EXAMENSARBETE CERTEC, LTH NUMMER 3:2003

Mikael Aili
Arvid Nilsson

Algoritmer för huvudstyrning av dator

Certec
Avdelningen för rehabiliteringsteknik
Institutionen för designvetenskaper
Lunds tekniska högskola

Förord

Det är med stort nöje vi, Mikael Aili och Arvid Nilsson, presenterar föreliggande examensarbete som utgör en fortsättning på Björn Breidegards imponerande arbete med *Minimetern*.

Historien bakom detta exjobb börjar i slutet av sommaren 2002. Arvid sökte efter ett examensarbete som utnyttjar algoritmer inom området datorseende för att lösa problemställningen att räkna ut vart en person riktar blicken. Både Matematikcentrum och institutionen för reglerteknik ställde sig positiva till ett sådant examensarbete, men påpekade hur svårt det är att genomföra den typen av datorseende i praktiken.

Den största tillämpningen av sådana system för *blickföljning* är som hjälpmedel för människor med funktionshinder. Därför togs frågan om ett examensarbete i rehabiliteringsteknik upp med Björn Breidegard på Certec. Både Björn och Arvid visste att datorseende är ett svårt område men beslutade ändå att göra ett försök att arbeta med detta inom ramarna för rehabiliteringsteknik och Björns tidigare arbete i området.

Mikael som var på jakt efter ett exjobb tyckte att det verkade intressant. Det skulle ju i grunden handla om matematisk bildanalys, något som både han och Arvid hade läst flera kurser om. Dessutom skulle det kunna bli utrymme för att experimentera med olika algoritmer från Artificiell Intelligens-området och det skulle kunna ingå konstruktion av någon grafisk applikation. Och sist men inte minst ingick det en engagerad handledare.

På Certec hade vi två en unik möjlighet att jobba med datorseende för en väldigt relevant tillämpning, nämligen hjälpmedel för personer med funktionshinder. Det som gjorde det ännu mer spännande var möjligheten att prova tekniken tillsammans med någon av de funktionshindrade personer Björn känner.

Vi vill framföra ett stort tack till alla som hjälpt oss under arbetets gång. Särskilt vill vi tacka Björn Breidegard, Magnus Andersson-Jardeby och Birthe Jensen samt Håkan Eftring och Finn Lindgren.

Denna rapport är skriven för reproduktion i färg, men av kostnadsskäl kan det exemplar läsaren håller i handen vara tryckt i gråskala. På Certecs hemsida finns en digital version i färg att ladda ner. Se rapportens baksida för internet-address.

Mikael Aili
Arvid Nilsson
Lund den 27 juni 2003

Abstract

Methods from statistical image analysis have been used to implement software for real time face tracking. The video stream from a camera connected to a personal computer is used as input. This face tracking software is intended to form the basis of a solution that allows people with severe physical disabilities to interact with computers. The software has been tested on a person with cerebral palsy.

Sammanfattning

Algoritmer från statistisk bildanalys har använts för att implementera mjukvara för ansiktsföljning. Indata är videoströmmen från en kamera som kopplas till en persondator. Vår avsikt med ansiktsföljningen är att huvudrörelser skall kunna användas som styrsignal i stället för datormusen för att interagera med en persondator.

Vi tror att för många människor med rörelsehinder kan det vara möjligt att utveckla datorprogram som kan vara till både nytta och glädje. Problemet är att dessa människor har svårt att interagera med datorn via de vanliga medlen. Vårt arbete presenterar en teknisk lösning som bygger på att användaren i någon utsträckning kan röra huvudet. Särskilt inriktar sig vårt arbete på svårt hjärnskadade människor. Eftersom både rörelseförmåga och kognitiv förmåga bland sådana användare kan variera stort är ett mål att inte bara applikationen, utan även tekniken skall kunna individanpassas. Detta åstadkommer vi genom att välja en mjukvarubaserad teknisk lösning.

Grundidéerna bakom vårt arbete har vi ärvt från Björn Breidegards *Minimetern.* Syftet med vårt arbete kan sammanfattas i en mening: att genomföra en teknikhöjning av *Minimetern* som gör tekniken användbar för fler funktionshindrade människor. Att utveckla tekniken för dess egen skull var inte det viktiga i vårt arbete.

Under arbetets gång har vi provat tekniken tillsammans med Magnus Andersson-Jardeby, vars individuella förmågor och behov varit viktiga för att formulera projektets fortsatta utveckling. Magnus har en CP-skada, men enligt vår bedömning verkar han ha en hel del kontroll över sina huvudrörelser. De slutsatser vi drog av provanvändningen var att rent tekniskt behöver våra algoritmer vidareutvecklas för att uppnå större robusthet.

När automatisk ansiktsdetektion visade sig väldigt svårt lät vi det slutligen göras manuellt. Rent praktiskt kan en assistent till användaren lätt utföra detta steg. Sedan kom vi in på ett mycket lovande spår för själva ansiktsföljaren, nämligen metoden med *egenansikten*, som också skulle kunna lösa problemet med ansiktsdetektion. Tyvärr närmade sig då slutet på den utsatta tiden.

Resultatet blev en ansiktsföljare som fungerade väl under förutsättningarna att ansiktet inte vreds för mycket eller flyttades för hastigt. Många konkreta idéer för hur den skulle kunna förbättras presenteras i vår rapport, så området är inte alls uttömt i och med detta examensarbete. Snarare presenterar det en språngbräda för framtida vidareutveckling

Innehåll

Inledning

Syfte

Syftet med detta examensarbete var att genomföra en teknikhöjning av Björn Breidegards *Minimeter* (Breidegard, 1999). Målet var att göra *Minimetern* tillgänglig för fler personer med funktionshinder genom att förbättra tekniken och göra det möjligt att bygga nya individanpassade applikationer på denna. För mer information om Björn Breidegards *Minimeter,* se även (Breidegard, 2001) och (Breidegard, 2002)

I övrigt var målet öppet och visionerna stora. Förhoppningen var att i tre etapper nå fram till blickföljning (eng *gaze tracking*), som innebär att man beräknar var på datorskärmen användarens blick är fokuserad. Varje etapp var ett möjligt slutmål eller delmål, beroende på vad examensarbetets tidsomfattning medgav. Steg ett var ansiktsföljning, varefter i steg två de olika ansiktsdragens lägen skulle beräknas och följas. Som en del av detta skulle man erhålla ögonens läge, och därefter i steg tre kunna beräkna blickpunkten.

Under fullföljandet av varje delmål skulle tekniken utvecklas med återkoppling från provanvändning tillsammans med möjliga användare. Två personer med olika funktionshinder var påtänkta för provanvändning av ansiktsföljning respektive blickföljning.

Syftet med denna rapport är att dokumentera arbetet som till största delen bestått av programmering med Microsoft Visual C++. Rapporten innehåller också resultaten av våra tester med en svårt rörelsehindrad provanvändare.

Bakgrund

Björn Breidegard har tidigare arbetat med *Minimetern* vars syfte är att med hjälp av modern och billig teknik hjälpa svårt hjärnskadade människor att kommunicera. Den individuella människan var drivkraften bakom arbetet. Ursprungligen var produkten skräddarsydd för en enda person, Emma Nilsson. Emma är i stort sett förlamad efter en olycka som orsakade en svår hjärnskada. Men det visade sig att hon ändå kunde göra vissa små rörelser. Genom god sjukgymnastik hade Emma övat upp förmågan att vrida huvudet lite till vänster och höger. Denna rörelse kunde ju användas som styrsignal! *Minimetern* blev en riktig succé för Emma, och ett antal applikationer utvecklades.

Björn hoppades även kunna hjälpa andra människor med svår nedsättning av rörelseförmågan. Efter att ha provat *Minimetern* med personer som hade andra typer av funktionshinder som också orsakats av hjärnskada kom han fram till att en teknikhöjning skulle behövas.

Rent tekniskt var problemet framför allt att följningen inte klarade stora förflyttningar från det ursprungliga kalibreringsläget. Eftersom Emma bara kan vrida huvudet lite åt vänster och höger fungerade programmet utmärkt för henne. Andra provanvändare som Björn varit i kontakt med hade betydligt yvigare huvudrörelser, och detta ställde till problem.

I princip kan man säga att examensarbetet var ett beställningsjobb från Björn till författarna. Uppdraget var att genomföra den nämnda teknikhöjningen.

Idén bakom *Minimetern*

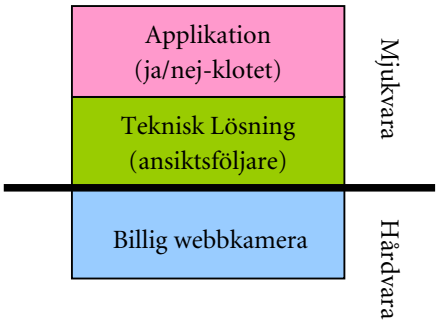
Arbetet vid Certec börjar och slutar alltid i människan. Målsättningen för varje projekt här är att det åtminstone ska vara till nytta eller glädje för någon funktionshindrad person.

”Målet har stora likheter med läkekonstens: att bota, lindra och/eller trösta.”

Bodil Jönsson, professor och forskningsansvarig vid Certec

I arbetet med datorapplikationer som kan vara till nytta eller glädje för människor med funktionshinder är individanpassning det viktigaste nyckelordet. Applikationens gränssnitt och funktionalitet måste i många fall skräddarsys för individens förmåga och intressen. Även tekniken som gör det möjligt för användaren att interagera med datorn behöver kunna individanpassas.

Lösningen måste vara flexibel för att kunna uppfylla dessa krav. *Minimetern* bygger därför på en kombination av mjukvara och en billig webbkamera. Både den tekniska lösningen och själva applikationen utgörs av mjukvara. Den tekniska lösningen bygger på algoritmer för bildbehandling och datorseende som i realtid beräknar olika styrsignaler med hjälp av videoströmmen från kameran. Till exempel kan näsans läge i bilden vara en tvådimensionell styrsignal som kan ses som en motsvarighet till den koordinatström datorns operativsystem får från en datormus. Sedan finns det otaliga möjligheter att använda denna styrsignal i olika applikationer.



Figur 1. Strukturschema för Minimeterns lösning.

En av grundförutsättningarna för att kunna individanpassa denna mjukvara är att källkoden är tillgänglig. I fallet med *Minimetern* är källkoden tillgänglig tack vare att den är utvecklad på Certec.

Ett alternativ till att skriva allt från grunden är att försöka hitta ett ”open source”-projekt som kan anpassas. Problemet är att det finns få projekt med öppen källkod för just bildbehandling och datorseende. De som finns har ofta inte uppnått den mognadsgrad som kunde motivera att man använder dem istället för att skriva egen kod.

Minimetern Björn Breidegard utvecklade åt Emma var en lösning av den typen som just diskuterades. *Minimetern* gjorde det möjligt för Emma att ge ifrån sig två olika uttryck, som kunde tolkas som ja eller nej, eller styrsignal åt någon applikation. Den enda hårdvaran som krävdes, förutom en vanlig persondator, var en webbkamera riktad mot Emmas ansikte. Dess tekniska lösning bestod av en mallmatchare (eng *template matcher*), ett program som kunde följa Emmas nästipp allt eftersom hon rörde på huvudet.

Figur 2. Emmas användargränssnitt – ett rullande klot som styrs med ansiktsvridningar.

En applikation som visade sig fungera mycket bra var Ja/Nej-klotet. Detta var ett grafiskt gränssnitt som ritade ett stort gult klot på skärmen. När Emma rörde sin nästipp åt vänster följde klotet hennes rörelse och ändrade gradvis färg till rött. När det nådde vänsterkanten av skärmen sa en röst ”nej”. På motsvarande sätt ändrades klotets färg till grönt när hon rörde sig åt höger men här sa rösten ”ja” istället. Denna visuella återkoppling gav henne en god känsla för gränssnittet och gjorde det mycket lättare för henne att använda.

Figur 3. Emma är på väg att svara Nej genom att vrida ansiktet åt vänster. När hon vridit lite till blir hela skärmen röd och ett Nej hörs i högtalarna.

Kravspecifikation

Följande kravspecifikation för examensarbetet sattes upp tidigt och var baserad på planeringen i föregående stycke. Som nämnts där slutfördes inte alla punkter i planeringen. Därför är inte alla krav uppfyllda i och med det här examensarbetet.

