Ansiktsigenkänning - Metoder och dess appliceringar

Andrea Åstrand, Jesper Larsson, Jonatan Ebenholm och Tobias Svensson
[andas158, jesla966, joneb274 och tobsv650]

28 maj 2026

1 Introduktion

Ansiktsigenkänning är en snabbt växande teknologi som kombinerar datorseendemetoder med maskininlärning för att kunna identifiera, verifiera och urskilja människor baserat på deras ansiktsdrag. Teknologins användningsområden är ständigt expanderande, där några av dagens tillämpningar är inom säkerhet, övervakning och användarautentisering. Alla appliceringar av ansiktsigenkänning kräver ett robust, träffsäkert och generellt system, då felbedömningar är en risk för säkerhet och integritet. Denna rapport ämnar att presentera en implementering av diverse metoder som sammansatt bygger upp en modell som identifierar ett ansikte från en databas, samt urskiljer ansikten som ej innefattas i databasen.

2 Metod

Det system denna rapport redogör för kräver flertalet metoder för att finna ansikten inom bilder - extrahera dess karaktäristik samt matcha dem mot bilder i en databas. Detta kapitel presenterar de olika metoder som används för projektets mål, redogör för hur de används och förklarar hur de kombineras med varandra för att få ett robust system.

2.1 Dataset

Projektet använder sig av tre olika uppsättningar av data. Den första uppsättningen med data, **db0**, innehåller fyra bilder på personer som inte ska ha tillträde i systemet. De sexton bilderna i **db1** är ansikten som bör ha tillträde till systemet, och det är detta dataset som en modell tränas på. Den sista uppsättningen data, **db2**, innehåller bilder av samma personer som i db1, men med svårare tillstånd att bedöma, såsom dålig ljussättning, dålig skärpa, bakgrunder med mycket information och andra ansiktsuttryck.

2.2 Vitpunkts Korrektion

Innan någon form av detektering utförs måste belysningen i bilden tas i hänsyn. Det finns många metoder för att göra detta, och för detta projekt implementerades Gray world och White patch korrekteringsmetoderna. Dock är det endast White patch som används för den slutgiltiga implementationen.

2.2.1 Gray world

Gray world utgår från antagandet att genomsnittet hos intensiteten av alla färgkanalerna bör vara lika. Alltså att medelvärdet hos färgkanalerna i bilden är okromatiska. Detta antagande kan uppnås genom att först ta fram medelvärdena hos färgkanalerna enligt ekvationerna 1, 2 och 3.

$$R_{avg} = \frac{1}{nm} \sum_{x=1}^{n} \sum_{y=1}^{m} R(x, y) \tag{1}$$

$$G_{avg} = \frac{1}{nm} \sum_{x=1}^{n} \sum_{y=1}^{m} G(x, y) \tag{2}$$

$$B_{avg} = \frac{1}{nm} \sum_{x=1}^{n} \sum_{y=1}^{m} B(x, y) \tag{3}$$

där n är bildens horizontella storlek, m är bilden vertikala storlek, R, G och B är färgkanalerna hos bilden och R_{avg}, G_{avg} och B_{avg} är medelvärdet hos färgkanalerna. Relationerna mellan färgkanalerna kan då tas fram genom att använda medelvärdena enligt ekvation 4 och 5.

$$\hat{\alpha} = \frac{G_{avg}}{R_{avg}} \tag{4}$$

$$\hat{\beta} = \frac{G_{avg}}{B_{avg}} \tag{5}$$

där $\hat{\alpha}$ är relationen mellan den gröna kanalen och den röda kanalen och $\hat{\beta}$ är relationen mellan den gröna kanalen och den blå kanalen. Relationerna mellan färgkanalerna används i ekvationerna 6, 7 och 8 för att ta fram dem vitpunkt korrigerade färgkanalerna.

$$\hat{R}(x, y) = \hat{\alpha} R(x, y) \tag{6}$$

$$\hat{G}(x, y) = G(x, y) \tag{7}$$

$$\hat{B}(x, y) = \hat{\alpha} B(x, y) \tag{8}$$

där $\hat{R}$, $\hat{G}$ och $\hat{B}$ är de vitpunkt korrigerade färgkanalerna hos bilden. De korrigerade färgkanalerna kan sättas ihop för att bilda den vitpunkts kompenserade bilden.

2.2.2 White Patch

White patch utgår från antaganded att dem ljusaste pixlarna i bilden är spektral reflektion från ljuskällan. Detta ger att färgen hos dem ljusaste pixlarna i bilden är färgen på ljuskällan. Kompensering av ljuskällan kan uppnås genom att först hitta

dem ljusaste pixlarna i bilden och sedan ta fram deras relation eligt ekvationerna 9 och 10.

$$\tilde{\alpha} = \frac{G_{max}}{R_{max}} \tag{9}$$

$$\tilde{\beta} = \frac{G_{max}}{B_{max}} \tag{10}$$

där G_{max}, B_{max} är värdet hos färgkanalerna ljusaste pixlar, $\tilde{\alpha}$ är relationen mellan max värdet i den gröna och röda färgkanalen och $\tilde{\beta}$ är relationen mellan max värdet i den gröna och blåa färgkanalen. Relationerna mellan maxvärdena insatta i ekvationerna 11, 12 och 13 ger då dem vitpunktskompenserade färgkanalerna.

$$\tilde{R}(x, y) = \tilde{\alpha} R(x, y) \tag{11}$$

$$\tilde{G}(x, y) = G(x, y) \tag{12}$$

$$\tilde{B}(x, y) = \tilde{\alpha} B(x, y) \tag{13}$$

I ekvationerna är $\tilde{R}$, $\tilde{G}$ och $\tilde{B}$ dem vitpunkts korrigerade färgkanalerna. De korrigerade färgkanalerna kan sättas ihop för att bilda den vitpunkts kompenserade bilden.

