KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Augmented Reality Bordspellen

Wouter Franken

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdspecialisatie Mens-machine communicatie

Promotor:
Prof. dr. ir. Philip Dutre

Assessor:
TODO

Begeleider:
Ir. J. Baert

Academiejaar 2014 – 2015

Voorwoord

DANKWOORD

Wouter Franken

Inhoudsopgave

Samenvatting

ABSTRACT

Lijst van figuren en tabellen

Lijst van figuren

Lijst van tabellen

Lijst van afkortingen en symbolen

Afkortingen

Symbolen

Hoofdstuk 3

Naïeve legoblokdetectie

In dit hoofdstuk bespreken we twee naïeve algoritmes om legoblokken te detecteren. Beide algoritmes zullen via een thresholding operatie de blokken lokaliseren. Vervolgens zal het eerste algoritme de geometrie van een legoblok gebruiken om te bepalen wat de pose is van deze eerste legoblok. Het tweede algoritme zal een 2 dimensionaal grid bepalen waar de blokken zich bevinden.

TODO: OVERZICHT VAN HET HOOFDSTUK

3.1 Blokdetectie op basis van simpele geometrische informatie

Dit eerste algoritme verloopt in twee delen om uiteindelijk de locatie en pose van een legoblok te bepalen: vinden van de contour van de legoblok en bepalen van de pose van de legoblok uit de contour. Eerst bespreken we deze twee onderdelen in een theoretische sectie, vervolgens wordt uitgelegd hoe het algoritme werd geïmplementeerd en ten slotte bespreken we de performantie, robuustheid en voor-en nadelen van deze methode.

3.1.1 Begrippen en veronderstellingen

Begrippen

Thresholding is een operatie die vaak in beeldverwerking wordt gebruikt om twee delen in een afbeelding te scheiden van elkaar (vaak voorgrond en achtergrond). Het resultaat is een binaire afbeelding waarin de achtergrond meestal wordt aangeduid met zwart en de voorgrond met wit.

YUV, **RGB** en **HSV** zijn drie verschillende kleurenruimtes die elk op een andere manier kleuren definiëren. Bij YUV gebeurt dit door de helderheid (Y) van de kleur te scheiden van twee kleurcomponenten (U en V). In RGB zijn er enkel drie kleurcomponenten waarbij de helderheid dus in deze componenten verwerkt zit. HSV scheidt tint (H), verzadiging (S) en helderheid (V) van elkaar. Omdat HSV drie componenten scheidt die een duidelijk verschillende invloed hebben op de uiteindelijke kleur wordt deze kleurenruimte vaak gebruikt voor kleurthresholding,

het is immers eenvoudiger om kleuren van elkaar te scheiden zonder de verzadiging of de helderheid te beïnvloeden.

Pinhole model is het model dat OpenCV gebruikt om 3D punten van een scene te transformeren naar de afbeelding via een perspectieftransformatie. In de algoritmes uit dit hoofdstuk wordt het gebruikt om 2D punten om te zetten naar 3D. Dit is de formule van het pinhole model:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Hierbij is (u, v) het 2D punt, (X, Y, Z) het 3D punt, de eerste matrix in het rechterlid de camera intrinsics matrix (bekomen door camera calibratie), de tweede matrix in het rechterlid is de camera extrinsics matrix (afgeleid uit de ligging van de markers) en s, tenslotte, is een schaalfactor. Wanneer de twee cameramatrices bekend zijn kunnen we een 3D punt omzetten in 2D en vice versa. Een belangrijke opmerking hierbij is dat, bij omzetting van 2D naar 3D, wat informatie te kort is (het aantal dimensies verhoogt immers). Deze extra informatie kan worden gegeven in twee vormen: ofwel kennen we de uiteindelijke Z coördinaat, ofwel kennen we de uiteindelijke X en Y coördinaten.

Veronderstellingen

In dit algoritme worden enkele veronderstellingen gemaakt, indien hier niet aan wordt gehouden zijn de resultaten onbepaald. In deze sectie bespreken we kort welke veronderstellingen zijn gemaakt en waarom.

- De achtergrond bestaat uit wit en zwart, belangrijk is vooral dat ze geen rode tint bevat. Deze restrictie werd opgelegd omdat we in het algoritme kleurthresholding gebruiken om de rode blokken te scheiden van de achtergrond.
- Omdat het algoritme gebruik maakt van kleurthresholding is het ook aangeraden om zo weinig mogelijk schaduw te hebben. Anders kan deze schaduw potentieel als een erg donkere rode blok worden aanzien.
- De blokken die worden gebruikt zijn steeds balkvormig, geen enkele blok of constructie van blokken mag een hoek bevatten. Dit is belangrijk omdat het algoritme ervan uit gaat dat de contour van een te detecteren legoblok bestaat uit maximaal zes hoekpunten. Dit impliceert ook dat legoblokken elkaar niet mogen aanraken vanuit het standpunt van de camera gezien (tenzij ze in elkaars verlengde liggen).
- Zoals hierboven reeds aangehaald gaat het algoritme ervan uit dat de contour van een blok maximaal zes hoekpunten bevat. Dit impliceert dat het algoritme beter werkt wanneer we legoblokken vanuit perspectief zien want dan bestaat de omtrek van een blok uit exact zes hoekpunten.

3.1.2 Algoritme

Bepalen van de contour

Eerst wordt de nieuwe frame van het YUV formaat omgezet naar het HSV formaat. Dit is een erg belangrijke stap vanwege twee redenen: ten eerste kan OpenCV niet werken met het YUV formaat en ten tweede is het HSV formaat (zoals hierboven reeds aangehaald) veruit het eenvoudigste formaat om te gebruiken bij thresholding operaties.

Op de HSV frame wordt vervolgens kleurthresholding toegepast om de legoblokken te scheiden van de wit-zwarte achtergrond. In dit eerste algoritme werd enkel gewerkt met de kleur rood. Experimenteel werd ondervonden dat om rood te scheiden van een wit-zwarte achtergrond de HSV waarden in het volgende interval moeten liggen:

$$160 < H < 180; 153 < S < 255; 30 < V < 255$$

TODO: IS CANNY EDGE DETECTIE WEL NODIG, AANGEZIEN WE AL EEN THRESHOLD HEBBEN?

