

Faculteit Computerwetenschappen



P&O Computerwetenschappen

Eindverslag

Blondeel Frédéric
Debeuf Martijn
Sauvillers Toon
Vanbeveren Dirk
Van den Bosch Bert
Van Steenbergen Seppe

Professor: Nuyens Dirk

Academiejaar 2019 - 2020

Inhoudsopgave

1 Inleiding	4
AUTEUR: DEBEUF MARTIJN	
2 Slave Detectie	4
AUTEUR: DEBEUF MARTIJN	
2.1 Beschrijving algoritmes	4
2.2 Resultaten scientific tests	7
2.2.1 Filters	7
2.2.2 Kleurdetectie	9
2.3 Doorgevoerde aanpassingen	10
2.3.1 Kleurperceptie	10
2.3.2 Barcode scanner	11
3 Synchronisatie	14
AUTEUR: BLONDEEL FRÉDÉRIC	
3.1 Beschrijving algoritmes	14
3.2 Resultaten scientific tests	16
3.3 Doorgevoerde aanpassingen	17
4 Tracking	18
4.1 Sensoren	18
AUTEUR: VAN STEENBERGEN SEPPE	
4.1.1 Rotatie	18
4.1.2 Translatie	20
4.1.3 Vinden van rotatie	21
4.2 Camera Tracking	21
AUTEUR: SAUVILLERS TOON	
4.2.1 FAST	22
4.2.2 BRIEF	23
4.2.3 Vinden van translatie	23
4.2.4 Nauwkeurigheid	24
4.2.5 Snelheid	25
4.3 Vinden van transformatie	26
AUTEUR: VAN STEENBERGEN SEPPE	
5 New Features	27
5.1 User interface	27
AUTEUR: VAN DEN BOSCH BERT	
5.2 Multithreading	29
AUTEUR: VAN DEN BOSCH BERT	
5.3 3D-scène	30
AUTEUR: VANBEVEREN DIRK	
5.4 Game	30
AUTEUR: VANBEVEREN DIRK	
6 Besluit	31
AUTEUR: SAUVILLERS TOON	

A Overzicht algoritme	35
AUTEURS: VANBEVEREN DIRK EN DEBEUF MARTIJN	
A.1 Algemeen	35
A.2 Hoekdetectie en filtering	37
A.3 Reconstructie	38
B Kleuren plots	39
AUTEUR: VAN STEENBERGEN SEPPE	
B.1 RGB plots	39
B.2 Histogrammen	40
B.3 HSL plots	41
B.3.1 3D	41
B.3.2 2D	42
C Handleiding Tracking	45

1 Inleiding

AUTEUR: DEBEUF MARTIJN

Dit verslag behandelt de verbeteringen en nieuwe features van het *Cat Caster* project. Een project waarin het mogelijk is om van verschillende schermen één groot scherm te maken aan de hand van een mobiele applicatie. Met de smartphone kan een afbeelding van de opstelling van de schermen genomen worden vanuit een bepaald perspectief. Daarna zullen de afbeeldingen, video's of animatie die op de schermen worden weergegeven, dat initiële perspectief volgen. Nu zijn er onderzoeken gebeurd naar kritieke punten binnen het project, om zo de performantie en algemene resultaten van de applicatie te verbeteren. Ook is de vraag gekomen om het initiële perspectief te kunnen wijzigen door middel van tracking.

Eerst wordt ingegaan op de scientific tests en het iterative development die zijn uitgevoerd omtrent twee belangrijke aspecten binnen de opdracht: Slave detectie (sectie 2) en synchronisatie (sectie 3). Na een herhaling van hoe deze twee initieel werkten, volgt een oplijsting van de onderzoeksresultaten. Daarna worden de aanpassingen aan elk van de aspecten nader bekeken en de performantieverbeteringen aangetoond. Een nieuwe belangrijke component is de tracking feature (sectie 4). Er zijn testen uitgevoerd met verschillende sensoren, camera's en methodes om op basis van deze resultaten een gestaafde beslissing te kunnen maken over welke het best gebruikt kan worden binnen dit project. Daarnaast is de performantie van de tracking zeer belangrijk om een zo realistisch mogelijk resultaat te bekomen. Daarbij is ook een driedimensionale scène gemaakt (sectie 5.3). Andere nieuwe features zijn een meer intuïtieve UI (sectie 5.1), multithreading waardoor de UI ten alle tijden interactief blijft (sectie 5.2) en een game (sectie 5.4).

2 Slave Detectie

AUTEUR: DEBEUF MARTIJN

Algemene overzichten en flowcharts van de architectuur en algoritmen na de aanpassingen zijn terug te vinden in Appendix A.3.

2.1 Beschrijving algoritmes

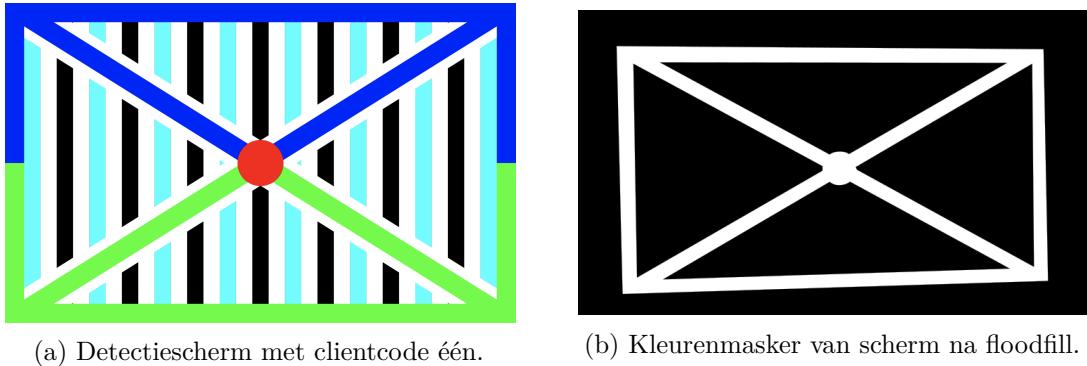
Voordat er scientific tests werden uitgevoerd en iterative development werd toegepast, ging het algoritme voor slave detectie als volgt. De genomen foto van de opstelling werd op het master device geanalyseerd. Na een herschaling volgens [1920 px × 1080 px], om performantie en reconstructie te bevorderen en omzetting van het RGB naar het HSL spectrum, kan deze analyse opgedeeld worden in vier grote stappen.

Floodfill Elk individueel clientscherm laat een vooraf bepaalde afbeelding zien met gekende kleurwaarden zoals op figuur 1a. Het combineren van een rand met een kruis geeft het meeste informatie over de associatie van de kleur-gefilterde pixels bij overlap of afgedekte delen van het scherm en bovendien meer informatie over de relatieve oriëntatie van het scherm op de foto. Om deze schermen te detecteren wordt de foto gefilterd op basis van berekende HSL-ranges (zie B.3.2) en een standaard *four-way floodfill* algoritme [1] om de associatie van de verbonden pixels te behouden. Na de executie van de floodfill is er voor elk gedetecteerd eiland een pixelmasker met een unieke ID per eiland opgeslagen in elke pixel, waar de verdere bewerkingen op uitgevoerd zullen worden. Het geïmplementeerde floodfillalgoritme groeit de vier pixelburen en een stack-based iteratieproces om

recursie–overflow tegen te gaan bij grote afbeeldingen en eilanden. In een worstcasescenario zal dit algoritme een eiland detecteren over de volledige afbeelding. Aangezien elke pixel maximaal vier keer in de stack terecht kan komen, door zijn vier buren, loopt deze floodfill volgens een tijdscomplexiteit van

$$O(4mn) = O(mn)$$

met m en n de dimensies van de afbeelding. De grootte van elk eiland zal in de praktijk over het algemeen een stuk kleiner zijn dan de volledige afbeelding.



Figuur 1: Detectiealgoritme

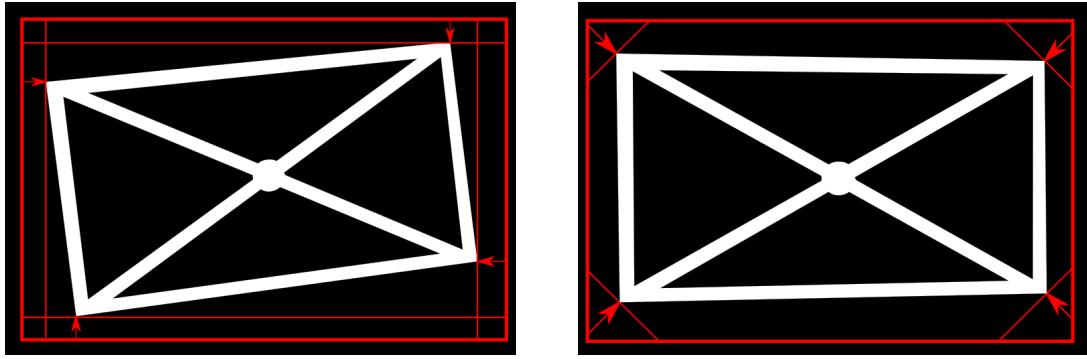
Niet enkel de associatie van de gemaskeerde pixels wordt op deze manier behouden, maar deze methode maakt ook dat de komende pixelbewerkingen maar over een minimale bounding box uitgevoerd worden ten opzichte van de volledige pixelmatrix. Elk resultaat van floodfill represeneert in een verbonden pixelverzameling, een island weergegeven in figuur 1b. De resulterende islands worden achteraf gefilterd opdat elk eiland de twee kleuren van de border en deze van het middelpunt bevatten. Als er aan deze voorwaarde voldaan is, wordt er een poging gedaan om lokaal een middelpunt te detecteren en geldige barcode te lezen 2.1. Bij het falen van een van deze verificatiestappen wordt het eiland verworpen, de overblijvende eilanden zijn geldige eilanden voor verdere hoekdetectie.

Hoekdetectie Vervolgens zoekt het algoritme naar hoeken binnen een island. In de eerste stap wordt er bepaald of het scherm voornamelijk recht of gekanteld is ten opzichte van de foto. Hiervoor wordt langs de linkerkant van het kader de standaarddeviatie van pixels in het masker berekend. Bij een standaardafwijking onder de 15% wordt een scherm als liggend of verticaal gezien op de foto en niet gekanteld.

Indien het scherm als gedraaid beschouwd wordt, zal er vanuit elke rand van de bounding box van het eiland de eerste mask–pixel als corner genomen worden, zie figuur 2a. In het geval dat het scherm relatief horizontaal of verticaal staat, zal er niet loodrecht op de randen gezocht worden, maar volgens een diagonaal tot een mask–pixel, zie figuur 2b. Beide varianten van het algoritme lopen in worstcasescenario volgens

$$O(4mn) = O(mn)$$

Maar zoals eerder vermeld zullen deze in praktijk maar tot de grens van het masker lopen.



(a) Loodrechte versie

(b) Diagonale versie.

Figuur 2: De twee versies van het hoekdetectiealgoritme.

Beide variaties van hoekdetectie zullen altijd vier hoeken als resultaat opleveren. Deze zullen door overlap en foutjes bij het maskeren niet altijd correcte hoeken zijn. Na het bepalen van de hoeken, wordt gecontroleerd of deze wel voldoen aan een vooropgestelde eis. Deze kwalificatie is gebaseerd op bepaalde eigenschappen die in de buurt van elke hoek moeten gevonden worden, namelijk twee lijnen die tot de boord behoren en een diagonaal die naar het middelpunt van het scherm loopt. Deze lijnen zijn bepaald door te filteren op de kleuren van de boorden en diagonalen van het detectiescherm die gescheiden zijn door een witte rand. Deze aanwijzingen moeten gevonden worden binnen een relatieve straal die gelijk is aan 25% van de maximale afstand van de hoekpunten tot het middelpunt. Als in een eerder gevonden hoek deze voorwaarden niet aanwezig zijn, wordt deze hoek verworpen als resultaat van de hoekdetectie.

Hoekreconstructie Na het filteren van de hoeken kan het dus voorkomen dat er geen vier meer overblijven. Volgende zaken kunnen hiervoor de oorzaak zijn. Enerzijds is het mogelijk dat bepaalde delen van schermen elkaar overlappen in de opstelling, anderzijds kunnen één of meerdere hoeken niet of slecht detecteerbaar zijn door een obstakel. Er wordt opgelegd dat minstens twee aanliggende hoekpunten en het middelpunt zichtbaar moeten zijn. Indien de gebruiker zich aan deze vooropgestelde eis houdt kan met volgend algoritme het scherm steeds volledig gereconstrueerd worden. De input die wordt meegegeven is een *dictionary* van de reeds gevonden hoekpunten. De sleutels zijn LU, RU, RD en LD m.a.w. posities van hoeken en als bijkomende waarden coördinaten voor deze die reeds gevonden zijn en een *null* als plaatshouder voor de nog te reconstrueren hoeken.

Eerst worden de vier punten bepaald die zich rond het middelpunt op de diagonalen bevinden. Deze worden ook in een *dictionary* opgeslagen met dezelfde structuur als de input. Daarna wordt hoek per hoek gekeken welke nog ontbreken. Indien reconstructie nodig is, wordt het overeenkomende punt van de diagonalen genomen. Samen met het middelpunt wordt hieruit een eerste reconstructielijn opgesteld. Daarna wordt vanuit een aanliggend hoekpunt het laatste hulppunt bepaald waardoor de tweede rechte wordt getrokken. De gezochte hoek is dan het snijpunt van de twee reconstructielijnen. Deze stappen herhalen voor andere ontbrekende hoeken resulteert in een *dictionary* met alle hoekpunten van het scherm.

Identificatie Met alle hoeken en middelpunt op zak is er een scherm gevonden en wordt het geïdentificeerd met behulp van een kleuren barcode. De barcode bestaat uit

een herhalend patroon van zwarte en witte lijnen telkens gevolgd door een cyaan lijn. Door het gebruik van deze cyaan lijn weet het algoritme waar het patroon eindigt en de volgende sequentie terug begint. De scanner leest het aantal wit-zwart schakeringen met een schakering gedefinieerd als een overgang van wit naar zwart of omgekeerd. Het detecteren van deze kleuren gebeurt aan de hand van opgestelde HSL ranges, zie B.3.2. Indien een gelezen pixel binnen de range van cyaan valt, zitten de scanner aan het begin en/of einde van een sequentie. Indien niet, wordt het onderscheid tussen zwart en wit gemaakt op basis van de waarde van de Lightness uit het HSL spectrum. Groter dan 50 is wit en kleiner dan 50 is zwart. De relatie tussen het aantal schakeringen en de clientcode is gelijk aan

$$aantalschakeringen = \left\lfloor \frac{clientcode}{2} + 2 \right\rfloor. \quad (1)$$

Daarbovenop start de sequentie met een witte streep indien de clientcode even is en een zwarte indien deze oneven is. Het is zeer belangrijk dat de sequentie ook eindigt met de kleur waarmee die begint. Dit omdat het scherm omgekeerd georiënteerd kan zijn in de opstelling, maar de barcode scanner hier geen rekening mee houdt. Zo is de barcode voor clientcode nul gelijk aan 'wit-zwart-wit' en deze voor clientcode één gelijk aan 'zwart-wit-zwart'. Het minimum aantal strepen in een barcodesequentie is dus drie. Het algoritme zal twee keer over alle pixels itereren. Hierdoor heeft het algoritme een tijdscomplexiteit van

$$O(2mn) = O(mn)$$

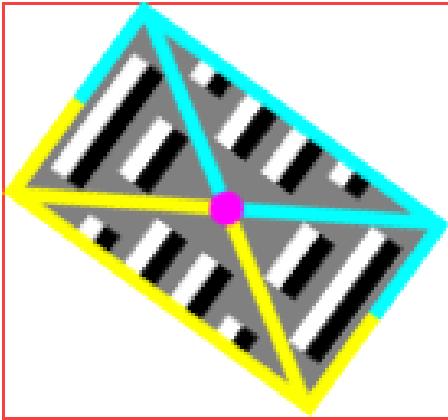
met m en n de dimensies van het eiland waarin de barcode gelezen wordt. Het algoritme gaat een keer horizontaal en een keer verticaal over de pixels. Op deze manier kan de barcode in alle mogelijke oriëntaties van het scherm gelezen worden. Aan de hand van de vier reeds bepaalde hoekpunten van het scherm wordt enkel de boundig box van deze vier punten overlopen. Figuur 3 illustreert een boundig box. Dit om storing van andere schermen te minimaliseren. Herhaling van het patroon heeft als resultaat dat bij overlap het scherm nog steeds geïdentificeerd kan worden. Het herhalend patroon is dus essentieel aan het correct inlezen van de barcode. Een groter aantal herhalingen stemt overeen met een hogere kans op mogelijke detectie, maar stemt ook overeen met een kleinere oppervlakte per herhaling. Deze kleinere oppervlakte is dan weer nadelig voor detectie. Aangezien hiermee de kans stijgt dat een kleur niet gedetecteerd wordt. Nadat het algoritme over alle pixels geweest is, wordt de ratio berekend tussen de code die het meeste keer gelezen is en de totale aantal codes die zijn gelezen. Dit gebeurt zowel in de lijst van horizontaal als verticaal gelezen barcodes. De barcode met de hoogste ratio wordt geselecteerd als gedetecteerde code.

2.2 Resultaten scientific tests

Twee experimenten werden uitgevoerd omtrent de interpretatie van kleur uit afbeeldingen. Hiervoor is een gegevensbank van 1378 afbeeldingen gebruikt. Een gegevensbank opgesteld met *MySQL* en *phpMyAdmin* houdt deze componenten bij.

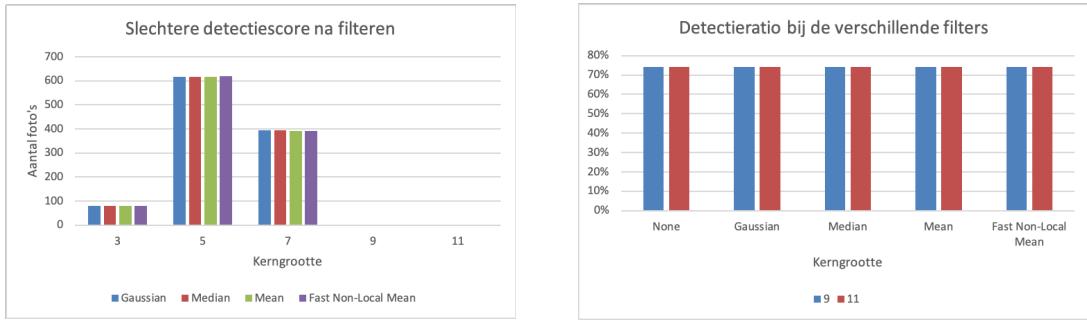
2.2.1 Filters

Een eerste bekeek het gebruik van filters om oneffenheden van kleuropervlakken uit een afbeelding te halen. Vier filters (Median Blur [2], Gaussian Blur [3], Mean Blur [4] en Fast Non-Local Mean [5]) werden vergeleken op twee vlakken. Enerzijds de stijging of daling in



Figuur 3: Rode omkadering geeft een bounding box weer.

correcte detecties en anderzijds de tijdsconsumptie van de filters beiden in functie van de kerngrootte.