I detta projekt används Matlabs inbyggda White Patch metod *illumwhite* då den har en inbyggd funktion för att använda en speciferad procent av de ljusaste pixlarna i bilden istället för endast den ljusaste. Detta är användbart ifall bilden är överexponerad och stora delar av bilden är helt vit.

2.3 Huddetektering med YCbCr-färgrymden

YCbCr-färgrymden är en modell som ofta används inom digital video- och bildteknik för att beskriva en bilds färg i tre komponenter: luminans (Y) och två krominanskanaler (Cb och Cr). Luminans representerar bildens ljusstyrka, medan Cb och Cr anger färginformation – specifikt den blå och röda komponenten i relation till ljusstyrkan. Transformeringen från RGB till YCbCr sker med hjälp av specifika matematiska uttryck som tar hänsyn till R, G och B-värden, vilka normalt sträcker sig mellan 0 och 255. Den resulterande Y-komponenten ligger i intervallet 0–255, medan Cb och Cr typiskt har värden mellan 16 och 240. Transformationen ges av ekvationerna 14, 15 och 16:

$$Y = 16 + \frac{1}{256}(65.738R + 129.057G + 25.064B) \tag{14}$$

$$Cb = 128 + \frac{1}{256}(-37.945R - 74.494G + 112.439B) \tag{15}$$

$$Cr = 128 + \frac{1}{256}(112.439R - 94.154G - 18.285B) \tag{16}$$

Där R, G och B representerar de ursprungliga röda, gröna och blå komponenterna i RGB-modellen. Y är luminanskomponenten. Cb är den blå krominanskomponenten, som visar mängden blått i förhållande till luminans, och Cr är den röda krominanskomponenten, som visar mängden röd färginformation i relation till luminansen.

YCbCr-modellen är särskilt lämplig för huddetektering eftersom den separerar ljusstyrka från färginformation, vilket gör det enklare att identifiera hudtoner. Framför allt har Cr-kanalen visat sig vara betydelsefull då den innehåller mycket av den röda färginformation som är typisk för hud. Studier har visat att denna kanal kan bidra till höga detektionsnivåer för hudområden.

I praktiken används tröskelvärden för Y-, Cb- och Cr-komponenterna för att avgränsa områden som troligen är hud. En pixel klassificeras som hud om dess värden ligger inom dessa intervall. Genom att kombinera resultat från de olika komponenterna med logiska operationer skapas en binär mask som markerar hudpixlar i bilden. Denna metod används för att filtrera ut potentiellt falska ögonkandidater, något som är ett av de mest kritiska momenten för ansiktsnormaliseringen[2] .

2.4 Ögondetektering

Den viktigaste delen av ansiktsigenkänningen är ögondetekteringen. Ögonens positioner används som invariabler i många nedanstående sektioner och även en liten skillnad i positionen hos ögonen kan innebära en felidentifiering. Därför testades tre olika metoder för ögondetektering YCbCr metoden, Cirkulär hough transform och en hybrid metod.

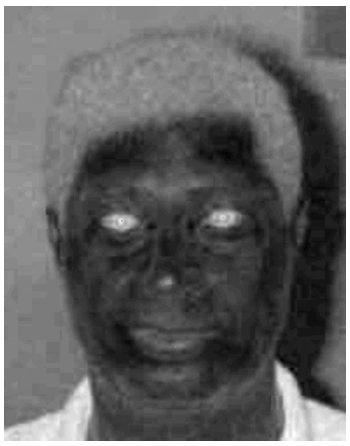
I den slutgiltiga implementationen av projektet användes en kombination av YCbCr metoden och Cirkulär hough transform. YCbCr metoden som tillsammans med ansiktsmasken skapad i föregående sektionen användes för att ta fram ungefärliga positioner av ögonen och den cirkulära hough transformen användes för att hitta mitten av ögonen med bättre precision.

2.4.1 YCbCr Metod

YCbCr metoden som föreslogs av Hsu, Mottaleb och Jain [3] hämtar två separata ögonmappar. Den ena mappen är från krominanskomponenten och den andra är luminanskomponenten hos bilden. Ögonmappen för krominansen beskrivs av ekvation 17, där C_b är den normaliserade blåa färgkanalen hos bilden, C_r är den normaliserade röda färgkanalen och $\bar{C}$ syftar på den negativa röda kroma komponenten vilket beräknas genom ekvation 18.

$$EyeMapC = \frac{1}{3}\{(C_b^2) + (\bar{C_r}^2) + \frac{C_b}{C_r}\} \tag{17}$$

$$\bar{C_r} = 1 - C_r. \tag{18}$$



Figur 1: C_b komponenten.

Figur 2: C_r komponenten.

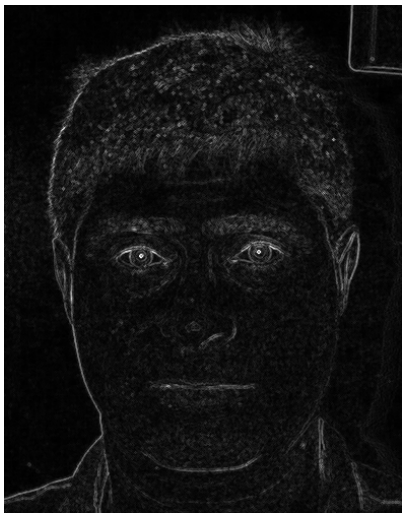
Figur 3: $\bar{C}_r$ komponenten.

