— Verslag P&O CW 2019–2020 Taak 3 —

Department Computerwetenschappen – KU Leuven 21 oktober 2019

Team 12	
Toon Sauvillers	60h
Seppe Van Steenbergen	54h
Bert Van den Bosch	52h
Frédéric Blondeel	20h

1 Introductie

Het herkennen van schermen, deze identificeren, lokaliseren uit een foto. Dit verslag behandeld deze uitdagingen. Het zoeken van schermen begint bij een foto. Deze foto bevat een scherm met een bepaald uitzicht, er is gekozen voor een groen-blauwe rand waarop gefilterd kan worden. Vervolgens zoekt het algoritme in de gefilterde foto naar aparte schermen en geeft aan hen een id. Met behulp van het kleurenverschil en bepaalde hoeken wordt de oriëntatie bepaald.

Het identificeren van het scherm gebeurd op dit moment nog apart met een kleurenbarcode. Deze barcode bevat 5 verschillende kleuren waardoor er 120 verschillende schermen geïdentificeerd kunnen worden. In de volgende weken worden deze twee, lokaliseren en identificeren, bij elkaar gezet.
Dit verslag behandeld de keuzes die gemaakt zijn alsook een uitleg bij de gebruikte algoritmen en hun tijdscomplexiteit. De mogelijke beperkingen worden met deze kennis geduid.

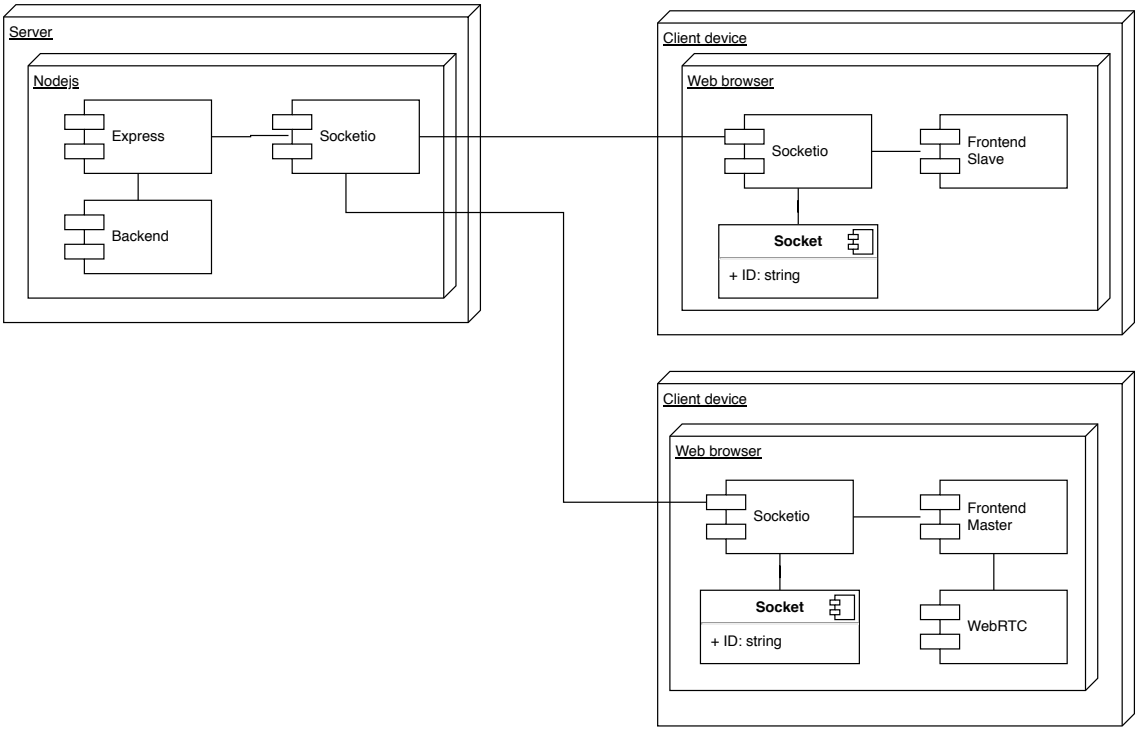
2 Design schematics and screenshots

(≈ 1 page. Depending on the need for a design overview.) *Design schematics, deployment diagrams, class diagrams, sequence diagrams, . . . Whatever you think we need to understand your design. You could add a little bit of text here but keep that under half a page. You reference the illustrations here from §3 when needed.*

All communication between clients runs through a nodejs server. We use the Socketio [**?**] library to set up a bidirectional communication channel between client and server. A deployment diagram can be found in Figure 1. Clients can take on a specific master or slave role by opening a specific webpage within the browser, i.e., the slaves will surf to `webaddress/slave`, while a client that wants to be the master loads the webpage found at `webaddress/master`.

We rely on webrtc [**?**] to collect a client-side video stream. This module offers the necessary tools for serialization, i.e., as a base64 encoded string. Serialization is required for transmission [**?**]. Server-side the backend code will process the image. In our demo we broadcast the image by sending it to all slaves. Slaves and masters can be reached by the server, as we use the concept of a namespace [**?**]. A namespace can be used to group clients with a similar role, and broadcast to them.

A socket is assigned a unique identifier by Socketio. This identifier can be used to emit messages from the server to a specific client. A list of connected clients is available through



Figuur 1: Deployment diagram of the multiscreen casting framework.

the API of Socketio [**?**].