Contouren worden dan bepaald met behulp van de `findContours` methode van OpenCV. Deze methode gebruikt het algoritme dat wordt beschreven in [10], het wordt hier niet besproken omdat dit out of scope van deze thesis is.

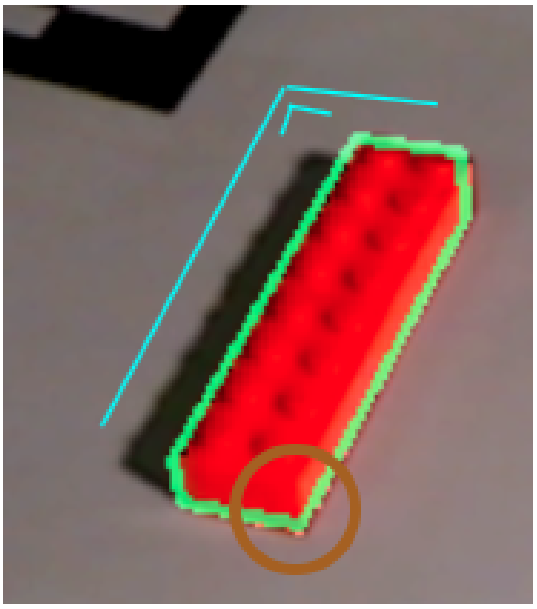
In de volgende stap wordt deze contour benadert met behulp van de `approxPolyDP` methode van OpenCV. Deze methode benadert deze contours met een polygon. Deze stap is nodig omdat een contour uit een enorme hoeveelheid punten bestaat, na deze benadering blijven de belangrijkste punten over: de hoekpunten van de legoblok. Deze methode bevat een parameter ϵ die aangeeft wat de maximale afwijking mag zijn van de benadering ten opzichte van de oorspronkelijke contour. Om ervoor te zorgen dat we een zo goed mogelijke benadering verkrijgen in alle gevallen, maken we ϵ steeds groter tot we net een maximum van zes hoekpunten hebben bereikt. Deze adaptieve techniek behoudt de kwaliteit van de contour maar in ruil daarvoor moeten we de veronderstelling maken dat de omtrek van de contour van de threshold bestaat uit maximaal zes hoeken.

Nu kan het zijn dat er contours zijn met een te kleine oppervlakte door ruis in de kleurthreshold. Om deze te verwijderen bekijken we de oppervlakte van alle contours en worden contours verwijderd met een oppervlakte die 100x kleiner is dan de oppervlakte van de grootste contour.

Het bepalen en verfijnen van de contour is afgelopen, nu moet bepaald worden wat de pose is van de legoblok. Dat gebeurt in het tweede deel van het algoritme.

Bepalen van de pose

Nu de contour is gevonden moet hieruit worden achterhaald wat de pose van de legoblok is, hiervoor bepalen we eerst in welk vlak van de legoblok elk hoekpunt van de contour ligt: onder- of bovenvlak. Dit geeft ons de Z coördinaat van al deze hoekpunten waardoor ze kunnen worden omgezet van 2D naar 3D via het pinhole model. Bij uitbreiding zijn dan alle hoekpunten van de legoblok in 3D gekend en dus kennen we ook de pose van de legoblok.



Figuur 3.1: Deze figuur toont hoe de pose vanuit een contour gevonden kan worden. Veronderstel dat de drie hoekpunten linksboven een rechte hoek vormen op het grondvlak (aangeduid met de lichtblauwe lijnen) en dat het bruin omcirkelde punt het dichtstbijzijnde punt bij de camera is. Dan zijn de drie punten deel van het bovenvlak omdat het omcirkelde punt zeker op het ondervlak ligt en niet tot deze drie punten behoort. Bovendien zijn alle andere punten zeker ook deel van het ondervlak van de legoblok.

Uit de geometrische informatie van een legoblok is geweten dat minstens drie opeenvolgende hoekpunten in hetzelfde Z-vlak liggen. Om te bepalen welke drie hoekpunten dit zijn zetten we alle 2D contourpunten om naar 3D (geprojecteerd op vlak $Z = 0$) en bepalen we welke drie opeenvolgende 3D hoekpunten (a, b en c) een hoek vormen die het dichtst bij 90 graden ligt. Van deze hoeken zijn we dan zeker dat ze in hetzelfde vlak liggen.

Vervolgens bepalen we het hoekpunt (d) dat in 3D (geprojecteerd op vlak $Z = 0$) zich het dichtst bij de camera bevindt, hiervan kan met zekerheid gezegd worden dat het werkelijk in het vlak $Z = 0$ ligt. Nu kan met zekerheid gezegd worden of a, b en c in het onder- of bovenvlak van de legoblok liggen: indien $d == a || d == b || d == c$ liggen a, b en c in het ondervlak, anders in het bovenvlak. De rest van de hoekpunten liggen in het andere vlak (zie figuur 3.1). Dit geeft ons (via het pinhole model) de 3D posities van alle hoekpunten van de contour en bij uitbreiding van de volledige legoblok.

Ten slotte kunnen nog de afmetingen en precieze positie van de blok worden bepaald. In plaats deze informatie uit slechts één frame te halen, wordt het achterhaald via een voting systeem over meerdere frames. Dit helpt om fouten in de vorige stappen van het algoritme te verkleinen. Het voting systeem werkt als volgt:

De blok wordt in verschillende frames gedetecteerd en wanneer alle hoekpunten dichter dan 0.8 cm bij elkaar liggen wordt dit gezien als dezelfde blok. In dat geval kan gestemd worden op de grootte en positie van de blok door het gemiddelde te nemen van de groottes en posities van alle blokken die zo dicht bij elkaar liggen. De waarde 0.8 cm is niet toevallig gekozen: het is exact de helft van de breedte van een 2x2 legoblok, zo dicht kunnen legoblokken dus nooit bij elkaar liggen.

