Projectverslag
Remote Rock Paper Scissors Lizard Spock

Klaas de Boer - 0165816

Begeleider:	Lennart Herlaar
Tweede beoordelaar:	Chris Janssen
Opleiding:	Cognitieve Kunstmatige Intelligentie - Bachelor
Opleidingsinstituut:	Universiteit Utrecht
ECTS:	10

25 november 2016

Inhoudsopgave

1 Inleiding **3**

1.1 Het spel: RPSLS 4

1.2 Computer Vision 5

1.3 Ontwerp van de RRPSLS applicatie 5

2 Relevante Literatuur **6**

3 Methode **7**

3.1 Onderzoeksaanpak 7

3.2 Van foto tot classificatie 7

3.3 Huidherkenning 8

3.3.1 Colorspace 8

3.3.2 Morfologische operaties 8

3.4 Gebaarclassificatie 9

3.4.1 De implementatie 9

3.5 Opbouw featurevector 10

3.6 Dataset 10

4 Experimenten **10**

4.1 Huidherkenning 11

4.2 SVM classificatie 11

4.2.1 kfolds principe 11

5 Resultaten **12**

5.1 Huidherkenning 12

5.2 SVM classificatie 13

5.2.1 YCrCb 14

5.2.2 LCCS 14

5.2.3 Resultaten met alle vijf de gebaren - Linear kernel 14

5.2.4 Resultaten met alle vijf de gebaren - RBF kernel 15

6 Conclusies & Discussie **16**

6.1 Programmeertaal 16

6.2 Huidherkenning 17

6.3 Android applicatie 17

6.4 Vervolg 17

1 Inleiding

Voor mijn Bachelor Eindwerkstuk voor de studie Cognitieve Kunstmatige Intelligentie aan de Universiteit van Utrecht heb ik gekozen voor een programmeeropdracht. Het doel bij aanvang van het project was het schrijven van een applicatie waarmee het mogelijk is om via een telefoon (*Remote*) het spel *Rock Paper Scissors Lizard Spock* (RPSLS) te spelen tegen een andere menselijke speler. De gebaren van de spelers worden herkend door het programma en de score wordt automatisch bijgehouden.
Er bestaan al Rock Paper Scissors (RPS) applicaties, maar dit zijn typisch applicaties waar tegen een computer gespeeld wordt en zonder de lizard-spock toevoeging. Een aantal van deze bestaande applicaties worden in het tweede hoofdstuk besproken.
In de applicatie zullen verschillende componenten samenwerken. Zo zal er een foto gemaakt worden door de camera van de telefoon, deze foto wordt vervolgens verwerkt en geanalyseerd met behulp van Computer Vision technieken opdat het gebaar bepaald kan worden. Hiervoor zal eerst het gebaar geïsoleerd worden op de foto met behulp van huidherkenning, vervolgens wordt het gebaar geclassificeerd als zijnde één van de vijf gebaren van het spel. Dit wordt voor beide spelers gedaan en de resultaten worden volgens de regels van het spel tegen elkaar gehouden om de winnaar (of het eventuele gelijkspel) te bepalen. De applicatie op de telefoons van de spelers heeft hiervoor een connectie gemaakt met een spelserver, deze spelserver maakt het mogelijk om de relevante informatie uit te wisselen.
Tijdens het verloop van het project is het initiële doel bijgesteld. Ik ben amibitieus begonnen aan een groot vraagstuk. Vanwege de grootte van het vraagstuk is het echter niet mogelijk geweest om binnen redelijke tijd een volledige Android applicatie te ontwikkelen. Wel heb ik nu meer inzicht in de bouwstenen van een Android applicatie, aangezien de eerste stappen die nodig zijn voor de bouw van een Android applicatie wel uitgevoerd zijn, zoals het instellen van een werkende ontwikkelomgeving en het maken van een applicatie die überhaupt te installeren is op een telefoon. Bij het nieuwe doel is het maken van de applicatie in Android-vorm komen te vervallen. De voornaamste uitdaging zit in het correct en robuust herkennen van de gebaren, er is gekozen om hierop te concentreren in dit project.

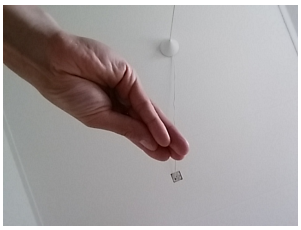
De code van het werkstuk is beschikbaar op de Subversion Repository: `https://svn.science.uu.nl/repos/sci.0323217.b0165816.project/`. Neem voor toegang contact op met Coert van Gemeren (c.j.vangemeren@uu.nl).

In dit project heb ik nauw samengewerkt met Coert van Gemeren en ik wil hem hierbij ook zeer bedanken voor al zijn hulp en inzichten. Ook wil ik Lennart Herlaar bedanken voor zijn geduld en behulpzaamheid. Ondanks de verkleining van de scope van het project heb ik een goed gevoel aan het geheel overgehouden en dat komt zeker mede door de goede samenwerking met hen beide.

