Ansiktsigenkänning, TNM034

Pelle Serander, Erik Olsson, Torsten Gustafsson, Marcus Lilja

13 december 2015

Sammanfattning

Den här rapporten beskriver en implementation av ansiktsigenkänning skriven i *Matlab*. En godtycklig bild på ett ansikte skickas in i programmet och jämföras med andra bilder i en databas. För att känna igen ett ansikte måste ansiktet först separeras från andra objekt i bakgrunden. Efter det används metoden *egenansikten* för att avgöra vilken person ansiktet tillhör. Förvrängningar appliceras som skalning, rotation, färgintensitet och placering, vilket programmet ska kompensera för.

Hur detta har implementerats beskrivs under Metod i denna rapport. Efteråt diskuteras resultatet och hur väl denna implementation fungerar samt vilka brister den har. Vår implementation klarar till exempel inte allt för kraftigt vinklade ansikten eller stora ändringar i ansiktsuttryck.

Innehåll

Sammanfattning i

1 Inledning 1

2 Bakgrund 2

3 Metod 3

3.1 Övergripande struktur . . . 3
3.2 Ljuskorrigering . . . 3
3.3 Hudidentifiering . . . 4
3.4 Framtagande av munmask . . . 5
3.5 Framtagande av ögonmask . . . 6
3.6 Rotation . . . 7
3.7 Ljusintensitetskorrigering . . . 8
3.8 Egenansikten . . . 8
3.8.1 Generera databasens egenansikten . . . 8
3.8.2 Jämförelse mellan databas och nytt ansikte . . . 9

4 Resultat 10

5 Diskussion 11

Kapitel 1

Inledning

Ansiktsigenkänning är ett populärt ämne inom datorseende och används mycket inom kommunikationssystem och passagekontroller [1]. Oftast går processen ut på att registrera ett ansikte på en digital bild, bearbeta ansiktet för att få fram den viktigaste informationen och jämföra den slutliga ansiktsbilden men en databas. Metoden för att upptäcka ett ansikte och sedan hitta rätt ansikte i databasen är långt ifrån enkel att tillämpa då det finns en rad olika faktorer som påverkar resultatet.

Kapitel 2

Bakgrund

För att känna igen ett ansikte krävs en databas över vilka ansikten som är kopplade till vilken person. Syftet med den här rapporten är att beskriva tillvägagångsättet för utvecklingen av ett ansiktsigenkänningsystem som utgår från en databas med en mängd olika bilder tagna på 16 olika personer.

Utvecklingen skedde i Matlab med kravet att ett porträtt skulle läsas in i ett program som sedan skulle returnera ett databas-id på personen, beroende på om personen tillhör databasen eller inte. Programmet skulle vara någorlunda stabilt och kunna känna igen ansikten som har blivit roterade ($\pm$ 5 % grader), skalade ($\pm$ 10 %), translaterade eller om bildens tonvärden blivit förvrängda ($\pm$ 30 %).

Kapitel 3

Metod

Här beskrivs tillvägagångssättet för implementation av systemet. Under ansiktsigenkänningen sker först en ljuskorrigering som förenklar framtagandet av ansiktsmasken, som i sin tur används för att plocka fram en *munmask* samt en *ögonmask*. Efter det används egenansikten för att avgöra vilken person ansiktet tillhör.

3.1 Övergripande struktur

Programmets funktionalitet är uppdelad i ett antal filer. De flesta funktioner är egenskrivna medan andra är hämtade från fria källor på internet. För igenkänningen används så kallade egenansikten. Innan programmet körs förebereds bilddatabasen genom att generera egenansikten i funktionen *initDB*. Detta görs i ett förberäkningspass eftersom det skulle ta för lång tid att räkna ut annars.

Efter att funktionen *initDB* körts så sparas en *.mat*-fil med de relevanta vikterna för att göra jämförelserna som behövs. När funktionen *tnm034* körs skickas en bild in och returnerar vilken person bilden föreställer, förutsatt att den hittas i databasen. *tnm034* använder funktionerna *faceDetection* och *compareToDB* för att upptäcka ansiktsområden samt jämföra dessa med databasen.