Figurerna 1, 2 och 3 visar en viktig aspekt i varför dessa komponenter är eftersedda i denna metod, nämnligen att väldigt höga värden i C_b och väldigt låga värden i C_r är närvarande just i ögonen hos personen i bilden vilket då kan användas för att mappa ögonen. Luminansen hos bilderna används då efter observation syns det att området på ansiktet där ögonen sitter har både höga och låga värden vad det gäller luminans. Med hjälp av morfologiska operationer såsom erosion och dilation kan en ögonmapp skapas. Ögonmappen som föreslogs i [5] beräknas genom ekvation 19, där $Y(x, y)$ är luminans komponenten hos bilden, $g_\sigma(x, y)$ är strukturerings elementet, $\oplus$ och $\ominus$ är morphologisk dilation respektive erosion. Den färdiga ögonmappen görs skapas genom ekvation 20, och efter ännu en morfologisk dilation och sedan en check med ett tröskelvärde skapas en ögonmask $EyeMaskIllu$. Resultatet visas i figur 7.

$$EyeMapL = \frac{Y(x, y) \oplus g_\sigma(x, y)}{Y(x, y) \ominus g_\sigma(x, y)} \tag{19}$$

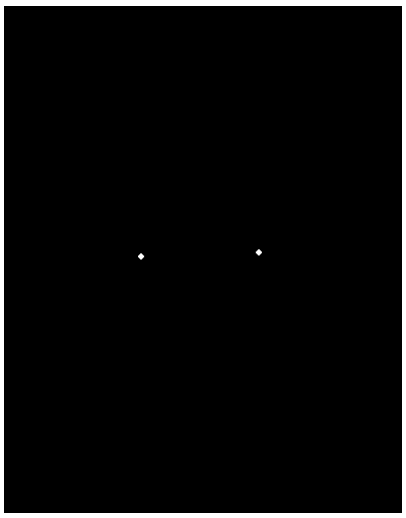
$$EyeMap = EyeMapC * EyeMapL \tag{20}$$

Figur 4: $EyeMapC$.



Figur 5: $EyeMapL$.

Figur 6: $Eye\bar{M}ap$.



Figur 7: $EyeMaskIllu$.

2.4.2 Circular Hough transform

Cirkulär Hough transform är en bildbehandlings algoritm som identifierar cirklar i bilden genom att omvandla pixelpunkter i bilden till parametricerad form som sedan undersöks med hjälp av hough transformen[4]. Cirkulära hough transformen hittar mitten av cirklar i bilden med en bra precision.

Första steget i den cirkulära hough transformen är att identifiera alla kanter i bilden. Detta kan göras med valfri kant detekterings algoritm. I varje identiferad kant pixel dras en cirkel i parametricerad form med den radie av cirkel som undersöks. Alla pixlar i bilden som ligger på cirkelns omkrets ackumuleras i en ackumulations matris med storlek nmk, där nm är bilden storlek och k är antalet cirkel radie som undersöks.

Ackumulations matrisen innehåller nu värden på hur många parametriserade cirklar av samma radie som går igenom varje pixel i bilden. De högsta värdena i matrisen motsvarar center koordinaterna av cirklar i bilden.

Den cirkulära hough transfromen har en tendens att ge falska cirklar ifall bilden är rörig och innehåller många element. Ett sett att öka robustheten hos den Cirkulära hough transformen är att minska området i bilden som undersöks för cirklar. I detta projekt görs detta genom att kombinera hough transformen med en annan ögon detekterings metod som är bättre på att hantera en rörig bild, men ger mindre exakta kooridinater gör ögon position. Detta görs genom att först detektera ögon med en annan metod. En mask skapas i ett område runt ögon positionerna som hittats. Den cirkulära hough transformen körs på det område i bilden som masken visar och mer exakta positioner hos ögonen fås ut.

I detta projekt användes matlabs inbyggda cirkulära hough transforms metod *imfindcircles*.

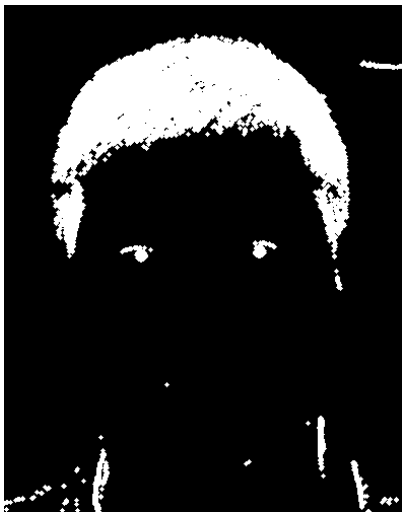
2.4.3 Hybrid metod

En annan metod för ögonigenkänning som föreslogs av Shafi Muhammad och Chung Paul i uppsatsen *A hybrid method for eyes detection in facial images*[5]. Metoden går ut på att kombinera tre metoder som har varsin svaghet, men tillsammans balanserar ut varandras svagheter. De tre metoderna som kombineras är:

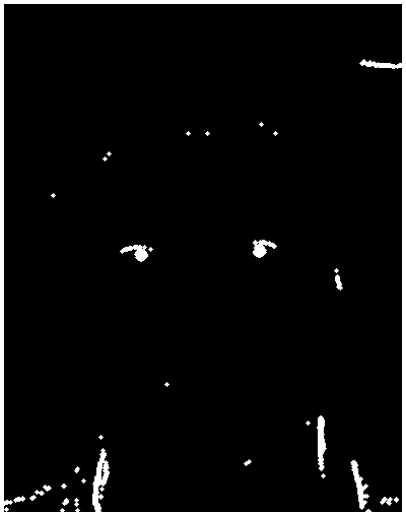
- Illumination-baserad metod
- Colour-baserad metod
- Edge density-baserad metod.

Illumination-baserad metod Illumination-baserade metoden är exakt samma som beskrivs i delkapitel 2.4.1

Colour-baserad metod Den färg-baserade metoden utgår från faktumet att ögonen är i det mörkaste området hos ansiktet. Metoden börjar därför med att konvertera bilden till ett histogram utjämnat gråskalebild. Därefter så används ett tröskelvärde för att skapa en ögonmask, då som sagt så är ögonen i ett mörkare område än resten av ansiktet. Hår brukar finnas med i masken, men tas bort med hjälp av komponent verifikations processer. Bilden kallas för $EyeMaskCol$.