Ett klassbibliotek med vars hjälp man kan erhålla en ström av koordinater vilka anger var användarens ansikte befinner sig och var på skärmen dennes blick är fokuserad. Dessa koordinater kan förutom läge och rotation beskriva användarens ansiktsuttryck.

Figur 4. Emma har svarat ja.

Klassbiblioteket skall skrivas i Microsoft Visual C++ och kunna användas på en standard-PC av år 2003, nämligen ca 2 GHz processor och 256 Mb arbetsminne. Den nyskrivna koden skall vara väl dokumenterad så att den i sin tur kan byggas vidare på.

De algoritmer som används skall resultera i robusta ansikts- och ögonföljare. Med robust menas framför allt att den skall kunna avgöra om följningen misslyckats och inte generera felaktiga utdata utan istället försöka återuppta följningen. Den bör dessutom vara robust för skiftande ljusförhållanden och olika användare.

För blickföljningen gäller att noggrannheten för uppskattningen av blickens fokuspunkt åtminstone bör vara två grader[1], och uträkningen ska kunna ske åtminstone 20 ggr per sekund.

Hårdvaran som används skall vara en kamera som arbetar med synligt ljus. Kameran skall finnas tillgänglig att köpa på marknaden, det skall alltså inte röra sig om specialdesignad hårdvara. Till datorn kan eventuellt ett video capture-kort behövas.

Det skall finnas ett antal demonstrationsprogram som visar vad man kan åstadkomma med de olika algoritmerna. Tanken bakom detta är att man lättare skall kunna förstå hur klassbiblioteken används genom att läsa källkoden till dessa program.

Rapportöversikt

Denna rapport innehåller till att börja med en marknadsöversikt. Detta för att läsaren skall få en överblick över området.

En viktig del i arbetet var att den nyskrivna koden blev väl dokumenterad. Syftet med denna rapport är alltså även att göra det lätt att sätta sig in i hur koden fungerar. En viktig del är att förklara den teori som ligger bakom de algoritmer som använts. Teoriavsnittet kan förstås vara av stort värde även för den som inte har intresse av just våra implementationer av algoritmerna.

Väl kommenterad kod är i princip självdokumenterande. Därför har det som en del i dokumentationsarbetet också varit viktigt med uttömmande kommentarer. Det kan dock vara svårt att få en överblick över de olika klassernas samband genom att läsa källkoden. Därför presenteras i ett appendix till denna rapport dessutom en översiktlig beskrivning av hur våra klasser är uppbyggda och hur de beror av varandra. Dessutom finns olika appendix som beskriver MATLAB-koden, ansiktsdatabasen och kameran som använts i examensarbetet.

[1] Här avses två graders rotation av själva ögongloben.

Metoder

Certecs metoder

Tidigare projekt (Jönsson, 1998) har lärt medarbetarna på Certec att funktionshindrade människors behov inte alltid formuleras av sig själva. Om användarna har funktionshinder som gör det svårt för dem att kommunicera, hur ska de då kunna uttrycka sina önskemål för omvärlden? Därför måste de som utvecklar hjälpmedlen ibland gå in och försöka etablera en dialog eller finna nya kommunikationsmedel. Längre fram i processen, när väl någon öppning mellan utvecklarna och användarna etablerats, så har man mötts av en ström av önskemål och idéer.

Tre olika metoder för att hjälpa funktionshindrade människor har identifierats på Certec:

- Papegojmetoden. En lösning som låter den funktionshindrade personen fungera nästan helt och hållet som en icke funktionshindrad.
- Kameleontmetoden. Låter den funktionshindrade personen göra samma sak som en icke funktionshindrad, fast på ett annat sätt.
- Pudelmetoden. Bygger på att identifiera problemets kärna, att hitta det innersta i behovet hos den funktionshindrade personen, för göra det möjligt att finna en helt annan lösning som uppfyller personens drömmar, önskningar och behov.

I vårt arbete har vi siktat på kameleontmetoden. Vårt mål är att låta funktionshindrade människor interagera med datorer, fast på ett annat sätt. I stället för den tvådimensionella styrsignalen från datormusen används huvudrörelser.

Ett viktigt motto i Certecs arbete är ”man kan aldrig veta förrän man har provat”. Därför var provanvändning ett mycket viktigt inslag i vårt arbete.

Programmeringsmetodik

Arbetet med detta examensarbete resulterade i mycket programkod. Programmering är ett komplicerat arbete som utan tvivel kan förenklas och göras mer effektivt med hjälp av olika teorier för mjukvarudesign och programmeringsprocessen.

Det har varit viktigt att kunna testa algoritmerna på ett tidigt stadium. Det vanliga sättet att testa algoritmer är att man har kända utdata med

tillhörande indata som man kan mata in i algoritmen. Därefter jämförs algoritmens resultat med de önskade resultaten, och föreligger skillnader måste man arbeta vidare på koden. Det var svårt att sätta upp testfall av den här typen för våra algoritmer där indata är något så komplicerat som en videoström, och utdata inte är definierat med pixelprecision. Våra önskemål på utdata kan typiskt formuleras som ”en tvådimensionell koordinat som anger var ansiktet befinner sig i bilden”.

Vad gäller bildbehandling med datorer så innehåller algoritmerna ofta mycket matrisberäkningar. Om vi konsekvent hade utvecklat prototyper i ett numeriskt orienterat språk som MATLAB hade vi kunnat använda dessa prototyper för att erhålla pålitliga utdata. I stället skrevs de flesta algoritmer direkt i C++. I ett specifikt fall, nämligen beräkning av PCA-bas (egenansikten) utifrån en databas av ansiktsbilder, bedömde vi att programmeringsinsatsen i C++ skulle bli så omfattande att vi skrev det programmet i MATLAB istället. Detta MATLAB-program skulle kunna utgöra prototyp till ett framtida C++-program för beskärning av bilderna och beräkning av PCA-basen.

I stället för testfall använde vi oss av visualisering av algoritmernas iterationer och resultat för att bedöma deras korrekthet. Visualiseringskoden implementerades på ett så tidigt stadium som möjligt, ofta långt innan algoritmen var helt färdig. Visualiseringen skedde i realtid och använde videoströmmen från kameran som indata. Denna omedelbara visualisering har i många fall varit ovärderligt för att upptäcka buggar. Det har också varit möjligt att variera vissa algoritmers parametrar i realtid genom interaktion med bilden eller genom att dra i reglage i gränssnittet.

Ofta har programmeringsarbetet varit iterativt, i och med att en mer eller mindre väl fungerade algoritm har utökats eller förbättrats i flera steg. Ibland har en iteration inneburit att vi skrivit om koden från grunden, efter att ha fått nya insikter i hur koden borde designas.

En viktig del av vår metodik var att vi redan från början planerat att ha provanvändning med någon funktionshindrad person. Beroende på hur långt vi kom på vägen från ansiktsföljning till blickföljning var två olika provanvändare påtänkta.

I vårt arbete ansåg vi provanvändning vara en mycket viktig metod för att bestämma inriktningen på projektet. De första två provanvändningarna genomfördes precis innan jul, vilket var i slutet av examensarbetets tidsrymd. Emellertid var då programmet fortfarande på ett ganska primitivt stadium, i och med att vi precis hade fått fram början på en ansiktsföljare. Därför vill vi ändå påstå att vi med tanke på programmets mognad i ett tidigt skede införde provanvändning som en viktig del av programmeringsarbetet.

Marknads-undersökning

Innan algoritmskapande och programmeringsarbete inleddes gjorde vi en utvärdering av några av de produkter som finns på marknaden. Dessutom försökte vi skapa oss en översikt av den forskning som bedrivs inom områden ansiktsföljning och blickföljning. Denna utvärdering genomfördes uteslutande via webben, och därför anges en mängd länkar i samband med kommentarerna nedan.

Existerande lösningar

Huvudföljning

http://www.naturalpoint.com/

Naturalpoint har utvecklat en IR-kamera som ansluts till USB-porten på en persondator med operativsystemet Windows. IR-kameran har dessutom fyra IR-dioder som sänder ut infrarött ljus. Genom att användaren placerar ett reflekterande klistermärke t ex i pannan kan kameran följa klistermärkets rörelse i realtid. Till hårdvaran hör en mjukvara för att styra muspekaren med denna metod. Dock finns ingen klickfunktion som är baserad på huvudrörelser. Musklick genereras genom att man trycker på en knapp på tangentbordet. Man kan även som tillval köpa så kallad *dwell clicking* vilket innebär att musklick genereras när man håller huvudet stilla ett tag.
En av författarna är ägare till denna produkt, och kan varmt rekommendera den med tanke på det låga priset. Tekniken är enkel, robust och billig, men kan inte utvecklas vidare på grund av den begränsade hårdvarulösningen. För ett arbete som syftar till att utveckla applikationer till huvudstyrning är denna produkt utmärkt. Emellertid var det ju inte syftet med vårt examensarbete. Nämnas bör att *Minimeterns* teknicknivå i slutet av vårt examensarbete var ungefär jämförbar med Naturalpoints produkt.

Ögonföljning

http://www.eyetechds.com/

Eyetech QuickGlance kostar ungefär $4000-$6500 beroende på modell. Produkten är den billigaste lösningen för ögonföljning i vår marknadsundersökning och är tänkt för funktionshindrade personer, särskilt de med ALS[2]. Tillämpningarna är många, men produkten marknadsförs framför allt som en ersättning för den vanliga datormusen. Det finns en variant som använder samma hårdvara, ”Eye Science”, men som är avsedd för forskare inom området.

[2] Amyotrofisk lateralskleros, en neurologisk sjukdom som påverkar nästan alla viljestyrda muskler i kroppen, utom t ex ögonrörelser.

Hårdvaran använder infrarött ljus som reflekteras i ögonen och fångas upp av en IR-kamera. Utrustningen har inga komponenter som fästs på användaren.

http://www.seeingmachines.com/
Vad gäller Seeing Machines produkt FaceLAB har vi en ungefärlig prisuppgift på $25 000. Emellertid finns ingen prisuppgift tillgänglig på hemsidan, så vi kan inte garantera att prisnivån inte ändrats sedan denna prisuppgift erhölls. Systemet är avsett för forskning och utveckling, främst inom bilindustrin. Det är baserat på två kameror, och beräknar inte bara ögonens utan även huvudets läge, samt vart man tittar. Systemet använder inte IR, utan genomför sina beräkningar utifrån bilder tagna med två vanliga gråskalekameror. Precisionen i ögonföljningen är ungefär 3 grader under goda förhållanden. Utrustningen fästs inte på användaren.

http://www.eyegaze.com/
LC Technologies tillverkar en produkt som heter ”The Eyegaze System”. Den är baserad på IR-ljus, och finns i två varianter. En för forskare, och en för att hjälpa människor med funktionshinder. På deras sida finns en gedigen genomgång av de medicinska problem som kan göra det omöjligt för en person att använda utrustningen. Dessa kan antagligen antas gälla för alla system som använder sig av IR-ljus med ”pupil-center/corneal-reflection” metoden. Dessutom finns en lista över vilka medicinska diagnoser de användare har haft som kunnat bli hjälpta av LC Technologies produkt.
Priser. Desktop Eyegaze System: $14 900. Portable Eyegaze System: $15 900. Det senare systemet kan monteras på rullstol.
Utrustningen fästs inte på användaren. Vi har inte funnit någon angivelse av noggrannheten varmed användarens blickpunkt beräknas.

http://www.a-s-l.com/
Applied Science Laboratories har IR-baserade system, både med zoom-pan-tilt[3] kameror och med huvudmonterade kameror. De är huvudsakligen avsedda för forskning om ögonrörelser. Prisuppgift saknas, men med tanke på användningsområdet är produkten förmodligen mycket dyr.

http://www.smi.de/
SensoMotoric Instruments har IR-baserade system, både med zoom-pan-tilt kameror och med huvudmonterade kameror. De är huvudsakligen avsedda för forskning om ögonrörelser, och är således förmodligen mycket dyra. Exakt prisuppgift saknas. Systemet kan inkludera dator och mjukvara, vilket ytterligare höjer priset. De har också system som är avsedda för forskning om nedsättningar av ögonrörelse.