3.2 Ljuskorrigering

Foton som tagits med olika förutsättningar som exempelvis olika kameror, belysning eller slutartid kommer att resultera i bilder med olika färgtemperaturer. Det är viktigt i bildanalyssammanhang att alla bilder har samma relativa färgnyans eftersom jämförelser mellan olika bilders färger kommer utföras.

Nyansskillnaderna kan korrigeras via en process som kallas för *vitbalans*. Det finns flera olika metoder för vitbalans, men i detta projektet användes en metod baserad på linjär skalning av bildens färgkanaler [2].

Ett medelvärde för alla pixlar i bilden beräknas för varje färgkanal (i detta fall röd, grön och blå). Dessa medelvärden divideras med medelvärdet av bildens gråvärden, *I*. På så sätt kan en skalfaktor för varje färgkanal bestämmas, se ekv. 3.1. Om skalfaktorerna appliceras på ursprungsbilden kommer dess färgvärden bli neutrala i alla färgkanaler.

$$[S_r, S_g, S_b] = \frac{[r, g, b]}{\overline{I}} \tag{3.1}$$

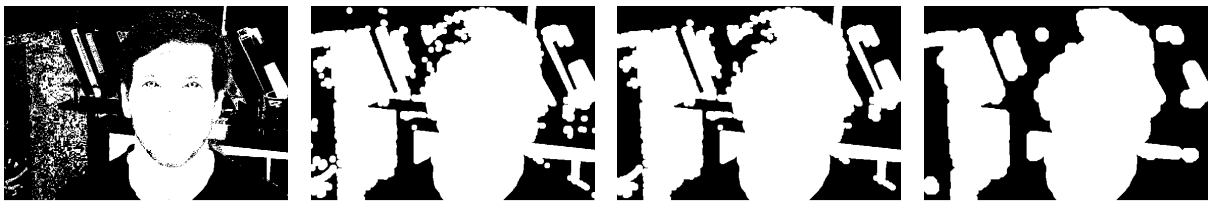
(a) Original bild. (b) Vitbalanserad bild

Figur 3.1: Ljuskorrigering enligt funktionen *whiteBalance*

3.3 Hudidentifiering

Syftet med hudidentifiering är att skapa en så kallad *hudmask* där all hud ska vara vit och allt övrigt ska vara svart. Detta leder till att ansiktet kan maskeras och blir lättare att identifiera. För att skapa hudmasken konverteras den vitbalanserade bilden till färgrymden YCbCr [4]. Varje värde från Cb och Cr-kanalerna jämförs med ett särskilt färgintervall som ska representera hud. Om krominansvärdena ligger inom intervallet sparas positionen för pixelen i en matris. En helt svart bild skapas som är lika stor som orginalbilden och med hjälp av positioner (från matrisen) sätts pixelvärdena i den svarta bilden till 1. Alltså kommer alla område som har identifierats som hud vara vitt och resten vara svart, se figur 3.2a. En brusreducering görs med morfologiska operationer vilket kommer göra det lättare att maskera ansiktet, se figur 3.2b

Om det finns områden i bilden som inte är hud men har en färg som ligger inom hudintervallet kommer även dessa område identifieras som hud och sättas till vitt. Därför måste dessa områden maskeras bort för att ansiktet ska kunna identifieras. Små områden i form av "öar" tas bort med hjälp av matlabfunktionen `bwareaopen` som tar bort alla vita områden som har en area som är mindre än 3.3 % av bildens area, se figur 3.2c. En ny brusreducering utförs genom att först utföra en erodering på vita områden och sedan förstora dessa område med dilation, se figur 3.2d.