Bovendien kan een legoblok nog in een verschillende state verkeren om te bepalen wanneer de blok een deel van het spel wordt (*actief*) en wanneer de blok er geen deel meer van uitmaakt (*inactief*): Wanneer een blok minstens in drie frames werd gedetecteerd wordt hij actief, maar indien hij minstens drie frames achter elkaar niet werd gedetecteerd wordt hij inactief en wanneer hij na vijf frames achter elkaar niet is gedetecteerd wordt hij zelfs verwijderd. Dit mechanisme maakt het algoritme flexibeler om blokken te kunnen toevoegen of verwijderen uit het spel wanneer de speler dat wil.

3.1.3 Resultaten, performantie en robuustheid

TODO

3.1.4 Voor- en nadelen

Aan deze methode zijn echter heel wat nadelen verbonden:

- Legoblokken mogen niet in een hoek naast elkaar staan omdat dan de veronderstelling dat een contour maximaal zes hoeken bevat niet meer geldig is.
- Enkel rode legoblokken kunnen worden gedetecteerd. Dit nadeel is echter eenvoudig weg te werken zoals wordt besproken in sectie 3.2.
- Legoblokken mogen niet in meerdere niveau's gebouwd worden. Muren zouden in principe wel kunnen (ze hebben immers maximaal zes hoekpunten) maar dan moet op één of andere manier de grootte van deze muur bepaald worden, dit wordt uitgebreider behandeld in hoofdstuk . Andere constructies in de hoogte kunnen echter niet vanwege dezelfde reden als het eerste nadeel.
- Het algoritme kan niet goed om met veranderingen qua belichting. Dit komt omdat we puur thresholden op basis van kleur maar deze waarden zijn sterk afhankelijk van welke belichting we gebruiken. In hoofdstuk komt een calibratiemethode aan bod die dit probleem verhelpt.

3.2 Blokdetectie met behulp van een 2D grid

Dit algoritme is een sterke vereenvoudiging van het vorige. In plaats van werkelijk de pose en positie van de legoblok te bepalen, gebruiken we enkel de threshold. Eerst behandelen we de aanpassingen ten opzichte van het vorige algoritme en vervolgens bespreken we de gevolgen die deze aanpassingen hebben.

3.2.1 Begrippen

Deze sectie behandelt enkele nieuwe begrippen die in het algoritme worden gebruikt.

Morphologische operaties zijn operaties die de geometrie van vormen in een binaire afbeelding kunnen wijzigen. De twee basis operaties zijn *dilate* en *erode*. In deze operaties komt er op neer dat we de convolutie nemen van de afbeelding met een kernel, die eender welke vorm of grootte kan hebben. Hierbij wordt de kernel over de afbeelding geschoven en dan wordt de pixel in het ankerpunt van de kernel vervangen door een maximum of minimum van alle pixels die binnen de kernel vallen. Bij een *dilate* operatie is dit het maximum en bij een *erode* operatie is dit het minimum. Bij een *dilate* en *erode* operatie wordt een zwart vlak dus respectievelijk kleiner en groter.

Naast de basis morphologische operaties bestaan ook nog de veel gebruikte *open* en *close* operatie. *Open* is in feite een *dilate* operatie toepassen op een afbeelding die eerder een *erode* operatie onderging. De *close* operatie is het omgekeerde. De *open* operatie wordt vaak gebruik om kleine witte vlekjes te verwijderen, terwijl de *close* operatie wordt gebruikt om kleine zwarte vlakjes te verwijderen.

3.2.2 Algoritme

Het eerste deel van het vorige algoritme wordt sterk ingekort aangezien we enkel een threshold nodig hebben. Maar er zijn ook een tweetal toevoegingen om tegemoet te komen aan de nadelen van het vorige algoritme:

Ten eerste berekenen we opnieuw een kleurenthreshold maar deze keer niet enkel voor de kleur rood, ook voor geel en blauw. Op die manier willen we aantonen dat het algoritme zonder problemen met meerdere kleuren overweg kan. Dit zijn kleurengrenzen die werden gebruikt om in een HSV afbeelding te thresholden zodat het resultaat een afbeelding is waarin de legoblokken wit zijn en de achtergrond zwart:

$$\begin{aligned} Rood &: 160 < H < 180; 153 < S < 255; 30 < V < 255 \\ Geel &: 135 < H < 160; 147 < S < 255; 30 < V < 255 \\ Blauw &: 0 < H < 112; 42 < S < 255; 13 < V < 255 \end{aligned}$$

Ten tweede, omdat we nu thresholden op meerdere kleuren kan het gebeuren dat er wat ruis optreedt in onze threshold. Deze ruis vertoont zich in hoopjes witte of zwarte lijntjes op plaatsen waar het niet hoort (zie figuur TODO). Om dit te vermijden worden de morphologische operaties *open* en *close* toegepast op de threshold.

Het tweede deel van het vorige algoritme ('het bepalen van de pose') valt volledig weg, logisch want we hebben geen contour bepaald. In plaats daarvan wordt het grid van onze virtuele wereld naar 2D omgezet (via het pinhole model). Dit 2 dimensionaal grid kunnen we vervolgens overlappen met de eerder berekende threshold en wanneer een gridnode zich in een witte regio van de threshold bevindt beschouwen we deze gridnode als 'bezet door een legoblok'.

3.2.3 Gevolgen

Dit algoritme is een grove versimpeling van het eerste algoritme aangezien de positie en grootte van een legoblok niet meer expliciet wordt bepaald. Nu vragen we ons af welke consequenties dat heeft.

Ten eerste wordt een fout gemaakt in het aantal gridnodes die bezet worden door een legoblokje. Deze fout wordt gemaakt omdat een onbezet deel schuilt achter de legoblok en dus niet zichtbaar is in een 2D camera frame. Sterker nog: we kunnen precies definiëren dat die fout even groot is als de schaduw op het grond vlak wanneer een lichtbron vanuit de camera schijnt in dezelfde richting als de camera kijkt (zie afbeelding TODO voor een voorbeeld).

Ten tweede heeft dit algoritme ook een aantal voordelen in die zin dat enkele nadelen uit het vorige algoritme werden weggewerkt:

- Legoblokken in meerdere kleuren kunnen worden gebruikt, het gevolg is wel dat morphologische operaties de threshold moeten oppoetsen om er kleine foutjes uit te halen. Dit werd hoogstwaarschijnlijk veroorzaakt door de belichting die niet altijd constant is, zodat de thresholdwaarden niet altijd even nauwkeurig zijn. Merk op dat dit even goed in het eerste algoritme had kunnen worden geïmplementeerd.