[3] Zoom-pan-tilt innebär att kamerans zoom och dess vridning kring två axlar är motorstyrda och kan ändras på signal från t ex en dator.

http://www.wirehub.nl/~skalar/
Skalar tillverkar bland annat ett system med en spole som är inbyggd i en ”kontaktlins”. Från kontaktlinsen går en sladd som kopplas till mätutrustning. Genom att mäta strömmen som induceras i spolen när man rör sig i ett varierande magnetfält, kan man beräkna ögats rörelse. Systemet har mycket hög noggrannhet. Produkten får endast användas på fullt friska personer för forskningsändamål. En fördel är att försök kan utföras även på djur. De tillverkar också huvudmonterade system. Prisuppgift saknas, och bedöms vara av mindre intresse i just det här fallet.

Forskning

http://www.isbe.man.ac.uk/~bim/
Tim Cootes utvecklade den populära och generella algoritmen Active Appearance Models, AAM. Den beskriver en statistisk modell som kan byggas med hjälp av träningsdata, och en algoritm för matchning till nya bilder. AAM bygger framför allt vidare på principalkomponentanalys, PCA. Länken leder till hans hemsida, och där kan man bland annat hitta en för våra ändamål intressant tillämpning på ansiktsföljning.

http://www.it-c.dk/research/EyeGazeInteraction/
Ett intressant projekt på IT-højskolen i Köpenhamn syftar till att konstruera en ögonföljare med låg upplösning som skall användas för kommunikation. Ögonföljaren skall kunna användas med en webbkamera. Sedan vårt examensarbete påbörjades har projektet resulterat i några intressanta publikationer som kan nås från ovanstående länk. Enligt en av dessa verkar det som om man bland annat använt sig av AAM i ögonföljningen.

http://www.lysator.liu.se/~eru/research/
Modellbaserad ansiktsföljning i 3D. Använder AAM och ansiktsmodellen CANDIDE. Utfört av Jörgen Ahlberg vid Linköpings Universitet.

Neurala nätverk och ögonföljning

Vår undersökning tydde på att de första mjukvarubaserade lösningarna för ögonföljning med hjälp av vanliga videokameror utnyttjade neurala nätverk. Detta innebär att man har en modell för hur hjärnans neuroner fungerar och utför datorsimuleringar med en sådan modell. Några intressanta publikationer i sammanhanget går att finna på webben.

http://www.cs.cmu.edu/afs/cs/user/baluja/www/papers/CMU-CS-94-102.ps.gz
Shumeet Baluja och Dean Pomerleau verkar vara pionjärerna i området. De presenterade redan 1994 denna uppsats som beskriver hur ett neuralt nätverk använts för att åstadkomma gaze tracking.

Dessutom finns i rapporten en beskrivning av resultaten från försöket, som är anmärkningsvärt goda.

http://isl.ira.uka.de/mie/eyegaze.html
Rainer Stiefelhagens projekt är stort sett en oberoende implementation av Baluja och Pomerleaus neurala nätverk.

http://isgwww.cs.uni-magdeburg.de/sg2002-crv/12-Ji.pdf
Även Qiang Ji och Zhiwei Zhu presenterar i ovanstående rapport resultaten av en blickföljare baserad på neurala nätverk. De använder en IR-baserad hårdvarulösning.

Kodbibliotek för bildanalys

Vi har tidigare diskuterat möjligheten att utnyttja ”open source” projekt. Det enda lovande projekt vi funnit är Intels kodbibliotek OpenCV. OpenCV står för Open Computer Vision och är en samling algoritmer för datorseende och grundläggande funktioner för bildbehandling som kan användas både i Windows och i Linux. De finns i färdigkompilerade versioner optimerade för Intels olika processorer.

Här följer två officiella länkar till projektet:
http://www.intel.com/research/mrl/research/opencv/
http://sourceforge.net/projects/opencvlibrary/

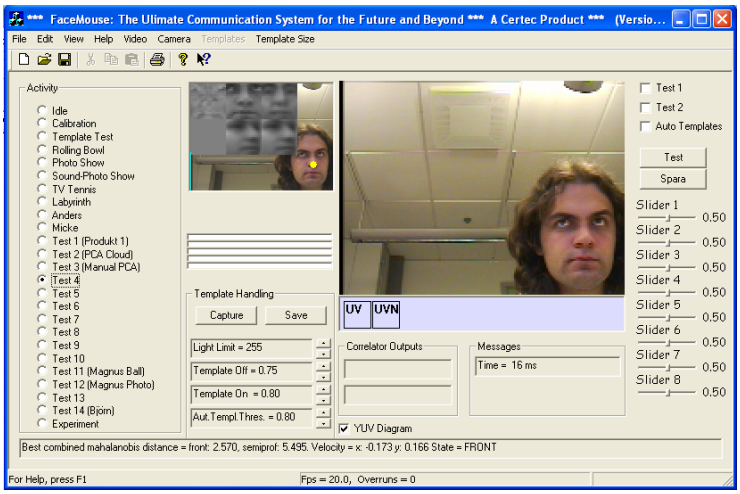
Så långt vi kom i planeringen gjorde vi bedömningen att algoritmerna vi använde var enkla nog att implementera själv i stället för att använda OpenCV. En fördel med att skriva dem från grunden är att man får större kontroll över deras beteende. Största nackdelen är att det är mycket mer tidskrävande.

Förutom OpenCV fann vi för C++ bara ett antal mer specialiserade kodbibliotek, som skrivits av forskare inom de respektive områdena. Det finns anledning att misstänka att mognadsgraden i dessa bibliotek är låg, eftersom de inte är avsedda för användning i kommersiella produkter med höga stabilitetskrav.

Efter att ha sökt på internet efter open-source projekt för datorseende drar vi slutsatsen att det är ett ovanligt område för open-source projekt. Vi skulle bara kunna spekulera i varför det är så. Kanske beror det på att det är relativt nyligen som datorkraften ökat och kamerapriserna minskat tillräckligt mycket för att möjliggöra avancerad bildbehandling på persondatorer.

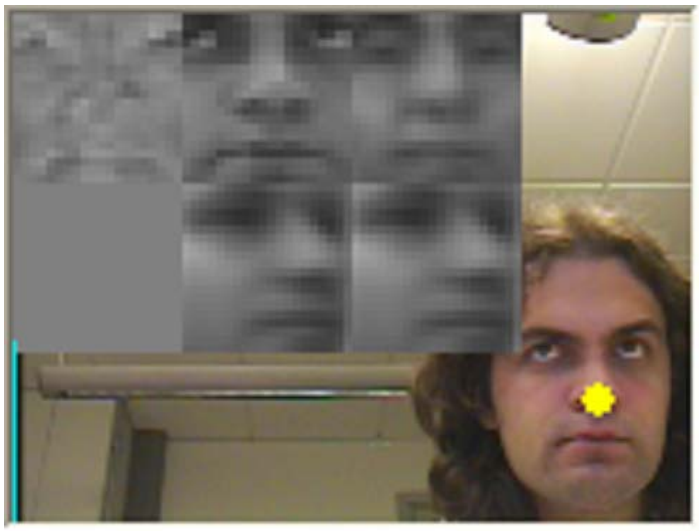
Prototyp

Vår handledare försåg oss med ett grafiskt gränssnitt som innehöll knappar länkade till tomma funktioner. I dessa tomma funktioner lade vi in vår experimentkod. Detta gav oss ett slags grafiskt laboratorium. I Figur 5 ser man hur det kunde se ut. Den stora bilden i fönstret visar videosekvensen som erhålles från kameran. Den mindre bilden är ett utritningsfönster där behandlade bilder ritades, eventuellt med visa tillägg, som skulle möjliggöra utvärdering av algoritmer. En viss interaktivitet var möjlig genom detta ritfönster. Det var t ex möjligt att låta experimentatorn klicka i ritfönstret för att välja ut ett område av intresse i bilden.



Figur 5. Det grafiska gränsnittet för vårt "algoritmlaboratorium". Kolonnen av radioknappar längst till vänster lät användaren välja mellan olika testfunktioner.

Figur 6 visar en förstoring av ritfönstret. De sex rutorna visar olika resultat från den egenansiktes-baserade algoritmen för ansiktsföljning. Den översta rutan i mitten visar det område i bilden som valdes ut som mest ansiktslikt. Till höger om denna syns dess rekonstruktion m h a egenansiktena och till vänster syns skillnaden mellan rekonstruktion och verklig bild. I skillnadsbilden betyder mörkt och ljust att det är stor skillnad (negativ eller positiv skillnad), medan mellangrått betyder liten skillnad. Rutorna på den undre raden kommer av experiment med egenansikten framtagna för ansikten i olika grader av profil. De två ansiktena man ser där är endast medelvärdesbilderna för två olika grader av profil. Pricken över näsan på ansiktet som skymtar längst ner till höger markerar centrum för den utvalda rektangeln.



Figur 6. En testfunktion som använde egenansikten och steepest descent för att följa ett ansikte. För mer om teorin bakom detta, se appendix 1 nedan.

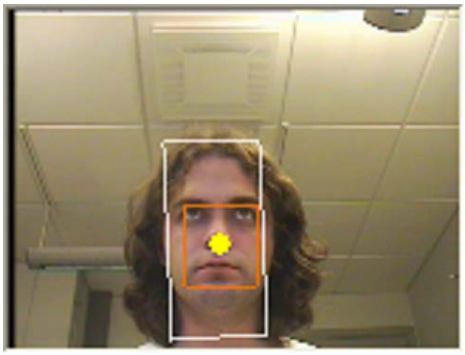
Figur 7 visar ritfönstret när en annan testfunktion använts. Här får användaren manuellt markera en rektangel, den mittersta av de tre. Till höger visas rekonstruktionen som gjorts med egenansiktena och till vänster visas skillnaden mellan rekonstruktion och den verkliga bilden i rektangeln. I figuren ser vi vad som händer när algoritmen får arbeta med något som inte alls föreställer ett ansikte. Resultatet av algoritmens försök att återskapa ett ansikte påminner om ett ansikte, men om man ser på skillnadsbilden så är det större skillnad här än i exemplet ovan (se Figur 6).
Detta illustrerar algoritmens förfarande. En bild projiceras på ansiktsrummet, vilket i någon mening tar fram de mest ansiktslika dragen i bilden. Man skulle alltså kunna säga att algoritmen är lite fantasifull. Därefter jämförs detta rekonstruerade ansikte med ursprungsbilden. Ifall de är lika så är det troligt att bilden föreställer ett ansikte.

Figur 7. I den här testfunktionen kunde man markera en ruta i bilden för att se hur mycket den liknar ett ansikte enligt en egenansiktesbaserad algoritm.

I Figur 8 visas till höger ett moln som gjorts på bilden till vänster. Molnet tillkommer genom att man sveper med ett sökfönster av fix storlek över bilden och ritar på varje bildpunkt en gråvärde som är ljusare ju mera området kring denna liknar ett ansikte (enligt vårt PCA-avstånd). Man ser här att det finns ett område som är ljusast och att det svarar ungefär mot ansiktets medelpunkt. Det är detta område som visas infällt i bilden till höger.

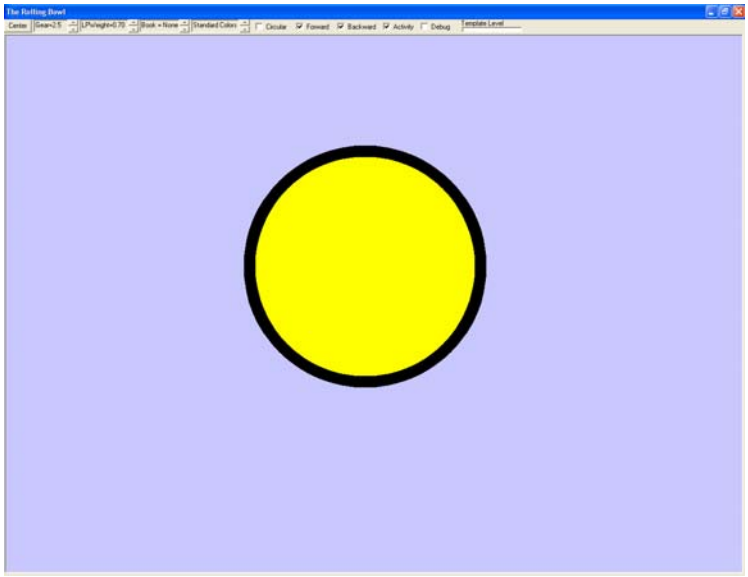
Figur 8. Ett ansikte och dess moln, framtaget med en egenansiktesbaserad algoritm för ansiktsigenkänning.