1.1 Het spel: RPSLS

Rock Paper Scissors Lizard Spock (RPSLS) is een variant en uitbreiding, bedacht door Sam Kass [1], op het Rock Paper Scissors spel(RPS). Deze variant is vooral bekend geworden door de uitleg van Sheldon Cooper [2] in de bekende televisieserie the Big Bang Theory.
Transcript van de regels in het Engels: *Scissors cuts Paper, Paper covers Rock, Rock crushes Lizard, Lizard poisons Spock, Spock smashes Scissors, Scissors decapitates Lizard, Lizard eats Paper, Paper disproves Spock, Spock vaporizes Rock (and as it always has) Rock crushes Scissors.* Hieronder een schematische weergave zie tabel 1 en een grafische weergave zie figuur 6.
In onderstaande figuren een voorbeeld foto van ieder gebaar:

Figuur 1: Rock



Figuur 2: Lizard

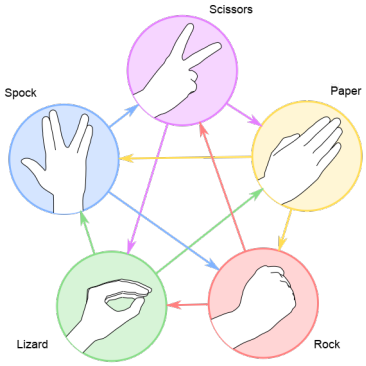
Figuur 3: Spock

Figuur 4: Scissors

Figuur 5: Paper

(1) rock	wint van (2) lizard	en van (4) scissors
(2) lizard	wint van (3) spock	en van (5) paper
(3) spock	wint van (4) scissors	en van (1) rock
(4) scissors	wint van (5) paper	en van (2) lizard
(5) paper	wint van (1) rock	en van (3) spock

Tabel 1: Schematische weergave van de regels



Figuur 6: Grafische weergave van de regels

1.2 Computer Vision

In het project wordt gebruik gemaakt van *Open Source Computer Vision* [3], dit is een bibliotheek van algoritmen waarmee beeld geanalyseerd kan worden. De keuze welke van de beschikbare technieken te gebruiken, de instelling hiervan en de aansluiting van de verschillende onderdelen zal in dit project uitgewerkt worden.

1.3 Ontwerp van de RRPSLS applicatie

Beide spelers starten de *Remote RPSLS* applicatie op zijn of haar (Android) telefoon en leggen hun telefoon met de rug op tafel, dit zorgt ervoor dat de front-side camera omhoog gericht is. Als de speler aangegeven heeft klaar te zijn, door op de startknop te drukken, wordt er afgeteld vanaf 3. Bij 0 dient de speler zijn gebaar boven de telefoon te maken. Op dat moment maakt de applicatie een foto en wordt deze foto geanalyseerd en vervolgens geclassificeerd door de Computer Vision software. Vervolgens worden de gebaren van beide spelers uitgewisseld en volgens de regels van het spel met elkaar vergeleken. De score van het spel wordt bijgehouden door de applicatie en de speler ziet dit op het scherm van zijn telefoon. Tevens krijgt de speler in het scorescherm te zien hoe zijn of haar gebaar geclassificeerd is.

Voor de communicatie tussen de twee telefoons wordt gebruikt gemaakt van een *gameserver*. Bij het starten van de applicatie wordt er een verbinding gemaakt met de *gameserver*. Bij de eerste keer starten wordt er automatisch een *game-id* aangemaakt op basis van het telefoonnummer. Op het moment dat een gebruiker wil starten kiest hij eerst een tegenstander, hiervoor gebruikt hij het telefoonnummer van de tegenspeler. Als dit telefoonnummer niet bekend is bij de *gameserver* zal er een foutmelding verschijnen. Zodra ook de tegenspeler de applicatie gestart heeft en het binnenkomende verzoek tot een spel heeft geaccepteerd, krijgen zij beide het startscherm te zien. Dit startscherm bevat een startknop, de laatst gemaakte gebaren en de totaalscore. Na het indrukken van de startknop wordt de ronde gestart en moet er een gebaar gemaakt

worden boven de telefoon nadat het aftellen voltooid is. Op dat moment wordt een foto gemaakt en geclassificeerd door de Support Vector Machine (SVM) [4]. De classificatie (een simpele *enum*, dus niet de foto) wordt naar de *gameserver* gestuurd die deze classificatie vervolgens doorstuurt aan de telefoon van de tegenstander en visa versa. Nu kan de applicatie op de telefoon de gemaakte gebaren weergeven, de score bepalen en deze toevoegen aan het totaal. Het is niet noodzakelijk dat de spelers tegelijkertijd hun actie uitvoeren, wanneer een speler wacht met het drukken op de startknop zal ook verwerken van de score bij de tegenstander wachten. Pas als beide telefoons via de *gameserver* gecommuniceerd hebben een gebaar te hebben ontvangen van de tegenstander kan er een volgende ronde gestart kan worden. Op dat moment wordt voor beide spelers de startknop weer beschikbaar.

In de applicatie zal een voorgetrainde SVM aanwezig zijn waarmee de gebaren herkend worden. Deze SVM wordt vooraf getraind met een zo groot mogelijke dataset op een ontwikkel computer en vervolgens kan deze SVM geëxporteerd worden. De artefacten van deze export worden toegevoegd aan de code van de Android applicatie. Hierna kan een applicatie *package* gemaakt worden. Deze *package* kan bijvoorbeeld via email gedistribueerd worden of toegevoegd worden aan de *Google App Store*.