(a) Första hudmasken (b) Brusreducerad hudmask (c) Borttagning av vita regioner (d) Erodering och dilation

Figur 3.2: Ansiktsmaskering

Om en bild skulle innehålla flera ansikten måste detta undersökas, dvs varje vitt område i hudmasken måste kontrolleras om det är ett ansikte eller inte. Det bestämdes att ansikten vars area är mindre än 2 % av totala bildarean inte kommer att bearbetas. För att identifiera potentiella ansikten används

matlabfunktionen `regionprops(image,'BoundingBox')` [3]. Denna funktion returnerar den minsta rektangeln som kan innehålla ett antal vita regioner, se fig 3.3a. Om rektangelns area är större än 2 % av bildens area beskärs den inlästa bilden efter rektangelns position och storlek. Den beskurna bilden kommer i detta skede representera en färgbild med ett identifierat ansikte. Samma procedur för att identifiera hud och filtrera bor vita område som gjordes i början utförs på den beskurna bilden för att beräkna en ny ansiktsmask.

Det finns bilder som har tagits med för hög eller för låg exponering vilket gör att hudfärgen på bilden blir förvrängd och hamnar utanför hudfärgsområdet när hudindetifieringen beräknas. Det leder till att det kan bli stora svarta partier i den slutliga ansiktsmasken, se figur 3.3b. Lösningen på att fylla ut ansiktsmasken gjordes genom en funktion som heter *FitElipse*. FitElipse använder minsta kvadratmetoden för att uppskatta en ellips baserat på den vita regionen i ansiktsmasken [5].

För att beräkna ellipsens koefficienter a och b och centrumpunkten (h, k), se ekvation 3.2, utgår funktionen från en kvadratisk ekvation enligt ekvation 3.3 där $\mathbf{x}$ är koordinaterna för den vita regionen i ansiktsmasken. Genom att införa ett koordinatbyte i ekvation 3.3 kan a, b och centrumpunktens koordinater lösas ut och därefter kan ellipsen beräknas med ellipsens ekvation 3.2.

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1 \tag{3.2}$$

$$\mathbf{x}^{\mathrm{T}}\mathrm{A}\mathbf{x} + \mathrm{b}^{\mathrm{T}}\mathbf{x} + \mathrm{c} = 0 \tag{3.3}$$

Ellipsens centrum kommer att ligga i ansiktsmaskens centroid [6] eftersom dessa två punkter adderas. Eftersom masken kan innehålla vita regioner som genererats från bakgrunden som har liknat hud, ersätts masken med ellipsen då den endast kommer täcka upp ansiktets område, se fig. 3.3c. Ifall ellipsen skulle bli för liten dileras den för att försäkra att ögon och mun hamnar innanför ellipsen. Till sist skickas den beskurna färgbilden och ellipsmasken till *mouthDetection*-funktionen.

(a) Identifierade ansikten (b) Beskuren faceMask (c) Maskerad ellips

Figur 3.3: Beskärning

3.4 Framtagande av munmask

För att hitta munnens område utnyttjar man att ett område innanför ellipsen kommer ha en rödare nyans jämfört med resten av bilden [4]. Alltså kommer *Cr*-kanalen ha ett betydligt större värde än *Cb*-kanalen. Med hjälp av ekvation 3.4 beräknas skillnaden mellan bildens rödaste regioner och bildens

minst röda regioner (*Cr*/*Cb*). Här motsvarar n bildens totala antal pixlar enligt ekvation 3.5. Resultatet blir en gråskalebild där munområdet bör ha den högsta ljus-intensiteten. Denna bild visas i figur 3.4a.

$$mouthMap = Cr^2 \cdot (Cr^2 - \eta \cdot \frac{Cr}{Cb})^2 \tag{3.4}$$

$$\eta = 0.95 \cdot \frac{\frac{1}{n}\sum Cr^2}{\frac{1}{n}\sum \frac{Cr}{Cb}} \tag{3.5}$$

För att ta bort övriga högintensitetområden som kan ha bidragit från bakgrunden multipliceras munmappen med ansiktsmasken. Högintensitetsområden som finns kvar utvidgas för att förtydliga munnens region. Via en loop trösklas varje intensitet i den nya munmasken (från 0.4 till 0.1) tills munnen har en area som är minst 0.28 % (minsta munarea) av munmaskens pixelarea . I loopen sätts först alla pixelvärden till 0 som ligger nedanför 60 % av bildens höjd, se bild 3.4b, därefter görs trösklingen med loopens aktuella tröskelvärde, erodering och dilation utförs. Till sist jämförs summan av alla nollskilda pixlar från munmasken med minsta munarean. Om antalet nollskilda pixlar är större än minsta munarean bryts loopen.