- Legoblokken kunnen nu ook in een hoek tegen elkaar staan, wat bij het eerste algoritme volledig uit den boze was. Een nadeel is echter wel dat meerdere niveau's in dit algoritme helemaal uitgesloten zijn, terwijl dit in het vorige algoritme, althans in theorie en met restricties, wel mocht. Dit komt omdat, als we de constructie hoger maken, dan de fout die we in dit algoritme maken (zie puntje 1) steeds groter wordt. Er is immers een nog groter gedeelte van het grondvlak, waar geen legoblokken op staan, verborgen achter de legoblokconstructie.

3.3

3.4 Besluit van dit hoofdstuk

Hoofdstuk 4

Legoblokdetectie op basis van CAD modellen

In het voorgaande hoofdstuk werd een naïeve methode aangebracht om legoblokken te detecteren. Deze methode worstelde echter met verschillende nadelen waardoor het niet mogelijk was om constructies van legoblokken te detecteren die bestaan uit meerdere niveau's. Om dit soort constructies wel te kunnen detecteren moeten we op zoek naar een meer generiek algoritme dat op basis van alle geometrische informatie een legoblok kan detecteren (zodanig dat de blok kan gevonden worden in eender welke legoconstructie). Daarom wordt in dit hoofdstuk een algoritme behandeld dat op basis van features, geëxtraheerd uit CAD modellen, legoblokken kan detecteren in een videoframe.

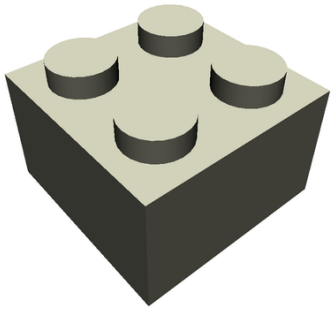
Eerst schetsen we kort in sectie 4.1 het algoritme. Vervolgens bespreken we in sectie 4.2 welke evaluatiemethodes verder in dit hoofdstuk worden gebruikt. Verschillende feature types en classificatie methodes voor dit algoritme komen aan bod en worden vergeleken in secties 4.3 en 4.4. Tenslotte geven we in sectie 4.5 aan waarom deze methode niet kan worden gebruikt in een AR spel om legoblokken te detecteren.

4.1 Begrippen en algoritme

Begrippen

CAD modellen zijn collecties van punten die de geometrie van een 3 dimensionaal object beschrijven, meestal worden ze gebruikt om objecten in een virtuele 3D wereld voor te stellen. Afbeelding 4.1 toont een afbeelding van het CAD model van een legoblok dat later in dit hoofdstuk nog wordt gebruikt.

Window sliding is een methode waarbij een rechthoek (het window) geschoven wordt over een afbeelding. Elke keer de rechthoek zich op een nieuwe plaats op de afbeelding bevind wordt meestal een andere afbeelding vergeleken met het gedeelte van de afbeelding in het window. Deze methode wordt onder andere gebruikt bij object detectie om een object te zoeken in een afbeelding. In dit geval wordt dan,



Figuur 4.1: Pose van het CAD model van een 2x2 legoblokje.

steeds wanneer de rechthoek verschuift, bekeken of het te detecteren object zich in dit window bevindt. Omdat er tijdens window sliding echter geen rekening wordt gehouden met de schaal van het object doen we dit meermaals en wordt de schaal van het window telkens aangepast.

Algoritme

Een CAD model bevat alle geometrische informatie over een object (het moet dat object immers in 3D voorstellen) en dus kan dit model gebruikt worden om een object in een afbeelding te detecteren. Het grote voordeel is dan dat, aangezien dit model alle geometrische informatie bevat, het object in om het even welke pose kan worden gedetecteerd. Dit is de methode die werd aangebracht in [1].

Om CAD modellen te kunnen gebruiken voor object detectie moet eerst nuttig informatie uit deze modellen worden gehaald. In een afbeelding is echter slechts een deel van het object te zien (de afbeelding is namelijk 2D), daarom is het beter eerst afbeeldingen te genereren van de verschillende poses van het object (figuur 4.1 toont een voorbeeld). Hierna kunnen deze afbeeldingen, via window sliding, vergeleken worden met de testafbeelding en afhankelijk van een goede score blijkt welke pose het object in de afbeelding heeft.

Het vergelijken van het window met de verschillende poses gebeurt door van het window en de poses features te berekenen en via een classificatie methode te vergelijken met elkaar. Hiervoor kunnen verschillende soorten features worden gebruikt en om aan te tonen waarom de keuze in het soort feature zo belangrijk is worden verschillende features in dit hoofdstuk vergeleken met elkaar. Verder worden ook twee verschillende classificatie methodes met elkaar vergeleken.

4.2 Evaluatiemethode

De evaluatie van de verschillende features gebeurt door eerst een classifier te trainen voor alle features en vervolgens de geleerde classifiers te gebruiken voor detectie.

Voor de **training** werden een 3000-tal positieve en 2946 negatieve samples gebruikt. De positieve samples zijn geconstrueerd door een set van 128 positieve afbeeldingen te transformeren op een willekeurig gekozen negatieve achtergrond (uit de negatieve samples). Deze transformaties helpen om een groter aantal aan positieve samples te genereren en om het effect van belichting en rotaties kleiner te maken. De set van 128 positieve afbeeldingen bestaat uit verschillende zichtpunten op een CAD model van een 2x2 legoblok en enkele foto's van dit legoblok, allemaal op een witte achtergrond. De zichtpunten zijn gegenereerd van slechts een kwart van de legoblok omdat de legoblok symmetrisch is. De foto's helpen de invloed van belichting op de classifier te minimaliseren aangezien ze werden getrokken in een meer realistische belichting. De parameters die tijdens de training werden gebruikt zijn aangegeven in tabellen 4.1 en 4.2.

De features werden steeds getraind door een cascade classifier te modelleren (zie sectie 4.4.1), dit om geen invloed op de resultaten te krijgen door een andere keuze in classificatie methode. Bij de vergelijking tussen de classificatie methodes werd steevast voor de HOG features gekozen om dezelfde reden (zie sectie 4.3.3).