2 Relevante Literatuur

- RPS tegen de computer [5]
  Het gaat hier voornamelijk om de AI van de computer waartegen gespeeld wordt. Er wordt gebruik gemaakt van online beschikbare *haarclassifiers* [6] voor de gebaren: rock, paper en scissors. In het algemeen geldt dat het trainen van *Haar-like features* erg veel werk is, zie het werk van Viola & Jones [7] waarin *Haar-like features* voor het eerst geïntroduceerd worden. Er zijn geen voorgetrainde *classifiers* beschikbaar voor de gebaren lizard en spock. Hierdoor valt het gebruik van deze methode voor mijn applicatie af.

- RPS robot die altijd wint [8]
  Bij deze installatie wordt het gebaar van de speler heel snel bepaald (één miliseconde), vervolgens kiest de applicatie het gebaar dat hiervan wint en laat de robot dit gebaar maken. Als mens heb je niet door dat de robot pas kiest nadat jij je gebaar hebt gemaakt, hierdoor lijkt het alsof de robot het spel volgens de regels speelt. Toch speelt de robot vals, aangezien deze methode tegen het basisidee van het spel ingaat: namelijk dat beide spelers (blind) een gebaar kiezen en niet dat je wacht tot de tegenstander zijn gebaar maakt en daarna pas kiest.

- RPS op Android [9]
  In deze implementatie wordt wederom tegen de computer gespeeld. De gebaren worden herkend door het tellen van vingers, met behulp van *convexity defects*[1]. Er is getest met dertig foto's en in 90% van de gevallen

[1] Zie hoofdstuk 3.2 voor een uitleg wat *convexity defects* zijn.

wordt de huid correct gedetecteerd [10]. Doordat in deze implementatie alleen de *convexity defects* gebruikt worden is het niet mogelijke om scissors van spock te onderscheiden aangezien deze twee gebaren hetzelfde aantal *convexity defects* heeft. Deze implementatie zal in dit eindwerkstuk uitgebreid en verbeterd worden.

3 Methode

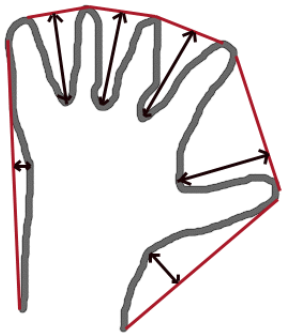
Hier zal eerst een korte beschrijving geven worden van het verloop van het onderzoek om inzicht te geven in de volgorde van de verschillende stappen, daarna wordt het algemene beeld geschetst om van een foto tot een classificatie te komen, tot slot wordt iedere stap in een eigen subhoofdstuk in detail toegelicht.

3.1 Onderzoeksaanpak

Stap één is het op het correct herkennen van huid met behulp van computer vision software. Hiervoor zijn er verschillende *colorspaces* met elkaar vergeleken gebruik makend van een initiële dataset van dertig foto's met daarop de vijf gebaren. Stap twee is het classificeren van de gebaren. In eerst instantie is er een contourvergelijking toegepast, dit bleek niet goed te werken. Vervolgens is in stap drie aan de slag gegaan met SVMs. Hierbij zijn de *featurevector* en de dataset in de loop van het onderzoek uitgebreid, tevens zijn de interne instellingen van de SVM (de kernel) veranderd. Door de uitbreiding van de dataset is ook de *colorspace* aangepast. Gebruik makend van de uiteindelijke dataset van 125 foto's bleek dat de LCCS *colorspace* een beter resultaat gaf.

3.2 Van foto tot classificatie

Er wordt een Support Vector Machine (SVM) [4] getraind met trainingsdata. Vervolgens wordt deze getrainde SVM gebruikt om een foto te classificeren. Belangrijk is dat de te classificeren foto op precies dezelfde manier verwerkt wordt als de foto's die gebruikt zijn tijdens het trainen.
Het verwerken van de foto's gaat als volgt: als eerste wordt er een masker gemaakt met behulp van huidherkenning. De contour van dit masker wordt gebruikt om de *Hu Moments* [11] te bepalen, ook wordt de contour gebruikt om de *convexity defects* te bepalen, zie figuur 7. Van deze *convexity defects* worden de hoeken bepaald. Deze hoeken worden verwerkt tot een histogram en samen met de *Hu Moments* vormen deze getallen de *featurevector*. Voordat deze *featurevector* gebruikt wordt door de SVM wordt de volledige *featurevector* genormaliseerd.
Voor iedere foto in de trainingsset wordt zo'n *featurevector* gemaakt en deze *featurevectors* worden samengevoegd in een matrix. Tevens wordt er een één dimensionale matrix gemaakt die de gebaren van de foto's uit de trainingsset bevat. De volgorde van de elementen van deze twee matrices moet met elkaar overeenkomen, zij vormen samen de trainingsdata voor de SVM.



Figuur 7: *convexity defects*

3.3 Huidherkenning

Om de gebaren van de speler te kunnen herkennen wordt als eerste stap de huid geïsoleerd op de foto. Er bestaan verschillende methodes om huid te herkennen. In de basis komt het erop neer dat de foto vertaald wordt naar een *colorspace* en dat vervolgens voor iedere pixel bepaald wordt of het huid is of niet. Dit resulteert in een zogenaamd 'masker'. Een masker is een binaire weergave van het plaatje waarin alle witte pixels huid representeren en zwarte pixels niet-huid. Zie voor een voorbeeld subhoofdstuk 3.3.2. Het masker wordt daarna morfologisch verwerkt met als doel het wegwerken van ruis. Ruis is in deze context foutief geclassifeerde pixels waar huid niet gedetecteerd is of juist pixels die onterecht als huid geclassifeerd zijn. Over het algemeen zijn dit kleine groepjes pixels die door middel van de morfologische operaties weggepoetst kunnen worden.