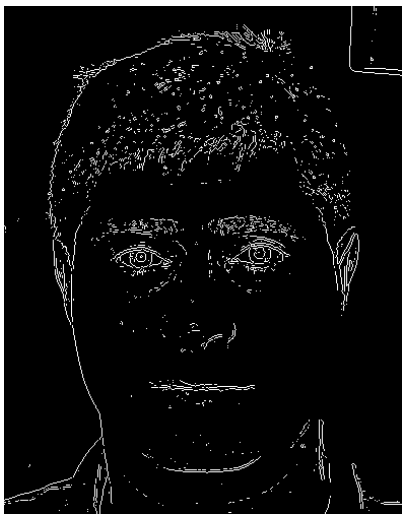


Figur 8: Histogram utjämnat gråskalebild som är trösklad.



Figur 9: $EyeMaskCol$.

Edge density-baserad metod Den sista metoden använder sig av faktumet att ögonområdet har högre *edge density* i jämförelse med andra områden i ansiktet. Enligt metoden som föreslås i [5] så konverteras bilden först till gråskala och hörn detekteras med *Sobel edge detection* vart efter morfologisk dilation utförs två gånger för att förbättra sammansatta områden i bilden. De sammansatta områderna fylls sedan igenom och därefter används morfologisk erosion tre gånger för att ta bort oönskade bitar av bilden. den resulterande bilden kallas för *EyeMaskEdge* och syns i figur 26.



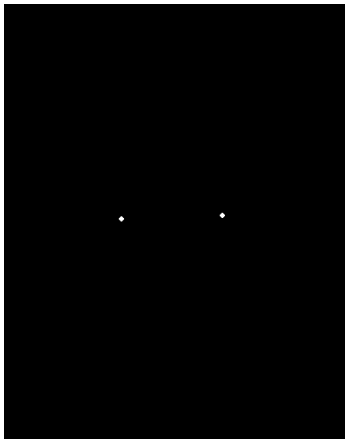
Figur 10: Edge-detekterad bild

Figur 11: *EyeMaskEdge*.

Kombinering av metoder Efter att de olika maskerna har beräknats så kan det fortfarande finnas artifakter kvar i bilderna som är oönskade. Därför så föreslogs det i [5] att applicera fyra regler på ögon maskerna för att förbättra dem:

- *Solidity* på en yta måste vara större 0,5.
- Bildförhållandet måste vara mellan 0,8 och 4,0.
- Sammanslutna ytor får inte röra kanterna av bilden.
- Orienteringen av en sammansluten yta ligger mellan -45 och 45 grader.

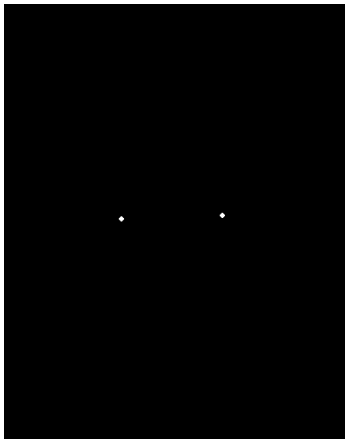
I fallen då en sammansluten yta inte uppfyller dessa krav så tas de bort. I figur 12 visas både bild på maskerna innan och efter reglerna har applicerats.



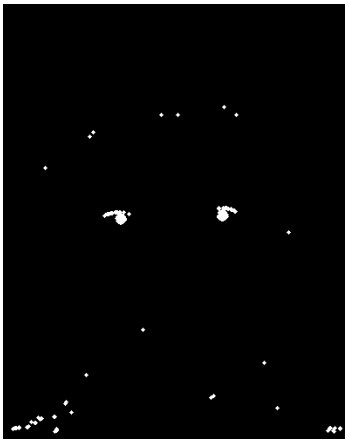
Figur 12: Illumination-method innan regler.

Figur 13: Colour-metod innan regler.

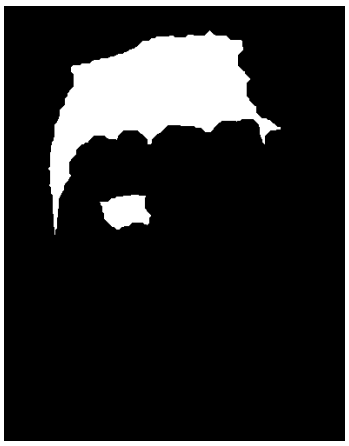
Figur 14: Edge-metod innan regler.



Figur 15: after rules.



Figur 16: after rules.



Figur 17: Edge mask after rules.

Därefter tas sammansattna ytor som är utanför en *bounding-box* bort och sedan skapas masker som tar blandningar av alla masker efter att reglerna har applicerats genom att elementvis multiplicera dem.