Trots trösklingen och maskeringen kan det fortfarande finnas oönskade vita regioner kvar, till exempel att munnen består av många små vita regioner eller att näsan har bidragit till masken. Målet är att det bara ska finnas en vit region som representerar munnens område, därför behöver ytterligare en filtrering göras. Detta görs genom att ta bort regioner som är mindre eller lika med minsta munarean. För att få en sammanhängande vit region för munnen körs funktionen `imFill` ifall det skulle finnas ett svart hål i den ”vita” munnen vilket slutligen ger figur 3.4c. Det sista som görs är att beräkna koordinaterna för centrumpunkten i den vita regionen och skicka dem till funktionen *eyeDetection*.

(a) Munmmap i gråskala (b) 60% av munmaskens övre del har svartas ut (c) Munmask efter dilation och tröskling

Figur 3.4: Berarbetning av munmasken

3.5 Framtagande av ögonmask

För att upptäcka ögon i ansiktet så används en metod som först gör om bilden till färgrymden YCbCr för att sedan räkna ut två ögonmasker, en krominansmap och en luminansmap enligt Rein-Lien Hsu:s

metod [4]. Luminansmap och krominansmap räknas ut enligt ekvationer 3.6 och 3.7. Figur 3.5a och 3.5b visar hur dessa kan se ut i en typisk färgbild.

$$EyeMapC = \frac{1}{3}(Cb^2 + Cr^2 + Cb/Cr) \tag{3.6}$$

$$EyeMapL = \frac{Y \oplus G}{Y \ominus G + 1} \tag{3.7}$$

Den slutgiltiga ögonmasken fås när ansiktsmasken, luminansmappen och krominansmappen multipliceras med varandra, se figur 3.5c. Då erhålls en ögonmask i gråskala där ögonområdena bör ha högst intensitet av alla bildens pixlar. Eftersom ögonens intensitet kan variera mellan bilder testas olika tröskelvärden, från 100 % intensitet och nedåt, tills två ögon har hittats, se figur 3.5d. För att ytterligare öka chansen att hitta korrekta ögonpunkter så konstruerades ytterligare en mask för att begränsa sökområdet. Det är på grund av denna mask som ögat "klipps"i figur 3.5d.

Ibland kan det hända att algoritmen hittar fler än två ögon trots den låga tröskelförändringen. I det fallet körs vissa tester på ögonen för att avgöra vilka två som är de korrekta. Exempelvis vet vi att ögonen måste vara på varsin sida om ansiktet. Om två möjliga ögon är väldigt nära varandra så slås dessa områden ihop till ett och jämförelse sker med det nya ögat istället.

(a) Krominans bild med histeq

(b) Normaliserad luminans bild

(c) Kombinerad ögonmask med dilation och ansiktsmask på

(d) Slutgiltig ögonmask efter tröskling

Figur 3.5: Ögonmasken beräknas i flera steg.

3.6 Rotation

Efter att munnen och ögonen har hittats så skickas deras positioner in i funktionen *triangulateFace* som har som uppgift att räkna ut vilken vinkel som behövs för att rotera bilden så att ögonen ligger längs med den horisontella axeln. Detta görs genom att beräkna skalärprodukten mellan en normaliserad vektor från vänster öga till höger öga och horistontalaxeln enligt ekvation 3.8.

$$\cos(\alpha) = \widehat{v} \cdot \widehat{x} \tag{3.8}$$

Bilden roteras därmed tillbaka med den funna vinkeln. En ny upprätad bild tas då fram med hjälp av bi-kubisk interpolation.

3.7 Ljusintensitetskorrigering

För att hantera att bilderna kunde ha olika ljusintensitet så konverterades bilden till färgrymden YCbCr och sedan användes matlabfunktionen `histeq` för att normalisera bildernas luminanskanal, se figur 3.6a och 3.6b.