3.3.1 Colorspace

Om de hand te isoleren is gebruik gemaakt van de *log-chromaticity colorspace* (LCCS) [12]. Dit is een *colorspace* die na tests met de YCrCb en de HSV [13] *colorspace* uiteindelijk het beste resultaat gaf met de uiteindelijke dataset die gebruikt is in dit onderzoek. De foto wordt omgezet in deze *colorspace* en voor iedere pixel wordt gekeken of de waarde binnen het gestelde bereik valt. Zie voor de onderzochte bereiken tabel: 2. Een waarde binnen het bereik betekent een huidpixel, een waarde buiten het bereik zorgt ervoor dat de pixel als niet-huid geclassificeerd wordt. Het bereik dat gebruikt is voor de LCCS *colorspace* komt uit het artikel van Khanal et al [12].

3.3.2 Morfologische operaties

Met morfologische operaties kunnen binaire beelden, zoals maskers, bewerkt worden. Het voornaamste doel hiervan is het wegwerken van ruis. Er zijn vier morfologische operaties: *dilation*, *erosion*, *closing* en *opening*. In dit huidherkenningsonderzoek heb ik steeds *erosion*, dan *dilation* en als laatste *closing* toegepast. Hieronder is het effect te zien van de gekozen morfologische operaties.

Figuur 8: De foto die verwerkt wordt tot masker

Figuur 9: Masker zonder morfologische operaties

Figuur 10: Masker met morfologische operaties

3.4 Gebaarclassificatie

Binnen het vakgebied van computervision bestaan er verschillende methodes om vormen te classificeren. Eén van de meer naïve methodes is het vergelijken van de waarde van iedere pixellocatie van het masker en de groundtruth[2] voor ieder gebaar. Het masker wordt dan geclassificeerd als het gebaar van de groundtruth waarbij de meeste pixels overeenkomen. Op het moment dat de oriëntatie of de groote van het masker niet overeenkomen met de groundtruth zal het gebaar niet of zeer slecht herkend worden. Een andere methode is het vergelijken van de contouren. Met behulp van de *imagemoments* [14] kan de contour van het masker vergeleken worden met de contouren van de groundtruths. Een moeilijkheid die geldt voor deze beide methodes is het maken/definiëren van de groundtruths. Typisch zal een deel van de arm aanwezig zijn op de foto's. In het geval dat de speler een trui aan heeft zal het resulterende masker geen arm hebben, maar als de spelen een t-shirt aanheeft of de mouwen heeft opgestroopt is de arm wel zichtbaar op het masker. Dit geeft moeilijkheden bij het definiëren van de groundtruths, omdat in het ene geval het masker een armvormige toevoeging heeft, en in het andere geval slechts de hand zichtbaar is. Hierdoor zal een methode die alleen de contouren gebruikt geen robuuste resultaten opleveren. In mijn applicatie heb ik dan ook gekozen voor een andere aanpak. Zie de volgende alinea 3.4.1.
Ook kan een *Haar-like feature* [15] *classifier* gebruikt worden. Met behulp van een grote trainingsset kan een *haarclassifier* getraind worden. Door gebrek aan voldoende trainigsmateriaal heb ik besloten deze methode niet te gebruiken in mijn project. Ook is het gebruik van de online beschikbare voorgetrainde *classifiers* geen oplossing omdat de gebaren spock en lizard niet beschikbaar zijn.

3.4.1 De implementatie

De methode waar ik op uitgekomen ben is een methode die gebruik maakt van een Support Vector Machine(SVM) [4]. De input van een SVM is per plaatje een *featurevector*. Voor een goed werkende SVM bevat deze *featurevector* precies genoeg informatie om de verschillende klasses te onderscheiden. De opbouw van de *featurevector* is van groot belang voor het uiteindelijke functioneren van de SVM. Als eerste wordt er een trainingset van *featurevectors* waarvan het

[2] De groundtruth is de weergave van een foto waartegen de getransformeerde foto vergeleken kan worden. Over het algemeen worden de groundtruth gemaakt door mensen om vast te kunnen stellen hoe goed een algoritme werkt.

gebaar bekend is gebruikt om de SVM te trainen. De SVM heeft verschillende parameters waarmee de interne werking van de machine ingesteld kan worden. Nadat de SVM getraind is kan er een nieuwe *featurevector* aangeboden worden die de SVM dan kan classificeren als één van de klasses uit de trainingsset.