Figur 9 visar ett experiment där både färg och PCA användes för att hitta och följa ett ansikte. Den stora rektangeln är det område som tagits fram m h a färg och den mindre det område som sedan funnits vara mest ansiktslikt inuti detta.



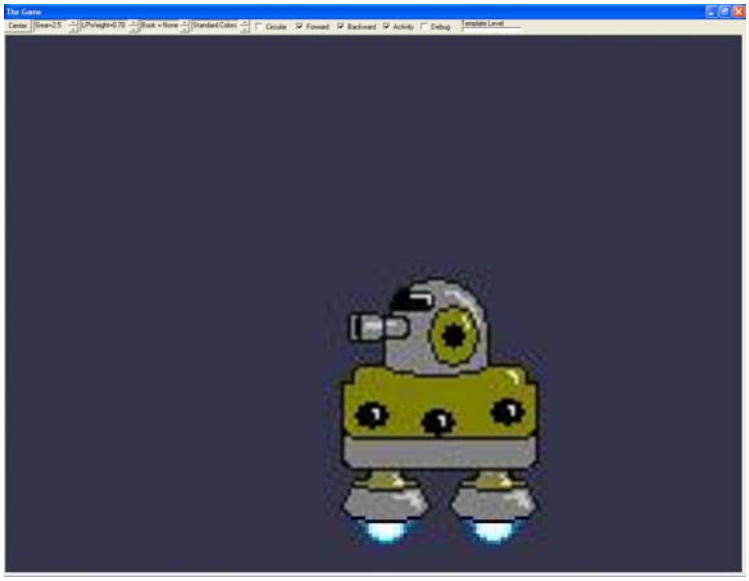
Figur 9. Ett experiment som använde både egenansikten och färg.

Den första applikationen åt Magnus var Ja/Nej-klotet. Vi gjorde en modifikation på det, en svart konturlinje runt klotet. Tanken med detta var att klotet skulle bli lättare att urskilja för honom, då vi trodde att han kanske såg lite dåligt.



Figur 10. Ja/Nej-klotet med svart konturlinje.

Vi började arbeta på ännu en applikation, som dock inte blev helt färdig. Istället för en enbart nyttig och användbar applikation tänkte vi oss något som skulle var roligt och underhållande för Magnus. Tanken här var att något som fångade hans intresse skulle få honom att röra sig mera och därigenom öva på de rätta rörelserna. Dessutom skulle denna applikation kunna förses med samma funktionalitet som Ja/Nej-klotet. Se Figur 11.



Figur 11. En applikation med samma styrsignal som Ja/Nej-klotet, där användaren kunde kontrollera ett litet rymdskepp med sina huvudrörelser.

Provanvändning med Magnus

Vid två tillfällen genomfördes provanvändning med Magnus Andersson-Jardeby, vars funktionshinder beskrivs nedan. Provanvändningarna dokumenterades på videofilm.

Figur 12. Här är vår provanvändare Magnus Andersson-Jardeby. Bilden har tagits av videokameran som är kopplad till datorn vi använt i examensarbetet.

Vår förhoppning var förstås att finna en teknisk lösning som var skräddarsydd åt Magnus. För att denna skulle kunna vara till långsiktig glädje för honom skulle det krävas någon specialutvecklad applikation. Detta låg utanför målet med vårt arbete, men vi hoppades kunna skriva någon speciell applikation som skulle kunna väcka Magnus intresse allteftersom arbetet med provanvändning fortskred.

Emellertid blev det bara två provanvändningar, och de skedde vid slutet av den utstakade tiden för examensarbetet. Men under provanvändningarna hade vi ändå flera roliga applikationer åt Magnus, nämligen de som Björn utvecklat åt Emma. De fungerade precis lika bra ovanpå vår tekniska lösning som de hade gjort i Björns ursprungliga version av *Minimetern*. Detta var en av fördelarna med att vi utnyttjade det kodbibliotek som Björn etablerat under sitt tidigare arbete.

Dels fanns Ja/Nej-klotet som beskrivits ovan, dels fanns ett program för bildvisning. Användaren kan gå till nästa bild genom att vrida huvudet åt höger, och föregående genom att vrida åt vänster. Till varje bild kan man lägga in något ljud, till exempel en musiksnutt.

Första mötet med Magnus

Här sammanfattas vad vi fick lära oss om Magnus och om vårt datorprogram när han och hans mamma Birthe Jensen var på besök 2002-12-16. Med var också en av Magnus två assistenter, Elisabet Einarsson. Magnus kommer också i mer eller mindre regelbunden kontakt med sjukgymnast, logoped, arbetsterapeut och lärare. Han går i skola på Fjärilen på Emaljskolan i Landskrona.

Magnus är 15 år gammal och har en CP-skada[4] sedan födseln. Han är stelopererad i ryggen. Magnus funktionshinder gör att han inte alls kan tala.

[4] Förkortning för cerebral pares, vilket betyder förlamning på grund av hjärnskada. Termen är ett samlingsbegrepp för rörelsehinder som orsakats av hjärnskada under graviditeten eller den första levnadsveckan. Ofta förekommer också andra funktionshinder än de motoriska.

Han har två rullstolar, och den nyaste av dem har nackstöd och pannband, som kan hålla huvudet på plats. Den nya rullstolen har två brister. Pannbandet glider lätt av p g a materialet och Magnus sluttande panna. Dessutom är nackstödet inte anpassat till Magnus sneda rygg, mer om detta nedan.

Figur 13. Eftersom Magnus har stort omfång i sina huvudrörelser, ställs stora krav på algoritmen för att följa hans ansikte

Här följer en beskrivning av Magnus viloläge. Han sitter alltid med munnen mer eller mindre öppen. Det är viktigt att veta eftersom en av våra algoritmer använde sig av träningsdata bestående av ansikten med stängd mun. Magnus vilar med huvudet lutande ner åt vänster. Magnus visade sig ha stort omfång för sina huvudrörelser. Från kamerans synvinkel kunde han ses titta både neråt, rakt till höger och snett ner till vänster.

Elisabet berättade att nackstödet är centrerat i förhållande till rullstolen, och inte går att flytta i sidled. Det är därför inte anpassat till Magnus sneda rygg, som gör att han när han har huvudet rakt fram inte befinner sig mitt i nackstödet.

Vi ville gärna veta lite om Magnus syn. Vi fick veta att Magnus vänstra öga var det mest aktiva för några år sedan, när de kontrollerade den saken sist. Magnus assistent berättade att det nog var dags för en synundersökning snart – Magnus syn hade troligen förändrats sen den förra. Om vi förstod Elisabet rätt hade det funnits en rätt lång tidsperiod då Magnus hade större nytta av hörseln än av synen, nämligen innan hans steloperation. Då brukade han mest titta nedåt.

Magnus har viss datorvana – han har en dator med olika spel och lekprogram hemma, som han styr genom att trycka på en knapp med handen.

Eftersom Magnus inte kan tala, undrade vi hur hans mamma och assistenterna kommunicerar med honom. Birthe berättade, att när det gäller någon ja/nej fråga, så brukar hon ställa sig rakt bakom Magnus och hålla hans huvud rakt fram. Sen ställer hon frågan, och låter honom få god tid på sig att röra huvudet vänster eller höger. Vänster betyder nej, och höger betyder ja! Elisabet berättade att hon brukar göra likadant, men ibland följer hon inte protokollet så noggrant, utan låter Magnus utgå från viloläget. Fortfarande betyder vänster nej, och höger ja. Vi tror att det finns två problem med att utgå från viloläget. Dels att det med Magnus rörelsemönster finns mer utrymme att röra sig åt höger, och dels att det kanske är mer naturligt att lyfta på huvudet när någon tilltalar en.

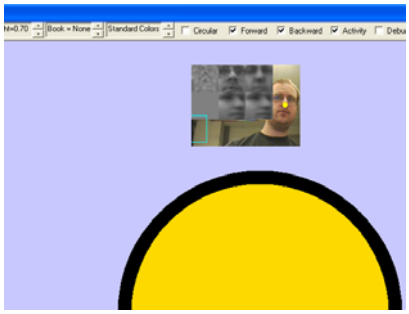
Ingående moment

Vår provanvändning innehöll följande moment:

1. Magnus provar Ja/Nej-klotet
2. Magnus provar Ja/Nej-klotet med pannbandet på.
3. Vi använder Björns algoritmer från *Minimetern* med pannbandet på. Vi använder munnen som mall. Applikationen är Ja/Nej-klotet.
4. Magnus provar Bildspelet – eller rättare sagt Björn visar bildspelet för att se vad han tycker om musik och bilder.
5. Vi spelar in en film med Magnus ansikte.
6. Vi spelar in en film med Magnus ögon. Björn håller i Magnus huvud för att han inte skall försvinna ur kamerans synfält.

Figur 14. I bilden visas bildskärmen och Ja/Nej-klotet, videokameran och handkontrollen.

För att förenkla för en assistent att hjälpa användaren med applikationerna används en handkontroll istället för mus och tangentbord när man kör applikationerna. Handkontrollen kan användas för att centrera Ja/Nej-klotet och för att visa algoritmens resultat infällt i skärmen. Tyvärr hade vi vid första provanvändningen inte skrivit programkoden för att använda den med vår version av programmet.



Figur 15. Detta är en förstoring av en del av gränssnittet för det rullande klotet. Då man trycker på B-knappen på handkontrollen skall en infälld bild dyka upp som illustrerar algoritmens arbete. Det är den som syns ovanför Ja/Nej-klotet. Konturlinjen i klotet fanns inte med i första provanvändningen, det var något vi införde till den andra.

Vid detta provtillfälle fungerade inte heller bildspelet med vår version av programmet. Bara enklast tänkbara möjliga version av rullande klotet. Däremot fungerade Björns ursprungliga algoritmer från *Minimetern* bra med bildspelet.

I moment 1 och 2 användes en färg- och rörelsebaserad algoritm för ansiktsdetektion och ansiktsföljning. Detta var vår allra första ansiktsföljare som fungerade någorlunda bra. Vi hade inte möjlighet att ta några skärmdumpar när Magnus använde programmet, så i bilderna nedan får Arvid vara användare för att illustrera hur de fungerar.

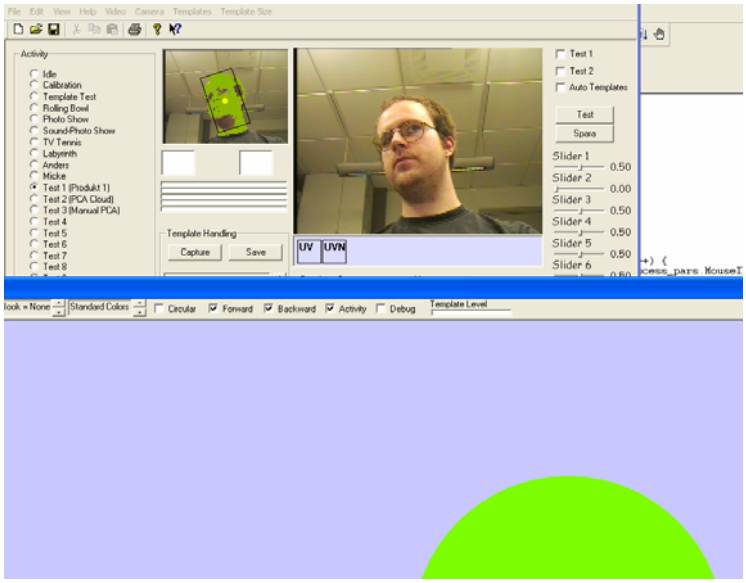
Först försöker algoritmen hitta några distinkta färger i bilden. De svarta ellipserna i färgkartan till vänster visar vilka färger den funnit.

Figur 16. Färgalgoritmen håller på att konvergera. Den försöker hitta ett antal distinkta färger i bilden till höger.

