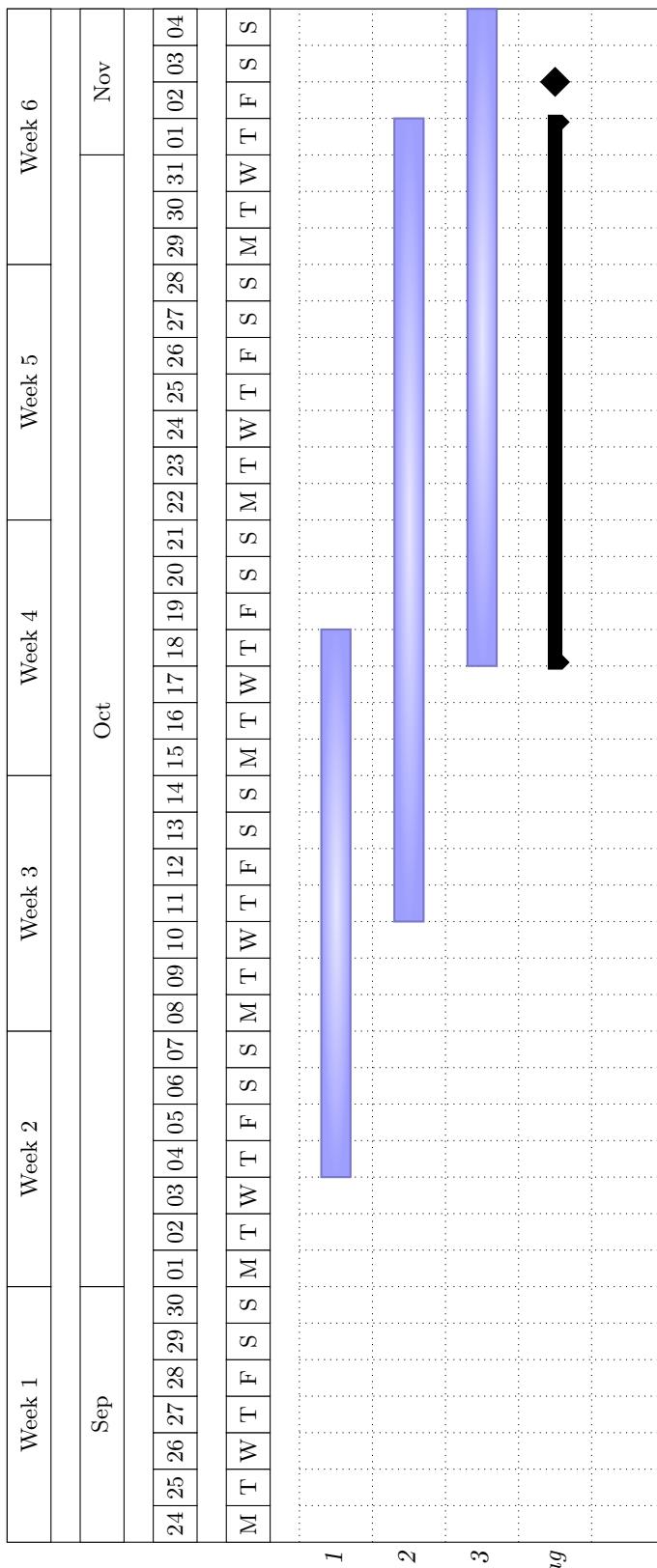
Helemaal achteraan dit verslag voegden we ook een Gantt-chart toe. Hierin staan enkele belangrijke deadlines vermeld en onze voorlopige planning. We hebben het project opgesplitst in vier verschillende taken die we elk ingepland hebben zoals afgebeeld. Ook de tijd die we rekenen voor het tussentijds - en het finaal verslag, zijn af te lezen in de chart.

## 6 Integratie van vakken

Om dit probleem op te lossen hebben we gebruik gemaakt van MATLAB. Het is dan ook een meerwaarde dat we in het eerste semester het vak 'Beginselen van Programmeren' gehad hebben om sneller vertrouwd te raken met deze nieuwe programmeertaal. Daarnaast leren we ook in 'nummerieke wiskunde' werken met MATLAB. Om de verschillende thresholds te bepalen in ons programma, maken we ook gebruik van en statistische analyse van de pixelwaarden. Het van 'Statistiek' draagt hier dus ook zijn steentje bij.

## 7 Besluit

Momenteel kunnen we besluiten dat we goed aan het vorderen zijn. Zoals eerder vermeld kunnen we de foto automatisch bijsnijden en de achtergrond ruisvrij maken. Onze focus ligt nu voornamelijk bij de implementatie van de kleurendetectie. Eénmaal deze ook op punt staat kunnen we beide programma's in de app steken en deze optimaliseren met de nadruk op gebruiksvriendelijkheid naar de klant toe. Verder houden we de klant op de hoogte over onze vooruitgang via onder andere dit verslag.



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

*Eindverslag*

*Eindpres.*





Taaknummer	Taakomschrijving
1	Achtergrond verwijderen en afbeelding bijsnijden
2	Kleuren detecteren en kwantificeren
3	Ontwerpen van een gebruikersvriendelijke app
4	Optimalisatie van het algoritme

Tabel 1: Omschrijvingen van de taken in de Gantt-chart