3.5 Opbouw featurevector

Voor de opbouw van de *featurevector* die gebruikt wordt in de SVM worden de zeven *Hu Moments* [11] van het masker gebruikt aangevuld met vier extra getallen zodat een *featurevector* van 11 getallen onstaat.
De vier extra getallen worden als volgt bepaald: van het masker worden de *convexity defects* bepaald. Dit zijn de inhammen van het masker ten opzichte van de buitenste omtrek of *convex hull* (zie de rode lijn in figuur 7). Voor iedere *convexity defect* wordt de hoek bepaald. Bij het gebaar paper zijn de meeste *convexity defects* mogelijk, namelijk vier: tussen iedere vinger één. In het masker kunnen ook erg kleine *convexity defects* gevonden worden, het algoritme filtert deze met behulp van een minimale en een maximale diepte. Alleen die defects worden opgepakt waarvan de diepte binnen het bereik van 70 en 250 pixels liggen. Deze waardes zijn empirisch gevonden. Let erop dat in de gebruikte testset alle foto's een afmeting van 640 bij 480 pixels hebben. Als er foto's van een ander formaat gebruikt worden zal dit bereik aangepast moeten worden.
Tevens wordt de hoek bepaald van de driehoek van de *convexity defect*. Alle *convexity defects* waarbij de hoek groter is dan 80° worden genegeerd. Als je je hand spreid kun je zien dat de grootste hoek die tussen de duim en de wijsvinger is en dat deze hoek niet groter is dan 80°.
De gevonden hoeken worden verwerkt tot een histogram. Er zijn vier bins gedefinieerd waarin de gevonden hoeken geplaatst worden. De waardes van de bins komen overeen met het aantal *convexity defects* dat in een bepaalde bin geplaatst is. De vier bins worden als volgt gevuld: de eerste bevat het aantal van *convexity defects* met hoeken tussen 0°en 20°, de tweede die tussen 20°en 40°, de derde 40°-60°en de vierde 60°-80°. Het aantal *convexity defects* per bin is maximaal vier en omdat er voor de *featurevector* getallen tussen de 0 en 1 gebruik moeten worden, wordt de waarde uit iedere bin gedeeld door 4.

3.6 Dataset

Doordat de juistheid van het resulterende masker van een foto zo nauw samenhangt met de lichtomstandigheden ten tijde van het maken van de foto, konden veel van de foto's die gemaakt zijn voor tijdens project niet gebruikt worden. De foto's waarbij het masker geen duidelijke handvorm heeft zijn afgekeurd.

4 Experimenten

Het eerste experiment dat uitgevoerd is in dit onderzoek was het bepalen van het beste huidherkenningsalgoritme. Er is gezocht naar de instellingen die de beste maskers opleverde. In de vervolg experimenten zijn gebaren geclassificeerd met verschillende methodes.

4.1 Huidherkenning

Met GIMP [16] zijn er met de hand huidmaskers gemaakt voor dertig foto's. Deze maskers worden gebruikt als "waarheid", de *groundtruth*, waarmee de maskers vergeleken worden die de verschillende computervision methodes opleveren. De vergeleken *colorspaces* zijn: LCCS, YCrCb (met twee verschillende instellingen) en HSV. Voor LCCS zijn de bereiken van Khanal & Sidibé [12] gebruikt. Voor HSV [13] heb ik een *bruteforce* optimalisatie algoritme gebruikt. Voor de *colorspace* YCrCb heb ik twee sets van waardes gebruikt. De waardes uit het werk van Chai & Ngan [17] en de waardes gevonden door het *bruteforce* optimalisatie algoritme. Zie tabel 2 voor een overzicht van deze waardes.
Ook zijn verschillende morfologische instellingen getest.

4.2 SVM classificatie

Om de kwaliteit van de implementatie te bepalen wordt er gekeken hoeveel foto's correct geclassificeerd worden door de SVM. Het doel is om het beter te doen dan 90% correct geclassificeerde gebaren. De gekozen *colorspace* (LCCS) zal vergeleken worden met de beste instellingen van YCrCb van het vooronderzoek. De foto's worden met beide *colorspaces* vertaald tot maskers en deze worden vervolgens met behulp van het *kfolds* principe getest. Tenslotte worden de *linear* kernel en de RBF kernel met elkaar vergeleken. Voor beide kernels is de LCCS *colorspace* gebruikt. De instelling van de kernel bepaalt hoe de zoekruimte ingedeeld wordt bij het trainen van de SVM. Ieder gebied in de zoekruimte representeert één van de gebaren. Bij een *linear* kernel wordt de ruimte verdeeld met rechte lijnen, bij een RBF (*Radial Basis Functions*) kernel zijn dit ronde of gebogen lijnen.

4.2.1 kfolds principe

Met behulp van het *kfolds* principe wordt de SVM getest. Het voordeel van het *kfolds* principe is dat met een relatief kleine dataset toch een gedegen SVM test uitgevoerd kan worden. Het idee van het *kfolds* principe is dat alle foto's van de dataset op de eerste na gebruikt worden voor de training van de SVM en dat deze eerste foto vervolgens door deze getrainde SVM geclassificeerd wordt. Vervolgens wordt de eerste foto aan de trainingset toegevoegd en de tweede foto eruit gehaald. Hierna wordt de SVM opnieuw (zonder voorkennis van de voorgaande stap te bewaren) getraind en wordt de tweede foto geclassificeerd. Dit wordt herhaald totdat iedere foto van de dataset geclassificeerd is. Hierna kan met de resulaten van deze tests het classificatiecorrectheidspercentage bepaald worden.

5 Resultaten

In dit deel worden de resultaten van de verschillende experimenten besproken.