De **detectie** gebeurt op twaalf afbeeldingen van een legoblok in verschillende poses. Dit geeft een idee over hoe robuust de features en classificatie methodes zijn tegen wijzigingen in pose. De parameters die gebruikt werden tijdens de detectie zijn ook aangegeven in tabellen 4.1 en 4.2.

Bij het vergelijken van features zal eerst naar performantie en robuustheid worden gekeken. De vergelijking tussen classificatie methodes focust zich eerst ook

Parameter	Waarde
Training	
# classif. in cascade	20
Min. hit rate	0.999
Max. false alarm rate	0.5
Mode (Haar-like)	ALL
Detectie	
Schaalfactor	1.1
Min. # buren	15
Min. grootte	10x10 (px)
Max. grootte	100x100 (px)

Tabel 4.1: Parameters gebruikt tijdens training en detectie van features met cascade classificatie als classificatie methode.

Parameter	Waarde
Training	
Padding (HOG)	0x0 (px)
Wind. stride (HOG)	8x8 (px)
Wind. size (HOG)	64x64 (px)
Detectie	
Schaalfactor	1.1
Hit threshold	0.4
Padding (HOG)	0x0 (px)
Wind. stride (HOG)	8x8 (px)
Wind. size (HOG)	64x64 (px)

Tabel 4.2: Parameters gebruikt tijdens training en detectie van features met SVM als classificatie methode.

op performantie en robuustheid, vervolgens wordt de impact van de *schaalfactor* parameter bekeken. Voor de experimenten die focussen op performantie werd de detectie 10x uitgevoerd op alle twaalf testafbeeldingen en hiervan het gemiddelde genomen. Omdat de training vaak een heel stuk meer tijd vraagt werd deze telkens maar eenmaal uitgevoerd om toch een groffe indicatie te geven van hoe lang zulke training duurt.

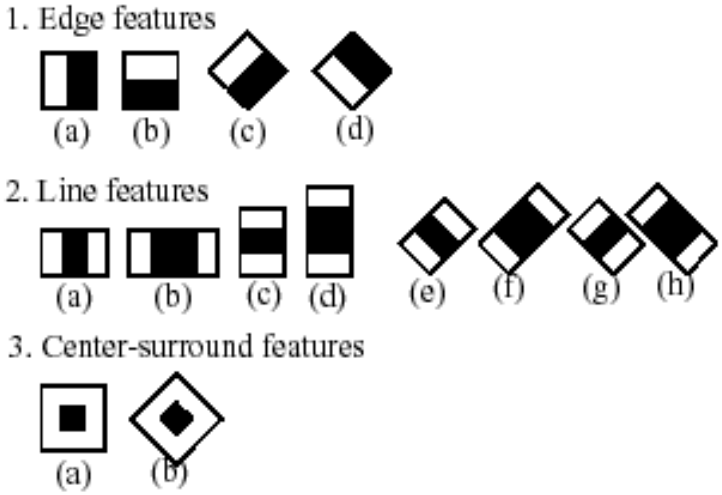
De experimenten werden uitgevoerd op een **machine** met een Intel Core 2 Duo 2,4 GHz processor, een NVIDIA GeForce 320M 256 MB grafische kaart, 8 Gb RAM en Mac OS X 10.10.2. Alle implementaties maken gebruik van OpenCV 2.4.9 (zowel training als detectie) en implementaties met SVM maken gebruik van SVMlight [6].

4.3 Features

In deze sectie worden drie verschillende soorten features besproken en met elkaar vergeleken: Haar-like [12], Multiscale Block Local Binary Patterns [7] (MB-LBP) en Histogram of Oriented Gradients [3] (HOG). Tenslotte wordt ook een parts-based feature besproken (gebaseerd op HOG), dat gebruikt werd in [1], en waarom deze niet is opgenomen in de vergelijking [4].

4.3.1 Haar-like

Bij de berekening van een Haar-like feature vector wordt het verschil genomen van de som van pixels tussen verschillende rechthoekige regio's. Oorspronkelijk werden vier soorten features gebruikt: twee die elk bestonden uit twee rechthoekige gebieden, één



Figuur 4.2: Uitgebreide Haar-like featureset. Het verschil wordt bepaald door de som te nemen van pixels in de witte regio's en af te trekken van de som van pixels in de zwarte regio's.

soort met drie rechthoekige gebieden en één soort met vier rechthoekige gebieden [12]. Later werden extra features toegevoegd tot de uiteindelijke featureset 14 soorten features bedraagt [8], zie figuur 4.2.

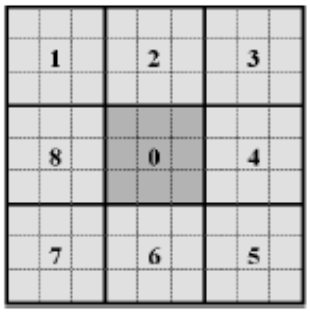
Om goede features te vinden wordt eerst een enorme hoeveelheid aan features aangemaakt. Aangezien dit aantal features een heel stuk hoger is dan het aantal pixels in het window moet een efficiënte methode gebruikt worden om de juiste features te selecteren. Hiervoor wordt AdaBoost gebruikt, een methode die origineel wordt gebruikt om een classifier sterker te maken. AdaBoost selecteert telkens die feature die de positieve en negatieve samples scheidt met een zo klein mogelijke fout [12]. Dit impliceert dat elke feature ook een threshold bevat die aangeeft wanneer deze feature een detectiewindow juist of onjuist classificeert. De uiteindelijke feature vector van het volledige detectiewindow is de set van de geselecteerde features.

4.3.2 MB-LBP

Een MB-LBP feature is een uitbreiding op de originele LBP feature waarin elke pixel werd vergeleken met zijn acht buren. Deze acht buren werden steevast in dezelfde volgorde overlopen en telkens een buur een grotere intensiteit had dan de pixel zelf werd een '1' genoteerd, in de andere gevallen een '0'. Zo wordt voor elke pixel een getal verkregen van acht bits dat de gradiënt van de pixel voorstelt in de verschillende richtingen [9].