(a) Orginalbild (b) Bild med normaliserad luminans kanal

Figur 3.6: Genom att normalisera luminansen så får bilden en högre kontrast.

3.8 Egenansikten

Ett *egenansikte* (eng. eigenface) är ett standardansikte som skapats utifrån flera ansikten i en databas och kan användas för jämförelser mellan ansikten [7]. Med tillräckligt många bilder i databasen kan ett tillräckligt precist standardansikte tas fram, som innehåller ett typiskt ansiktes viktigaste egenskaper.

Denna databas skapas genom att *PCA* (principialkomponentanalys) utförs på alla bilder i databasen och en serie egenvärden för varje ansikte beräknas. Dessa egenvärden utgör en minimalistisk representation av respektive ansikte och fångar dess viktigaste egenskaper. Egenvärden för nya ansikten kan beräknas med samma metod och jämförs med databasens egenvärden för att hitta det ansiktet som är mest likt den nya bilden.

3.8.1 Generera databasens egenansikten

Alla bilder läses in i databasen och vitbalanseras. För att bilderna ska kunna jämföras skalas de om till samma storlek, vilket gör jämförelseprocessen skalningsinvariant. För att få fram de karaktäristiska komponenterna för ett ansikte subtraheras varje ansikte med medelvärdet utav alla ansikten.

Nästa steg är att beräkna alla egenvärden och egenvektorer för varje bild. Detta görs utifrån kovariansen av alla ansikten. Det karaktäristiska drag som är intressant för varje komponent är de med stor varians. Antal egenvärden kommer bli samma som antalet bilder i databasen. Efter PCA så kommer de relevanta egenvärdena ha minskat och dessa används sedan som våra vikter vid mätning av likheten på vår testbild.

Databasen med de ursprungliga bilderna ifrån databasen beräknas i ett förpass och sparas i en *.mat*-fil. Detta är för att snabba upp jämförelseprocessen, eftersom det ofta blir långa beräkningstider för att skapa databasen med egenansikten.

3.8.2 Jämförelse mellan databas och nytt ansikte

Efter att databasen med egenansikten har tagits fram kan denna användas för att jämföra nya ansiktsbilder med databasen. Den nya bildens vikter beräknas på samma sätt som för ansikten i databasen. De euklidiska avstånden mellan den nya bildens vikter och de ifrån databasen jämförs. Det minsta avståndet ger det ansiktet i databasen som är mest likt det nya ansiktet. Ifall skillnaden mellan dessa skulle vara mindre än ett givet tröskelvärde stämmer ansiktsbilderna tillräckligt bra överens med varandra för att säkerställa att de visar samma person.

Kapitel 4

Resultat

Resultatet fungerar väl för bilder med enklare bakgrund. Det kan dock skilja sig avsevärt beroende på vilka förvrängningar som har applicerats på bilden. Exempelvis så kan en rotation ge en avsevärt förändrad ansiktsmask vilket i sin tur ändrar hur den slutgiltiga beskärningen av bilden blir. Kravet med att systemet ska vara stabilt är uppfyllda för fleratalet av bilderna.

Tabell 4.1 är baserad på en databas med 16 bilder som har transformerats på olika sätt och sedan jämförts med vår databas. En intressant egenskap som framkom under tester är att metoden som används inte ger några falska positiva resultat, det vill säga att en person skulle identifieras som någon annan, under genomförda tester. Andelen falska negativa resultat är i de flesta fall relativt låg, det är främst vid nedskalning av bilden som det uppstår problem.