5.1 Huidherkenning

Het *bruteforce* optimalisatie algoritme past één waarde uit het bereik aan en rekent uit hoeveel pixels correct als huid geclassificeerd worden met behulp van de maskers gemaakt in GIMP. Omdat het doorzoeken van de volledige zoekruimte (256^6 verschillende instellingen) teveel tijd kost, aangezien voor iedere instelling alle dertig foto's vertaald moeten worden naar een masker en vervolgens vergeleken worden met de groundtruth, kunnen als parameters voor de optimalisatie voor iedere min en max (dus in totaal 6 waardes) een bereik opgegeven dat het optimalisatie algoritme afgaat. Op deze manier kan gericht gezocht worden in bepaalde ranges van de *colorspace*. Na een aantal tests konden hele gebieden van de *colorspace* uitgesloten worden. Zo bleek voor YCrCb dat voor de Y waardes de Ymin het beste op 0 kon blijven en Ymax in op 255, dat voor Cr de waardes onder de 120 en boven de 180 niet interessant waren en dat dat gold bij Cb voor waardes onder 70 en boven de 160. De resulterende zoekruimte is klein genoeg om *bruteforce* aan te pakken. Op deze manier zijn de onderstaande "myBest" bereiken gevonden.

LCCS (Khanal & Sibidé):	**log[R/G]**(0.15,1.1)	**log[B/G]**(-4,0.3)	
YCrCb (Chai & Ngan):	**Y**(0,255)	**Cr**(133,173)	**Cb**(77,127)
YCrCb (myBest):	**Y**(0,255)	**Cr**(136,166)	**Cb**(97,144)
HSV (myBest):	**Hue**(0,17)	**Sat**(38,215)	**Val**(59,255)

Tabel 2: De gebruikte bereiken

In tabel 3 worden de resultaten weergegeven die verkregen zijn door vijftien foto's te analyseren met de verschillende methodes. **Avg fdr** is de *average false detection rate*, dit zijn de pixels die als wit geclassificeerd worden (als huid worden gezien) door de methode, maar die zwart zijn (geen huid) in de groundtruth. **Avg cdr** is de *average correct detection rate*, dit zijn de witte pixels de ook wit zijn in de *groundtruth*. **Avg cr** is het totaal aantal pixels dat correct geclassificeerd is, correct wit plus correct zwart gedeeld door het totaal aantal pixels.
Het beste resultaat (dik gedrukt in de tabel) werd bereikt met een *dilation* en *erosion* van 9 pixels en een *closing* van 11 pixels. De foto's die gebruikt zijn in dit deel van het onderzoek bleken het beste resultaat te geven met de YCrCb *colorspace*, hoewel de verschillen klein zijn. Uitgezonderd het verschil met HSV. Later in het project, toen er meer foto's werden gebruikt, bleek dat de LCCS *colorspace* betere resultaten gaf. Hieruit kan geconcludeerd worden dat de keuze voor de *colorspace* en de gebruikte bereiken sterk afhankelijk is van de gebruikte dataset.

5.2 SVM classificatie

Met een *confusion matrix* kan de performance van een algortime gevisualiseerd worden. Op de verticale as staat het gebaar zoals het echt gemaakt is, op de horizontale as staat de classificering van het gebaar door het algoritme. *Precision* geeft weer hoeveel van de classificaties (van één type, bijvoorbeeld rock)

Morph Settings		Literature		myBest	
		LCCS [12]	YCrCb [17]	YCrCb	HSV
Without morph	avg cdr	0.83	0.90	0.88	0.60
	avg fdr	0.12	0.24	0.05	0.08
	avg cr	0.86	0.80	0.93	0.83
erode(13,13)	avg cdr	0.84	0.91	0.89	0.59
dilate(13,13)	avg fdr	0.12	0.26	0.05	0.07
close (31,31)	avg cr	0.87	0.79	0.94	0.83
erode(13,13)	avg cdr	0.83	0.90	0.88	0.57
dilate(13,13)	avg fdr	0.11	0.24	0.04	0.06
close (16,16)	avg cr	0.87	0.80	0.94	0.84
erode(9,9)	avg cdr	0.83	0.90	**0.88**	0.58
dilate(9,9)	avg fdr	0.11	0.24	**0.04**	0.06
close (11,11)	avg cr	0.87	0.80	**0.94**	0.83
erode(5,5)	avg cdr	0.85	0.91	0.90	0.62
dilate(5,5)	avg fdr	0.12	0.26	0.05	0.08
close (21,21)	avg cr	0.87	0.79	0.94	0.83

Tabel 3: Skindetection Results Table.

ook daadwerkelijk van dat type zijn. Als er bijvoorbeeld ook een aantal spocks als rock geclassificeerd worden gaat de precision naar beneden. Met *recall* kan worden gezien hoeveel van de gebaren geclassificeerd zijn als het gebaar dat het is. Met andere woorden, hoeveel van de rocks ook geclassificeerd zijn als rock. Als er een rock incorrect als lizard geclassificeerd wordt dan gaat de recall van rock omlaag. Je kan het zien als: *precision* gaat over de classificaties en *recall* gaat over de daadwerkelijke gebaren. In de ideale situatie wordt ieder gebaar correct geclassificeerd, in de *confusion matrix* zal dan alleen de diagonaal nummers bevatten en *precision* en *recall* is dan overal precies één.
De *total* aan de rechterzijde van de matrix geeft weer hoeveel gebaren er van een bepaald type in de dataset aanwezig zijn. De *total* aan de onderzijde geeft weer hoeveel van een bepaald type geclassificeerd zijn door het algoritme. Op de kruising van de twee staat de optelling: het totaal aantal items in de dataset. De *accuracy* geeft in één getal de weer hoe goed de gebruikte methode presteert. Dit getal is het totaal aantal juist geclassificeerde gebaren, de aantallen op de diagonaal, gedeeld door het totaal aantal aangeboden gebaren. Dit getal staat in de matrix op rechtsonder met "acc:" ervoor.