$$Image_{IlluCol} = Image_{Illu} * Image_{Col}, \tag{21}$$

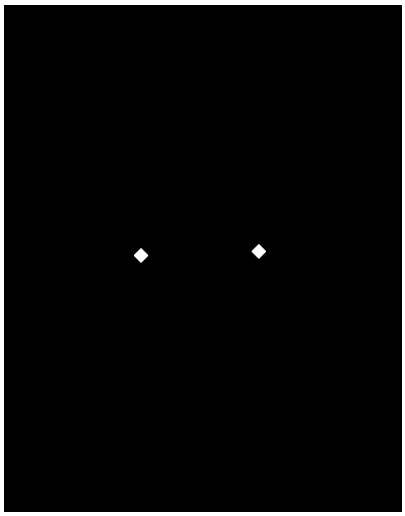
$$Image_{IlluEdge} = Image_{Illu} * Image_{Edge}, \tag{22}$$

$$Image_{ColEdge} = Image_{Col} * Image_{Edge}. \tag{23}$$

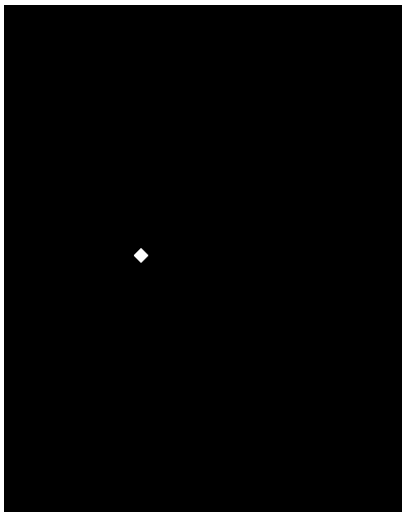
Sedan adderas resultaten ihop som sådan:

$$Image_{combined} = Image_{IlluCol} + Image_{IlluEdge} + Image_{ColEdge} \tag{24}$$

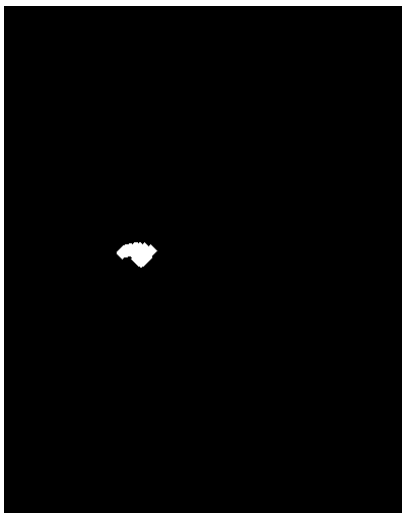
för att skapa en slutgiltlig bild, som sedan används för att hitta ögon positionen. Bilderna syns i figur 18 till 21.



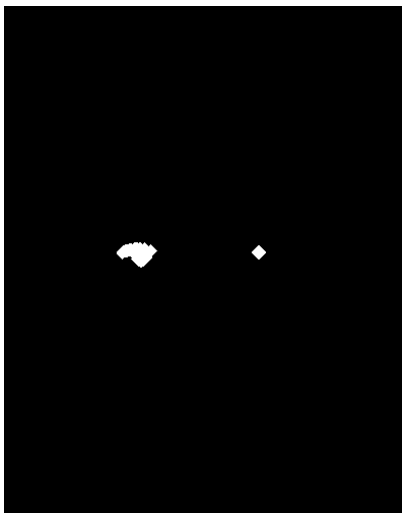
Figur 18: $Image_{IlluCol}$.



Figur 19: $Image_{IlluEdge}$.



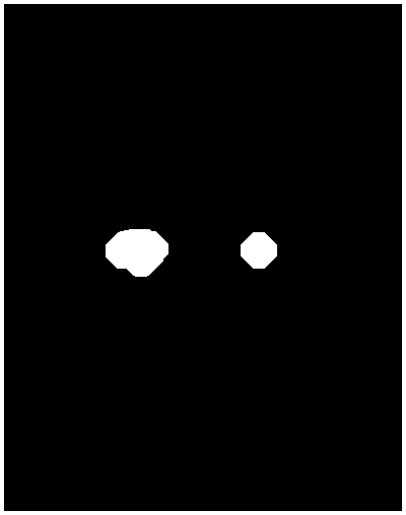
Figur 20: $Image_{ColEdge}$.



Figur 21: $Image_{combined}$.

Efteråt kan positionen hos ögonen ganska enkelt finnas genom att hämta centroiden hos ytorna i bilden, även om en del morfologiska operationer kan komma att användas först för att förbättra bilden.

Figur 22: Startbild



Figur 23: Slutresultatet.

2.5 Ansiktsnormalisering

Innan ansiktet kan identifieras måste den normaliseras. Variationer i bilderna gör det svårt att utföra PCA med ett bra resultat. Normaliseringen görs för att förminska dessa variationer så mycket som möjligt. Normaliseringen innehåller många steg så som translation, rotation, beskärning, skalning och kontrast ökning. Ordningen som stegen utförs är viktigt då varje steg utgår från föregående steg. Normaliseringen kräver att ögondetekteringen utförts och gett positioner för två ögon.

2.5.1 Translation

För att bilden ska kunna normaliseras måste ögonen ligga i samma höjd. Om ögonen i den identifierade bilden ligger i olika höjd måste då bilden roteras så dem ligger lika. För att underlätta rotationen translateras bilden så mittpunkten av ögonen hamnar i mitten av bilden. Den inbyggda matlabfunktionen *imtranslate* används för detta. Ögon positionerna translateras också med bilden.

2.5.2 Rotation

Efter translationen roteras bilden utifrån ögonens position. Bilden roteras med den vinkel som gör att ögonen ligger i höjd med varandra. Eftersom mitten av ögonen ligger i mitten av bilden kan rotationen enkelt göras med hjälp av den inbyggda matlabfunktionen *imrotate*. Imrotate roterar en bild runt sitt center med en specificerad vinkel. I *imrotate* specificerades det också att bilden ska klippas så den är lika stor efter rotation som innan så bilden är invariant.

2.5.3 Beskärning

Efter rotationen beskärs bilden så ointressanta områden tas bort som hals och bakgrund. Detta görs genom att ta fasta punkter i bilden som ögonens position och distans och sedan lägga till en marginal runt dem i alla riktningar. Detta ger att endast de intressanta delarna av ansiktet blir kvar. Den inbyggda matlabfunktionen *imcrop* användes för detta.