En av färgerna antas vara hudfärg. För att lista ut vilken det skulle kunna vara, undersöker algoritmen vilken färg som uppvisar mest rörelse. Bakgrunden måste alltså vara stilla.

Figur 17. Algoritmen har funnit en färg som uppvisar mycket rörelse. Den har markerats med grönt i bilden till vänster.

I Figur 17 visas med grön markering vilken färg algoritmen fann. I bästa fall är det ansiktet, men det skulle ju också kunna vara något annat som rörde sig. En rektangel, markerad med svart i figuren, anpassas till området. Sedan används den rektangelns centrum, markerad med en gul prick, som styrsignal. Ja/Nej-klotet behöver en endimensionell styrsignal, och därför används bara den gula prickens x-koordinat.



Figur 18. Bilden visar hur Arvid styr det rullande klotet mot ”ja” genom att röra huvudet åt vänster. (Bilden är beskuren, och visar bara en del av det som syns på datorskärmen).

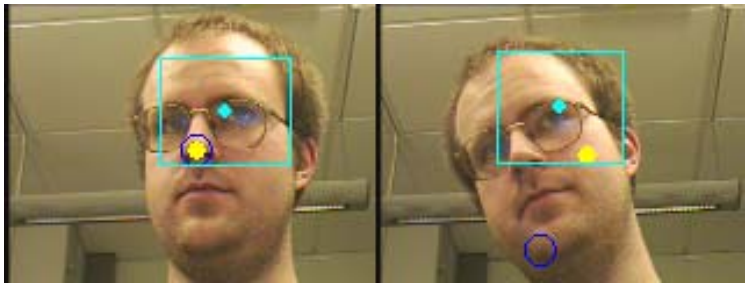
Algoritmen är mycket robust för förändringar i det ansiktsfärgade områdets form. Problem kan dock uppstå om ett nytt objekt med samma färg dyker upp, till exempel en hand eller ett annat ansikte. Det som innan var en fördel, att den var robust för förändringar i ansiktets form, vänds nu till dess nackdel. I Figur 19 tror algoritmen att handen är en del av ansiktet, eftersom den kommer nära. Efter det följer algoritmen läget av det kombinerade objektet hand plus ansikte.

Figur 19. En hand dyker upp i bilden. Den har samma färg som ansiktet, och när den kommer för nära tror algoritmen att den är en del av ansiktet. Då ”fångas” den upp av rektangeln, och pricken anger inte längre ansiktets position.

I moment 3 och 4 användes Björns mallmatchare från den förra versionen av *Minimetern*. Därför berättar vi här också kortfattat hur den fungerar.

Först skall assistenten ange ett område, eller mall, som skall följas. Följningen går till så att algoritmen i varje ny bild från kameran letar efter ett område som liknar mallen man angav. Om algoritmen inte hittar något, stannar följningen av tills den hittar ett liknande område igen. Den här algoritmen fungerar utmärkt om användaren bara gör små rörelser.

Figur 20. I bilden till vänster har assistenten just markerat näsan som det område algoritmen skall följa. I bilden till höger har den hittat något som liknar innehållet i kryssrutan. Det som algoritmen hittat, i det här fallet näsan, markeras med en prick.



Figur 21. Algoritmen har till höger misslyckats med att hitta något som liknar en näsa tillräckligt mycket. Problemet är att Arvid har roterat huvudet, och alltså även näsan, så att den inte längre liknar det ursprungliga området.

Man kan också använda något annat än näsan, till exempel munnen.

Figur 22. Först markeras munnen. I nästa bild har algoritmen hittat nåt som liknar det ursprungliga området tillräckligt mycket. Sen stänger Arvid munnen, och algoritmen tappar spåret. Längst till höger har den hittat något igen.

Resultat och diskussion

Moment 1. Magnus flyttar klotet åt vänster och åt höger. Han fastnar gärna i extremlägena ja och nej. Hans naturliga viloläge är inte i mitten, alltså inte tittande rakt fram. Programmet fungerar över förväntan, men den färgbaserade ansiktsföljaren absorberar gärna andra hudfärgade saker i bakgrunden. Magnus viftar till med vänsterhanden ibland – hans mor säger att det kan vara en tillfällighet för just idag. Om handen kommer i kamerans synfält kan den störa följningen om den rör sig för nära ansiktet. Vi provar att be Magnus ”Ge mig ett Ja-klot”/”Ge mig ett Nej-klot” med svårtolkade resultat – han har en tendens att hamna på fel ställe, eller vila vid ja eller nej långa tider. Han har svårt att återvända till neutralläge. Trots att Magnus normala viloläge är vårt ”Nej” d v s vänster, fastnar han ibland borta på Ja också.

Moment 2. Det var svårt att säga om pannbandet var någon förbättring. Kanske stretar Magnus emot pannbandet? Resultaten var ungefär som i punkt ett – han svarar ja och nej omväxlande, och hamnar gärna längre stunder i ändlägena.

Moment 3. Björns mallmatchare funkade ganska bra, med munnen som mall, men inte lika bra med näsan som mall. Ungefär samma beteende hos Magnus som i test 1 och 2. Med pannbandet på fungerade alltså Björns ursprungliga algoritm någorlunda väl.

Moment 4. Under det här momentet bytte vi bild i bildspelet manuellt med hjälp av handkontrollen. Magnus verkar intresserad av musiken, men tittar åt lite olika håll.

Moment 5. Magnus sitter i sin gamla stol, utan pannband. Tanken med filmningen var att vi skulle kunna använda den för att träna en algoritm på just Magnus ansikte.

Moment 6. Björn håller i Magnus huvud, för att han inte skall falla utanför kamerans synfält. Magnus verkar nästan inte alls följa med olika föremål med ögonen. Vi provar olika föremål som låg i närheten: färgade papper och en färgglad skruvmejsel, utan någon större respons. Ett något nedslående resultat. Birthe berättar att han brukar kunna följa rörelser bättre än så. Kanske har Magnus blivit trött, eller kanske är han distraherad av att någon håller i hans huvud.

Under provanvändningen märkte vi att Magnus vid två eller tre tillfällen får en till synes ofrivillig muskelsammandragning. Då drogs hans huvud ner åt vänster och ögonen rullade uppåt. Det ställer förstås krav på att algoritmen klarar att återhämta sig efter denna snabba förflyttning. Den färgbaserade ansiktsföljaren har inga problem med detta.

Björn påpekade hur viktigt det var med sjukgymnastik i fallet med Emma. Han föreslog att Magnus ser till att träna på huvudrörelser, och att hålla ett neutralt huvudläge. Träningen skulle förutom att vara nyttig rent allmänt förstås kunna förbättra Magnus förmåga att svara ja och nej.

Birthe sade att hon ändå var imponerad över hur ofta Magnus höll huvudet rakt fram under besöket här.

Vi bestämde oss för att inte skriva ett program som har Magnus viloläge som neutral position, eftersom det då inte är fråga om två likvärdiga rörelser för ja och nej.

Vi vågade inte säga så mycket om Magnus syn. Vid försöket hade vi bildskärmen ungefär så långt in från bordskanten att kameran fick plats framför, och Magnus fick sitta så långt från bordet att händerna inte skulle slå i bordet om han gjorde en ofrivillig rörelse. Han ser antagligen inte bra på långt avstånd, siffran 3 dm nämndes som lagom bildskärmsavstånd. Det var ingen som tänkte så noga på att kontrollera vart Magnus tittade under försöket. Men Arvid fick intryck av att det var lite hit och dit, inte så noga fixerat på klotet i alla fall. Vi bestämde oss för att i fortsättningen använda mycket kontraster, gärna tydlig grafik med svarta konturlinjer. Detta kanske hjälper ifall Magnus ser dåligt.

Vi funderade på något roligt program, roligare än Ja/Nej-klotet, som kunde uppmuntra Magnus att träna neutrala läget, samt de två extremlägena för ja och nej. Magnus gillar nog ljud, så vi borde ha något roligt stereoljud som följer rörelsen när han går utmed

skärmen. Klassisk musik och lite tuffare musik kan båda vara intressanta – Magnus har visat intresse för olika typer av musik.

Till nästa möte bestämde vi oss för att se till att handkontrollen fungerade som den skulle. Vi borde försöka specialanpassa en *egenansiktsbas* (se appendix 1 - Teori) till Magnus. Vi borde även ha en tumstock och kontrollera monitoravståndet för att se var Magnus ser bra. För algoritmerna är det viktigt att tänka på att Magnus ansikte ofta syns i profil från kameran.

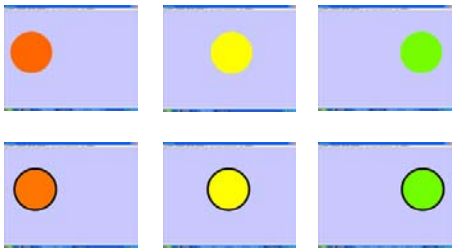
Vi bestämde oss för att prova samma metod som Birthe använder hemma när hon frågar Magnus något – att hålla i huvudet och låta honom välja Ja/Nej. Det är viktigt att huvudföljaren klarar att följa Magnus huvud, och inte blir lurad av att Birthes händer eller ansikte dyker upp i bilden. Detta skulle innebära problem för den färgbaserade ansiktsföljaren vi använde vid detta tillfälle.

Andra mötet med Magnus

Eftersom vi inte hunnit ändra särskilt mycket sedan första mötet blev den här provanvändningen ganska oorganiserad. Endast en arbetsdag hade förflutit mellan mötena.

Vi hade förberett en uppsättning *egenansikten* som var tränade på några bilder hämtade ur filmerna vi spelade in vid förra mötet.

Det visade sig finnas två problem med algoritmen. Vi hade inte tillräckligt många träningsbilder, så den visade sig inte kunna generalisera tillräckligt bra till bilderna av den verklige Magnus som satt framför datorn.



Figur 23. Bilderna visar det rullande klotet före och efter tillägget av konturlinje.

Tanken var att algoritmen skulle byta till semiprofil-läge när Magnus började vända bort ansiktet. Det innebär rent tekniskt att den byter till en annan uppsättning egenansikten som är tränade på semiprofil-bilder, när dessa beskriver bilden bättre. Vi hade i vårt arbete haft problem med detta byte, och det fungerade tyvärr inte under provanvändningen.

Vi provade att lägga en tjock svart konturlinje runt det rullande klotet, så att det skulle synas bättre. Vi tänkte att för till exempel en närsynt person kanske det blev dålig kontrast mellan den blå bakgrunden och det gula klotet. Magnus svarade ”ja”, eller rättare sagt rörde huvudet åt höger, på frågan om han såg klotet. Eftersom vi inte frågade förra gången vet vi inte om han såg klotet ordentligt då.

Vi provade att låta Magnus bläddra bland låtarna i musik och bild-spelet. Magnus fick ingen bra kontroll över det, utan bytte lite slumpmässigt mellan låtarna. Vår ansiktsföljare var att skylla där,

eftersom den som sagts tidigare inte klarade av att följa hans ansikte när det gick över i semiprofil.

Vi talade lite om vilken typ av program vi skulle specialskriva åt Magnus. Hans mamma ville gärna ha något som väckte hans intresse med hjälp av ljud, och som sen stimulerade hans syn. Hon ville gärna att han skulle få öva mer på att använda synsinnet. Men vi kom fram till att det kanske var svårt att väcka Magnus intresse om det inte var ljud med i programmet. Vi borde testa att lägga in positionerat ljud i Ja/Nej klotet eller i någon annan tillämpning.

Birthe tyckte det var synd att Magnus oavsiktligt bläddrade bakåt när han tittade på musik/bild-spelet. Hon föreslog att vi gjorde någon slags bok eller film som man kunde bläddra i. Hon sa att man kunde ta bort den funktionen och bara låta honom bläddra framåt, för att undvika att det blir så rörigt.

Birthe tyckte att ett intressant tema för ett bildspel skulle vara Harry Potter. När hon nämnde det namnet gav Magnus ifrån sig ett ljud som visade förtjusning! Birthe brukar nämligen läsa högt ur J. K. Rowlings böcker om denne unge trollkarlselev för Magnus.

Vi anser att det vore viktigt att anpassa ett styrsystem till Magnus nuvarande rörelseförmåga. Sjukgymnastik skulle kunna förbättra denna, men innan dess borde det ändå gå att utveckla någon applikation som stimulerar huvudrörelser i allmänhet.