Hier wordt de MB-LBP feature gebruikt in plaats van de oorspronkelijke LBP omdat deze meer robuust is. In de MB-LBP variant worden, in plaats van een pixel met zijn buren te vergelijken, blokken van pixels met hun buren vergeleken (zie figuur 4.3). Hierdoor is de feature meer robuust en encodeert het bovendien, naast microstructuren, ook macrostructuren in een afbeelding [7].

Om de feature vector van een volledig window te berekenen wordt het window in cellen opgesplitst en vervolgens histogrammen berekend van de MB-LBP features binnen een cel. De histogrammen van alle cellen worden ten slotte geconcateneerd om de feature vector te bekomen.



Figuur 4.3: Een 9x9 MB-LBP operator: elk blok bestaat uit 9 pixels.

4.3.3 HOG

Bij een HOG feature worden eerst gradiënten van alle pixels in een window berekend. Vervolgens wordt het window onderverdeeld in cellen en wordt voor elke cel een histogram van de gradiënten gemaakt. Om problemen met belichting en contrast te voorkomen worden cellen gecombineerd tot blokken waarover deze histogrammen worden genormaliseerd. Ten slotte worden alle histogrammen geconcateneerd (net als bij MB-LBP) om een feature vector van het window te bekomen [3].

4.3.4 Deformed parts-based

Deformed parts-based modellen is een type feature dat bestaat uit twee onderdelen: een HOG feature van de volledige window (root filter) en aantal kleinere HOG features van een subwindow (part filters). Deze verschillende part filters worden gedefinieerd door een anker punt ten opzichte van de plaats van de root filter en een functie die alle mogelijke plaatsen van deze part filter beschrijft relatief ten opzichte van het anker punt. De score van de hypothese dat een window het te detecteren object bevat wordt berekend door de sommatie te nemen van de verschillen tussen de score van een filter (HOG feature) en een kost voor de vervorming van de filter. Dit model maakt het HOG feature een stuk meer flexibel voor vervormingen of andere poses van een object [4].

Een groot nadeel van deze methode is dat ze nogal wat rekenwerk vraagt om uit te voeren en dus ongeschikt is voor gebruik in real time. Dit probleem is eerder al aangehaald in andere state-of-the-art papers die de performantie hebben kunnen verhogen. Hoewel bijvoorbeeld in paper [13] wordt aangegeven dat 40 FPS haalbaar is, is dit slechts na parallellisatie en bovendien op een desktop computer. Zulke resultaten zullen dus (voorlopig) niet gehaald kunnen worden op een mobiel apparaat om te gebruiken voor AR. Wegens tijdsbeperkingen is dit niet aangetoond.

4.3.5 Vergelijking

In deze sectie worden de eerste drie besproken feature types met elkaar vergeleken. Ze worden eerst vergeleken qua performantie en robuustheid.

Performantie

De vergelijking van performantie tussen verschillende soorten features bestaat uit twee onderdelen: enerzijds de snelheid bij training van de classifier en anderzijds de snelheid bij detectie.

Tabel 4.3 toont dat een classifier op basis van HOG features sneller kan worden getraind dan een classifier op basis van LBP features en dat deze laatste sneller kan worden getraind dan een classifier op basis van Haar-like features. We merken vooral op dat er een groot verschil is tussen de tijd nodig om een HOG classifier te trainen en tijd nodig om een LBP of Haar-like classifier te trainen. Dit valt als volgt te verklaren: bij het trainen van een Haar-like classifier wordt eerst voor elke window een enorme hoeveelheid aan features gegenereerd die vervolgens wordt uitgeselecteerd

met AdaBoost [5]. Deze hoeveelheid aan features is vele malen groter dan het aantal pixels in het window, in tegenstelling tot MB-LBP en HOG waarbij voor elk window enkel een aantal histogrammen worden opgesteld.

Uit de snelheid in detectie kunnen we vooral opmerken dat de detectie van MB-LBP een stuk sneller gaat dan detectie van Haar-like en HOG. De reden hiervoor is simpelweg dat bij MB-LBP gewoon integer waarden worden vergeleken terwijl bij Haar-like en HOG features floating point getallen moeten worden vergeleken.

Robuustheid

Uit de resultaten van figuur 4.4 kan worden afgeleid dat detectie met behulp van HOG de beste resultaten geeft: geen enkele afbeelding uit de testset was vals positief en op 3 afbeeldingen uit de testset na werd de legoblok overal correct gevonden. Hoewel met MB-LBP één legoblok meer wordt gevonden, vertoont het heel wat meer valse positieven.

TODO: bespreek HAAR features (resultaten hiervan zijn nog onvolledig).

4.4 Classificatie methodes

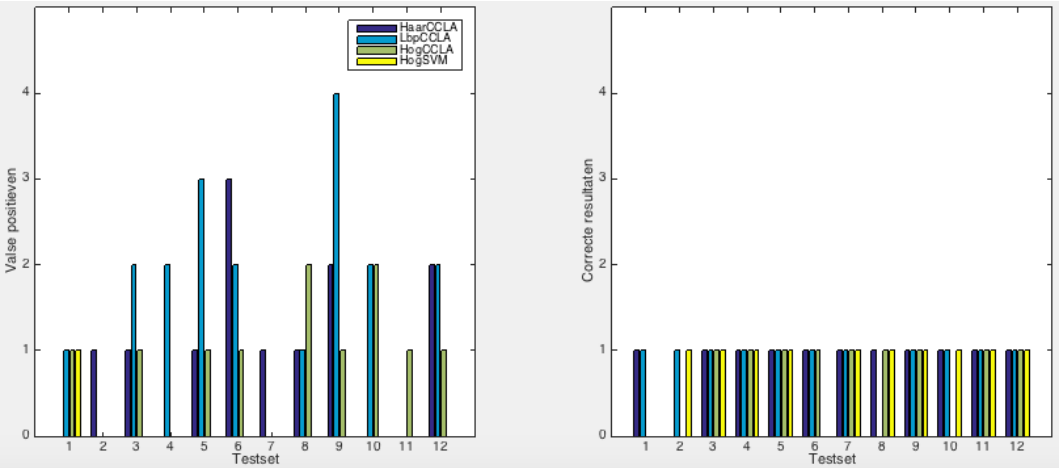
In deze sectie worden de volgende classificatie methodes besproken: Cascade Classifiers [12] (CCLA) en Support Vector Machines [6] (SVM). Vervolgens worden beide methodes vergeleken op vlak van robuustheid en performantie. Ten slotte wordt de impact bekeken van de *schaalfactor* parameter op robuustheid en performantie voor zowel CCLA als SVM.