Tabell 4.1: Tabell över hur bilderna i DB1 lyckas

Rotation	Falsk positiv	Falsk negativ	Lyckade
+5 °	0%	18.75%	81.25%
-5 °	0%	18.75%	81.25%
Ljusförvrängning	-	-	-
+30	0%	18.75%	81.25%
-30	0%	25%	75.00%
Skalning	-	-	-
+10%	0%	6.25%	93.75%
-10%	0%	50.00%	50.00%

Kapitel 5

Diskussion

Det var svårt att få programmet till att returnera bra resultat för alla bilder i databasen. Ansikten med mycket bakgrund var svårast att känna igen eftersom *skinDetection*-funktionen kan hitta ansikten i bakgrunden där det egentligen inte finns några. Lösningen för detta är som tidigare nämnt att filtrera bort så mycket vita regioner som möjligt. Problemet är att på vissa bilder kan det filtreras bort för mycket och på andra för lite. Därför krävdes det mycket testning av olika bilder för att komma fram till bra tröskelvärden. I början uppstod det problem med att antingen ögonen eller munnen filtrerades bort och därmed kunde inte processen fortsätta. Detta löstes genom att maskera bort den övre halvan av bilden för att inte få några onödiga bidrag när munnen skulle hittas. Den nedre halvan från munnen maskerades bort när ögonen skulle hittas, dessutom så konstruerades ytterligare en mask för att ytterligare förfina sökområdet.

I början av projektet fanns det aldrig en funktion för att kontrollera vilken ljusintensitet mun eller ögon skulle ha för att bli identifierade. I stället fanns det bara en tröskel som bestämde att så länge intensiteten var högre än 28 % kunde det vara en mun eller ett öga. Detta ledde till stora problem eftersom ögonen kunde hittas på helt fel ställen. Även om den nya algoritmen för att hitta ögon blev bättre än att använda en tröskel finns det inget krav för att ögonen ska ligga i samma horisontella linje, istället körs algoritmen tills det bara finns korrekta vitområde kvar med kravet att det ska finnas möjliga punkter på båda sidor av ansiktet. Detta är dock inget som senare tas hänsyn till då algoritmen baserar sig på avståndet till munnen. Här kan problem uppstå ifall två punkter på samma sida av ansiktet används då rotationen är beroende av dessa.

I *generateSkinMap*-funktionen finns det en tröskel för att bestämma vilka färger som ska representera hud. Vi valde att ha en relativt smal tröskel för att varken för mörka eller ljusa färgområden ska identifieras som hud. Detta blir ett problem i högexponerade och lågexpenerade bilder. Vissa bilder är så ljusa att huden nästan är vit vilket gör det svårt att få ansiktsmasken att hamna rätt då centroidpunkten inte hamnar i mitten av ansiktet.

Vår metod genererar sällan falska positiva resultat vilket är önskvärt då det i praktiska sammanhang innebär att en olegitimerad person inte kommer godkännas. Den får dock problem att identifiera personer då bilden har skalats ned, vilket kan bero på att upplösningen på ansiktet sjunker. Ett problem som inte hanteras väl av vår algoritm är icke-kvadratiska bilder. Efter att ansiktet hämtats kommer bilden skalas om till en kvadratisk bild vilket kan sträcka ut bilden på oönskade sätt. Det kan också vara en bidragande orsak till de dåliga resultaten vid nedskalingen.

Litteraturförteckning

[1] Inseong Kim, Joon Hyung Shim, and Jinkyu Yang *Face detection* 2003 https://web.stanford.edu/class/ee368/Project_03/Project/reports/ee368group02.pdf

[2] Matlab Central, Kye Taylor, *Whitebalance function* http://se.mathworks.com/matlabcentral/fileexchange/41089-color-balance-demo-with-gpu-computing/content/GPUDemoCode/whitebalance.m

[3] MathWorks, *regionprops* http://se.mathworks.com/help/images/ref/regionprops.html

[4] Mohamed Abdel-Mottaleb, Anil K. Jain, Rein-Lien Hsu, *Face Detection in Color Images* http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1000242

[5] Walter Gander Gene H. Golub Rolf Strebel, *Least-Squares Fitting of Circles and Ellipses,* December 1996 http://www.emis.de/journals/BBMS/Bulletin/sup962/gander.pdf

[6] Wolfram MathWorld *Geometric Centroid* http://mathworld.wolfram.com/GeometricCentroid.html

[7] *Eigenface Tutorial, Santiago Serrano* www.pages.drexel.edu/~sis26/Eigenface%20Tutorial.htm