2.5.4 Skalning

Efter beskärningen så skalades bilden så distansen mellan ögonen blir en specificerad storlek. Detta görs genom att ta fram en faktor mellan den önskade längden mellan ögonen och den faktiska längden mellan ögonen och sedan skala hela bilden med den faktorn. Skalning av bilden görs med imscale.

2.5.5 Kontrast ökning

Sista steget i normaliseringen är att förbättra kontrasten i bilden. Bilden görs först svartvit med hjälp av matlabfunktionen *rgb2gray*. Kontrasten förbättras sen med matlabfunktionen *histeq* vilken sprider ut bildens histogram för att förbättra kontrasten.

2.6 PCA

När det gäller att identifiera ett okänt ansikte för de om det finns bland ett par kända ansikten finns det många olika metoder att använda. Den enklaste metoden är skulle vara att jämföra varje pixelvärde i den okända bilden med varje pixelvärde i dem kända bilderna och välja den som har minst skillnad. Problemet med denna metod är att även en liten bild innehåller tusentals pixlar vilket gör denna metod väldigt långsam ifall man behöver jämföra med många kända ansikten. Ett sätt

att lösa detta problem är att reducera dimensionen hos jämförelsena till en mer hanterbar storlek. I detta projekt används PCA för detta.

PCA går till på att skapa en lägre dimensionellt känneteckens rymd (*feature space*) som innehåller dem största variationerna i datan. För att identifera ett ansikte måste vi då bara projecera ansiktet på denna rymd och kolla skillnaden till kända ansikten i denna rymd. Detta görs genom att skapa egenansikten.

2.6.1 Medelansikte

För att skapa eigen faces i sectionen nedan måste först ett medel ansikte tas fram. För att underlätta beräkningarna omvandlas bilden först från en n x m matris till en nm x 1 vektor enligt ekvation 25, där I_i är matrisen för ansiktsbilden i med storlek $n \times m$ och Γ_i är ansiktsvektorn för bilden i med storlek $nm \times 1$. Medelansiktet kan då beräknas genom ekvation 26, där Ψ är medelansikts vektorn och M är antalet bilder i datasetet.

$$I_i = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}_{n \times m} \rightarrow \begin{bmatrix} a_{11} \\ \vdots \\ a_{1m} \\ a_{21} \\ \vdots \\ a_{2m} \\ \vdots \\ a_{nm} \end{bmatrix}_{nm \times 1} = \Gamma_i \tag{25}$$

$$\Psi = \frac{1}{M} \sum_{i=1}^{M} \Gamma_i \tag{26}$$

2.6.2 Egen-ansikten

För att ta fram egenansiktena måste en kovarians matris tas fram genom att först ta fram skillnaden mellan ansiktsvektorerna enligt ekvation 27 och sedan kovarains matrisen enligt ekvation 28.

$$\Phi_i = \Gamma_i - \Psi \tag{27}$$

$$C = AA^T \tag{28}$$

där differensvektorn Φ_i är skillnaden mellan ansiktsvektorn i och medelansiktsvektorn, C är kovarians matrisen och $A = [\Phi_1, \Phi_2 ... \Phi_M]$ och har storleken $nm \times M$. Det uppstår dock ett problem ifall man räknar ut kovarians matrisen så här. Kovarians matrisen kommer få storleken $nm \times nm$ vilket skulle ge nm antal egenvektorer med storlek $nm \times 1$ vilket är en ohanterbar mängd för en bild. Därför räknas kovarians matris ut enligt ekvation 29 istället.

$$C = A^T A \tag{29}$$

där C är kovarians matrisen med storlek $M \times M$. En kovarians matris med storlek $M \times M$ kommer ge M antal egenvektorer med storlek $M \times 1$. Eftersom M är mycket mindre än nm då det endast är antalet tränings bilder kommer denna storlek av kovarians matris att vara en mer hanterbar storlek.

En egenskap hos matriser ger att vi kan använda ekvation 30 för att få fram egenvärdena hos egenvektorerna.

$$u_i = Av_i \tag{30}$$

där u_i är egenvektorn i från kovariansmatrisen C enligt $C = [u_1, u_2...u_M]$ och v_i är egenvärdet för egenvektorn u_i. Storleken på egenvärdet ger hur mycket egenvektorn påverkar den slutgiltiga bilden. Genom att sortera egenvektorerna enligt deras egenvärden kan dem K bästa egenvektorerna väljas.

Eftersom egenvektorerna efter en omvandling till en $n \times m$ matris följande ekvation 25 i motsatt riktning liknar ansikten ansikten brukar dem kallas egenansikten eller eigenfaces. Varje ansikte i träningssättet , minus egenansiktet, kan nu representeras av en linjär kombination av egenvektorerna enligt ekvation 31.

$$\Phi_i = \sum_{j=1}^{K} w_j u_j \tag{31}$$

där Φ_i är differens vektorn för ansikte i, u_j är egenansiktena och w_j är vikten hos egenansiktet u_j för ansiktet i. Vikten w_j räknas ut enligt ekvation 31.

$$w_i = u_j^T \Phi_i \tag{32}$$

Varje tränings bild kan då bli representerad av en viktvektor $\Omega_i = [w_1, w_2...w_K]^T$ där $i = 1, 2...M$. En viktvektor för varje träningsbild skapas vilken tillsammans med alla egenansikten och medelansiktet sparas för senare användning.