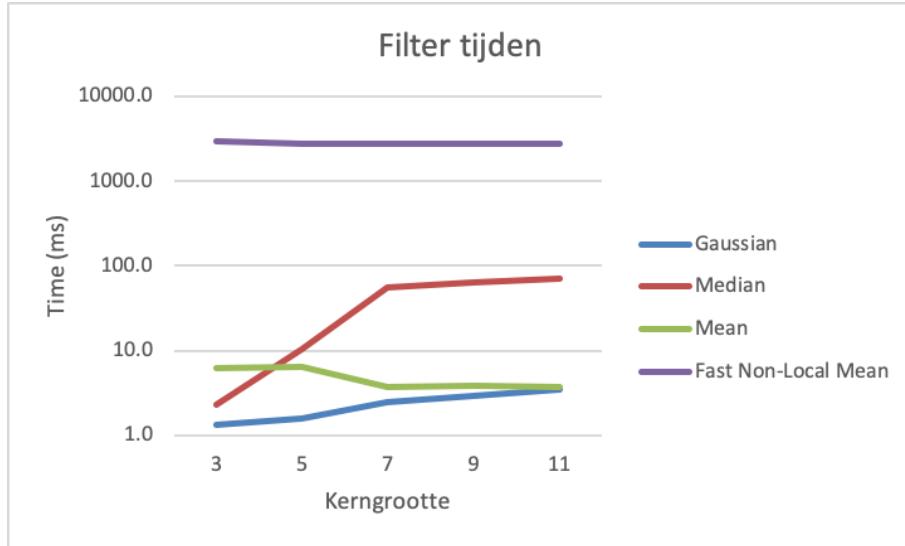
(a) Procent correct gedetecteerd, in functie van toegepaste filter en kerngrootte.

(b) Aantal correct gedetecteerde schermen in functie van filter en kerngrootte.

Figuur 4: Resultaten filters

Conclusies Op basis van de resultaten weergegeven op figuur 4a en figuur 5, kan besloten worden dat het toepassen van een filter op de te analyseren foto geen significant resultaat teweegbrengt. Ook volgt dat de keuze van de filter geen rol speelt. De resultaten zijn gelijklopend tot op enkele afbeeldingen na. Bij kerngroottes kleiner dan negen zijn er zelfs foto's die minder goed gedetecteerd worden na het toepassen van een filter. Een foto die dus niet correct geïdentificeerd werd zonder filter, zal ook niet geïdentificeerd kunnen worden na het toepassen van een filter met kerngrootte kleiner dan negen. Daarom werd dan gekeken wat het effect zou zijn indien kerngroottes negen en elf worden gebruikt. De resultaten zijn weergegeven op figuur 4b. Hier valt meteen op dat het toepassen van een filter met een kerngrootte groter dan negen ook geen significant betere resultaten oplevert. Daarenboven moet men het tijdsverbruik van de filters in acht nemen.

Op basis van bovenstaande bevindingen is besloten om geen filter toe te passen op de te analyseren afbeeldingen in het project. Daar deze er enkel voor zorgen dat de analyse-tijd groter wordt zonder een enigszins beter resultaat.



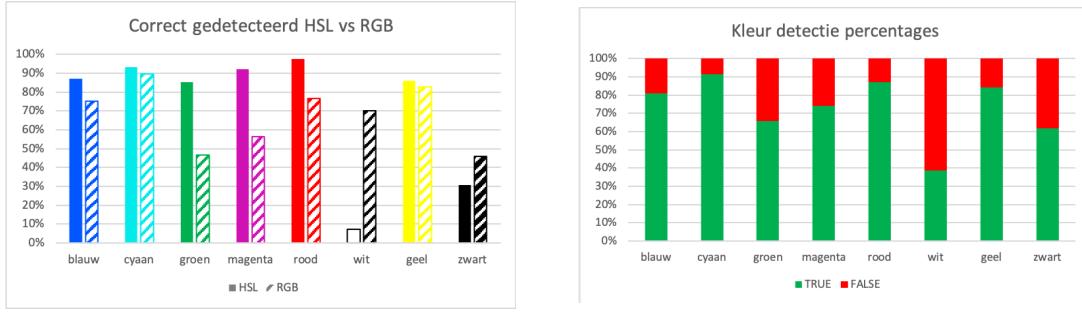
Figuur 5: Benodigde tijd (ms) om filter toe te passen in functie van de kerngrootte.

2.2.2 Kleurdetectie

Een tweede experiment keek hoe goed de verschillende kleuren werden gedetecteerd onder de verschillende omstandigheden alsook binnen de twee kleurruimten. De grootste onderzoeks vragen hier waren welke kleuren er best gebruikt worden in het detectiescherm en binnnen welke kleurruimte.

Omgevingsfactoren Er is een onderscheid gemaakt tussen de hoeveelheid omgeving die meegenomen is in de afbeelding, onderverdeeld in vier categoriën. Elke kleur is gefotografeerd geweest met geen, 10%, 40% en 80% omgeving. Daarnaast werd ook de helderheid van de schermen gewijzigd. De gebruikte standen zijn 25%, 50%, 75% en 100%. Als laatste factor zijn de foto's eens genomen in het donker, dus in afwezigheid van lichtbronnen maar ook eens met artificieel licht.

De afbeeldingen werden ook in de HSL en RGB ruimte apart uitgelezen om deze te kunnen vergelijken.



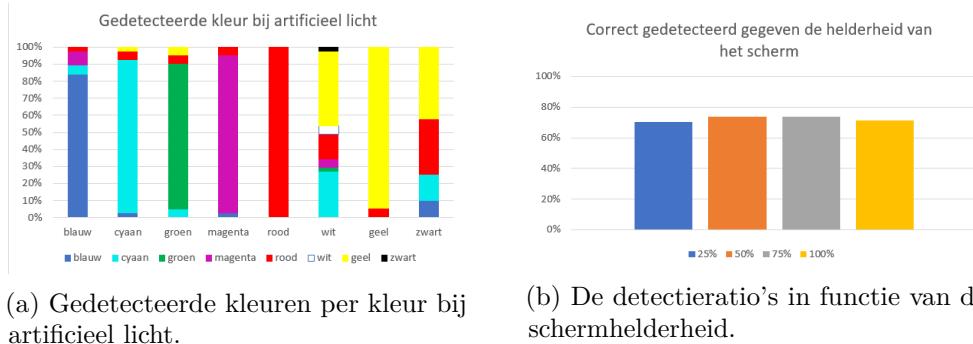
(a) De detectieratio's binnen de HSL en RGB kleurruimten.

(b) Percentage correct gedetecteerd volgens de weergegeven kleur.

Figuur 6: Resultaten kleurdetectie

Conclusies Uit de bevindingen is gebleken dat de interpretatie van kleur sterk afhankelijk is van de manier waarop kleur wordt voorgesteld en de omgevingsfactoren die hierbij aanwezig zijn.

- De kleurruimte waarin een kleur wordt voorgesteld speelt een grote rol bij de detectie, zie figuur 6a. Van de verschillende kleurruimten die bekijken zijn, is er niet een die eenduidig beter is dan de andere. Beiden hebben hun voordeelen. Zo is HSL beter voor de detectie van alle kleuren dan RGB. Maar de detectieratio van zwart en wit is dan wel weer aannemelijk beter in RGB dan in HSL.
- Op figuur 6b is te zien dat de best gedetecteerde kleuren cyaan, rood en geel zijn. Dit zouden dus de beste kleuren zijn om te gebruiken in het detectiescherm. In plaats van rood zou dan wel beter magenta gebruikt worden omdat deze kleur zich in het HSL spectrum meer centreert tussen geel en cyaan dan rood.
- Omgevingsfactoren spelen ook een belangrijke rol bij de detectie van de kleuren. Hiervoor werd gekeken naar zowel de omgeving, de lichtinval en de helderheid van het scherm. De belangrijkste bevindingen hierbij hebben te maken met de lichtinval en de omgeving. Door lichtinval komt er bij artificieel licht extra rood in de kleuren door de rode schijn die het licht achterlaat, zie figuur 7a. Hoe groter de omgeving, hoe meer de kleuren fout gedetecteerd gaan worden. Dit is te wijten aan het feit dat de camera hierbij op de omgeving gaat focussen in plaats van op het scherm. Tot slot is ook gebleken dat helderheid geen groot effect heeft op de detectie, figuur 7b.



(a) Gedetecteerde kleuren per kleur bij artificieel licht.

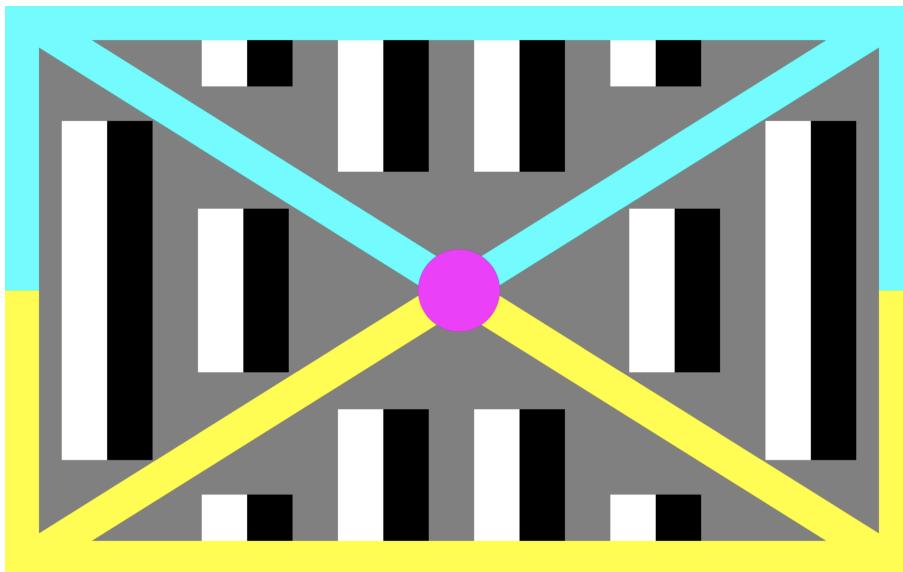
(b) De detectieratio's in functie van de schermhelderheid.

Figuur 7: Resultaten kleurdetectie

2.3 Doorgevoerde aanpassingen

2.3.1 Kleurperceptie

Voor het detectiealgoritme kan het best gebruik gemaakt worden van de HSL kleurruimte zoals nu al het geval is. Omdat binnen HSL enkel gekeken werd naar de S- en L-waarde indien zwart of wit gedetecteerd moet worden, zullen deze twee parameters achterwege gelaten worden. Om kleuren te detecteren zal dus enkel nog rekening gehouden worden met de H-waarden. De keuze voor de drie kleuren die weergegeven worden op het detectiescherm valt op magenta, geel en cyaan. Deze drie kleuren hebben de grootste detectieratio's en de omgevingsfactoren hebben het minste invloed op deze kleuren. Een voorbeeld van het nieuwe detectiescherm is te zien op figuur 8.

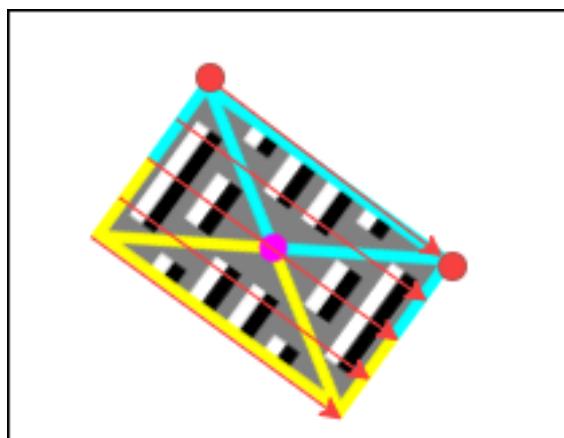


Figuur 8: Het nieuwe detectiescherm, gebaseerd op de resultaten van de onderzoeken.

2.3.2 Barcode scanner

De meest invloedrijke aanpassing is gebeurd aan de barcode scanner. De verandering van het uitlezen in HSL naar RGB en de iterator hebben ervoor gezorgd dat veel meer schermen en opstelling correct gedetecteerd kunnen worden. Hiervoor is de manier waarop de barcode gelezen wordt ook gewijzigd.

Iterator Een eerste element die hierin meespeelt is het gebruik van een pixeliterator om de barcode te lezen. Hieraan kunnen de linker- en rechterbovenhoek worden meegegeven. Daarnaast worden ook de breedte en hoogte van het scherm op de afbeelding meegegeven in aantal pixels. De iterator geeft dan volgens de oriëntatie van deze twee hoekpunten telkens een rij van pixels terug, tot het volledige scherm is overlopen, weergegeven op figuur 9.



Figuur 9: Werkwijze iterator om de barcode te lezen.

Doordat de barcode nu altijd van links naar rechts gelezen wordt is het niet meer nodig dat de laatste sequentie streep dezelfde kleur heeft als de eerste. Dit resulteert in kortere

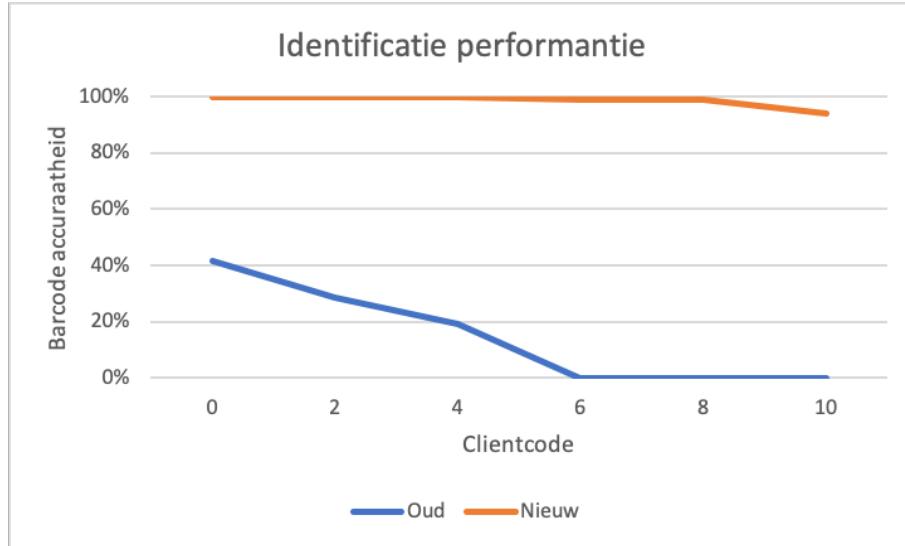
barcodes, het minimum aantal strepen is nu twee. Clientcode nul heeft nu sequentie 'wit-zwart' en clientcode één heeft sequentie 'zwart-wit'. Kortere sequenties betekent bredere lijnen en resulteert in een meer nauwkeurig identificatiealgoritme.

HSL naar RGB Het uitlezen van de pixels gebeurt nu in RGB waarden. Hiervoor is eerst het detectiescherm wat gewijzigd. Dit is zo aangepast dat enkel volledige barcodes worden weergegeven. Op het oude detectiescherm liepen deze achter de diagonalen door waardoor rond de diagonalen soms halve barcodes werden weergegeven. Dit verkleint de kans op foute scans. Doordat de interpretatie van kleur nu volgens het RGB spectrum gebeurt en voor de barcode enkel wit en zwart onderscheiden moet kunnen worden, is er geen nood meer aan de gekleurde lijnen die het begin en eind van een barcode sequentie weergeven. In de HSL versie was dit wel nodig omdat in dat spectrum wit, zwart en grijs onderscheiden niet nauwkeurig genoeg kan gebeuren. De achtergrond is nu grijs (RGB waarde van [128, 128, 128]).

Vooraleer er begonnen wordt met lezen wordt de scanner gecalibreerd. Door binnen de volledige afbeelding de meest witte en zwarte pixelwaarde te zoeken. Dit gebeurt door voor elke pixel, zijn afstand te bekijken tot deze witte [255, 255, 255] en zwarte [0, 0, 0] waarde aan de hand van vergelijking (2). De twee pixelwaarden die dan het dichtst bij elk van deze theoretische waarden aanliggen worden dan gebruikt als referentie voor zwart en wit tijdens het lezen van de barcode. Verder blijft het identificatiealgoritme gelijk.

$$afstand = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2} \quad (2)$$

Performantietest Als laatste wordt de performantieverbetering van deze aanpassingen bekeken voor het identificatiealgoritme. Hiervoor is een experiment opgesteld met het oude en nieuw algoritme. In gelijke omstandigheden werd telkens een detectiescherm met clientcode gaande van nul tot tien weergegeven voor beiden algoritmen. Deze werden dan door de analyse gehaald en hierbij werd de accuraatheid van de barcode weergegeven. Een voorbeeld van de opstellingen is weergegeven in figuur 10. Het gebruik van artificieel licht is zeer uitdagend voor het oud algoritme door de rode schijn die in de afbeelding wordt gecreëerd maar helpt om de verbeteringen te benadrukken. Ook staat de helderheid van beide schermen telkens op 75%.



Figuur 11: Resultaten van de perfromantietest.



Figuur 10: Opstellingen perfromantietest

De resultaten zijn weergegeven op figuur 11. Hierbij ziet men meteen de stijging in accuraatheid van de gescande barcode. Het oude algoritme gaf vanaf clientcode 6 geen of de verkeerde barcode terug. Het nieuw algoritme had hierbij geen enkel probleem en de accuraatheid daalde niet onder de 90%. De stijging is vooral toe te wijten aan de volgende punten.

- De barcode scanner weet nu, door de hoeken en iterator, de richting waarin gescand moet worden.
- Door de kortere sequenties kunnen de lijnen breder weergegeven worden wat de kans op een correct gelezen sequentie verhoogt.

- De overgang van HSL naar RGB die zorgt voor een beter onderscheid tussen zwart en wit.
- Enkel volledige barcodes worden nog weergegeven.

3 Synchronisatie

AUTEUR: BLONDEEL FRÉDÉRIC

3.1 Beschrijving algoritmes

De applicatie gebruikt de synchronisatie voor 3 verschillende doeleinden: countdown, video en animatie. Het basisidee is het berekenen van het tijdsverschil tussen slave en master, dit verschil af te trekken van de slave's interne tijd (opgevraagd met *Date.now()*) om zo een perfecte synchronisatie te bekomen.

Klokken synchronisatie De server en slave devices staan in synchronisatie aan de hand van een *syncDelta*. Dit is het verschil tussen de klok van de slave en van de server.

$$currentTime = Date.now() + syncDelta$$

Deze waarde wordt berekend door gebruik van het NTP protocol [6] en wordt toegelicht in ST1.1.

Video De video synchronisatie werkt aan de hand van het versnellen of vertragen van de video afspeelsnelheid. Elke slave zal een gesynchroniseerde klok hebben en aan de hand hiervan berekenen op welk punt van de video hij moet zijn. Elke slave zal om de 30ms checken of de video achterloopt of voorloopt (dit gebeurt aan de hand van een *setInterval()*) en zal de afspeelsnelheid respectievelijk versnellen of vertragen. Als het verschil te groot is ($>1\text{sec}$) zal een sprong naar het correcte stuk van de video plaatsvinden.

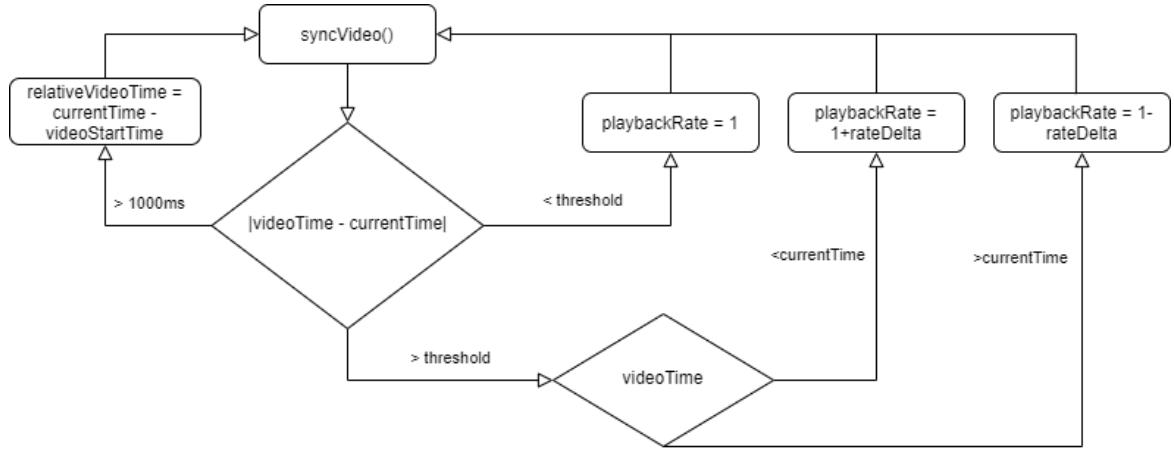
$$videoTime = videoStartTime + relativeVideoTime$$

$$|videoTime - currentTime| < threshold$$

De *relativeVideoTime* is de actuele vooruitgang van de video. De *threshold* is een voorgedefinieerde waarde om af te wegen of de video al dan niet gesynchroniseerd is. Als de absolute waarde van het verschil de *threshold* overschrijdt zal de video versnellen of vertragen.

$$playbackRate = 1 \pm rateDelta$$

Met *rateDelta* een voorgedefinieerde waarde die de playbackrate zal aanpassen.