5.2.1 YCrCb

Hieronder in tabel 4 een *confusion matrix* met daarin de resulaten die verkregen zijn bij gebruik van de YCrCb *colorspace*. Van de foto's wordt 85% correct geclassificeerd bij gebruik van deze *colorspace*.

		Classified as				
		rock	**paper**	**scissors**	**total**	**recall**
True gesture	**rock′**	25	0	0	25	1.000
	paper′	4	19	2	25	0.760
	scissors′	2	3	20	25	0.800
	total	31	22	22	75	
	precision	0.806	0.864	0.909		**acc: 0.85**

Tabel 4: Confusion Matrix - YCrCb

5.2.2 LCCS

Hieronder in tabel 5 een *confusion matrix* met daarin de resulaten die verkregen zijn bij gebruik van de LCCS *colorspace*. Bij deze *colorspace* wordt 100% van de gebaren correct geclassificeerd.

		Classified as				
		rock	**paper**	**scissors**	**total**	**recall**
True gesture	**rock′**	25	0	0	25	1.000
	paper′	0	25	0	25	1.000
	scissors′	0	0	25	25	1.000
	total	25	25	25	75	
	precision	1.000	1.000	1.000		**acc: 1.0**

Tabel 5: Confusion Matrix - LCCS

5.2.3 Resultaten met alle vijf de gebaren - Linear kernel

Een SVM die gebruik maakt van de *linear* kernel kan de vijf de gebaren niet goed van elkaar onderscheiden. De klasse van rock wordt niet genoeg onderscheiden van lizard en hetzelfde geldt voor de klassen scissors en spock. Hieronder de resulaten met vijf gebaren, zie tabel 6. Het aantal juist geclassificeerde gebaren is hier 45%.
Zoals gezien kan worden in de *confusion matrix* worden alle rocks als lizard geclassificeerd en visa versa. Dit komt omdat de SVM deze twee gebaren niet van elkaar kan onderscheiden en dat er bij *kfolds* altijd met één minder gebaar getraind wordt van het type dat er in die *kfold* geclassificeerd wordt. Dus als er een rock geclassificeerd wordt dan heeft de SVM getraind met 24 rocks, 25 papers, 25 scissors, 25 lizards en 25 spocks. Waardoor de weegschaal voor lizard of rock in dit voorbeeld doorslaat naar lizard. Het blijkt dat dit verschil van één bij het trainen genoeg invloed heeft dat het te classificeren gebaar geclassificeerd wordt als het gebaar waarvan er 25 aanwezig zijn in de trainingsset.

		Classified as						
		rock	**paper**	**scissors**	**lizard**	**spock**	**total**	**recall**
True gesture	**rock′**	0	0	0	25	0	25	0.000
	paper′	0	22	0	0	3	25	0.880
	scissors′	0	0	25	0	0	25	1.000
	lizard′	25	0	0	0	0	25	0.000
	spock′	0	4	11	1	9	25	0.360
	total	25	26	36	26	12	125	
	precision	0.000	0.846	0.694	0.000	0.750		**acc: 0.45**

Tabel 6: Confusion Matrix - Linear Kernel

5.2.4 Resultaten met alle vijf de gebaren - RBF kernel

Bij een SVM die gebruik maakt van de *Radial Basis Function* (RBF) kernel gaat het aantal juist geclassificeerd gebaren naar 75%. Spock blijkt het moeilijkste gebaar om te classificeren; het heeft de laagste *precision* en de laagste *recall*. Wat misschien niet zo verwonderlijk is omdat dit het enige gebaar is dat dicht

in de buurt komt van twée andere gebaren: paper en scissors. Het wordt zelfs een keer geclassificeerd als lizard!

		Classified as						
		rock	**paper**	**scissors**	**lizard**	**spock**	**total**	**recall**
True gesture	**rock′**	16	0	0	9	0	25	0.640
	paper′	0	19	0	0	6	25	0.760
	scissors′	0	0	24	0	1	25	0.960
	lizard′	4	0	0	21	0	25	0.840
	spock′	0	5	7	1	12	25	0.480
	total	20	24	31	31	19	125	
	precision	0.800	0.792	0.774	0.677	0.632		**acc: 0.74**

Tabel 7: Confusion Matrix - RBF Kernel

6 Conclusies & Discussie

In dit deel worden de resultaten besproken, wat de leeraspecten zijn geweest van dit bachelor eindwerkstuk en wat de toekomstige werkzaamheden zijn bij een eventueel vervolg van dit onderzoek.

6.1 Programmeertaal

Voordat ik begon aan deze opdracht had ik het idee om de computervision code te schrijven in Python. Maar omdat de C++ interface de native interface is en niet alle mogelijkheden beschikbaar zijn in de Python interface, heb ik besloten om het OpenCV gedeelte van het project in C++ te schrijven. Achteraf gezien erg leuk, ik had voorheen slechts in één project eerder met deze programmeertaal gewerkt. Door dit onderzoek heb ik veel geleerd op het gebied van C++, met name het concept van pointers was interessant en zeer handig om te leren voor toekomstige programmeerprojecten.