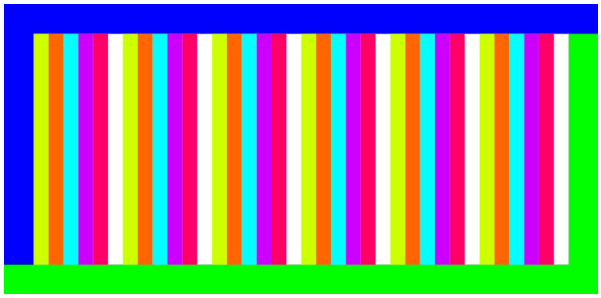
3 Algorithms

(≈ 1 page. Depending on necessity) *When you design or use an algorithm you should be able to explain how well it performs and if it satisfies the requirements. Often there will be alternatives or system parameters that need to be chosen. We want to know why you chose a specific set of parameters, or why you chose an approach over another. We supply some example text of last year.*

3.1 Scherm identificatie met barcodes

3.1.1 Algemene uitleg

De identificatie van schermen gebeurt aan de hand van een kleuren barcode. Het slave-scherm zal binnen zijn rand(gebruikt voor edge detection) een herhalend patroon van 5 kleuren en wit tonen. Aan de hand van de volgorde kan een unieke code per scherm gelezen worden en zo is elk scherm duidelijk gedefinieerd. De 5 kleuren zijn speciaal gekozen om zo ver mogelijk uit elkaar te liggen in het HSL spectrum om fouten lezingen te vermijden. Ze dragen elk een cijfer van 1 tot en met 5 en worden maar één keer gebruikt in de barcode. Dit zorgt ervoor dat we in theorie een totaal van 5! (= 120) verschillende schermen op één moment kunnen detecteren. Het herhalend patroon geeft ons de mogelijkheid om fouten te vermijden. Na het vinden van het scherm zelf (zie sectie 3.2) zal de barcode gelezen worden. Indien een reflectie of een overlap plaatsvind kan een stuk van het scherm bedekt zijn. De scanner() zal het midden van het scherm lezen in rechte lijn (volgens de orientatie), aangezien een stuk bedekt is zal deze blijven gaan tot hij 5 kleuren en wit tegenkomt. Zo



Figuur 2: Voorbeeld van een barcode met 6 herhalingen.

niet, zal de lijst met codes leeggemaakt worden en verder gaan. Het herhalend patroon is dus essentieel aan het correct inlezen.

3.1.2 Data set

Om het scanner() algorithme te testen hebben we een aantal foto's genomen vanop verschillende afstanden, met verschillende computers (kleuren kunnen afwijken van scherm tot scherm) en met verschillende aantal barcodes dat op het scherm past. Aangezien een herhalend patroon gebruikt wordt kan men kiezen hoeveel er per scherm staan, dit kan van 3 tot 15 gaan (meer heeft niet veel zin want het onderschijden van kleuren wordt lastig).

3.1.3 Experiments

3.2 Image

3.2.1 Algemene uitleg

3.2.2 Data set

3.2.3 Experiments

4 Testen

Voor het algemene testen van de hierboven beschreven algoritmen zijn er testen opgesteld. De testen bekijken enkele zelfgemaakte foto's. Deze zijn zelf gemaakt om de perfecte oriëntatie en grootte te weten zodat deze kunnen worden nagegaan. Er is een lijst opgesteld met alle testfoto's en daarbij hun eigenschappen. Er zijn drie verschillende testen, elk voor een belangrijke eigenschap. Zo zal het aantal schermen, de grootte van de schermen en de oriëntatie worden nagegaan. Met behulp van *console.assert* worden de verschillende testfoto's afgegaan. Bij het testen zijn er geen fouten ontdekt. Indien nodig kunnen er testen worden bijgeschreven om verdere functionaliteiten te bekijken.

5 Besluit

De *basic slave screen detection* zoals beschreven in de opgave bij taak 3 is nagenoeg volledig geïmplementeerd en werkende. Waar er in het begin *openCV* werd gebruikt, zijn er nu eigen algoritmen die al het werk leveren. Verschillende schermen kunnen worden gedetecteerd al moet er voor het zoeken naar de hoeken van deze schermen wel nog een vernufter algoritme komen. De oriëntatie van de schermen wordt snel en correct gedetecteerd. Het algoritme om de oriëntatie te vinden, gaat wel uit van correct gevonden hoeken en kan dus nog

verbeterd worden. De implementatie van de algoritmen is zo efficiënt en zo simpel mogelijk gehouden, er wordt vaak niet meer gedaan dan enkel over de aparte pixels gelopen.

De identificatie van de schermen gebeurd met een kleurenbarcode. Deze kan op een zeer klein scherm nog worden gedetecteerd. Aangezien de code herhaalt kan worden op het scherm is het ook mogelijk om bij gedeeltelijke bedenking een scherm te herkennen. Deze methode berust nu nog op de assumptie dat het scherm recht staat. Er is al een oriëntatiematrix opgesteld om het scherm naar behoren te draaien maar dit is nog niet geïmplementeerd.

De basis voor *basic slave screen detection* is gelegd. De komende weken zal de code nog wat opgeruimd moeten worden. Ongebruikte functies moeten verdwijnen en geheugenmanagement moet worden herbekeken. Ook voor scaling moet er nog gewacht worden. Deze kan nog niet herkent worden. Als later de communicatie mogelijk is tussen de *slaves* en de *master* zal de grootte van het scherm worden opgevraagd. Door deze te vergelijken met de grootte van het scherm, kan de scaling worden berekend. Er is nog veel werk aan de winkel maar de basis die er nu is, mag er toch al wel zijn.