Figuur 12: Video synchronisatie protocol

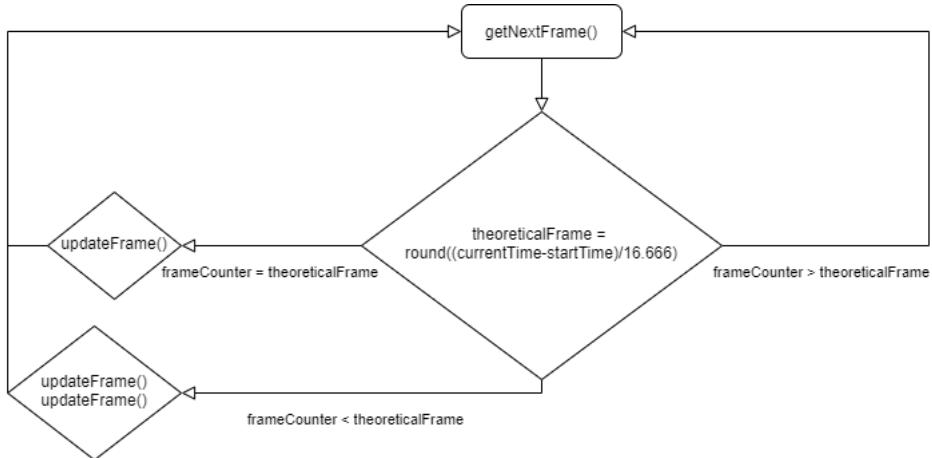
Animatie De animatie gebeurt op een analoge manier als de video synchronisatie. De klokken van de slave devices staan in sync met de server. Wanneer de animatie wordt opgeroepen zal elke slave de starttijd opslaan, en aan de hand van *requestAnimationFrame()* om de 16.66 ms (60fps) een nieuwe frame tekenen, hierbij wordt ook een *frameCounter* bijgehouden. In het geval dat een computer trager zou lopen en een frame niet zou tekenen zal de volgende vergelijking niet kloppen,

$$frameCounter = theoreticalFrame$$

met

$$theoreticalFrame = \text{round}\left(\frac{\text{currentTime} - \text{startTime}}{16.66}\right)$$

Er zal dan ofwel een extra of geen *updateFrame()* plaatsvinden naargelang *frameCounter* groter of kleiner is dan de *theoreticalFrame*.



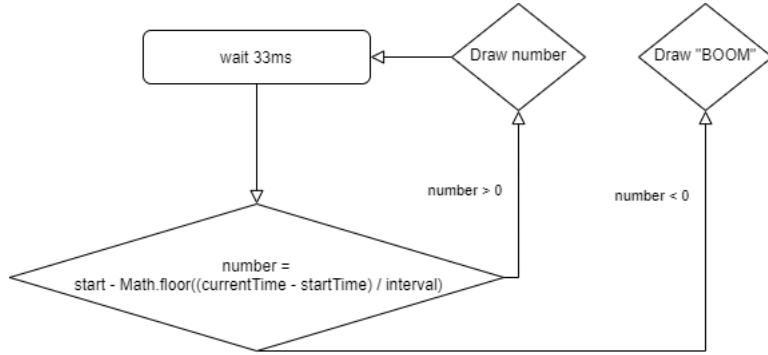
Figuur 13: Animation synchronisatie protocol

Countdown De countdown gebruikt een simpel algoritme en is toch heel accuraat (voornamelijk door de correcte *syncDelta*). Elke slave is zoals gewoonlijk in synchronisatie met de server. Elke slave zal apart een *setInterval()* functie draaien die om de 33ms een

getal zal tekenen op het canvas. Dit getal zal de countdown zijn die berekend is op de volgende manier

$$number = start - Math.floor((currentTime - startTime) / interval)$$

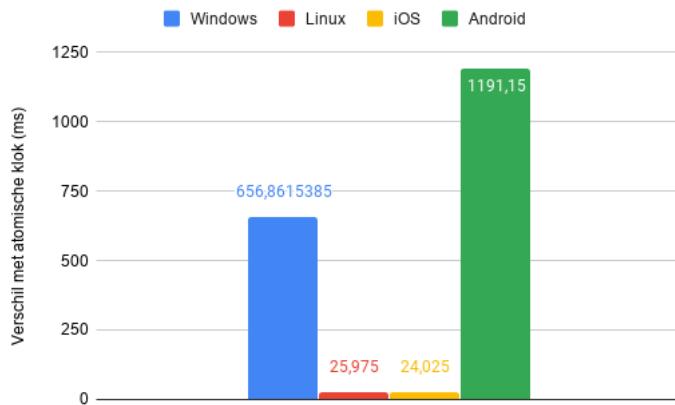
Met *start* het start getal van de countdown, *interval* het tijdsinterval tussen 2 getallen.



Figuur 14: Countdown protocol

3.2 Resultaten scientific tests

Uit de testen is bewezen dat er een groot verschil kan zijn tussen de klok van de clients en de wereldtijd (en bijgevolg de server), zie figuur 15. Als oplossing is het NTP protocol gebruikt om het tijdsverschil te berekenen tussen slave en server.

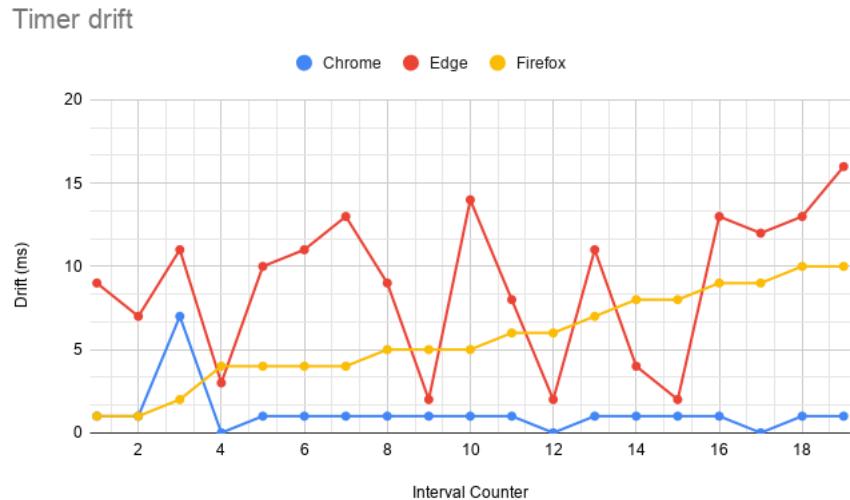


Figuur 15: Tijdverschil van diverse apparaten

Vervolgens zijn er ook fluctuaties in het netwerk die er voor zouden kunnen zorgen dat dit tijdsverschil een slechte benadering is van de realiteit. Dit kan opgelost worden door 10 keer de ping [7] te berekenen en vervolgens het gemiddelde te nemen.

Verder is het ook gebleken dat de klokdrift weinig impact heeft over een korte tijdsspanne en is daardoor verwaarloosbaar. Het is dus genoeg om enkel bij de start van animatie, video of countdown te synchroniseren.

Daarenboven volgt dat animaties niet altijd even snel verlopen op verschillende toestellen, zie figuur 16.



Figuur 16: Timer drift

Dit heeft te maken met hoelang het toestel nodig heeft om een frame te tekenen (meestal door hardware, of ook door netwerk latency).

Daarnaast zijn er ook nog verschillende manieren om elke frame te starten. Waarbij de `setTimeout()` functie bijvoorbeeld meerdere keren de berekeningen maakt voor een stap voordat de frame helemaal getekend is. Dit kan zorgen voor een animatie die veel sneller loopt dan normaal. De oplossing hiervoor is de functie `requestAnimationFrame()` die het mogelijk maakt de berekeningen te maken voordat de frame getekend wordt, al kan dit op tragte hardware de framerate verlagen.

Deze redenen zorgen ervoor dat synchronisatie niet mogelijk is op basis van de assumptie dat elke frame perfect om de 16.66ms (60fps) wordt getekend. Wel mogelijk is het vergelijken van een theoretische frame (gebaseerd op de gesynchroniseerde tijd met de server) en de actuele frame. Indien deze niet meer gelijk zouden zijn, kan een versnelling of vertraging ingevoerd worden om ze terug gelijk te stellen.

3.3 Doorgevoerde aanpassingen

De belangrijkste aanpassing is de synchronisatie met de server zodat elke client de exacte tijd van de server heeft, dit gebeurt dus aan de hand van het NTP protocol. Met dit gegeven kunnen allerlei algoritmes geschreven worden om een video of animatie foutloos af te beelden (uitgelegd in sectie 3.1). Vorig semester werd dit niet gedaan, in de plaats werd simpelweg een start commando gestuurd naar alle verbonden apparaten en "hopelijk" ging dit snel genoeg om "ongeveer" synchroon te zijn. Er was ook geen check naar drift dit wil zeggen geen versnelling of vertraging naarmate de synchronisatie op andere apparaten anders verliep. Deze aanpak was uiteraard zeer onderhevig aan netwerk fluctuaties en indien de hardware trager werkte liep dit helemaal mis.

Een grote verbetering was ook het gebruik van `requestAnimationFrame()` om de animatie af

te beelden. Dit verliep vroeger zeer slecht met andere methodes zoals *setInterval()* want deze methode houd geen rekening met de mogelijkheid dat een frame niet op tijd wordt berekend en getekend. De nieuwe methode is ook speciaal gemaakt voor dit doeleinde.

4 Tracking

4.1 Sensoren

AUTEUR: VAN STEENBERGEN SEPPE

Eén manier om beweging van een apparaat te detecteren is met behulp van zijn sensoren. Vandaag de dag zitten smartphones vol met sensoren die allerlei verschillende soorten data verzamelen [8]. Diegene die nuttig kunnen zijn voor het bepalen van de beweging zijn de accelerometer, gyroscoop en magnetometer.

De accelerometer meet de acceleratie die het apparaat ondervindt. Hiermee kan deze sensor de oriëntatie bepalen van het apparaat aan de hand van de zwaartekracht of de beweging in eender welke lineaire richting.

De gyroscoop geeft ook informatie over de oriëntatie van het apparaat maar is nauwkeuriger dan de accelerometer. Ook kan de gyroscoop rotatie meten wat een accelerometer niet kan.

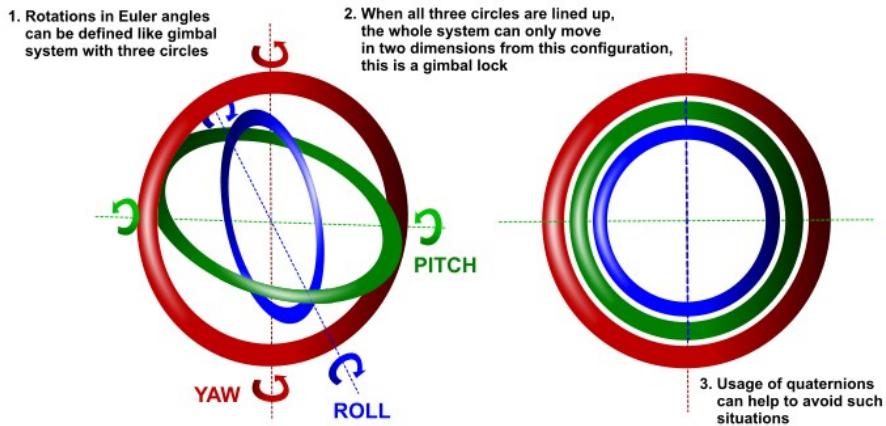
De magnetometer kan magnetische velden detecteren. Dit wordt gebruikt om het noorden te vinden aan de hand van het aardmagnetisch veld.

Voor het uitlezen van de waarden van deze sensoren wordt gebruik gemaakt van de *Generic Sensor API* [9]. Deze is standaard ingeschakeld in Chrome vanaf Chrome 67. Voor Firefox en Safari is de compatibiliteit niet vermeld in de documentatie [10]. Uit eigen testen blijkt dat deze browsers de *Generic Sensor API* niet ondersteunen. Vervolgens kunnen de uitgelezen waarden gebruikt worden om de beweging van het apparaat te bepalen. Deze beweging kan opgesplitst worden in zijn rotatie en translatie. Dit verslag bespreekt zowel de implementatie als de nauwkeurigheid van deze beiden.

4.1.1 Rotatie

Voor de rotatie wordt de *OrientationSensor* interface gebruikt van de *Generic Sensor API*. Hierbij is nog de keuze tussen de *AbsoluteOrientationSensor* interface en de *RelativeOrientationSensor* interface. Het verschil tussen beiden is het assenstelsel waarin de rotatie gedefinieerd wordt. Bij de absolute is dit het referentie coördinatensysteem van de aarde. Bij de relatieve is dit een stationair coördinatensysteem. Er wordt voor de relatieve geopteerd omdat deze geen gebruik maakt van de magnetometer voor het bepalen van het referentie coördinatensysteem van de aarde. Deze kan namelijk verstoord worden door magnetische velden en dus de accuraatheid aantasten [11].

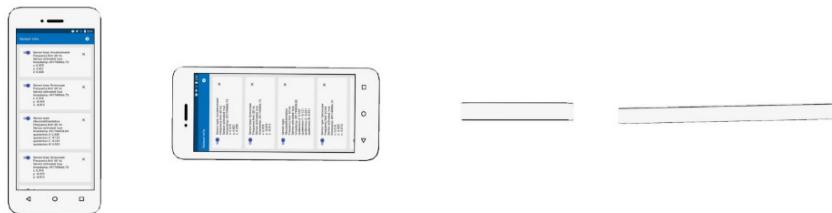
Om de oriëntatie te beschrijven is er de keuzen tussen Eulerhoeken en quaternionen [12]. Hiervoor worden de quaternionen gekozen omdat deze een Gimbal lock vermijden. Indien Eulerhoeken worden gebruikt kan dit worden voorkomen, waardoor er een vrijheidsgraad verloren gaat en dus informatie over de oriëntatie. Zie figuur 17 voor een duiding.



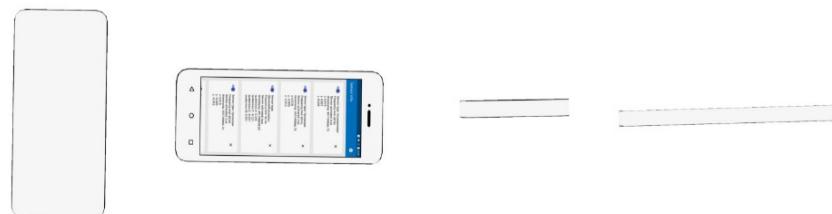
Figuur 17: Illustratie van een Gimbal lock, een vrijheidsgraad gaat verloren [13]

Om een zicht te krijgen op de nauwkeurigheid van de sensoren in een smartphone, zijn twee experimenten uitgevoerd met een Samsung Galaxy S9. De eerste test toont de oriëntatie van de smartphone in enkele gekende posities.

x: 0°, y: 0°, z: 0° x: 0°, y: 0°, z: 90° x: 90°, y: 0°, z: -2° x: 92°, y: 1°, z: 90°



x: 180°, y: 0°, z: -179° x: 0°, y: 0°, z: -91° x: -90°, y: -1°, z: -177° x: 91°, y: 2°, z: -90°

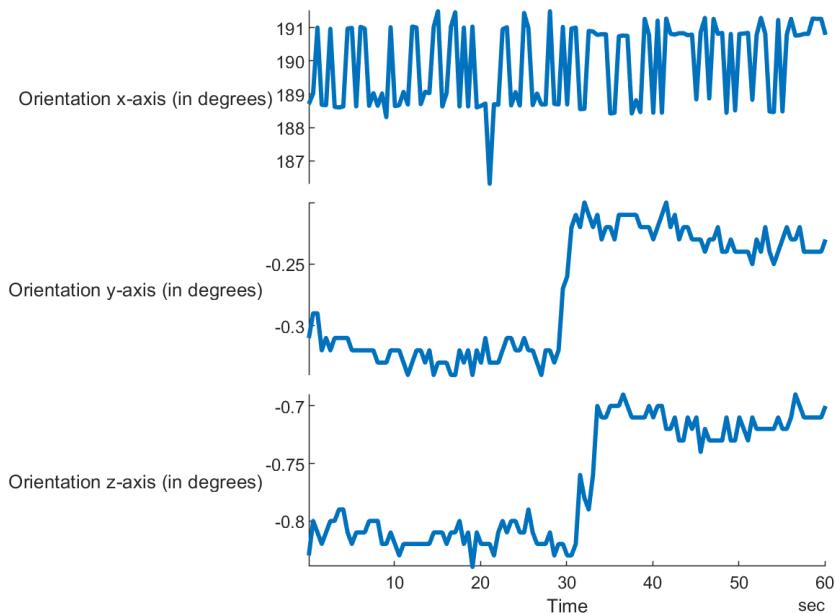


Figuur 18: De uitgelezen waarde van de sensoren in Euler hoeken en weergegeven op een 3D-model vanuit bovenaanzicht.

Zie figuur 18 voor de verschillende oriëntaties en de bijbehorende uitgelezen waarde van de

sensoren. Het verschil tussen de effectieve oriëntaties en de uitgelezen oriëntaties zijn op enkele graden na gelijk.

Voor het tweede experiment werd de smartphone gedurende 60 seconden plat op tafel gelegd. In deze tijd werden de accelerometer en gyroscoop uitgelezen aan de hand van de applicatie "AndroSensor"[\[14\]](#).



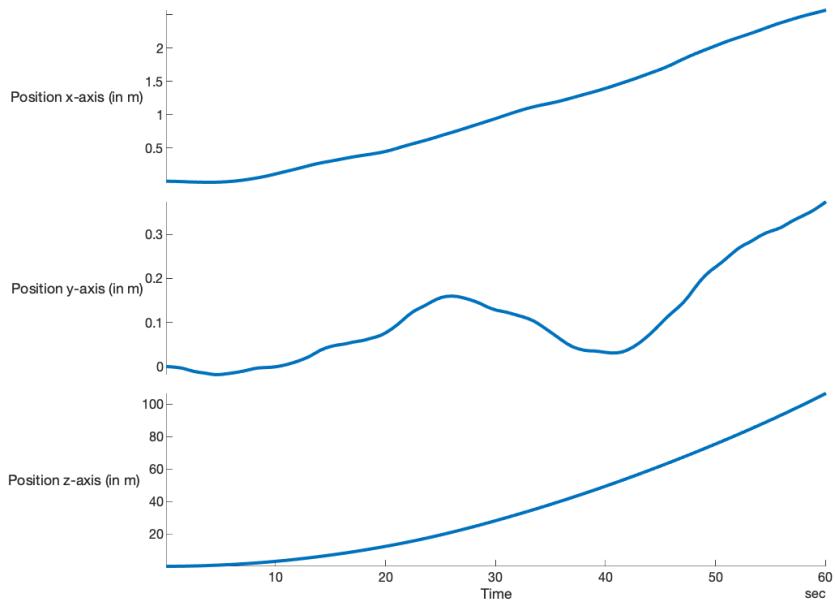
Figuur 19: Oriëntatie apparaat in rust, plat liggend op tafel gedurende 60 seconden volgens de x-as, y-as en z-as met behulp van de *RelativeOrientationSensor* interface.