6.2 Huidherkenning

Het is moeilijk om de hand te isoleren in alle mogelijke lichtomstandigheden. Vooral kunstlicht met een gebroken wit plafond bleek uitdagend. Voor vervolgonderzoek zou ik een beter algoritme voor het detecteren van huid willen ontwikkelen, bijvoorbeeld door een calibratiestap toe te voegen, gebruik te maken van background substraction of door het toepassen van een algoritme dat gebruik maakt van de kans dat een pixel een huidpixel is. Voor mijn dataset bleek de LCCS *colorspace* het beste te werken. Bij een ander algoritme zou het best eens kunnen dat YCrCb of zelfs HSV ook goed werken.

6.3 Android applicatie

De ontwikkeling van de Android applicatie valt buiten het kader van dit onderzoek. Wel kan uit de eerste experimenten bij de bouw van de applicatie opgemaakt worden dat het werkend krijgen van een ontwikkelomgeving geen sinecure is. Zo moet er gelet worden om de juiste plugins te installeren van de *Eclipse Marketplace*: 'Andmore Development Tools for Android' & 'Android ADT Extensions'. Een ander belangrijk punt is dat de standaardgeheugen-instelling van de *Java Runtime* moet worden uitgebreid naar -Xms512m & -Xmx1024m in het jmc.ini bestand uit de *bin-folder* van de *Java JDK* en niet alleen, zoals veel *support websites* vermelden, in de eclipe.ini van de *Eclipse* installatie.
Voor het renderen van een *preview* was het voor de gebruikte omgeving noodzakelijk om dit te doen voor API versie 17, ondanks dat de *target-build* op API 25 stond. Het draaien van de applicatie in een virtuele omgeving werkte zeer langzaam en was geen optie om te gebruiken in de ontwikkeling. De *workaround* was het maken van een installatie *package*, die te sturen naar een telefoon en daar de applicatie te installeren en te testen.
De keuze om te werken in de *Eclipse IDE* is gestuurd door het voorbeeldproject dat beschikbaar is vanuit het *Open Source Computer Vision* project. Het is interessant om te onderzoeken of het mogelijk is om te werken met OpenCV in de officiële IDE voor Android: *Android Studio*.

6.4 Vervolg

Eén van de openstaande punten van de implementatie is het toevoegen van de laatste twee gebaren aan het spel zonder een daling van de *accuracy*. Het zal nog een uitdaging zijn om spock van scissors te onderscheiden en lizard van rock. Er zullen extra onderscheidende eigenschappen, vertaald naar een getal in de *featurevector*, gevonden moeten worden. Denk hierbij aan de dikte (één of twee vingerdiktes) van de delen naast een *convexity defect* (voor spock) of aan het verloop van hoek van de raaklijn waarmee het meer stompe van rock van het meer langwerpige van lizard onderscheiden kan worden. Een ander openstaand punt is het maken van een Android applicatie zoals bij de start van project het plan was. Het lijkt me nog steeds erg leuk om dit spel te kunnen spelen via mijn telefoon met bijvoorbeeld iemand aan de andere kant van de wereld.

Referenties

[1] Rock Paper Scissors Lizard Spock by sam kass. http://www.samkass.com/theories/RPSSL.html. Accessed: 2016-10-15.

[2] De uitleg van Sheldon Cooper in The Big Bang Theory op youtube.com. https://youtu.be/x5Q6-wMx-K8. Accessed: 2016-10-15.

[3] OpenCV - open computer vision. http://opencv.org/. Accessed: 2016-10-15.

[4] A. Ya. Lerner V.N. Vapnik. Recognition of patterns with help of generalized portraits. *Avtomat.i Telemekh.*, 24(6):774–780, 1963.

[5] RPS tegen de computer op youtube.com. https://youtu.be/mF5kyLTczuE. Accessed: 2016-10-15.

[6] Haarclassifiers voor rock, paper & scissors op github. https://github.com/Aravindlivewire/Opencv/tree/master/haarcascade. Accessed: 2016-10-15.

[7] Paul A. Viola and Michael J. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. In *Computer Vision and Pattern Recognition*, volume 1, pages 511–518, 2001.

[8] Computer die altijd wint op youtube.com. https://github.com/Aravindlivewire/Opencv/tree/master/haarcascade. Accessed: 2016-10-15.

[9] RPS op Android op youtube.com. https://youtu.be/3nxjjztQKtY. Accessed: 2016-10-15.

[10] Paper van RPS op Android op idc.ac.il. http://www1.idc.ac.il/toky/ImageProcAndroid-14/RockScissorsPaper/Report.pdf. Accessed: 2016-10-15.

[11] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *Information Theory, IRE Transactions on*, 8(2):179–187, February 1962.

[12] Bishesh Khanal and Désiré Sidibé. Efficient Skin Detection Under Severe Illumination Changes and Shadows. In *ICIRA 2011 - 4th International Conference on Intelligent Robotics and Applications*, pages 1–8, Aachen, Germany, December 2011.

[13] George H. Joblove and Donald Greenberg. Color spaces for computer graphics. *SIGGRAPH Comput. Graph.*, 12(3):20–25, August 1978.

[14] Moments in Image Processing by robert bailey 2002. http://www.csie.ntnu.edu.tw/~bbailey/Moments%20in%20IP.htm. Accessed: 2016-10-15.

[15] Haar-Like Features op links.springer.com. http://link.springer.com/content/pdf/10.1007/978-0-387-73003-5_258.pdf. Accessed: 2016-10-15.

[16] GIMP - gnu image manipulation program. `https://www.gimp.org/`. Accessed: 2016-10-15.

[17] D. Chai and K.N. Ngan. Face segmentation using skin-color map in videophone applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(4):551–564, 1999.