4.4.1 CCLA

Bij een **CCLA** wordt een classifier getraind door een aantal feature classifiers te combineren. Elk van deze classifiers bestaat uit een aantal features op basis waarvan elke subwindow van een frame wordt geclassificeerd. Zulke cascade classifier werkt door eerst elke subwindow van de frame te laten evalueren door de eerste classifier, indien de eerste classifier de subwindow aanvaardt triggert dit een evaluatie door de volgende classifier in de cascade. Enkel wanneer een window door alle classifiers is aanvaard, kan er vanuit worden gegaan dat dit window het te detecteren object bevat. Door een cascade van classifiers te gebruiken moet elke classifier van de cascade niet

	Gemiddelde performantie	
Feature	Training	Detectie (ms)
Haar-like	> 2 dagen	TODO
MB-LBP	TODO	43.71
HOG	5h3m3s	189.36

Tabel 4.3: Feature performantie.



Figuur 4.4: Feature robuustheid.

al te sterk zijn waardoor negatieve windows al sneller kunnen worden afgevoerd, wat de detectie des te sneller maakt.

4.4.2 SVM

Een **SVM** classificeert op een andere manier: elk subwindow wordt voorgesteld als een p-dimensionale vector in een p-dimensionale ruimte. Met behulp van (p-1)-dimensionale hyperplane worden de subwindows gesplitst in twee groepen: de ene groep wordt aanvaard, de andere niet. De *hit threshold* parameter bepaald hoe ver een vector van de hyperplane moet liggen om te worden aanvaard of afgekeurd. Deze lineaire classifier was het oorspronkelijke algoritme [11], later werd ook een methode ontwikkelt voor non-lineaire classificatie [2].

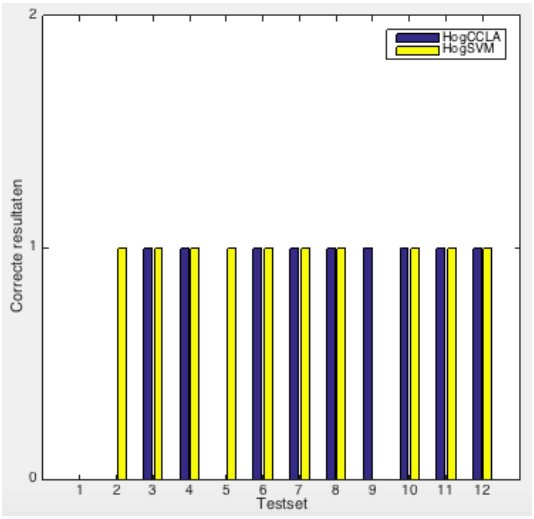
4.4.3 Vergelijking

Robuustheid

Figuur 4.5 toont de robuustheid van de twee classifier methodes. Merk op dat de grafiek met valse positieven is weggelaten omdat deze in beide gevallen overal nul was. Er is duidelijk weinig verschil tussen de twee methodes qua robuustheid: de SVM methode detecteert slechts één legoblokje meer.

Performantie

Tabel 4.4 toont een vergelijking van de performantie tijdens de training en de detectie. Dit toont aan dat de snelheid van CCLA hoger ligt bij detectie maar niet bij training. Dat de detectie langer duurt bij een SVM is logisch aangezien het een lineaire classifier is (geen cascade die de snelheid verhoogt). De training snelheid ligt lager bij een SVM omdat deze geen cascade van classifiers moet opstellen.



FIGUUR 4.5: Robuustheid classifier methodes.

	Gemiddelde performantie	
Feature	Training	Detectie (ms)
CCLA	5h3m3s	189.36
SVM	1m55s	244.94

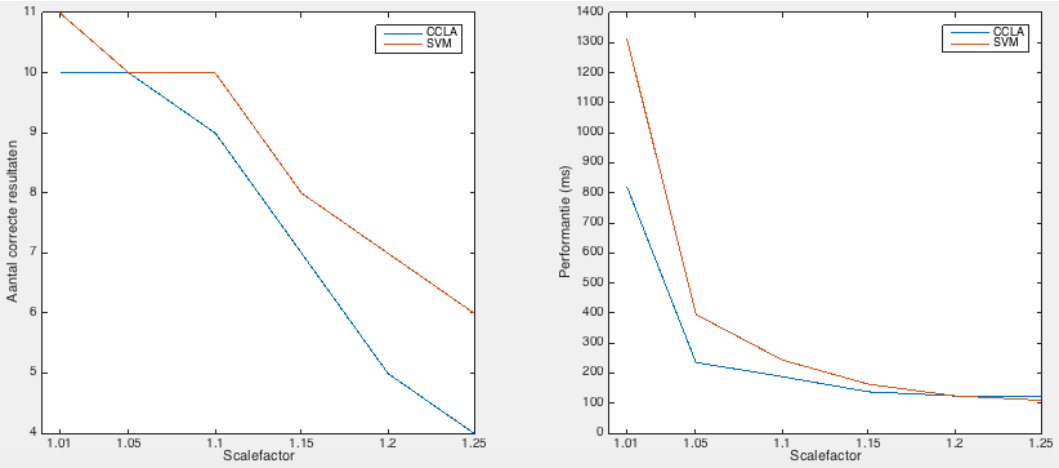
TABEL 4.4: Performantie classifier methodes.

Schaalfactor

De schaalfactor is een parameter die bepaalt hoe sterk de schaal van het window wordt gewijzigd tussen een minimum en maximum grootte. Dus wanneer deze kleiner wordt zullen normaal gezien meer iteraties nodig zijn, terwijl een nauwkeurigere detectie plaatsvindt.