Zie figuur [19](#) voor de oriëntatie van het apparaat dat plat ligt op een tafel en niet beweegt. Gedurende deze 60 seconden aan data is te zien dat er altijd een fout zit op de gemeten waarde. Deze is wel beperkt tot maximaal enkele graden.

Uit beide testen blijkt dat de oriëntatie van een apparaat op enkele graden na juist kan afgelezen worden aan de hand van zijn sensoren. Dit maakt het accuraat genoeg om de oriëntatie van een apparaat te bepalen aan de hand van zijn sensoren voor de *ScreenCaster*.

4.1.2 Translatie

Voor de translatie moet gekeken worden naar de acceleratie van het apparaat. Hiervoor wordt de *LinearAccelerationSensor* interface gebruikt van de *Generic Sensor API*. Hierbij is de zwaartekracht al van de gemeten waarde afgetrokken. De positie wordt dan berekend aan de hand van een dubbele integraal over de acceleratie omdat de acceleratie de tweede afgeleide is van de positie.



Figuur 20: Positie apparaat in rust, plat liggend op tafel gedurende 60 seconden volgens de x-as, y-as en z-as met behulp van de *LinearAccelerationSensor* interface.

Zie figuur 20 voor de positie van het apparaat dat plat ligt op een tafel en niet beweegt gedurende 60 seconden. Hierop is te zien dat de positie volgens de z-as een kwadratische drift heeft. Na 60 seconden loopt de fout hierdoor al op tot ongeveer 100 meter. Indien voor de positie, in functie van de tijd, een lineaire regressie wordt opgesteld verkrijgt men een R^2 gelijk aan 0.59. Hierdoor kan er verondersteld worden dat deze fout zich lineair zal verder zetten. Na 60 seconden bedraagt de fout hier ook al twee meter. De fout op de positie in de y-as blijft bescheiden maar heeft toch een enigszins stijgend verband. Deze bevindingen komen doordat ook bij de accelerometer er een fout zit op de gemeten waarde. Dit zorgt ervoor dat de translatie van een apparaat berekenen aan de hand van sensoren niet mogelijk is voor de applicatie.

4.1.3 Vinden van rotatie

Om de rotatie een apparaat te beschrijven wordt er gekeken naar het relatief verschil tussen de beginpositie en de dan gemeten positie. Hiervoor worden de beginpositie en de dan gemeten positie opgeslagen in een 4×4 transformatiematrix. Het relatief verschil is dan het product van de gemeten rotatie met de inverse van de beginpositie.

4.2 Camera Tracking

AUTEUR: SAUVILLERS TOON

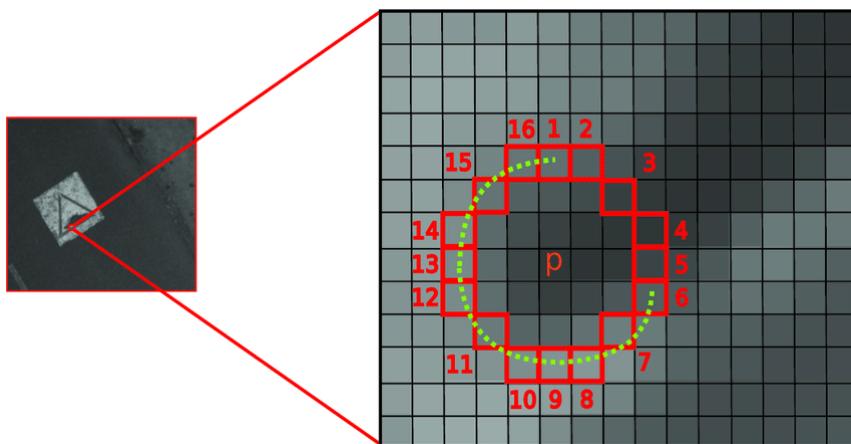
Een tweede manier om beweging van een apparaat te detecteren is met behulp van de camera. Deze beweging vertaalt zich vervolgens in een transformatie. Om beweging te detecteren zijn er punten nodig waarop gefocust kan worden, dit zijn de *keypoints*. Deze punten springen uit in een afbeelding waardoor steeds dezelfde punten gevonden kunnen worden. Tussen verschillende frames zullen deze keypoints zich verplaatsen. Het linken van de keypoints tussen deze twee frames zal resulteren in een beweging.

Een algoritme dat in de *Technical Committees* naar voren kwam is het *ORB* algoritme [15]. ORB staat voor *Oriented FAST and Rotated BRIEF*. In *ORB: an efficient alternative to SIFT or SURF (2011)* bespreken E. Rublee, V. Rabaud, K. Konolige en G. Bradski waarom ORB te verkiezen is boven SIFT of SURF, twee andere keypoint detectiealgoritmes.

Deze sectie bespreekt eerst FAST, een algoritme om keypoints te detecteren. Vervolgens gaat het over naar BRIEF, dit algoritme koppelt de gevonden keypoints aan elkaar.

4.2.1 FAST

FAST staat voor *Features from Accelerated Segment Test* en werd geïntroduceerd door E. Rosten en T. Drummond in 2005 [16]. FAST zoekt naar hoeken in de afbeelding waar de intensiteit van de omgeving hard verschilt. Rond een pixel p worden alle pixels in een *Bresenham cirkel* met straal drie bekeken. Wanneer er bij twaalf opeenvolgende pixels de intensiteit verschilt met minimaal t , een voorafbepaalde drempelwaarde, wordt p als een *keypoint* beschouwd, zie figuur 21. Om het algoritme snel genoeg te laten verlopen maar niet teveel keypoints te vinden, wordt een drempelwaarde tussen de 20 en 70 gebruikt. Afhankelijk van de omgeving kan deze manueel worden aangepast in de applicatie. In geavanceerde systemen zal FAST zichzelfs steeds verbeteren door middel van *machine learning*, in de applicatie is deze optie weggelaten omdat dit te verdiepend zou zijn voor een 'proof of concept'.



Figuur 21: Pixel p voldoet aan alle voorwaarden om een *keypoint* te zijn. [17]

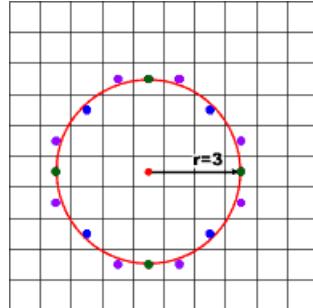
Intensiteit De intensiteit van een pixel is gelijk aan zijn grijswaarde. Bij de implementatie wordt de foto eerst omgezet naar een grijswaarde foto. De intensiteit zal zo een gewogen waarde worden van de RGB-waarden van de pixel. De coëfficiënten zijn de waarden gebruikt en vastgelegd door de *NTSC*¹.

$$\text{Intensiteit} = 0.299 * R + 0.587 * G + 0.114 * B$$

Bresenham cirkel Het Bresenham algoritme is geïntroduceerd in de *Computer Graphics* om lijnen en andere vormen als pixels af te beelden. Door dit algoritme toe te passen zal

¹NTSC is *The National Television System Committee*, een analoge norm voor kleurentelevisie geïntroduceerd in 1954.

elke cirkel die genomen wordt, altijd éénzelfde vorm aannemen [18]. Zie figuur 22 voor een visualisatie van een Bresenham cirkel met straal drie.



Figuur 22: Bresenham cirkel met straal drie. De aangeduide pixels worden gebruikt om de cirkel te definiëren. [19]

4.2.2 BRIEF

BRIEF staat voor *Binary Robust Independent Elementary Features*. Dit algoritme gebruikt de gevonden keypoints volgens FAST (4.2.1) en probeert voor elk keypoint een match te vinden met een keypoint van een andere set. [20]

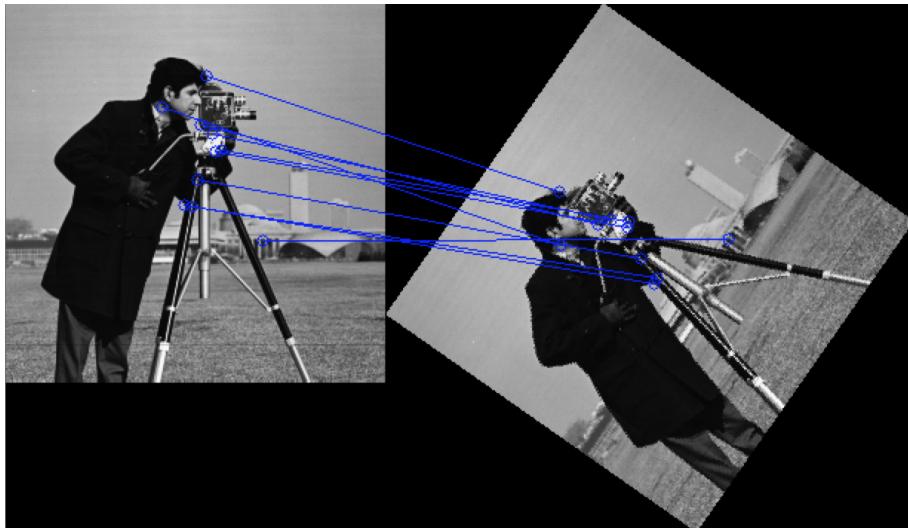
BRIEF koppelt paren van keypoints door een binaire string op te stellen voor elk keypoint van dat paar. Door deze binaire strings te vergelijken aan de hand van de *Hamming Distance* [21] kijkt het algoritme of deze keypoints een match zijn of niet. Daarbij bekijkt het ook hoe nauwkeurig de koppeling is, des te kleiner de *Hamming Distance* des te zekerder het algoritme is dat deze twee keypoints bij elkaar horen.

Keuze voor BRIEF in ORB Verschillende keypoint-matching algoritmes zoals SIFT [22] en SURF [23] beschrijven een keypoint in een vector van 512 bytes. Het aanmaken van al deze beschrijvingen loopt al snel op bij een groot aantal gevonden keypoints en neemt veel geheugen in beslag. Daarenboven duurt het matchen langer hoe meer geheugen er gebruikt wordt voor de beschrijvingen. BRIEF slaat de stap van de beschrijving maken over om keypoints te matchen wat een enorme winst aan geheugen oplevert zonder accurate matching op te geven. [24]

Implementatie De implementatie en parameters zijn volledig gebaseerd op de BRIEF-implementatie van *trackingjs* [25]. Dit is een in de praktijk bewezen correcte en efficiënte implementatie.

4.2.3 Vinden van translatie

Met behulp van de keypoints, gevonden door *FAST* en gekoppeld door *BRIEF*, kan er een translatie gevonden worden. Alle gekoppelde keypoints hebben een nauwkeurigheid, deze geeft weer hoe zeker *BRIEF* ervan is dat de twee keypoints bij elkaar horen. Vanaf dat deze nauwkeurigheid boven de zeventig procent ligt, kijkt het algoritme wat de translatie volgens de x- en y-as is. De gemiddelde translatie van alle keypoints die aan de genoemde criteria voldoen, wordt als algemene translatie van het apparaat genomen. Zie figuur 23 voor een illustratie.



Figuur 23: Het matchen van de keypoints in twee frames. [26]

4.2.4 Nauwkeurigheid

Vooraleer dat de applicatie de gevonden translatie kan gebruiken, onderzoekt dit verslag eerst de nauwkeurigheid van het geïmplementeerde *ORB* algoritme. Het algoritme zal verschillende frames van eenzelfde afbeelding, zie figuur 24, analyseren. Deze frames zijn op een gecontroleerde manier enkele pixels opgeschoven. De afwijking die er al dan niet zal zijn, bepaalt de nauwkeurigheid.

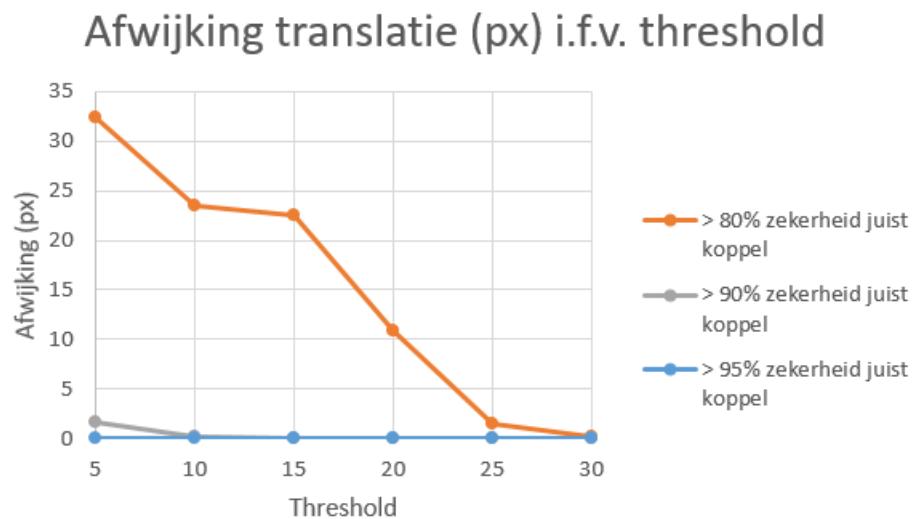


Figuur 24: De gebruikte afbeelding voor het testen van de nauwkeurigheid van *ORB*.

Als eerste behandelt dit verslag de invloed van de intensiteitsthreshold, zie vorige sectie 4.2.1. op de afwijking van de translatie. Hiervoor wordt voor elke thresholdwaarde 50 keer een willekeurige translatie van maximaal 30 pixels (in zowel x- als y-richting) gecreëerd en

vervolgens gezocht. Hiervan wordt dan de mediaan van afwijkingen genomen om uiterste waarden uit te filteren, een slecht gekozen achtergrond leidt namelijk snel tot weinig tot geen keypoints wat tot een zeer grote afwijking resulteert. Dit experiment wordt driemaal herhaald waarbij steeds een hogere zekerheid moet zijn van de keypoints die aan elkaar gelinkt worden. Uit figuur 25 kan men twee dingen concluderen. Als eerste wordt de afwijking steeds kleiner naarmate de zekerheid tussen de koppels van keypoints hoger is. Het tweede is dat wanneer deze keypoints gevonden worden met een hogere threshold, de afwijking ook steeds vermindert.

Bij 90% zekerheid is de mediaan van de 50 translaties steeds onder de 1 pixel. Vanaf een thresholdwaarde van 25 is de afwijking zelfs nul. Bij 95% is de afwijking zelfs steeds nul pixels, zie tabel 1 voor de precieze cijfers van de koppels met 90% zekerheid. Uit deze gegevens is duidelijk dat voor een hoge zekerheid gekozen moet worden, liefst in functie van een goede threshold.



Figuur 25: De afwijking van de translatie in functie van de thresholdwaarde.

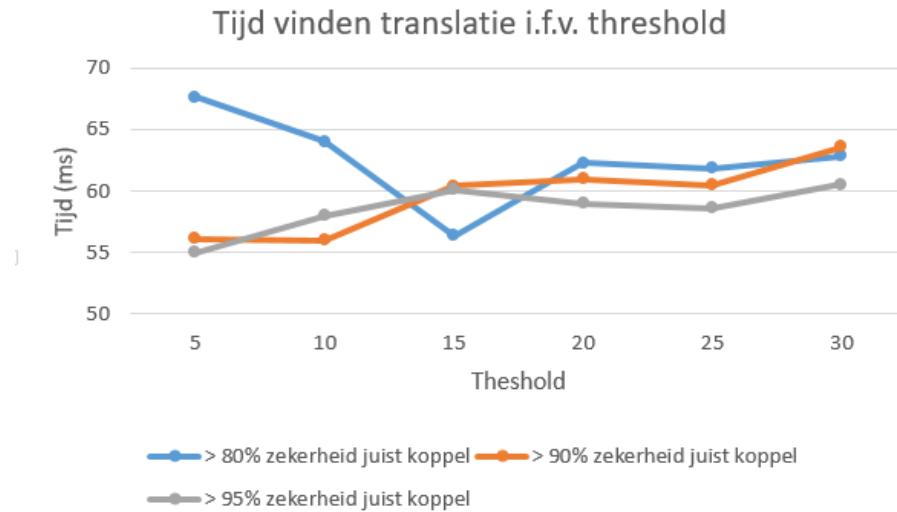
Threshold	Afwijking 80% (px)	Afwijking 90% (px)	Afwijking 95% (px)
5	32.33	1.66	0.0
10	23.54	0.15	0.0
15	22.57	0.07	0.0
20	10.91	0.04	0.0
25	1.48	0.0	0.0
30	0.18	0.0	0.0

Tabel 1: De afwijking in pixels in functie van de intensiteitsthreshold

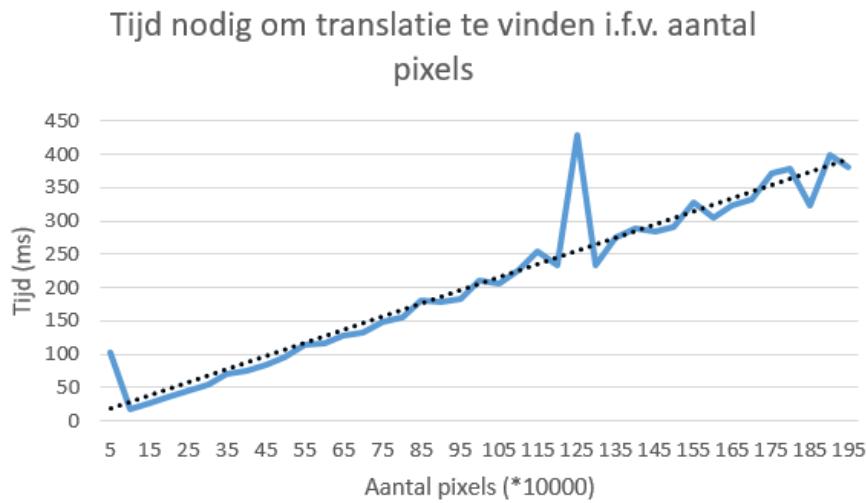
4.2.5 Snelheid

Ook de snelheid van het algoritme is belangrijk. In figuur 26 is te zien dat de tijd lichtjes stijgt met een hogere threshold, de zekerheid van de koppels heeft geen invloed op de snelheid van het algoritme. Figuur 27 laat aan de andere kant zien dat het aantal pixels

dat wordt onderzocht een veel grotere invloed heeft op de tijd. Al stijgt de tijd lineair (zie de zwarte lineaire trendlijn), toch is het belangrijk in een real-time trackingsysteem om het zo snel mogelijk te laten lopen. Enkel zo zal het bewegen in de drie dimensionale omgeving vloeihand aanvoelen voor de gebruiker.



Figuur 26: Tijd nodig om de translatie te vinden in functie van de threshold.



Figuur 27: Tijd nodig om de translatie te vinden in functie van aantal pixels

4.3 Vinden van transformatie

AUTEUR: VAN STEENBERGEN SEPPE

Om een volledige transformatie te vinden, moet de rotatie van de sensoren gecombineerd worden met de translatie van de camera. Echter bij een pure rotatie, lijkt het voor de camera dat er ook een translatie is opgetreden. De juiste translatie is dus niet degene die *ORB* vindt. Deze translatie moet eerst nog gecompenseerd worden met informatie over de rotatie van de sensoren. Dit gebeurt door de omgekeerde rotatie, gevonden door de sensoren,

toe te passen op de gevonden translatie. Aangezien er enkel een rotatie gevonden wordt op een driedimensionaal punt en de translatie tweedimensionaal is, krijgt de translatie een fictieve diepte. Een volledig juiste transformatiematrix wordt hier niet bekomen, doordat het bewegen van het master device intuïtief is en er enkel een verplaatsing wordt nagebootst in de 3D-scène, is dit geen probleem. In de applicatie wordt de keuze voor translatie en/of rotatie aan de gebruiker gelaten. Deze kan kiezen tussen ofwel één van beide ofwel beiden tesaamen.