Vi hann alltså inte göra något utstuderat test med Ja/Nej-klotet. Vi tyckte inte det var lönt med den nya halvfärdiga ansiktsföljaren.

Diskussion och Slutsatser

Programmeringsmetod

Ett av de mest lyckade besluten i början av exjobbet var att bygga vidare på Björns kod i stället för att börja om från grunden. Detta ledde till att vi kunde sätta igång med bildbehandling och datorseende omedelbart. Det vi hade gratis var klasser för att komma åt videoströmmen från kameran, och ett grafiskt gränssnitt för att interagera i realtid med de algoritmer vi skrev.

Därför drar vi slutsatsen att det borde löna sig för våra efterföljare att jobba vidare i samma ramverk och kanske också förbättra våra klasser eller kombinera dem på nya sätt och med nya algoritmer.

Vi vill påpeka att även om programspråket C++ tillåter högre abstraktionsnivå än vanliga C, så finns det programspråk på ännu högre nivå som kan vara lämpliga för utveckling av prototyper. Till exempel kan MATLAB vara lämpligt för prototyper som innehåller mycket numeriska beräkningar. För den slutgiltiga implementationen är det dock svårt att se något alternativ till C++ om beräkningarna skall kunna utföras i realtid med en videoström som indata.

Det kan vara lämpligt att vid den slutgiltiga implementationen i C++ utnyttja ett färdigt kodbibliotek för mer avancerade delalgoritmer. I så fall skulle vi rekommendera Intel OpenCV som beskrivs mer ingående i avsnittet Kodbibliotek för Bildanalys nedan.

Applikationer

Vi anser att även om tekniken i vår lösning har sina begränsningar kan man ändå utveckla användbara applikationer på den bas vi etablerat. Många tillkortakommanden i tekniken bakom styrsättet, i det här fallet följning av huvudrörelser, kan man kompensera för i applikationens gränssnitt. Björn Breidegards applikationer till den förra versionen av *Minimetern* visar att det är möjligt att finna nyttiga och roliga tillämpningar trots att Emma bara kunde röra huvudet vågrätt, och därför bara kunde utnyttja en av de två dimensionerna i styrsignalen.

Vidare anser vi att det är viktigt att applikationen även anpassas till individens förmåga. Det är inte på förhand givet att en applikation måste kunna hjälpa användaren att göra något som han eller hon

annars inte kunde göra. Till exempel kan en applikation som vid första betraktande bara verkar ha nöjesvärde även ha nyttovärde i och med att den stimulerar användaren att träna vissa rörelser.

Robusthet

Ansiktsföljaren som den nu står kräver manuell kalibrering för att starta. Rent praktiskt innebär detta att man med musen drar en ruta runt det ansikte man vill att programmet skall följa.

För att förstå nedan beskrivna problem är det viktigt att förstå teorin bakom principalkomponentanalys, PCA, och hur denna metod utvecklas till så kallade egenansikten (eng *eigenfaces*). Detta beskrivs i teorikapitlet.

Robustheten förbättrades betydligt då vi använde en detaljbild istället för hela ansiktet, d v s en rektangel runt ögon, näsa och mun som träningsdata istället för att även inkludera öron och hår. Dessa delar av ansiktet uppvisar mindre variation eftersom egenansiktena inte behövde ta hänsyn till variation i frisyr. Rutan man ritar för att sätta igång följningen skall alltså omfatta ögon, näsa och mun, men inte håret.

Robustheten sätts på spel vid hastiga rörelser – då kan det hända att följaren ”tappar spåret” och fastnar. Då krävs en ny manuell kalibrering för att den skall hitta rätt igen. Detta är en begränsning som beror på valet av följningsalgoritm, *steepest descent*, som beskrivs i avsnittet ”PCA-baserad ansiktsföljning” nedan. I programmeringstekniska termer säger man att steepest descent är en *girig* algoritm. Detta innebär att den alltid gör vad som verkar bäst för stunden.

Om ansiktet vänds bort från porträttläge till profil kan följningen också tappa spåret. Då liknar nämligen inte ansiktet i profil egenansiktena tillräckligt mycket, eftersom dessa är tillverkade med hjälp av ansiktsbilder som är tagna rakt framifrån.

För att lösa problemet med hastiga rörelser skulle man behöva en långsammare ansiktsfinnare som kan ta god tid på sig för att hitta ansiktet igen så man slipper interagera med programmet manuellt. Ett sätt att göra detta skulle vara att göra en uttömmande sökning i bilden – låt en rektangel vandra runt i bilden tills man hittar ett område som beskrivs väl av egenansiktena.

För att lösa problemet med profil skulle man kunna ha flera uppsättningar egenansikten och byta mellan dessa när en ger bättre likhet än de andra. Alternativt skulle man kunna inkludera även profilbilder i träningsmängden. Problemet är att den stora variation

som då uppstår i träningsmängden skulle kunna försämra dess förmåga att skilja på ansikten och icke-ansikten.

Vad händer sedan?

Björn Breidegard kommer att arbeta vidare med *Minimetern*, och förhoppningsvis kommer också något framtida examensarbete att bygga vidare på projektet.

En viktig fortsättning på vårt arbete skulle vara att utveckla en individanpassad applikation för någon funktionshindrad person. I slutändan är förstås tekniken ingenting värd utan en applikation, samtidigt som en applikation inte kan implementeras utan att tekniken fungerar.

När det gäller vidare teknikutveckling rekommenderar författarna att man utvärderar Active Appearance Models. För mer information om denna statistiska modell med tillhörande matchningsalgoritm, se avsnittet Forskning i kapitlet Marknadsundersökning.

Appendix 1 –Teori

Detta kapitel innehåller avsnitt som beskriver de metoder och algoritmer som använts i programmet, och matematiken bakom dessa. Avsnitten presenteras i en ordning som på ett naturligt sätt leder fram till de viktigaste begreppen: *Principalkomponentanalys* för att beräkna *egenansikten* och *EM-algoritmen* för färgsegmentering.

Läsaren kan behöva slå upp definitioner av vissa grundläggande matematiska begrepp som används nedan. Därför kan läroböcker i linjär algebra och matematisk statistik vara bra att ha till hands. Författarna rekommenderar (Sparr, 1995) för linjär algebra samt (Blom, 1984) och (Blom, 1998) för matematisk statistik.

Notation

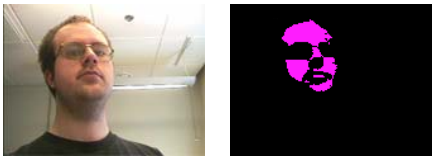
Här beskrivs delar av den matematiska notation som använts i detta kapitel.

- Kursiv stil betecknar en skalär: x
- Kursiv stil används även för att beteckna en endimensionell stokastisk variabel: x
- Fetstil betecknar en vektor: $\mathbf{x}$
- Fetstil används även för att beteckna en flerdimensionell stokastisk variabel: $\mathbf{x}$
- Om man infört en vektor $\mathbf{x}$ betecknar x_i den i:te komponenten i vektorn: $\mathbf{x} = (x_1, x_2, x_3, \ldots)$. Motsvarande gäller för stokastiska variabler.
- Fetstil och stor bokstav betecknar en matris: $\mathbf{A}$
- Lilla sigma med index betecknar standardavvikelsen för motsvarande stokastiska variabel: σ_{x_1}
- Lilla sigma med två index betecknar kovariansen för den kombinationen av stokastiska variabler: $\sigma_{x_1 x_2}$
- Upprepat index betyder varians: $\sigma_{x_1 x_1} = \sigma_{x_1}^2$
- Stora sigma med fetstil betecknar kovariansmatrisen:
$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_{x_1 x_1} & \sigma_{x_1 x_2} & \cdots \\ \sigma_{x_1 x_2} & \sigma_{x_2 x_2} & \\ \vdots & & \ddots \end{pmatrix}$$
- Väntevärdet av den stokastiska variabeln $\mathbf{x}$ betecknas $\mathsf{E}[\mathbf{x}]$.

Enkel statistisk modellering

Ett statistisk synsätt har använts i många av de algoritmer som använts i detta projekt. Metoden vi använt kan övergripande beskrivas så här:

- Tänk efter vad som skall modelleras och vad som skall beräknas
- Finn en statistisk modell
- Finn en algoritm som beräknar de önskade storheterna, och som bygger på den statistiska modellen



Figur 24. Till höger har man med någon metod maskat ut ett intressant område i bilden. Observera att bilden bara är en illustration, inte en skärmdump.

Ett enkelt exempel är följande problemställning:

> *Av alla pixlarna i en bild har vi maskat fram ett område som intresserar oss. Vi vill få en högnivårepresentation av området som sammanfattar dess läge, rotation och storlek.*

Figur 24 illustrerar maskningen som lett fram till det intressanta området. Vitsen med en högnivårepresentation är att man slipper lagra listan över varenda pixel som ingår i området – man behöver bara spara fem värden: bredd, höjd, x-koordinat, y-koordinat och rotation. Dessutom skulle några av dessa kunna vara intressanta styrsignaler för en applikation.

En mängd element som är samlade, så att de kan uppfattas som närliggande, som en klunga, brukar kallas för ett kluster. Säg att vi kommer fram till följande:

> *Det som skall modelleras är ett kluster av pixlar. Det som skall beräknas är en approximation av klustrets storlek och orientering.*

Figur 25. Rektangeln visar en approximation av klustrets storlek och orientering.

Rektangeln i Figur 25 visar vad det är vi vill beräkna – klustrets storlek illustreras av rektangelns storlek och dess orientering av rektangelns lutning. Läs i avsnittet ”Approximation av ett 2D-klusters form” för detaljerna om vilken modell som valdes och vilken algoritm som använts för att beräkna de sökta storheterna.

Normalfördelningen

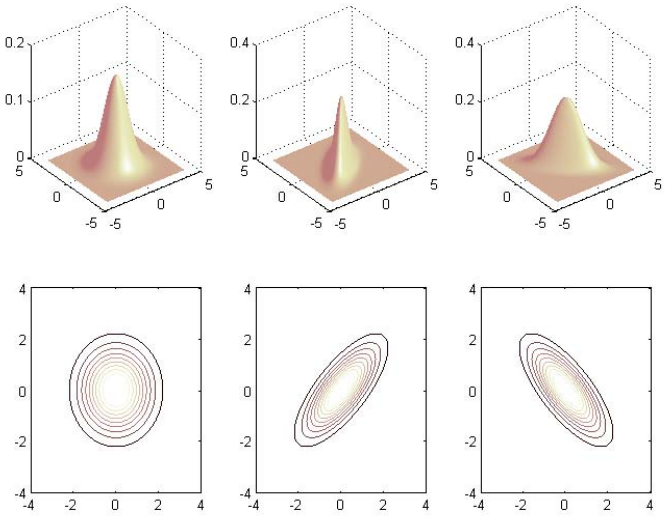
I förra avsnittet diskuterades statistisk modellering. Utgångspunkten brukar vara att man antar att det som skall modelleras är ett utfall av en stokastisk variabel med en viss fördelning. Denna fördelning antas i våra algoritmer alltid vara normalfördelningen.

Låt $\mathbf{x}$ vara en flerdimensionell stokastisk variabel. $\mathbf{x}$ är *normalfördelad* om den har följande simultana täthetsfunktion:

$$f_{\mathbf{x}}(\boldsymbol{\xi}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} \cdot e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})},$$

där d är antalet dimensioner, $\boldsymbol{\mu}$ är väntevärdet och $\boldsymbol{\Sigma}$ är kovariansmatrisen.

Graferna i Figur 26 visar några exempel på tvådimensionella normalfördelningar. Väldigt kortfattat kan man säga att väntevärdet anger var sannolikhetsmassan är belägen, medan kovariansmatrisen ger fördelningens form.



Figur 26. Bilden visar normalfördelningens simultana täthetsfunktion för några olika kovariansmatriser. Från vänster: ingen, positiv respektive negativ kovarians mellan x1 och x2. Väntevärdet är (0,0) i alla tre fördelningarna. Under graferna visas några nivåkurvor.