2.7 Matchning

Denna sektion går ut på att använda det som gjordes i dem föregående sektionerna till att identifiera ett okänt ansikte. Detta sker i följande steg:

1. Vitpunkts kompensera det okända ansiktet enligt metoden i sektion 2.2.
2. Hitta ögonpositionerna i den vitpunkts kompenserade bilden enligt metoden som beskrevs i sektion 2.4.
3. Använd dem identifierade ögonpositionerna till att normalisera bilden enligt sektion 2.5.
4. Omvandla den normaliserade bilden till en ansiktvektor enligt ekvation 25.
5. Ta fram differens vektorn för ansiktsvektorn enligt ekvation 27 med det sparade medelansiktet från förra sektionen.
6. Projektera differens vektorn på dem sparade egenvektorerna och ta fram vikterna enligt ekvationen 32. Okända ansiktet kan då representeras som viktvektorn $\Omega_{okänd} = [w_1, w_2...w_K]^T$

7. Hitta minsta skillnaden mellan viktvektorn för det okända ansiktet och viktvektorerna för träningsansikten enligt ekvation 33.

8. Tröskla minsta skillnaden med ett tröskelvärde som valts efter testning där en minsta skillnad under tröskelvärdet identifierar okända ansiktet med det id som motsvarar viktvektorn som gav minsta skillnad och en minsta skillnad över tröskelvärdet inte blir identiferat som något av träningsansiktena och returnerar därför id = 0.

$$d = min\|\Omega_{okänd} - \Omega_i\| \tag{33}$$

där $\Omega_{okänd}$ är viktvektorn för det okända ansiktet, Ω_i är viktvektorn för träningsansiktet i där $i = 1, 2...M$ och d är den minsta skillnaden mellan viktvektorerna.

3 Resultat

För att testa systemets robusthet och träffsäkerhet testades systemets respons på bilder av olika typer. Systemet testades på bilder som alla tillhör databasen som systemet tränades på, både utan och med modifikationer. De modifikationer som användes var translation av ansiktet inom bilden, mindre rotation av bilden ($\pm 5°$), skalning ($\pm 10\%$) och tonförändring ($\pm 30\%$). Systemet testades även mot bilder med ansikten som inte tillhör db1, alltså de i db0, även här med och utan modifikationer, samt med andra bilder på personerna som db1 innehåller, de i db2.

De metoder presenterade i föregående kapitel ger, i kombination, ett system vars träffsäkerhet varierar under olika förutsättningar. I vissa förhållanden, såsom bilder med enkla modifikationer, är träffsäkerheten nästintill perfekt, såsom de bilder som finns i dataseten db0 och db1. För de mer utmanande bakgrunderna i datasetet db2 fallerar systemet då det *kan* ge rätt utfall, men ger oftast fel.

När systemet presenteras för bilder ur db0, alltså bilder med ansikten som ej finns med i datasetet, skall systemet returnera ett värde av noll. Resultatet från tester gjorda med bilder ur datasetet db0 visas i tabell 1. För bilderna i db0 är det intressanta resultatet så kallade falska positiva bedömningar, det vill säga att systemet bedömer att ansiktet ur en bild finns med i den databas den är tränad utefter. Som tabell 1 visar, kan systemet med perfekt träffsäkerhet avgöra att ansikten ur db0 ej finns i db1, både med de fyra ursprungliga bilderna, och med fyrtio modifierade bilder av dem, tio per bild.

Tabell 1: Resultat av tester med bilder ur db0, omodifierade och med variationer.

Typ av bild	Antal bilder	Falska positiva	Korrekthet
Omodifierade	4	0	100%
Modifierade	40	0	100%

När systemet testades för de bilder som systemet är tränat utefter, gavs även här ett nästintill felfritt resultat, som tabell 2 visar. Till skillnad från tester med ansikten som ej tillhör träningsdatan, är inte falska positiva bedömningar intressanta. Istället är dess motsats, falska negativa bedömningar av intresse. Falska negativa

bedömningar sker när systemet, givet ett ansikte ur dess träningsdata, ger utfallet att det ej är med i databasen. Ett annat intressant felaktigt utfall är om systemet bedömer att ett ansikte är med i datasetet, men att systemet bedömer att det mest lika ansiktet inte är det riktiga.

Tabell 2: Resultat av tester med bilder ur db1, omodifierade och med variationer.

Typ av bild	Antal bilder	Felidentifiering	Falska negativa	Korrekthet
Omodifierade	16	0	0	100%
Modifierade	160	0	1	$\approx 99\%$
Modifierade	1600	0	27	$\approx 98\%$

För de 16 omodifierade bilderna, alltså de exakta bilderna som systemet är tränat på, är resultatet som väntat helt felfritt. Däremot, när samma modifieringar som för de tidigare testerna introduceras, sker, om än sällsynt, felbedömningar. Systemet testades mot 160 och 1600 bilder med variation från db1, med 10 respektive 100 bilder per ansikte, och de felbedömningar som sker är samtliga falska negativa bedömningar. Utav de 16 personer som db1 innefattar, var den som hade flest falska negativa bild sju visad i figur 24. Mer om varför bilden blir felklassificerad, samt hur det kan motarbetas presenteras och diskuteras i nästkommande kapitel.

De sista testerna som utfördes var test av bilderna från db2. Som tidigare skrivet är de ansikten i datasetet av samma personer som i db1, alltså skall alla bilder få ett korrekt id. Likt testerna på db1 är felkällorna av intresse när systemet identifierar ansikten fel eller ger falskt negativt svar. En annan intressant felkälla här är om systemet inte hittar några ögonkandidater, eller använder sig av fel ögonkandidater i normaliseringsprocessen. Eftersom normaliseringsprocessen blir fel, kommer de två bilderna vars värden jämförs inte vara jämförbara. Tabell 3 visar resultaten av testerna med datasetet db2. Värt att notera av den statistik som visas är att alla felidentifieringar som skedde egentligen var falska negativa. Anledningen till att de räknas som felidentifieringar är att bedömningen av vilket ansikte ur db1 som testansiktet inte var korrekt, alltså hade de varit felidentifieringar ifall de inte var falska negativa.

Tabell 3: Resultat av tester med bilder ur db2, samtliga omodifierade.