5 New Features

5.1 User interface

AUTEUR: VAN DEN BOSCH BERT

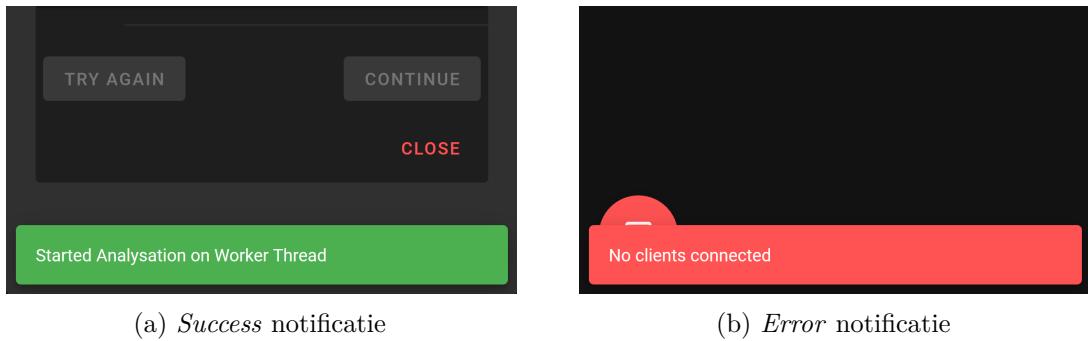
Aan de hand van feedback van onervaren gebruikers die de applicatie voor het eerst gebruiken en de professor tijdens de demo, werden grote delen van de applicatie aangepast, geherstructureerd en geoptimaliseerd.

Automatisering Hoe minder user interactie er nodig is voor hetzelfde resultaat, hoe beter. Er werden zo veel mogelijk knoppen weggewerkt zodat de UI zo duidelijk en overzichtelijk mogelijk blijft zonder features te laten vallen. Dit gebeurde in volgende elementen:

- Room sluiten en starten, zie figuur 29a.
- Start panelen van *Screen Detection* automatiseren
- Minimaal aantal buttons voor *Image Projection*

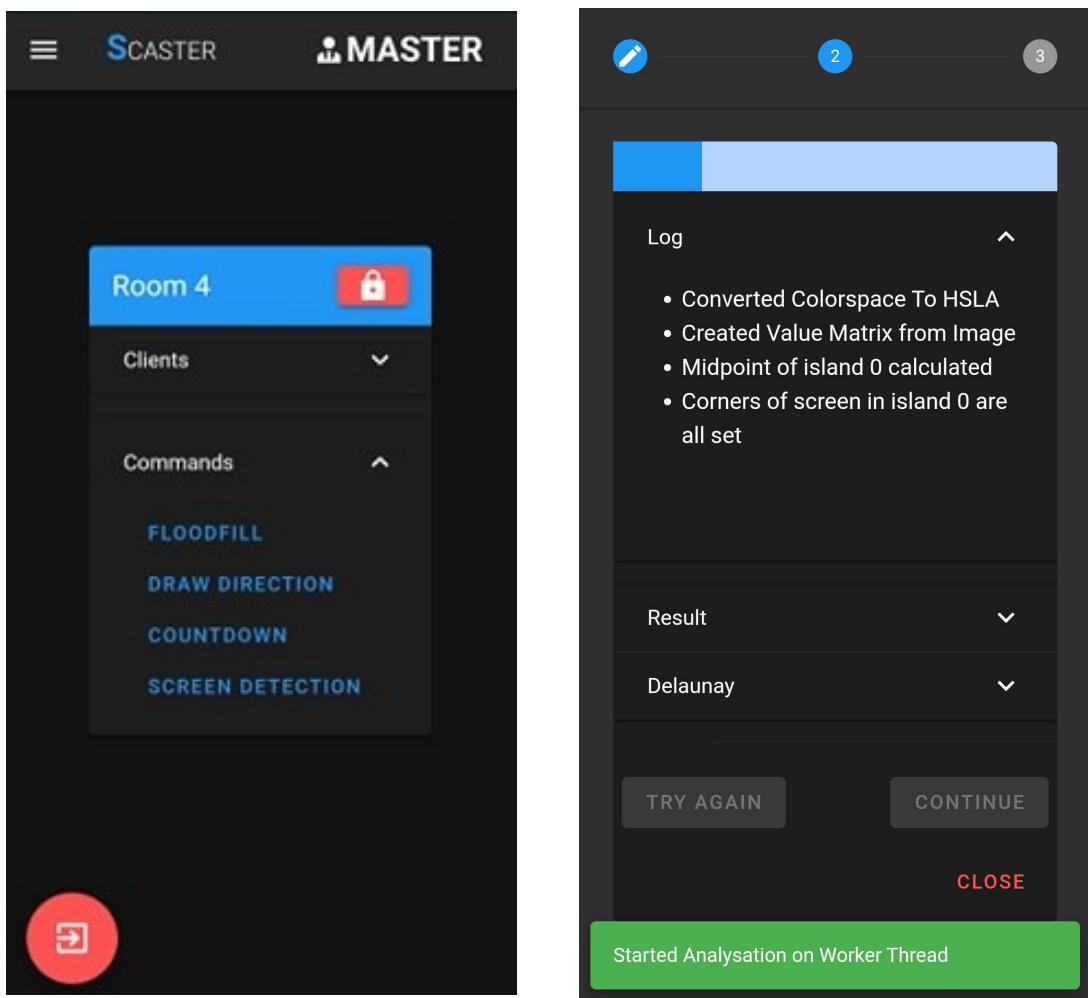
User Experience Om een gebruiker die geen voorkennis heeft met de applicatie te helpen sturen door alle functies werden veel buttons correcter hernoemd en grote stukken van de layout geherstructureerd om verzorgd en duidelijk over te komen. Een kleine, maar zeker *Quality Of Life* verbeterende feature is een voorbeeld weergave van hoe de geselecteerde afbeelding of video geprojecteerd zal worden op de verschillende slave devices. Dit geeft een goed beeld of enerzijds de devices correct zijn gedetecteerd en anderzijds ook een referentie om de uiteindelijke projectie mee te vergelijken en te controleren op fouten.

Notificatie Systeem Om de gebruiker meer informatie en updates te geven over wat er gebeurt, werden notificaties toegevoegd. Op deze manier kan er in real time feedback en updates op acties van de gebruiker gegeven worden. Deze functionaliteit wordt vooral gebruikt als de gebruiker een functie van de applicatie wil gebruiken voordat aan alle voorwaarden voldaan is. Zoals bijvoorbeeld analyse zonder een afbeelding te maken of selecteren, de verdere functionaliteiten van een *room* gebruiken zonder dat er clients aan de master zijn verbonden etc.



Figuur 28: Twee voorbeelden van een notificatie.

Result Display De pagina die tijdens het analyseren feedback geeft werd volledig vernieuwd. Niet enkel werden vorige punten toegepast, maar er is nu een extra communicatie kanaal toegevoegd waar een log en een progress bar van de analyse weergegeven wordt. Dit geeft de gebruiker een duidelijk overzicht hoe ver de executie staat, wat er op dat moment berekend wordt en waar het eventueel fout gaat tijdens de uitvoering. Zie onderstaande figuur 29b.



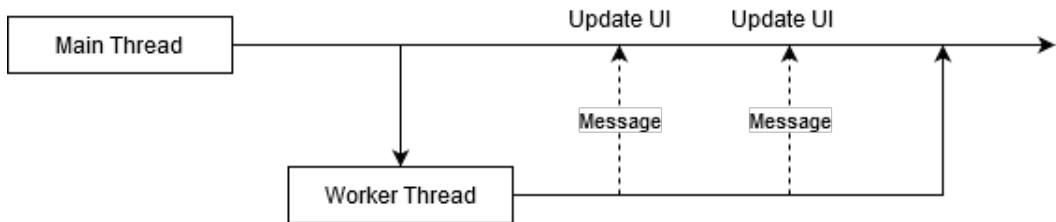
Figuur 29: Twee voorbeelden verbetering UI/UX.

5.2 Multithreading

AUTEUR: VAN DEN BOSCH BERT

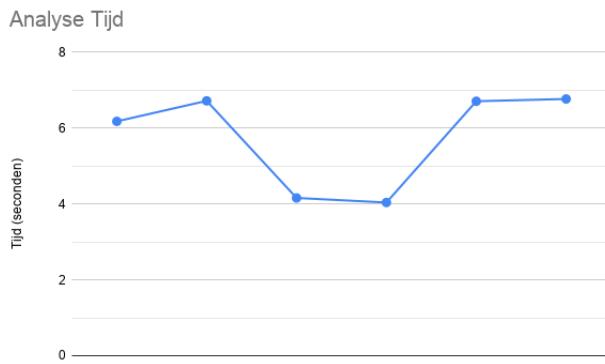
Om de volledige web applicatie responsive te houden, tijdens de berekeningen en analyse van de afbeelding, wordt deze taak afgeschoven naar een *worker thread* [27]. Deze *workers* kunnen javascript uitvoeren in parallel met de main thread zonder dat de pagina moet wachten op de executie en het resultaat van dat script. Door de image manipulation scripts door te geven aan een *worker*, heeft de main thread de mogelijkheid om de UI interactief te houden en te blijven updaten, zie 5.1. Via message passing wordt de main thread op de hoogte gehouden van het werk dat de *worker* al geleverd heeft, of waar het eventueel mis loopt zonder de web pagina te blokkeren.

Deze aanpak geeft geen performance voordeel van multithreaden omdat de berekeningen zelf nog lineair worden uitgevoerd. Het voordeel is dat de applicatie vloeiend blijft werken tijdens een lange rekentijd, zie 5.2, zeker als het een grote afbeelding is en de berekeningen op een mobile device gebeuren.



Figuur 30: Webworkers parallel aan de Main thread

Pagina Performance Zoals eerder vermeld is er geen verbetering in de uitvoeringsnelheid van de analyse bij het gebruik van één *worker*, maar blijft de pagina tijdens de uitvoering wel responsief. In figuur 31 worden enkele rekentijden van de analyse weergegeven. Als deze executie op de main webpagina uitgevoerd wordt, zal deze pagina voor deze tijden ook volledig niet reageren op user input omdat deze aan het wachten is op de executie van het javascript.



Figuur 31: Tijden van image analysis algoritmes op verschillende slave opstellingen uitgevoerd op een laptop ²

²Laptop met een Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz processor

5.3 3D-scène

AUTEUR: VANBEVEREN DIRK

Voor de 3D-scène wordt er een library *three.js* [28] gebruikt om dit weer te geven in de browser. Deze library laat toe om objecten te plaatsen in een 3D-omgeving en ook een model in te laden. Met de tracking data, zie 4, wordt de camera in de scene geroteerd en getransleerd. Naast de transformatie van de camera heeft de canvas nog een transformatie matrix die het juiste deel van de 3D-scène weergeeft.



Figuur 32: 3D-scène

5.4 Game

AUTEUR: VANBEVEREN DIRK

In dit project is er een simpele multiplayer game geïntegreerd. In dit spel is het mogelijk in de wereld te bewegen en kogels af te vuren naar elkaar. Elke speler heeft daarbij een levensbalk, deze zal verlagen wanneer de speler geraakt wordt door een kogel. Spelers kunnen ook punten verdienen wanneer het hun lukt om een andere speler uit te schakelen.



Figuur 33: Het spel

Display Om dit te implementeren wordt het spel getekend op de canvas. De client krijgt alle relevante data van de server per *game state update* om het spel te kunnen weergeven.

De *game state update* zal zo de locaties van de spelers en de kogels doorspelen alsook de gegevens over de levensbalk en de namen van de spelers.

Controller Een aparte pagina is gewijd aan de controller. Deze bestaat uit twee joysticks op het scherm die geïmplementeerd worden door de library *NippleJS*[29]. Eén voor de positie te bepalen en een ander om in een richting te schieten. De data om te bewegen en te schieten wordt continu doorgestuurd zolang de gebruiker de joysticks aanraakt. Deze data bevat enkel de richting van beweging en van het schieten. Dus niet de positie van de speler, dit berekend de server zelf.



Figuur 34: Spel controller joysticks

Game state De volledige game logica wordt berekend op de server. Bij elke update wordt de data naar de clients doorgestuurd die het spel moeten weergeven. Bij deze actie worden er verschillende aspecten geüpdatet. Bijvoorbeeld de positie van de speler met de gegeven controller input, positie van de kogels, toevoegen/verwijderen van een speler en aanmaken/verwijderen van kogels. Het voordeel van dit allemaal op de server te berekenen is dat de spelklok dan uniform is voor alle aangesloten spelers. Als de berekening van de spelerspositie op de controller pagina berekend zou worden zou die sneller of trager kunnen updaten dan anderen en zou de positie ook gemanipuleerd worden door de speler om een voordeel te krijgen.

6 Besluit

AUTEUR: SAUVILLERS TOON

Uit de *Scientific Tests* is gebleken dat de gebruikte kleuren uit het eerste semester voor detectie en identificatie niet de beste kleuren waren. Door de detectie naar cyaan, geel en roos te veranderen worden de schermen nu veel beter gedetecteerd. De identificatie is van HSL naar RGB gegaan om wit en zwart te detecteren, dit resulteert in een bevordering van de accuraatheid tot wel 500%. Ook de synchronisatie is uitvoerig getest geweest. Deze resultaten wezen erop dat de geïmplementeerde synchronisatie niet voldoende was om op alle schermen samen een video af te spelen. Door een nieuwe implementatie loopt de video nu mooi gesynchroniseerd en past het zichzelf realtime aan.

Ook een nieuwe feature, tracking, is onderzocht en toegevoegd aan de applicatie. Met behulp van de sensoren en van *keypoint tracking* kan het master apparaat gevuld worden. Hiermee kan de gebruiker zich in een driedimensionale omgeving wanen.

Het project werd nog op verschillende andere vlakken verbeterd. Zo is de userinterface getest geweest bij onervaren gebruikers. De pijnpunten en niet-intuïtieve opties zijn verwijderd of aangepast. Dit verhoogt het gebruiksgemak. Ook kan de gebruiker het detectiealgoritme volgen. Voor meer ervaren gebruikers kan dit zo een indicatie geven van wat er is fout-gelopen. Zo kan deze eventueel een nieuwe foto nemen of de opstelling aanpassen. Door middel van *multithreading* is deze opvolging mogelijk en zal het detectiealgoritme zonder onderbrekingen kunnen worden uitgevoerd.

Een heel jaar uitgebreid werken aan een project met mensen die je niet kent. Niet alleen de opdracht maar ook samenwerking als uitdaging. "Als team zijn we trots op wat we bereikt hebben: een werkende applicatie die uit slechts één foto schermen kan herkennen, identificeren en daarop verschillende elementen kan projecteren."

Referenties

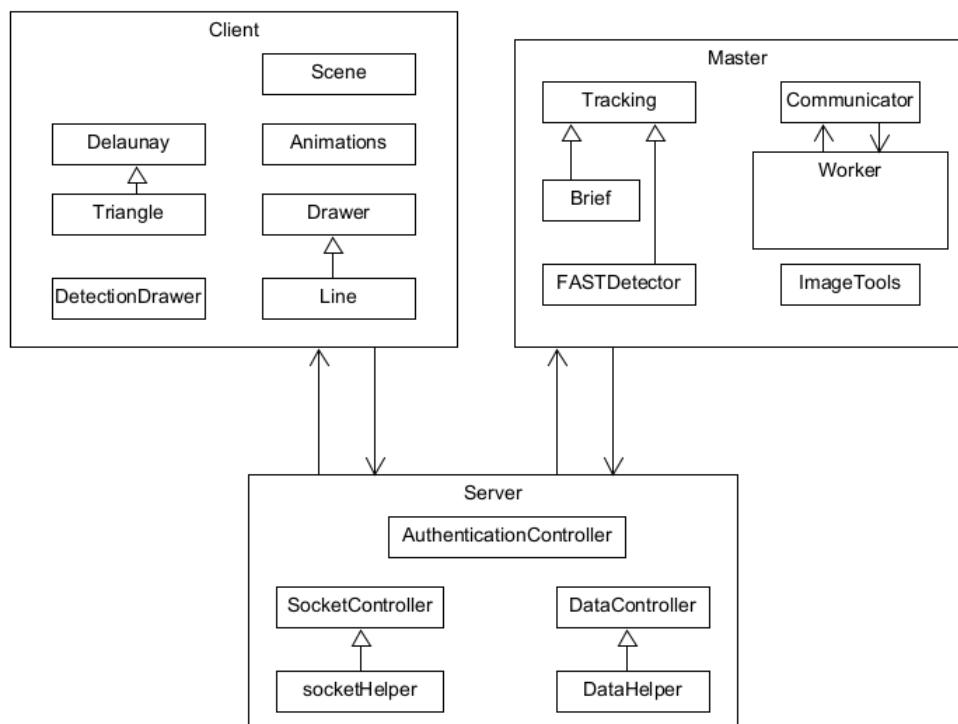
- [1] Flood fill. https://en.wikipedia.org/wiki/Flood_fill. geraadpleegd oktober 2019.
- [2] T. Huang, G. Yang, and G. Tang. A fast two-dimensional median filtering algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(1):13–18, 1979.
- [3] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. Gaussian smoothing. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm>, 2003.
- [4] Geometric mean filter. https://en.wikipedia.org/wiki/Geometric_mean_filter.
- [5] Yan-Li Liu, Jin Wang, Xi Chen, Yan-Wen Guo, and Qun-Sheng Peng. A robust and fast non-local means algorithm for image denoising. *Journal of Computer Science and Technology*, 23(2):270–279, 2008.
- [6] Ntp protocol. https://en.wikipedia.org/wiki/Network_Time_Protocol. geraadpleegd 2020.
- [7] Ping. [https://en.wikipedia.org/wiki/Ping_\(networking_utility\)](https://en.wikipedia.org/wiki/Ping_(networking_utility)). geraadpleegd 2020.
- [8] Manisha Priyadarshini. Which Sensors Do I Have In My Smartphone? How Do They Work? <https://fossbytes.com/which-smartphone-sensors-how-work/>, 2018.
- [9] Alexander Shalamov and Mikhail Pozdnyakov. Sensors For The Web!, Jan 2019.
- [10] MDN contributors. Sensor apis, Feb 2020.
- [11] Kenneth Rohde Christiansen and Anssi Kostiainen. Orientation Sensor. <https://www.w3.org/TR/orientation-sensor/>, 2019.
- [12] Moti Ben-Ari. A tutorial on euler angles and quaternions. *Department of Science Teaching Weizmann Institute of Science*, 2014.
- [13] Avoid gimbal lock for rotation/direction maya manipulators. <https://around-the-corner.typepad.com/adn/2012/08/avoid-gimbal-lock-for-rotationdirection-maya-manipulators.html>.
- [14] fivasim. AndroSensor for Android. <http://www.fivasim.com/androsensor.html>, 2015.
- [15] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. *2011 International Conference on Computer Vision*, 2011.
- [16] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. *Tenth IEEE International Conference on Computer Vision (ICCV05) Volume 1*, 2005.
- [17] Software/ fast corner detection. <https://computervisiononline.com/software/1105138582>.
- [18] Jurg Nievergelt and Claus Hinrichs. *Algorithms and data structures - applications to graphics and geometry*. Textbook Equity Editions, 2014.