Kovariansmatrisen

Kovariansmatrisen kommer att dyka upp i många av algoritmerna nedan. Nyckeln till förståelse av principalkomponentanalys ligger i diagonaliseringen av kovariansmatrisen som vi går igenom nedan, varför detta avsnitt är särskilt viktigt.

Kovariansmatrisen $\boldsymbol{\Sigma}$ för den stokastiska variabeln $\mathbf{x}$ definieras som:

$$\boldsymbol{\Sigma} \stackrel{def}{=} \mathrm{E}\left[\left(\mathbf{x} - \mathrm{E}[\mathbf{x}] \right) \left(\mathbf{x} - \mathrm{E}[\mathbf{x}] \right)^{\mathrm{T}} \right]$$

Väntevärdet $\mathrm{E}[\cdot]$ är linjärt i sitt argument. Man kan utnyttja detta för att skriva $\boldsymbol{\Sigma}$ på formen

$$\begin{aligned}
\boldsymbol{\Sigma} &= \mathrm{E}\left[\left(\mathbf{x} - \mathrm{E}[\mathbf{x}] \right) \left(\mathbf{x}^{\mathrm{T}} - \mathrm{E}[\mathbf{x}]^{\mathrm{T}} \right) \right] = \\
&= \mathrm{E}\left[\mathbf{x}\mathbf{x}^{\mathrm{T}} - \mathbf{x}\mathrm{E}[\mathbf{x}]^{\mathrm{T}} - \mathrm{E}[\mathbf{x}]\mathbf{x}^{\mathrm{T}} + \mathrm{E}[\mathbf{x}]\mathrm{E}[\mathbf{x}]^{\mathrm{T}} \right] = \\
&= \mathrm{E}\left[\mathbf{x}\mathbf{x}^{\mathrm{T}} \right] - \mathrm{E}[\mathbf{x}]\mathrm{E}[\mathbf{x}]^{\mathrm{T}} - \mathrm{E}[\mathbf{x}]\mathrm{E}[\mathbf{x}]^{\mathrm{T}} + \mathrm{E}[\mathbf{x}]\mathrm{E}[\mathbf{x}]^{\mathrm{T}} = \\
&= \mathrm{E}\left[\mathbf{x}\mathbf{x}^{\mathrm{T}} \right] - \mathrm{E}[\mathbf{x}]\mathrm{E}[\mathbf{x}]^{\mathrm{T}}
\end{aligned}$$

På diagonalen innehåller kovariansmatrisen *spridningsmåttet* varians för varje komponent i **x**. Resten av matrisen består av *beroendemåttet* kovarians mellan varje par av komponenter i **x**.

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_{x_1 x_1} & \sigma_{x_1 x_2} & \cdots \\ \sigma_{x_1 x_2} & \sigma_{x_2 x_2} & \\ \vdots & & \ddots \end{pmatrix}$$

Observera att $\mathbf{\Sigma}$ alltid är en symmetrisk matris eftersom $\sigma_{x_i x_j} = \sigma_{x_j x_i}$.

Vad händer om man byter koordinatsystem med någon basbytesmatris V? Basbytesmatrisen definieras av

$$\mathbf{x} = \mathbf{V}\mathbf{x}',$$

där **x** är vektorns koordinater i det ursprungliga koordinatsystemet, och $\mathbf{x}'$ är koordinaterna i det nya.

V konstrueras genom att låta de nya basvektorerna $\mathbf{e}_i'$, uttryckt i det gamla koordinatsystemet, vara kolonner i V.

$$\mathbf{V} = \begin{pmatrix} \mathbf{e}_1' & \mathbf{e}_2' & \cdots & \mathbf{e}_\mathbf{n}' \end{pmatrix}$$

Genom att utnyttja att väntevärdet är linjärt i sitt argument får vi

$$\begin{aligned} \mathbf{\Sigma} &= \mathrm{E}\left[\mathbf{x}\mathbf{x}^{\mathrm{T}}\right] - \mathrm{E}[\mathbf{x}]\mathrm{E}[\mathbf{x}]^{\mathrm{T}} = \\ &= \mathrm{E}\left[\mathbf{V}\mathbf{x}'(\mathbf{V}\mathbf{x}')^{\mathrm{T}}\right] - \mathrm{E}[\mathbf{V}\mathbf{x}']\mathrm{E}[\mathbf{V}\mathbf{x}']^{\mathrm{T}} = \\ &= \mathrm{E}\left[\mathbf{V}\mathbf{x}'\mathbf{x}'^{\mathrm{T}}\mathbf{V}^{\mathrm{T}}\right] - \mathbf{V}\mathrm{E}[\mathbf{x}']\left(\mathbf{V}\mathrm{E}[\mathbf{x}']\right)^{\mathrm{T}} = \\ &= \mathbf{V}\mathrm{E}\left[\mathbf{x}'\mathbf{x}'^{\mathrm{T}}\right]\mathbf{V}^{\mathrm{T}} - \mathbf{V}\mathrm{E}[\mathbf{x}']\mathrm{E}[\mathbf{x}']^{\mathrm{T}}\,\mathbf{V}^{\mathrm{T}} = \\ &= \mathbf{V}\left(\mathrm{E}\left[\mathbf{x}'\mathbf{x}'^{\mathrm{T}}\right] - \mathrm{E}[\mathbf{x}']\mathrm{E}[\mathbf{x}']^{\mathrm{T}}\right)\mathbf{V}^{\mathrm{T}} = \\ &= \mathbf{V}\mathbf{\Sigma}'\mathbf{V}^{\mathrm{T}} \end{aligned}$$

Låt oss nu för enkelhetens skull anta att den nya basen är ortonormerad så att V är en ortogonal matris. Då gäller att $\mathbf{V}^{-1} = \mathbf{V}^{\mathrm{T}}$, och vi kan enkelt lösa ut $\mathbf{\Sigma}'$:

$$\mathbf{\Sigma}' = \mathbf{V}^{\mathrm{T}}\mathbf{\Sigma}\mathbf{V}$$

Eftersom $\mathbf{\Sigma}$ är en symmetrisk matris kan man enligt en sats från linjär algebra *diagonalisera* den med en ortogonal matris. Detta innebär att det finns en ortogonal basbytesmatris V så att $\mathbf{\Sigma}'$ är diagonal.

Att en kovariansmatris är diagonal innebär att alla elementen utanför huvuddiagonalen, det vill säga kovarianserna, måste vara noll. Det är ganska anmärkningsvärt att man alltid kan åstadkomma detta genom att göra ett basbyte, som dessutom är ortogonalt.

För normalfördelningen är detta resultat särskilt intressant, eftersom två okorrellerade normalfördelade stokastiska variabler är oberoende.

Så med hjälp av ett basbyte kan man se till att komponenterna i $\mathbf{x}$ är oberoende stokastiska variabler.

Hur konstruerar man detta V? Den nya basen skall bestå av alla egenvektorer till $\mathbf{\Sigma}$. Symmetriska matriser har alltid ortogonala egenvektorer, och genom att normera dem kan vi se till att den nya basen är ortonormerad.

Antag nu att vi valt basbytesmatrisen V på detta sätt. Det gäller att den diagonalmatris som erhålles vid diagonalisering består av egenvärdena $(\lambda_1, \lambda_2, \ldots, \lambda_n)$ till den ursprungliga matrisen $\mathbf{\Sigma}$.

$$\mathbf{\Sigma}' = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

Dessa diagonalelement är förstås varianserna i det nya koordinatsystemet:

$$(\sigma_{x_1'x_1'}, \sigma_{x_2'x_2'}, \ldots, \sigma_{x_n'x_n'}) = (\lambda_1, \lambda_2, \ldots, \lambda_n)$$

Det kan vara av intresse för avsnittet om principalkomponentanalys nedan att veta att totala variansen inte påverkas av diagonaliseringen, eftersom

$$\sum_i \sigma_{x_i'x_i'} = \sum_i \lambda_i = \operatorname{tr}(\mathbf{\Sigma}) = \sum_i \sigma_{x_i x_i}$$

Vi har här utnyttjat en sats från matristeorin som säger att summan av egenvärdena för en matris $\mathbf{\Sigma}$ är lika med *spåret* av $\mathbf{\Sigma}$. Spåret tr($\mathbf{\Sigma}$) är definierat som summan av diagonalelementen.

I det här sammanhanget kan vi också nämna att kovariansmatrisen alltid är en *positivt semidefinit matris*. Detta innebär att egenvärdena alla är större än eller lika med noll, vilket ju är viktigt eftersom de har en tolkning som varianser – varianser kan ju inte vara negativa.

Skattning av medelvärde och kovarians

Antag att vi har ett stickprov $\{\mathbf{x}_i\}_{i=1}^n$ från en statistisk fördelning. Låt oss nu påminna om formlerna för två väntevärdesriktiga skattningar av väntevärde och kovariansmatris. De är i nämnd ordning *stickprovets medelvärde* som ges av

$$\boldsymbol{\mu}^* = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i \,,$$

och *stickprovets kovariansmatris* som ges av

$$\mathbf{\Sigma}^* = \frac{1}{n-1}\sum_{i=1}^n (\mathbf{x}_i - \boldsymbol{\mu}^*)(\mathbf{x}_i - \boldsymbol{\mu}^*)^{\mathsf{T}}$$

Läsaren ser säkert ur formeln ovan att $\mathbf{\Sigma}^*$ är symmetrisk, och att den påverkas på samma sätt som $\mathbf{\Sigma}$ av ett basbyte. Alltså kan även $\mathbf{\Sigma}^*$

diagonaliseras genom att välja en ny ortonormerad bas som består av dess egenvektorer. Egenvektorerna till $\mathbf{\Sigma}^*$ kan användas som skattningar av egenvektorerna till $\mathbf{\Sigma}$. Dessutom är $\mathbf{\Sigma}^*$ också positivt semidefinit.

I praktiken vill man förstås inte först iterera över hela stickprovet för att beräkna $\boldsymbol{\mu}^*$ och sedan iterera en gång till för att beräkna $\mathbf{\Sigma}^*$. Då kan följande omformning komma till pass:

$$\mathbf{\Sigma}^* = \frac{1}{n-1}\left(\sum_{i=1}^{n}(\mathbf{x}_i)(\mathbf{x}_i)^{\mathsf{T}} - n\boldsymbol{\mu}^*\boldsymbol{\mu}^{*\mathsf{T}}\right)$$

I fortsättningen kommer vi att utelämna asterisken som beteckning för skattning. I praktiken känner man oftast inte de exakta värdena på $\boldsymbol{\mu}$ och $\mathbf{\Sigma}$, utan har bara en skattning att tillgå. Vi kommer inte heller att hålla isär beteckningarna för en stokastisk variabel $\mathbf{x}$ och ett utfall av densamma. Sammanhanget får utvisa vilket det rör sig om.

Mahalanobis-avståndet

Läsaren är säkert bekant med den euklidiska metriken

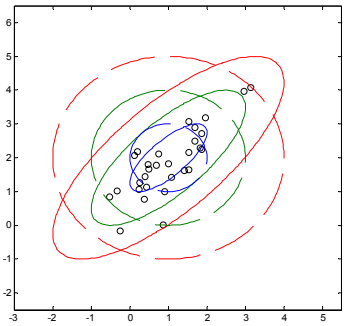
$$d(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2 = \sqrt{(\mathbf{x}-\mathbf{y})^{\mathsf{T}}(\mathbf{x}-\mathbf{y})}$$

för att mäta avståndet mellan två vektorer $\mathbf{x}$ och $\mathbf{y}$.

Antag att $\mathbf{x}$ är ett utfall av en flerdimensionell stokastisk variabel. Från ett stickprov kan vi få reda på något om dess fördelning, till exempel genom att beräkna stickprovets medelvärde $\boldsymbol{\mu}$ och kovariansmatris $\mathbf{\Sigma}$. Vi söker nu ett enkelt mått på hur långt $\mathbf{x}$ befinner sig från *medelvärdet*.

Ett exempel där detta är av betydelse är följande enkla algoritm. Man önskar klassificera en punkt som tillhörande någon av flera olika fördelningar vars medelvärden och kovariansmatriser man har skattat. Detta görs genom att man beräknar avståndet från punkten $\mathbf{x}$ till varje fördelnings medelvärde och väljer den som ger minst avstånd.