Typ av bild	Antal bilder	Felidentifiering	Falska negativa	Ögon ej funna	Korrekthet
Suddig	9	2	3	2	$\approx 22\%$
Svår bakgrund	16	3	2	5	$\approx 38\%$
Dåligt ljus	6	3	0	2	$\approx 17\%$
Ansiktsuttryck	7	1	3	2	$\approx 14\%$

Figur 24: Bild sju från db1

4 Diskussion

De fel som uppstår när bilder ur db1 modifieras med rotation, skalning, tonförändringar och translation är i alla förekomster så kallade falska negativa, alltså bedömer systemet att likheten mellan det ansikte som testas och de ur träningsdatan är för liten. Det ansikte som systemet anser att testansiktet är mest likt var under alla tester rätt ansikte, men systemet sållar bort dessa eftersom skillnaden mellan dem är för stor. Som tidigare skrivet är det ansikte vars modifierade bilder får falskt negativt utfall bild nummer sju ur db1, visat i 24. De faktiska, normaliserade, bilderna som jämförs mellan bild sju och den med variationer visas i figur 25 respektive figur 26. Olikheterna mellan de normaliserade bilderna är knappt noterbara, den enda märkbara skillnaden är en liten förskjutning, annars är de identiska. Ett enkelt sätt att lösa detta på är att öka toleransen för likheten mellan bilderna, men det introducerar andra problem.

Figur 25: Normaliserad bild ur db1, för träningsdata.

Figur 26: Normaliserad bild som testas.

Genom att öka toleransnivån ökar risken för att få falska positiva bedömningar. För de flesta appliceringar av ansiktsigenkänning är falska positiva resultat ofta allvarligare än falska negativa, eftersom de kan leda till felidentifiering av individer. Till exempel, i säkerhetssystem kan ett falskt positivt innebära att en obehörig person ges tillgång, vilket kan få allvarliga konsekvenser. Därför är det viktigt att hitta en balans mellan toleransnivån och systemets noggrannhet. Anledningen till att den toleransnivå som används i det implementerade systemet inte höjdes för att minska antalet falska negativa, var att marginalen mellan likhetsfaktorn för de sanna ansiktena och andra ansikten var mycket liten, ibland när främmande ansikten testades ansågs de som mer lika de i db1 än vad ansikten ur db1 med modifikationer var.

Systemet som skapats är känsligt till hur vältagna bilderna är. Bilderna i db2 är därför problematiska då deras tillstånd ger stor variation i systemet. Som det visas i 3 så är korrektheten låg, även fast bilder med svår bakgrund är nästan lika bra som ett myntkast. För de olika felen som kan påverka resultatet i db2 så är det vanligt att ögonen inte hittas alls. För att lösa detta skulle hybrid-metoden som beskrevs i 2.4.3 kunna användas för att hitta ögonen, men som syns i figur 17 så är metoden inte optimiserad och får artifakter som påverkar slutresultater även på enklare bilder i db1. Med en bättre variant av hybrid-metoden så hade ögonen hittats. Vidare

så använder systemet också eigenfaces för ansiktsigenkänningen, men enligt Belhumeurs, Hespanhas och Kriegmans resultat i deras rapport *eigenfaces vs. fisherfaces: recognition using class specific linear projection*[1] så syns det att fisherface-metoden är mer robust och ger färre fel än Eigenfaces. En implementation av denna metod hade möjligtvis löst problemen då ögonen hittades, men felidentifiering skett eller falska negativa dykt upp.

När bilder normaliseras i färg för huddetektering med YCbCr-modellen, uppstår en utmaning med att fastställa tröskelvärden som fungerar universellt. Detta beror på att normaliseringen bygger på antagandet att den utförs korrekt, vilket inte alltid är fallet. En dåligt genomförd normalisering kan leda till att vissa hudtoner hamnar utanför de definierade tröskelvärdena för Y-, Cb- och Cr-komponenterna. Detta är problematiskt, särskilt i applikationer där mångfalden av hudfärger är viktig att hantera för att undvika diskriminerande eller inexakta resultat.

För att mildra problemet kan morfologiska operationer användas. Dessa operationer bygger på antagandet att hud finns i vissa delar av bilden och kan vidga identifierade hudområden för att inkludera angränsande pixlar som sannolikt också är hud. Till exempel kan en dilation-operation öka storleken på de upptäckta hudområdena, vilket förbättrar robustheten mot små variationer i färg eller belysning.

Även om morfologiska operationer kan vara användbara i många fall, är de inte en perfekt lösning. De kan misslyckas i situationer där hudfärgerna är mycket olika eller när belysningsförhållandena är extrema. Dessutom finns risken att sådana operationer inkluderar felaktiga områden som inte är hud, vilket kan påverka precisionen negativt. Därför behövs ofta en kombination av tekniker, såsom adaptiva metoder eller användning av ytterligare informationskällor, för att förbättra noggrannhet och pålitlighet i huddetektering.

Referenser

[1] P.N. Belhumeur, J.P. Hespanha, and D.J. Kriegman. Eigenfaces vs. fisherfaces: recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720, 1997.

[2] Jose M Chaves-González, Miguel A Vega-Rodríguez, Juan A Gómez-Pulido, and Juan M Sánchez-Pérez. Detecting skin in face recognition systems: A colour spaces study. *Digital signal processing*, 20(3):806–823, 2010.

[3] Rein-Lien Hsu, M. Abdel-Mottaleb, and A.K. Jain. Face detection in color images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):696–706, 2002.

[4] Simon just Kjeldgaard Pederson. Circular hough transform. *Aaldborg University, Vision, Graphics, and Interactive Systems*, 123(6):2–5, 2007.

[5] Shafi Muhammad and Chung Paul. A hybrid method for eyes detection in facial images. loughborough university. conference contribution, (2008). https://hdl.handle.net/2134/10195.