- [19] Penny Rheingans. Finding pixels that lie on a circle. <https://www.csee.umbc.edu/~rheingan/435/pages/res/gen-3.Circles-single-page-0.html>, Sep 2003.
- [20] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. *Computer Vision – ECCV 2010 Lecture Notes in Computer Science*, page 778–792, 2010.
- [21] Nitya Raut. What is Hamming Distance? <https://www.tutorialspoint.com/what-is-hamming-distance>, 2018.
- [22] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
- [23] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.
- [24] Abid K. Alexander Mordvintsev. BRIEF (Binary Robust Independent Elementary Features). https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_brief/py_brief.html, 2013.
- [25] trackingjs. Source: features/Brief.js. <https://trackingjs.com/api/Brief.js.html>, 2014.
- [26] module' object has no attribute 'drawmatches' opencv python. <https://stackoverflow.com/questions/20259025/module-object-has-no-attribute-drawmatches-opencv-python/26227854#26227854>.
- [27] MDN contributors. Using Web Workers. https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers. Geraadpleegd op: maart 2020.
- [28] Threejs - library for 3d visuals in javascript. <https://threejs.org/>.
- [29] yoannmoinet. Nipplejs - a vanilla virtual joystick for touch capable interfaces. <https://github.com/yoannmoinet/nipplejs>.

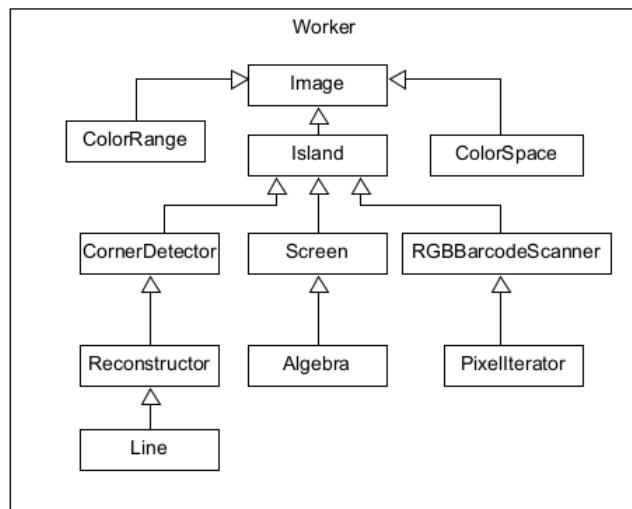
A Overzicht algoritme

AUTEURS: VANBEVEREN DIRK EN DEBEUF MARTIJN

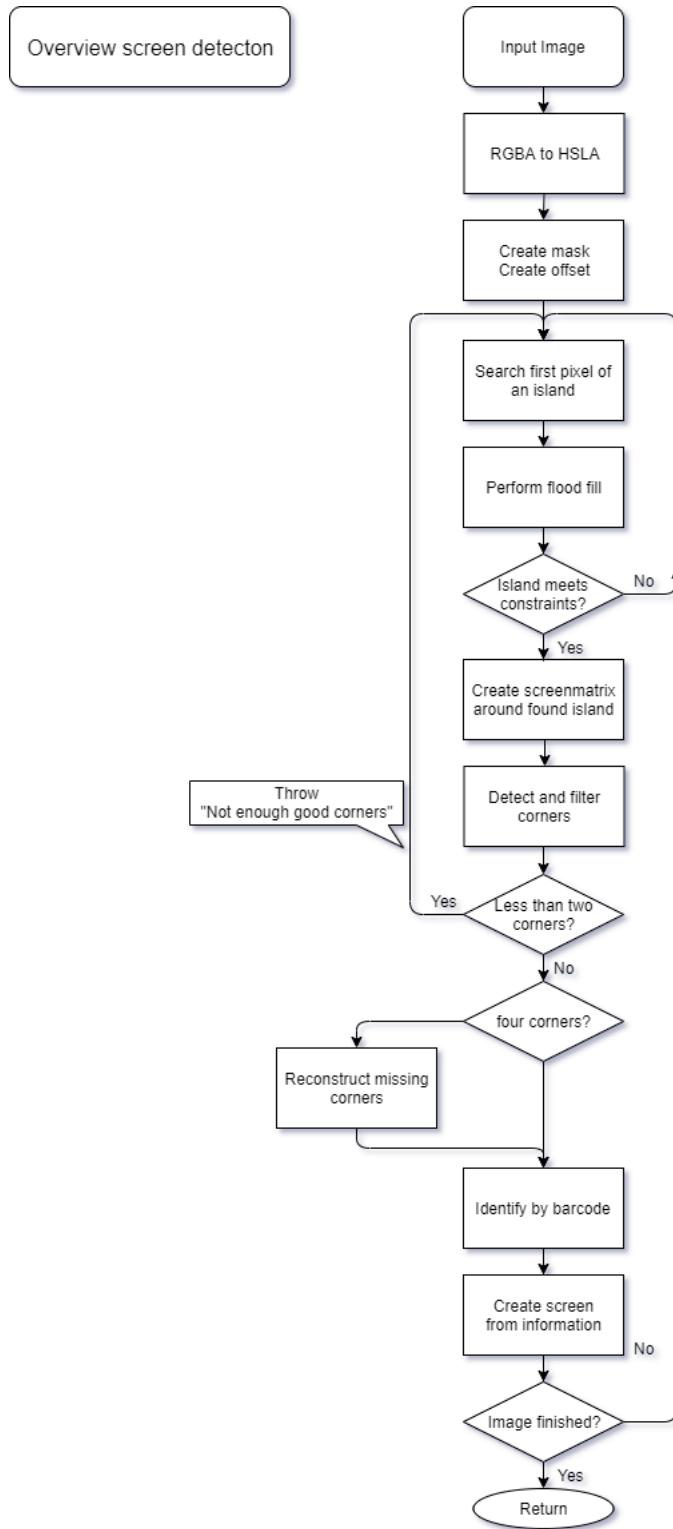
A.1 Algemeen



Figuur A.1: Architectuur overzicht

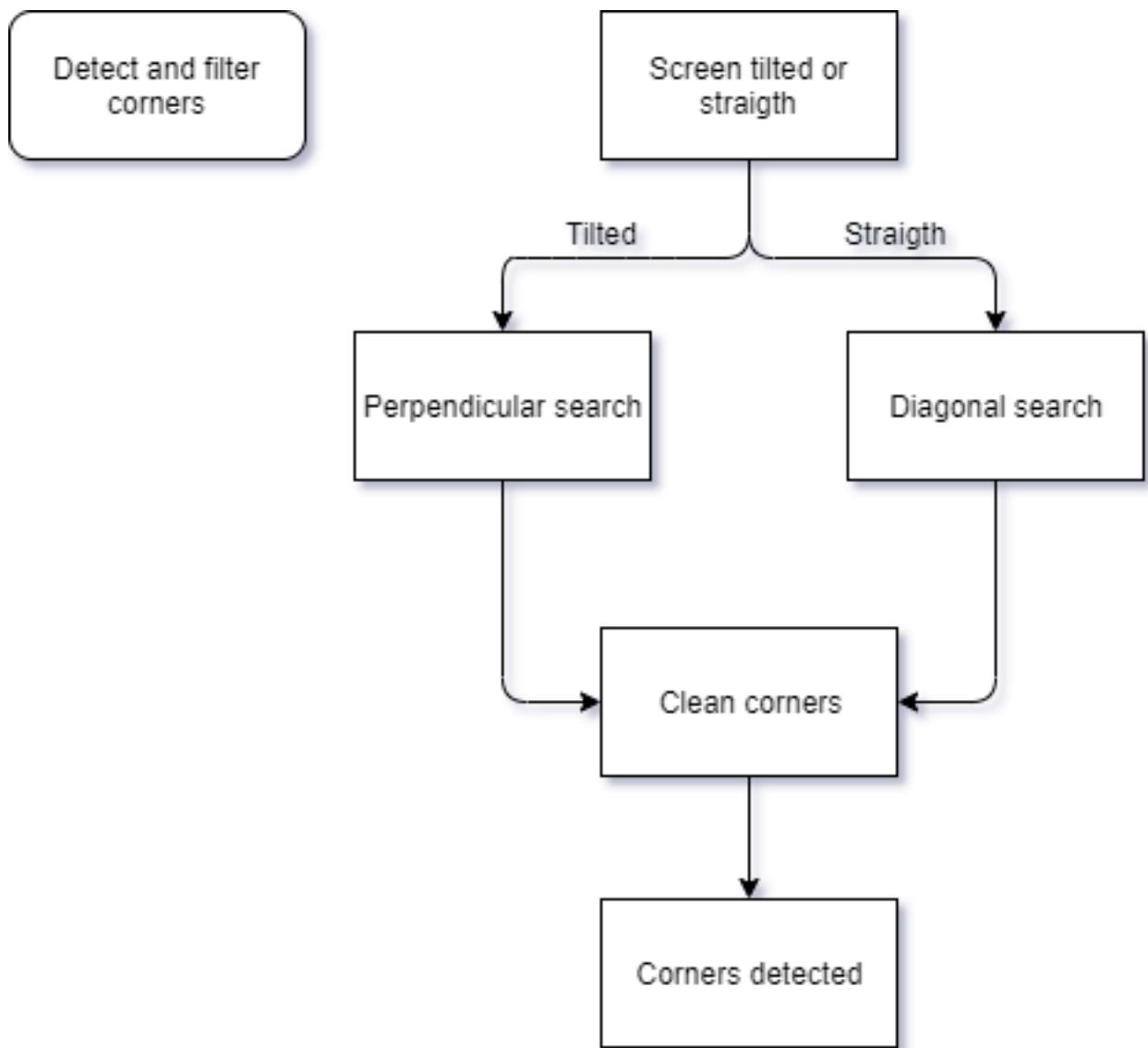


Figuur A.2: Worker overzicht



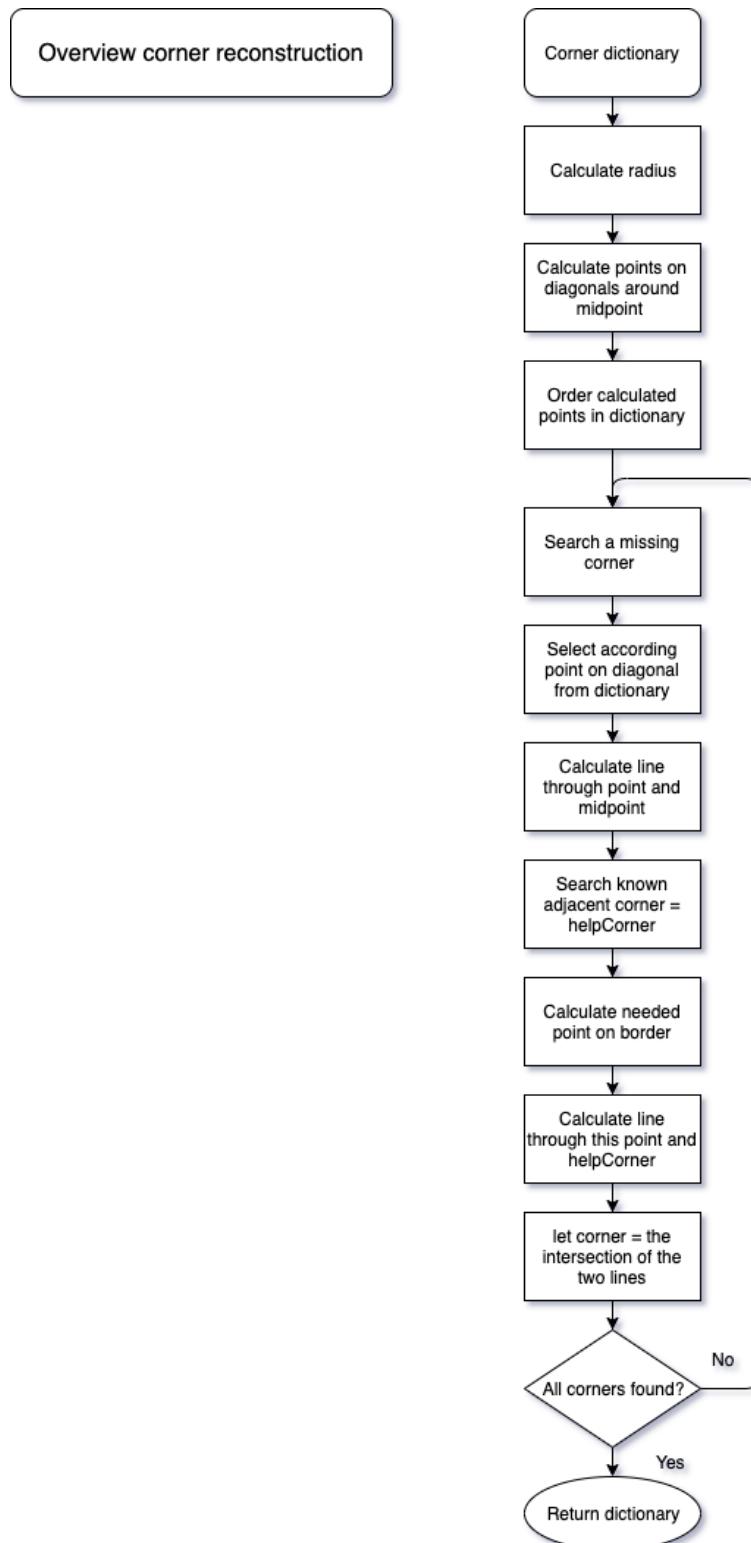
Figuur A.3: Overzicht detectie algoritme

A.2 Hoekdetectie en filtering



Figuur A.4: Overzicht hoekdetectie en filtering

A.3 Reconstructie

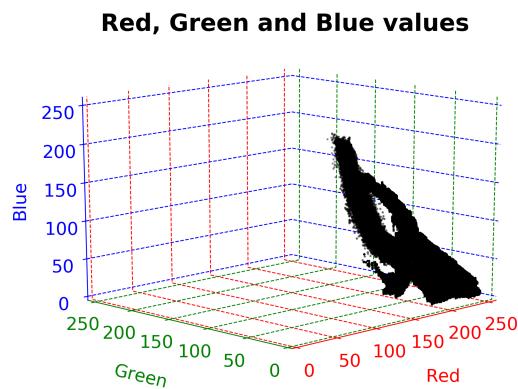


Figuur A.5: Overzicht reconstructie

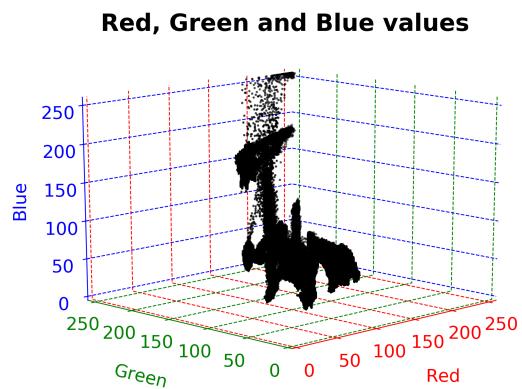
B Kleuren plots

AUTEUR: VAN STEENBERGEN SEPPE

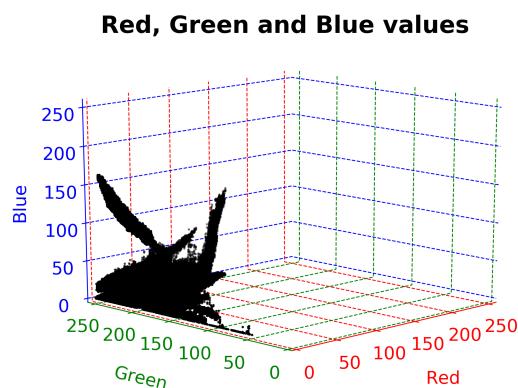
B.1 RGB plots



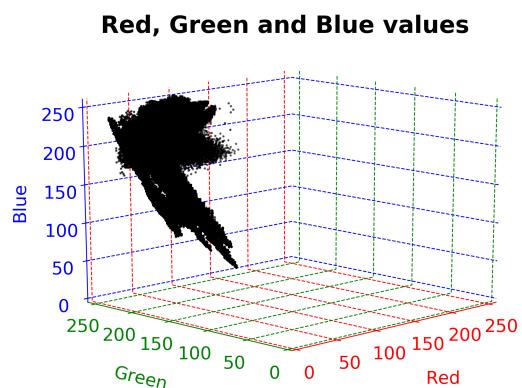
Figuur B.6: RGB plot voor de kleur rood.



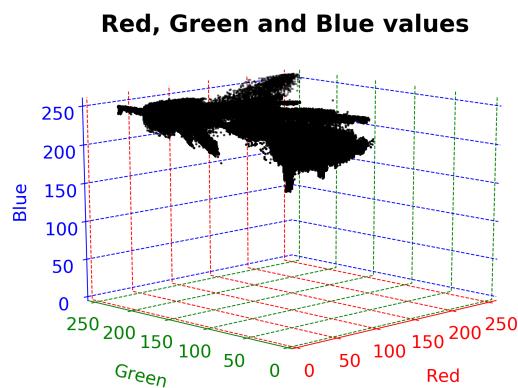
Figuur B.7: RGB plot voor de kleur geel.



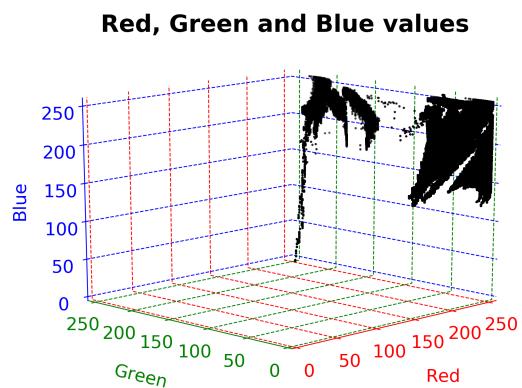
Figuur B.8: RGB plot voor de kleur groen.



Figuur B.9: RGB plot voor de kleur cyaan.

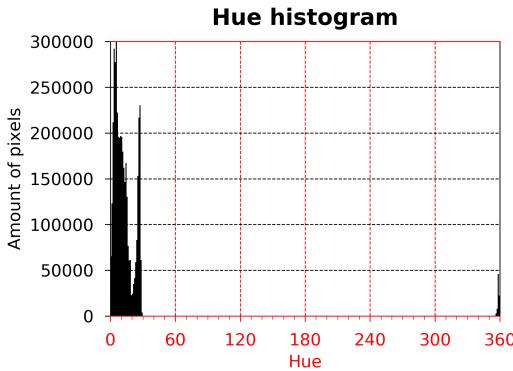


Figuur B.10: RGB plot voor de kleur blauw.

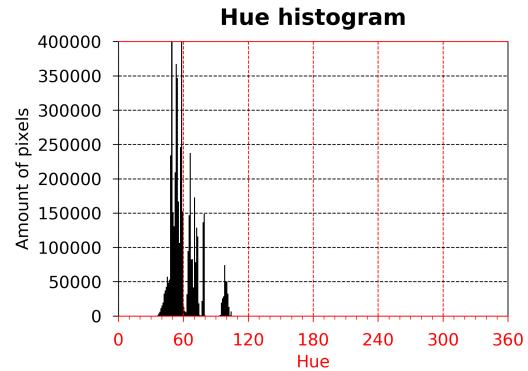


Figuur B.11: RGB plot voor de kleur magenta.

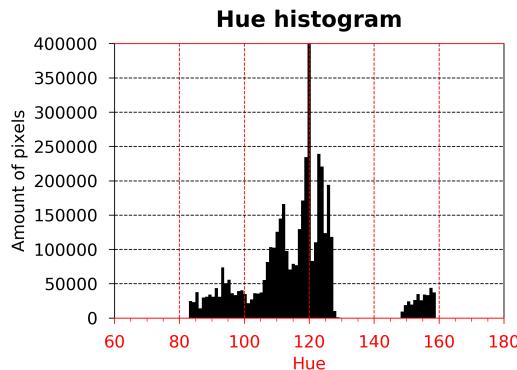
B.2 Histogrammen



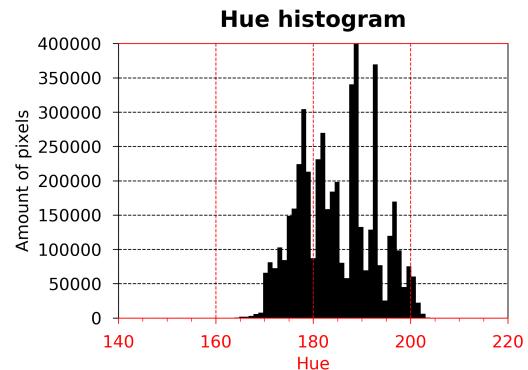
Figuur B.12: Histogram van de tint in functie van het aantal waargenomen pixels voor de kleur rood.