Om de impact van de schaalfactor te bepalen, wordt de performantie en robuustheid van de HOG feature opgemeten wanneer de schaalfactor 1.01, 1.05, 1.10, 1.15, 1.20 en 1.25 bedraagt. Uit figuur 4.6 is het duidelijk dat aan de verwachtingen voldaan is: de performantie is hoger (het aantal ms daalt) bij een hogere schaalfactor, terwijl het aantal correcte detecties daalt. Het aantal valse positieven werd ook gemeten maar deze was voor beide classifiers bijna altijd 0.

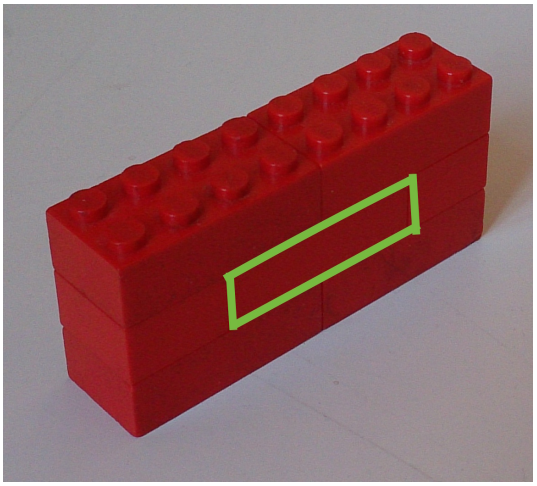
Opmerkelijk is echter dat het aantal correcte poses hoger ligt bij SVM dan bij CCLA. Maar hoewel de detectie beter is laat de performantie echter te wensen over. De oorzaken hiervoor kunnen als volgt verklaard worden: Een SVM is een lineaire classifier die vergeleken kan worden met één enkele classifier uit een cascade. Om een voldoende goede detectie te behalen moet een SVM dus erg correct kunnen classificeren, dit vergt meer tijd aangezien een cascade juist is ontwikkeld om efficiënter te zijn dan een enkele classifier. Aangezien de SVM een hogere robuustheid haalt is het dan ook logisch dat de performantie lager ligt.



Figuur 4.6: Impact schaalfactor op performantie en robuustheid van HOG feature.

4.5 Discussie / Besluit

In dit hoofdstuk werd onderzocht welke feature types en classifier methodes beter werken voor detectie van legoblokken. Zo is aangetoond dat HOG de meer robuuste feature type is en dat MB-LBP de meest performante van de drie is. Verder werd aangegeven dat een vierde type feature (parts-based deformable modellen) te traag is om nuttig te gebruiken in een AR applicatie. Ten slotte toonde de vergelijking tussen classifier methodes aan dat detectie met een SVM classifier vaak trager is maar tegelijk ook meer robuust. De methode die in dit hoofdstuk werd onderzocht is echter niet nuttig om te gebruiken in een AR spel om legoblokken te detecteren en



Figuur 4.7: Een voorbeeld van een legoblokconstructie waarin het groen omlijnde vlak het enige is wat we van die legoblok zien.

hier is waarom:

Hoewel het wel mogelijk is om een enkele legoblok te detecteren is het erg moeilijk om in een constructie van legoblokken de blokken afzonderlijk te detecteren. Dit resulteerde telkens in erg slechte resultaten, zelfs wanneer het beste type feature en de beste classificatie methode werd gekozen. Dit wordt hoofdzakelijk veroorzaakt door de enorme hoeveelheid aan occlusie die een legoblok in een constructie grotendeels verbergt. Hierdoor is slechts een erg beperkt deel van de legoblok (in sommige gevallen slechts één vlak, zie figuur 4.7) zichtbaar wat het erg moeilijk maakt om met features een legoblok te detecteren. Zelfs indien op voorhand een optimale pose zou worden gekozen kan voor sommige legoblokken enkel één vlak zichtbaar zijn in de afbeelding.

Deze redenering leidt tot een volgend resultaat: het is erg moeilijk, zo niet onmogelijk, om in een legoconstructie alle legoblokken apart te detecteren. Daarom moeten we proberen om, in plaats van de blokken individueel te proberen detecteren, ze als groter geheel te detecteren. Zulke legoconstructie kunnen we zien als een flexibele legoblok die eender welke hoogte, diepte en breedte kan hebben. Dat is exact waarvoor parts-based deformable modellen voor worden gebruikt maar zoals aangehaald in sectie 4.3.4 is deze methode hoogstwaarschijnlijk te traag om te gebruiken in AR. Dat is de reden waarom in de volgende sectie een sneller alternatief wordt voorgesteld dat gebruik maakt van het idee dat lego constructies moeten gedetecteerd worden in plaats van individuele legoblokken. Hierbij wordt echter wel afgestapt van het idee om CAD modellen te gebruiken.

Bijlagen

Bibliografie

[1] M. Aubry, D. Maturana, A. A. Efros, B. C. Russell, and J. Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 3762–3769. IEEE, 2014.

[2] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[4] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

[5] Y. Freund and R. E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.

[6] T. Joachims. Svmlight: Support vector machine. *SVM-Light Support Vector Machine http://svmlight. joachims. org/, University of Dortmund*, 19(4), 1999.

[7] S. Liao, X. Zhu, Z. Lei, L. Zhang, and S. Z. Li. Learning multi-scale block local binary patterns for face recognition. In *Advances in Biometrics*, pages 828–837. Springer, 2007.

[8] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages I–900. IEEE, 2002.

[9] T. Ojala, M. Pietikäinen, and D. Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29(1):51–59, 1996.

[10] S. Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[11] V. Vapnik. Pattern recognition using generalized portrait method. *Automation and remote control*, 24:774–780, 1963.

[12] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[13] J. Yan, Z. Lei, L. Wen, and S. Z. Li. The fastest deformable part model for object detection. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 2497–2504. IEEE, 2014.

Fiche masterproef

Student: Wouter Franken

Titel: Augmented Reality Bordspellen

Engelse titel: Augmented Reality Boardgames

UDC: 681.3

Korte inhoud:
Hier komt een heel bondig abstract van hooguit 500 woorden. LaTeX commando's mogen hier gebruikt worden. Blanco lijnen (of het commando) zijn wel niet toegelaten!

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdspecialisatie Mens-machine communicatie

Promotor: Prof. dr. ir. Philip Dutre

Assessor: TODO

Begeleider: Ir. J. Baert