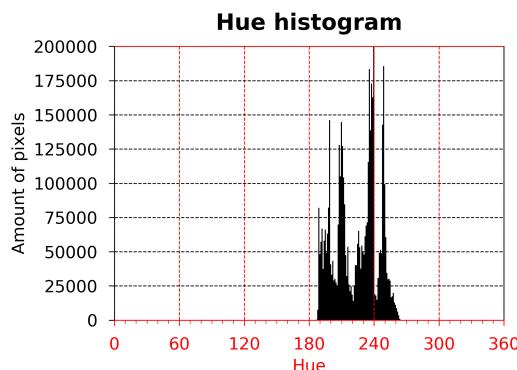
Figuur B.13: Histogram van de tint in functie van het aantal waargenomen pixels voor de kleur geel.



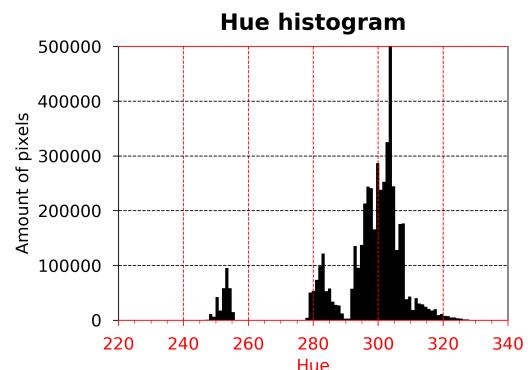
Figuur B.14: Histogram van de tint in functie van het aantal waargenomen pixels voor de kleur groen.



Figuur B.15: Histogram van de tint in functie van het aantal waargenomen pixels voor de kleur cyaan.



Figuur B.16: Histogram van de tint in functie van het aantal waargenomen pixels voor de kleur blauw.

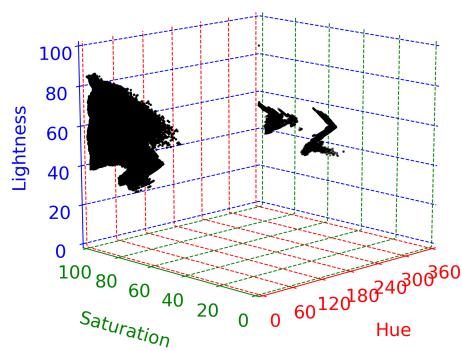


Figuur B.17: Histogram van de tint in functie van het aantal waargenomen pixels voor de kleur magenta.

B.3 HSL plots

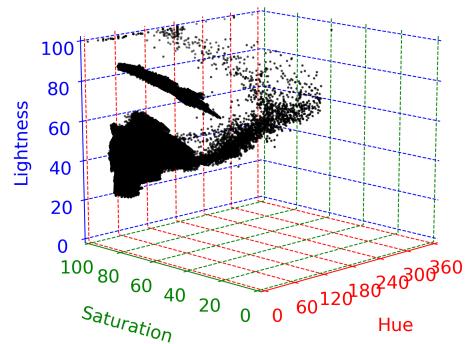
B.3.1 3D

Hue, Saturation and Lightness values



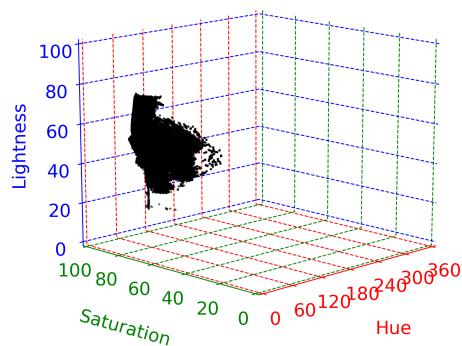
Figuur B.18: HSL plot voor de kleur rood.

Hue, Saturation and Lightness values



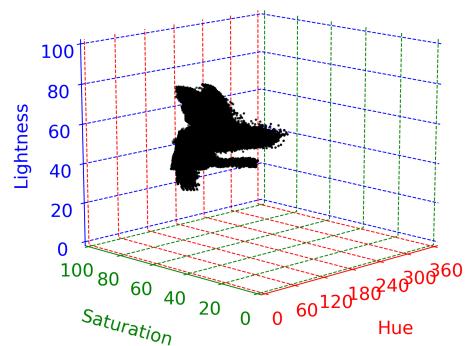
Figuur B.19: HSL plot voor de kleur geel.

Hue, Saturation and Lightness values



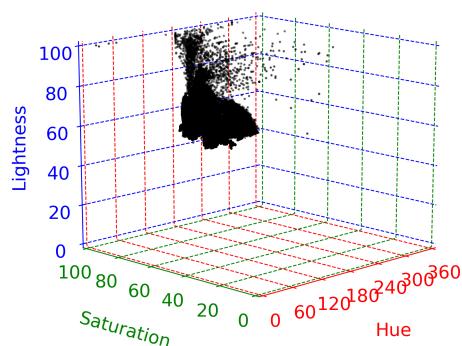
Figuur B.20: HSL plot voor de kleur groen.

Hue, Saturation and Lightness values



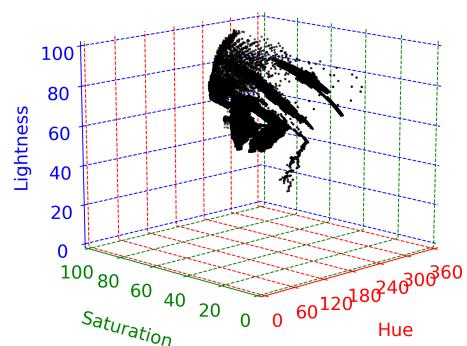
Figuur B.21: HSL plot voor de kleur cyaan.

Hue, Saturation and Lightness values



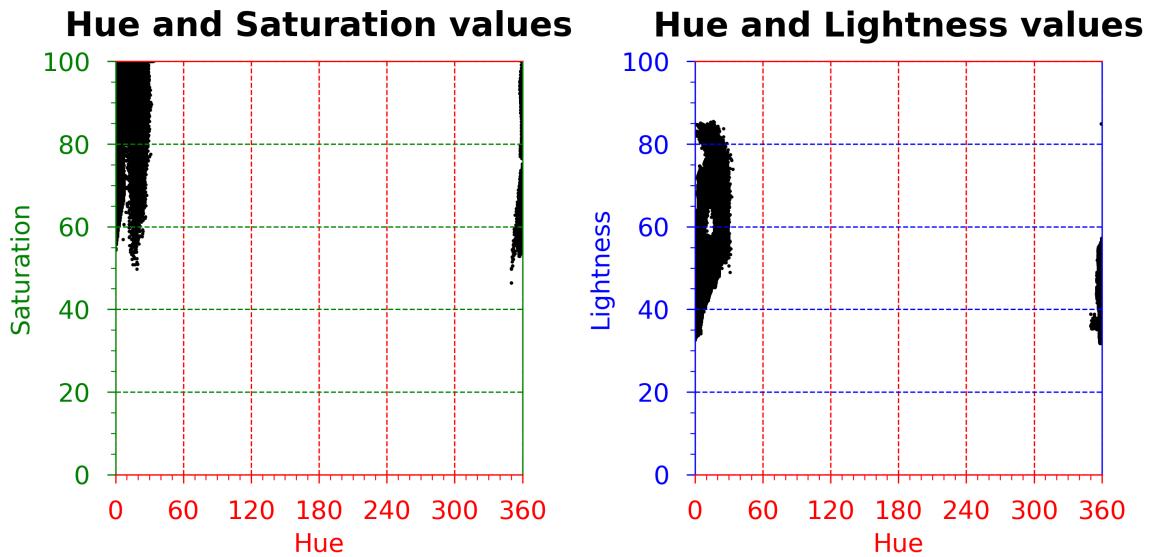
Figuur B.22: HSL plot voor de kleur blauw.

Hue, Saturation and Lightness values

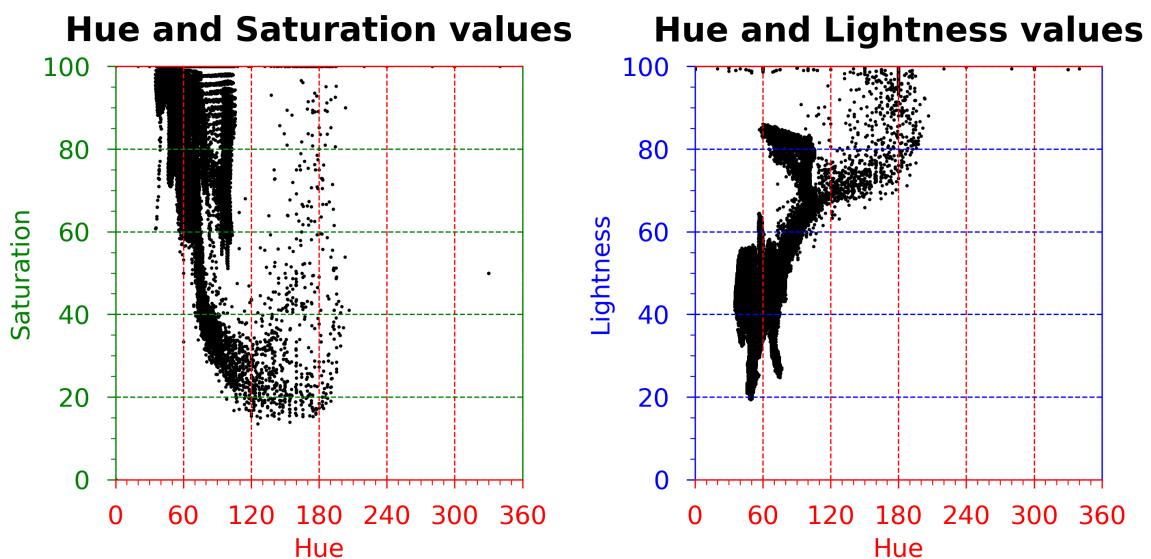


Figuur B.23: HSL plot voor de kleur magenta.

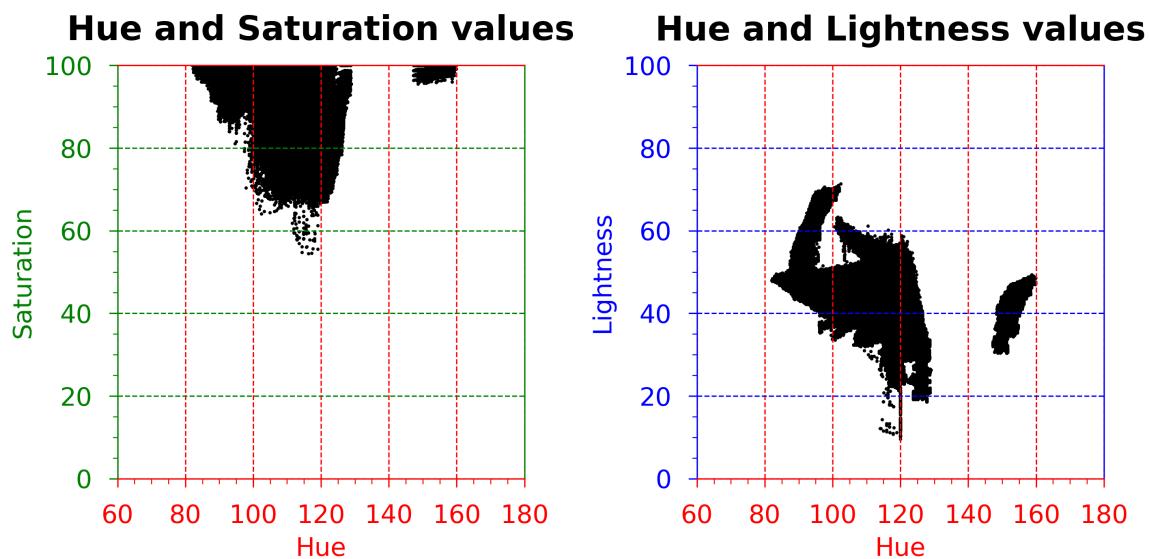
B.3.2 2D



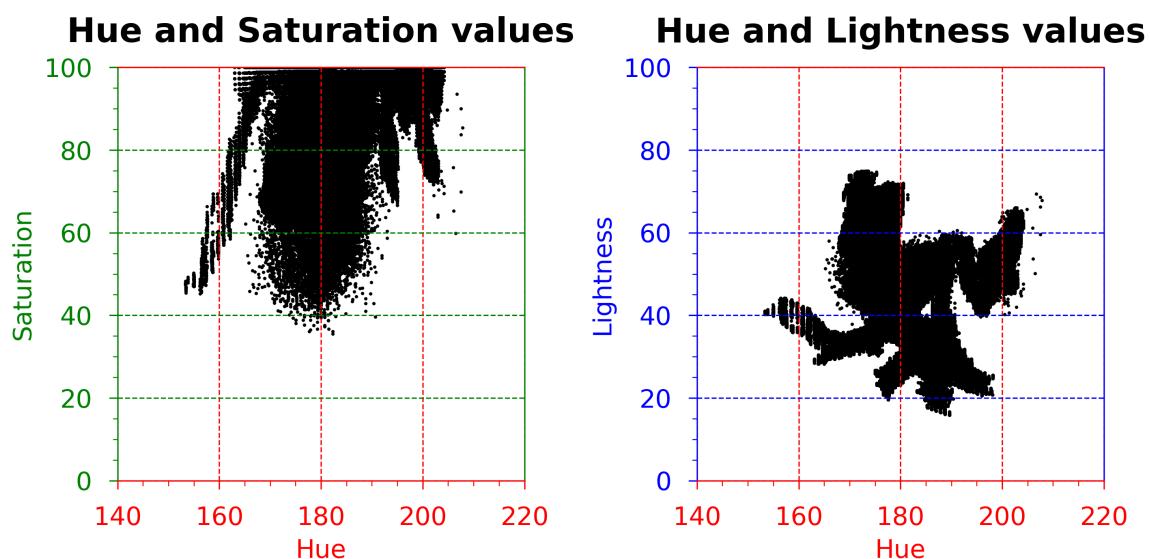
Figuur B.24: Scatter plots in functie van tint en saturatie, alsook in functie van tint en lichtheid voor de kleur rood.



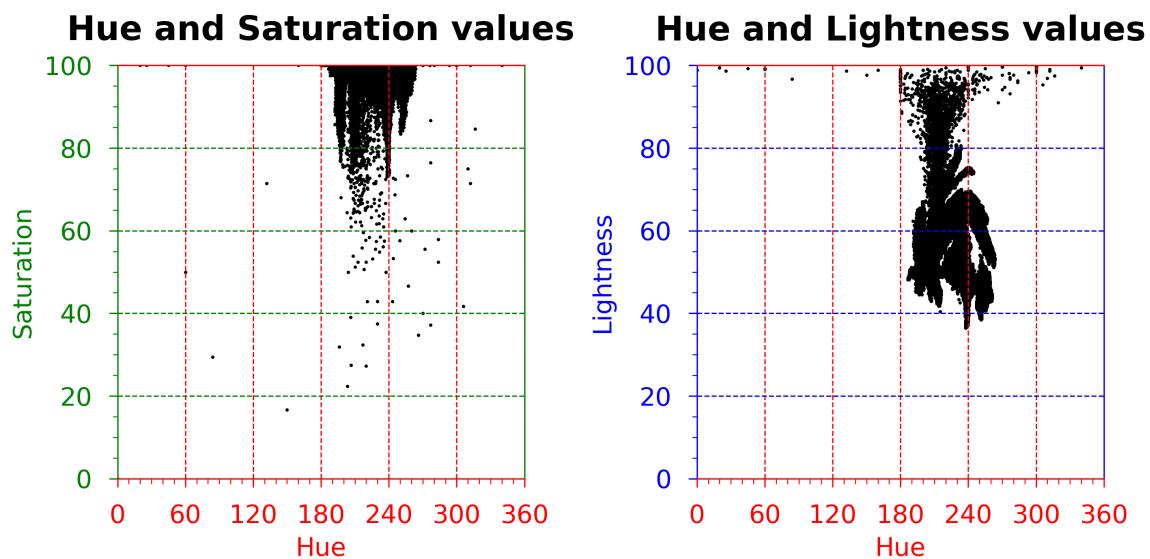
Figuur B.25: Scatter plots in functie van tint en saturatie, alsook in functie van tint en lichtheid voor de kleur geel.



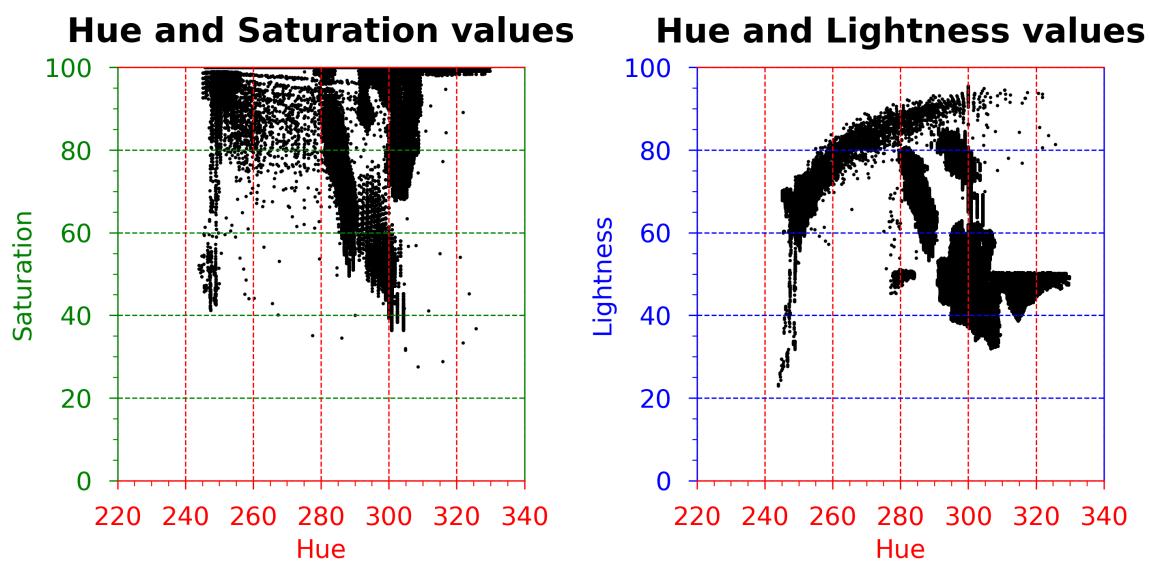
Figuur B.26: Scatter plots in functie van tint en saturatie, alsook in functie van tint en lichtheid voor de kleur groen.



Figuur B.27: Scatter plots in functie van tint en saturatie, alsook in functie van tint en lichtheid voor de kleur cyaan.



Figuur B.28: Scatter plots in functie van tint en saturatie, alsook in functie van tint en lichtheid voor de kleur blauw.



Figuur B.29: Scatter plots in functie van tint en saturatie, alsook in functie van tint en lichtheid voor de kleur magenta.

Handleiding ScreenCaster tracking

Team 12

Auteurs: Blondeel Frédéric en Sauvillers Toon



1 Inleiding

Deze handleiding geeft een stap voor stap uitleg om de tracking implementatie van de applicatie te gebruiken. De benodigde hardware en software wordt opgeliist en vervolgens zal het stappenplan uitgelegd worden.

2 Benodigde hard- en software

Om gebruik te maken van de *ScreenCaster* heeft u volgende zaken nodig:

- Smartphone met volgende specificaties:
 - Camera
 - Gyroscoop
 - Webbrowser
- Eén of meerdere schermen die een webbrowser kunnen openen.

Ondersteunende besturingssystemen Het slave gedeelte kan op elk besturingssysteem worden gedraaid. Voor alle functionaliteiten bij de master is er Android versie 8.0 of hoger nodig. Indien er geen gebruik van de tracking feature gewenst is, kan gebruik gemaakt worden van een iOS apparaat voor de master.

Ondersteunde webbrowsers Niet alle browsers worden ondersteund, ook moeten alle schermen een andere browsersessie openen. De vermelde browsers garanderen een volledige ondersteuning voor alle toepassingen binnen de applicatie. Ondersteunde webbrowsers voor een *master* apparaat:

- Chrome versie 80.0 of hoger
- Opera versie 66 of hoger

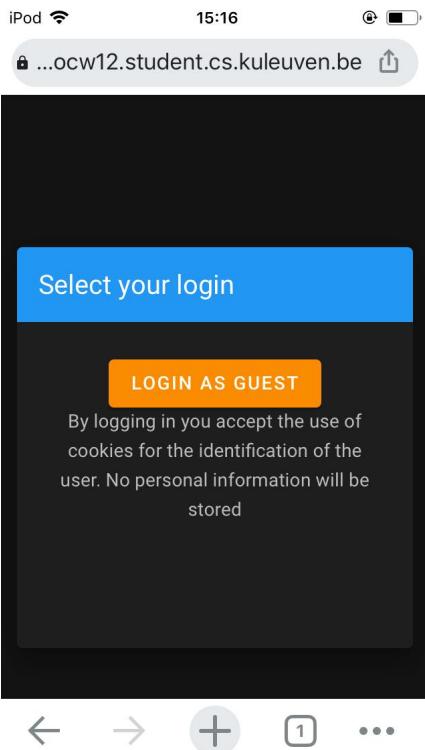
Ondersteunde webbrowsers voor een *client* apparaat:

- Chrome versie 80.0 of hoger
- Safari versie 13.1 of hoger
- Opera versie 66 of hoger
- Firefox versie 71.0 of hoger

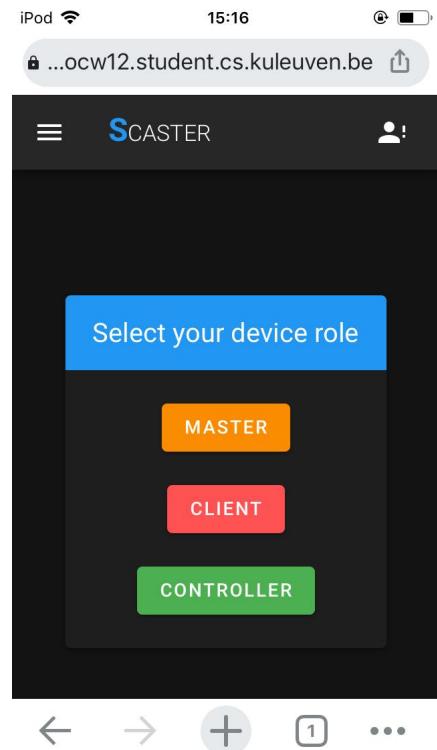
3 Praktische handleiding

3.1 Stap 1 – Verbinden en inloggen

Om naar de applicatie te gaan, volgt u volgende link: <https://penocw12.student.cs.kuleuven.be>. Een login scherm zal verschijnen met extra informatie over de cookies. Na het klikken op **Login as guest** zal u op de homepagina terecht komen met verschillende opties om een rol te kiezen voor uw apparaat.



(a) Login Scherm



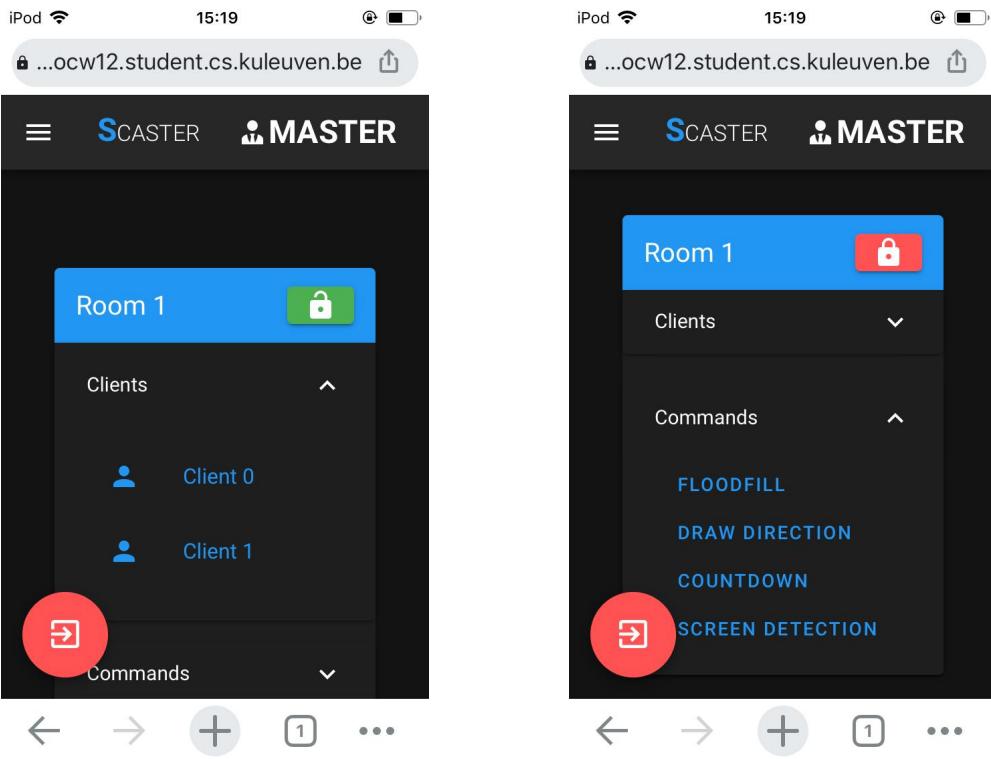
(b) Homepagina

Figuur 1: Stap 1 en 2

3.2 Stap 2 – Een rol kiezen

Men kan kiezen uit drie verschillende rollen, zoals te zien op figuur 1b. Om te kiezen drukt u simpelweg op de bijhorende knop.

- **Master:** Als master kan u één of meerdere clients hebben. Eerst wordt er automatisch een room gecreëerd, deze kan u in de blauwe box vinden (zie figuur 2). Wanneer alle clients, die u wil bedienen, verbonden zijn, kan u de room sluiten en verdergaan met de clients te besturen. Dit doet u door op **Commands** te klikken of op het groene slotje. Dit slotje wordt rood nadat de room gesloten is.

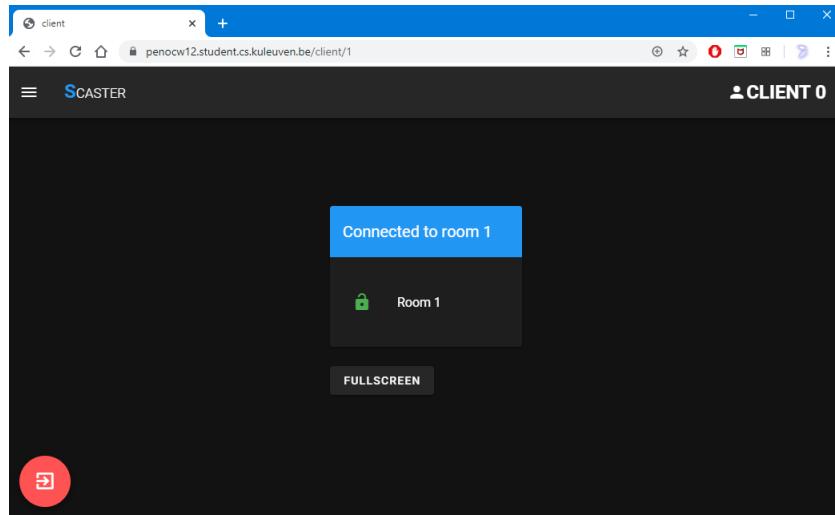


(a) Room waar alle slaves op verbinden

(b) Commands

Figuur 2: Master scherm

- **Client:** Als client kan u één master kiezen. Eerst kiest u een room die nog open staat, deze staan aangeduid met een groen slotje. Wanneer de kamer gesloten wordt door de master, kan deze het scherm gebruiken. U vindt de kamernummer waarmee u bent verbonden in de blauwe box.



Figuur 3: Client 0 is verbonden aan room 1

Het apparaat kan zich elk moment afmelden door op de rode exit knop links onderaan te klikken. U komt dan vanzelf weer op de homepage terecht.

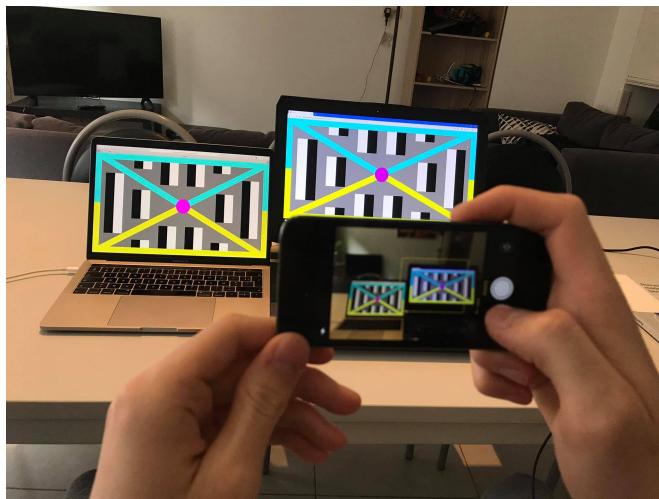
3.3 Commands

U kan kiezen voor verschillende commands, via het command *Screen Detection* komt u bij de schermdetectie uit, zoals te zien op figuur 2b. Verdere uitleg staat in sectie 4.

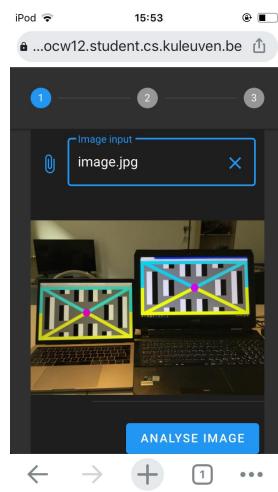
4 Screen Detection

Wanneer het pop-up scherm van de schermdetectie opent, zullen alle schermen een detectiescherm weergeven. Elk scherm heeft zijn eigen uniek detectiescherm.

4.1 Stap 1 – Foto trekken



(a) Fotograferen van de slave opstelling



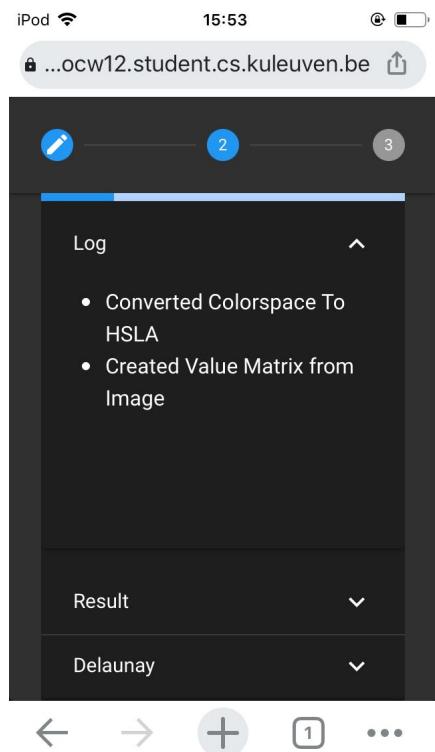
(b) Inladen afbeelding

Figuur 4: Input image

Als eerste moet een foto van de opstelling worden genomen. Dit doet u door op het paperclip icoontje te drukken. U zal een foto kunnen kiezen of een nieuwe foto trekken met behulp van de camera applicatie van uw apparaat. Wanneer de foto genomen is, kan u best nog even controleren of alle schermen er duidelijk op staan. De applicatie kan omgaan met kleine oneffenheden en reflecties op het scherm, ook mogen er hoeken ontbreken van schermen en mogen deze gekanteld staan. Van elk scherm **moet** wel het magenta middelpunt te zien zijn op de foto én twee aanliggende hoeken.

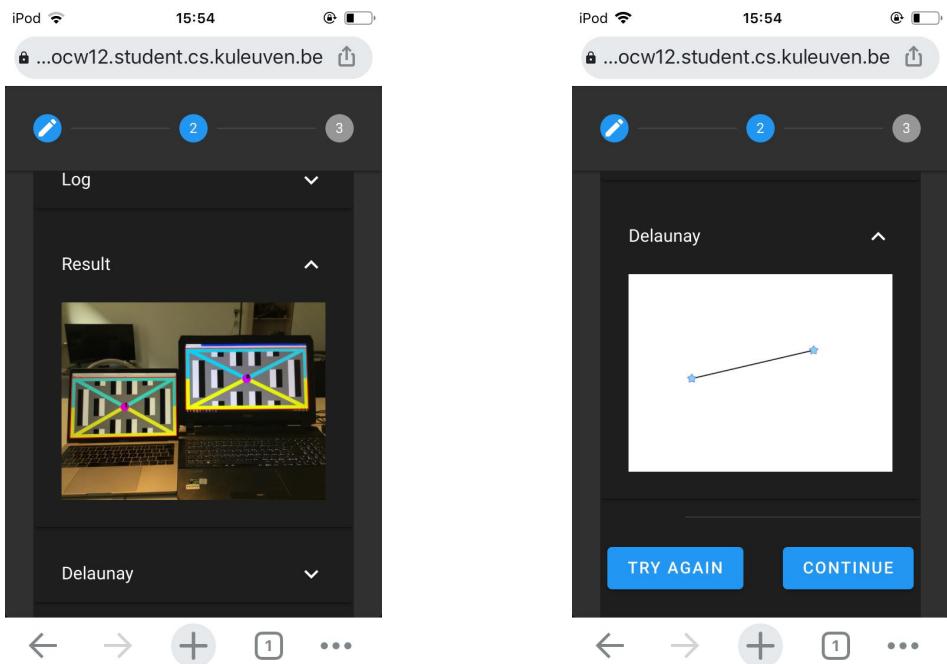
4.2 Stap 2 – Detecteren

Vervolgens drukt u op **analyse image** om de detectie van de schermen te starten. U kan in de log de verschillende stappen van het algoritme volgen.



Figuur 5: Log tijdens de analyse

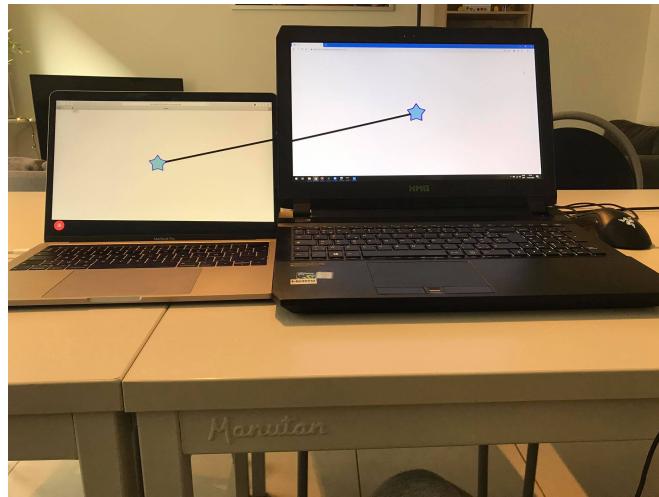
Wanneer het algoritme klaar is met het detecteren van de schermen, verschijnt de foto waarop de detectie is gebeurd. Hierop kan u alle gevonden schermen vinden, deze zijn rood gemarkeerd, het midden van het scherm is gemarkeerd met de client-id. In het tabblad **Delaunay** is er de gevonden Delaunay triangulatie te zien die ook op de schermen is weergegeven.



(a) Resultaat scherm detectie

(b) Berekende Delaunay triangulatie.

Figuur 6: Resultaten van de analyse



Figuur 7: Resultaat op de slave schermen

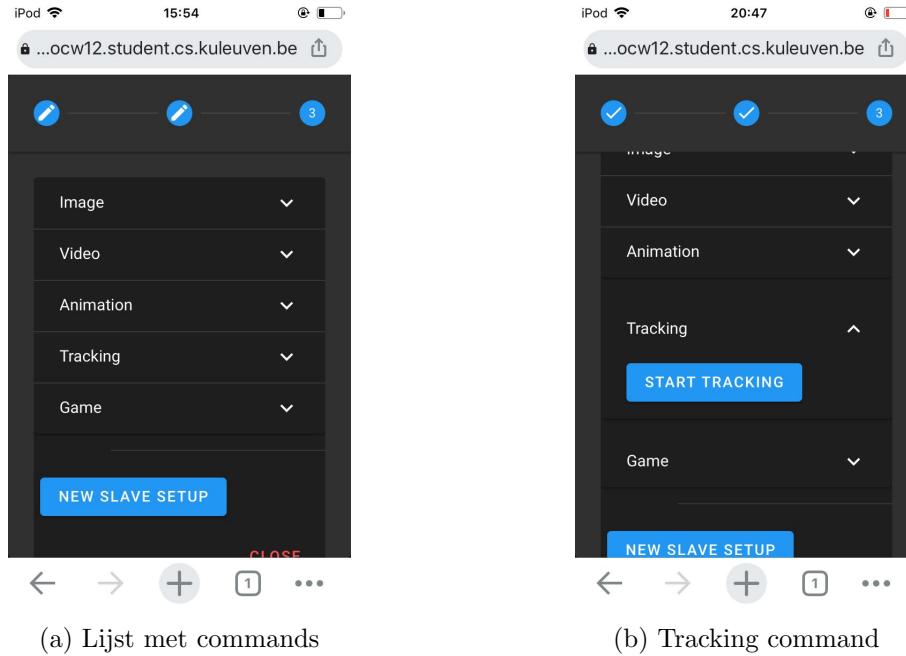
In het geval dat één of meerdere schermen niet (goed) zijn gedetecteerd kan u via **Try Again** een nieuwe foto nemen in de log zal meer informatie staan waar de detectie is fout gegaan. Via **Continue** gaat u verder naar een keuzemenu.

4.3 Stap 3 – Commando’s

Eénmaal op het keuzemenu zal een lijst met mogelijke commando’s gevonden worden. Hieruit kan **Tracking** gekozen worden.

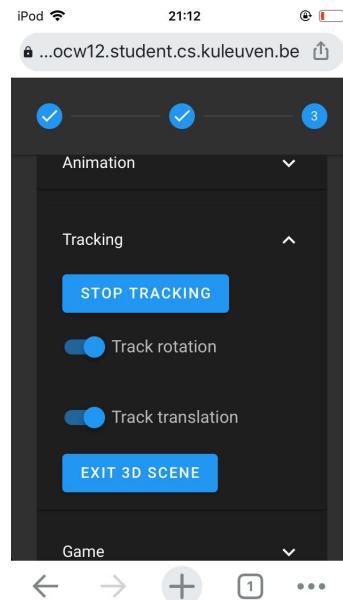
De tracking wordt uitgevoerd op eender wat er op dat moment op het scherm wordt weergegeven, zei het een image, video, animatie, game of simpelweg de Delaunay triangulatie.

Deze moeten dan wel eerst ingeladen worden indien dit gewenst is. Indien u een 3D-scène wenst kan deze later bij tracking gekozen worden.



Figuur 8: Commands

Wanneer de tracking wordt gestart aan de hand van de **Start Tracking** knop, zullen extra opties mogelijk zijn. Er kan gekozen worden tussen pure rotatie tracking (aan de hand van het apparaat zijn sensoren) of pure translatie tracking (aan de hand van de camera–input). Men kan ook beide opties kiezen voor een volledige tracking. In het tracking dropdown menu kan ook gekozen worden om een 3D–scène te tonen door op de **View 3D scene** knop te drukken.



Figuur 9: Verschillende opties voor tracking