Är den euklidiska metriken lämplig för den här typen av avståndsmätning? Nej, om man nu har en skattning av $\mathbf{\Sigma}$ är det bättre att införa ett annat avståndsmått som tar hänsyn till formen på den stokastiska variabelns fördelning.



Figur 27. Illustration av två olika metriker, euklidisk metrik (streckad linje) och mahalanobis-metrik (heldragen).

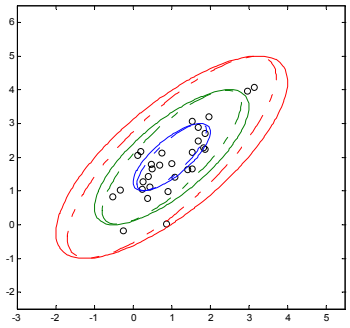
Figur 27 visar en mängd slumptal tagna från en tvådimensionell normalfördelning. Punkterna tenderar att lägga sig i ett ellipsformat kluster, vars form beror på kovariansmatrisen. De streckade cirklarna visar vilka punkter som har avståndet ett, två respektive tre från

medelvärdet mätt med euklidiskt avstånd. De heldragna ellipserna visar ett avståndsmått som är anpassat till klustrets form.

Vi inför den så kallade Mahalanobis-metriken, som bär namn efter den indiske statistikern Prasanta Chandra Mahalanobis.

$$d(\mathbf{x},\boldsymbol{\mu}) = \|\mathbf{x}-\boldsymbol{\mu}\|_{\boldsymbol{\Sigma}} = \sqrt{(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\,\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}\,.$$

I Figur 27 har den sanna kovariansmatrisen använts. Figur 28 visar även resultatet när man använder den skattade kovariansmatrisen istället. Felet i skattningen beror bland annat på stickprovets storlek – ju större stickprov man har, desto bättre blir skattningen. Hur som helst ser man att även den skattade kovariansmatrisen leder till en bättre metrik än den euklidiska.



Figur 28 Jämförelse mellan de två metriker som erhålles när man använder den skattade (punktstreckad) och sanna (heldragen) kovariansmatrisen.

En viktig koppling till normalfördelningen bör påpekas. Uttrycket under rottecknet känner vi igen från formeln för normalfördelningens täthetsfunktion. Detta antyder att Mahalanobis-metriken är särskilt lämplig att använda då man har en sådan fördelning. Vi skall undersöka detta närmare nedan.

Betrakta de punkter som ligger på konstant avstånd från medelvärdet, mätt med Mahalanobis-metrik. Med en ekvivalent formulering ligger de på nivåytor till avståndsfunktionen. Då ligger de alltså även på nivåytor till normalfördelningens simultana täthetsfunktion, eftersom

$$\sqrt{(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\,\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} = \text{const.} \;\Leftrightarrow\; e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{T}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} = \text{const.}$$

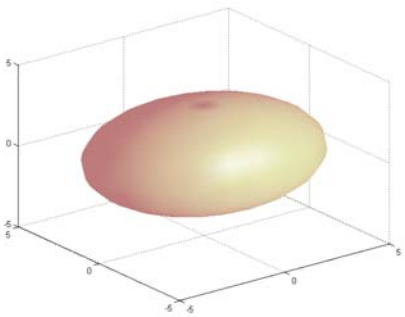
Den intressanta ekvationen i sammanhanget är tydligen

$$(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\,\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) = \text{const.}$$

Denna ekvation definierar en mångfald som är en kurva, yta eller motsvarande beroende på dimensionen på **x**. Låt oss för enkelhetens skull kalla det en yta. Vilken typ av yta rör det sig om? Det är ju en *kvadratisk form* vi har i vänsterledet, och eftersom kovariansmatrisen alltid är positivt semidefinit kan man bara erhålla en enda typ av *kvadratisk yta* ur formeln, nämligen hyperellipsoiden. I två och tre dimensioner motsvarar detta en ellips respektive en ellipsoid.

Om man har ett koordinatsystem med axlar som ligger utmed ellipsoidens axlar, har man diagonaliserat kovariansmatrisen. Det kan inses om man känner till att ekvationen för en ellipsoid vars halvaxlar är parallella med koordinataxlarna är

$$\sum_i \frac{(x_i-\mu_i)^2}{r_i^2} = 1\,.$$



Figur 29. En ellipsoid.

Vi skall visa att längden r_i av ellipsoidens *i*:te halvaxel svarar mot roten ur variansen $\sigma_{x_i x_i}$. Vi går baklänges och ersätter r_i^2 med variansen och skriver uttrycket i vänsterledet på matrisform

$$\sum_i \frac{(x_i - \mu_i)^2}{r_i^2} = \sum_i \frac{(x_i - \mu_i)^2}{\sigma_{x_i x_i}} =$$
$$= (\mathbf{x} - \boldsymbol{\mu})^{\mathsf{T}} \operatorname{diag}\left(\frac{1}{\sigma_{x_i x_i}}\right)(\mathbf{x} - \boldsymbol{\mu}) =$$
$$= (\mathbf{x} - \boldsymbol{\mu})^{\mathsf{T}} \left(\operatorname{diag}\left(\sigma_{x_i x_i}\right)\right)^{-1} (\mathbf{x} - \boldsymbol{\mu}) =$$
$$= (\mathbf{x} - \boldsymbol{\mu})^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})$$

Om $\boldsymbol{\Sigma}$ är diagonal får vi alltså följande avståndsformel:

$$\|\mathbf{x} - \boldsymbol{\mu}\|_{\Sigma} = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})} = \sqrt{\sum_i \frac{(x_i - \mu_i)^2}{\sigma_{x_i x_i}}},$$

där $\sigma_{x_i x_i}$ är variansen för komponenten x_i. Uttrycket i högerledet visar hur Mahalanobis-avståndet skiljer sig från det vanliga euklidiska. Det skalar om varje term för att ta hänsyn till vad vi vet om variansen i den riktningen. När vi skattade variansen $\sigma_{x_i x_i}$ skattade vi ju enligt variansens definition väntevärdet av uttrycket i täljaren,

$$\sigma_{x_i x_i} = \mathsf{E}\left[\left(x_i - \mathsf{E}[x_i]\right)^2\right]$$

Vad gäller om man inte använder ett till ellipsoiden anpassat koordinatsystem?

Eftersom $\boldsymbol{\Sigma}$ är symmetrisk kan den diagonaliseras med en ortogonal basbytesmatris **V**. **V** uppfyller alltså i egenskap av basbytesmatris $\mathbf{x} = \mathbf{V}\mathbf{x}'$, där $\mathbf{x}'$ är koordinaterna i den nya basen. Det gäller också att $\mathbf{V}^{-1} = \mathbf{V}^{\mathsf{T}}$ eftersom den är ortogonal. Vi får följande faktorisering av kovariansmatrisen:

$$\boldsymbol{\Sigma} = \mathbf{V}^{\mathsf{T}} \operatorname{diag}\left(\sigma_{x_i' x_i'}\right) \mathbf{V}.$$

Inversion ger

$$\boldsymbol{\Sigma}^{-1} = \mathbf{V}^{\mathsf{T}} \operatorname{diag}\left(\frac{1}{\sigma_{x_i' x_i'}}\right) \mathbf{V},$$

och vid insättning i uttrycket under rottecknet i Mahalanobis-avståndet får vi

$$(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\,\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})=$$

$$=(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\,\mathbf{V}^{\mathrm{T}}\,\mathrm{diag}\left(\frac{1}{\sigma_{x_i'x_i'}}\right)\mathbf{V}(\mathbf{x}-\boldsymbol{\mu})=$$

$$=\left(\mathbf{V}(\mathbf{x}-\boldsymbol{\mu})\right)^{\mathrm{T}}\,\mathrm{diag}\left(\frac{1}{\sigma_{x_i'x_i'}}\right)\mathbf{V}(\mathbf{x}-\boldsymbol{\mu})=$$

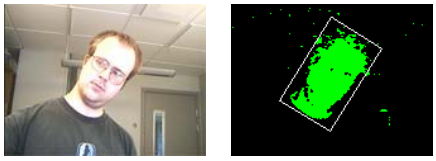
$$=(\mathbf{x}'-\boldsymbol{\mu}')^{\mathrm{T}}\,\mathrm{diag}\left(\frac{1}{\sigma_{x_i'x_i'}}\right)(\mathbf{x}'-\boldsymbol{\mu}')=$$

$$=\sum_i\frac{(x_i'-\mu_i')^2}{\sigma_{x_i'x_i'}}$$

Det visar sig alltså att normeringen av termerna i avståndsformeln sker i det diagonaliserande koordinatsystemet, d v s det ortonormerade koordinatsystem vars axlar är parallella med ellipsoidens axlar.

Approximation av ett 2D-klusters form

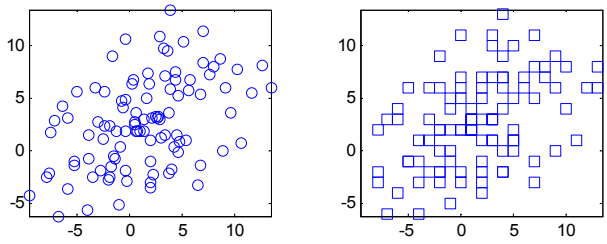
Om man har ett kluster av punkter i xy- eller uv-planet, kan det vara intressant att beräkna klustrets läge, orientering och storlek. En anledning kan vara att man önskar en mer kompakt representation av klustret än vad en lista av de tillhörande punkterna utgör. Om man vet att pixlarna i klustret föreställer ett ansikte sett framifrån, kan det följande även användas för att approximera ansiktets lutning i sidled. Denna lutning är i och för sig knappast en intressant styrsignal.



Figur 30. Rektangeln till höger visar en approximation av klustrets storlek och orientering. Till vänster syns originalbilden.

Vi antar här att pixlarna i klustret är ett stickprov från en normalfördelning. I praktiken är så förstås inte fallet, utan man får betrakta antagandet som en approximation. Om klustret är av någorlunda rektangulär eller elliptisk form, kommer den approximationen att duga bra.

Ett annat tillkortakommande i modellen är att ett stickprov från en normalfördelning ger flyttalskoordinater. Enligt modellen kan då ett kluster innehålla överlappande pixlar. Hos pixlarna i ett kluster kan varje koordinat bara finnas med högst en gång. Modellen är nämligen kontinuerlig, och den verkliga bilden är diskret.



Figur 31. Till vänster syns ett kluster som slumpats fram från modellen vi valt i detta avsnitt. Till höger visas ett exempel på hur ett kluster skulle kunna se ut i en riktig bild. Skillnaden mellan modellen och verkligheten är att modellen tillåter överlappande koordinater. Modellen är nämligen kontinuerlig, och verkligheten är diskret i det här fallet.

Detta avsnitt är en enkel tillämpning av de egenskaper hos kovariansmatrisen som redovisats ovan. Det som följer utgör en utmärkt förberedelse till avsnittet om principalkomponentanalys nedan. Vi skall nämligen diagonalisera en 2×2 kovariansmatris för hand.

Beräkna först en skattning av kovariansmatrisen för pixlarna i klustret. Vi använder i detta avsnitt följande beteckningar för elementen i matrisen:

$$\mathbf{\Sigma} = \begin{pmatrix} a & b \\ b & d \end{pmatrix}$$

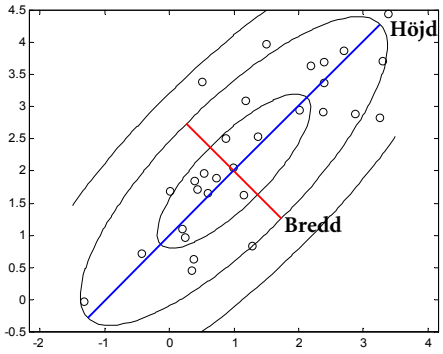
Antag att kovariansen b är skild från noll. Annars är ju $\mathbf{\Sigma}$ redan diagonal!

Det karakteristiska polynomet för matrisen ges av

$$p(\lambda) = |\mathbf{\Sigma} - \lambda\mathbf{I}| = \begin{vmatrix} a-\lambda & b \\ b & d-\lambda \end{vmatrix} = (a-\lambda)(d-\lambda) - b^2$$

Den karakteristiska ekvationen $p(\lambda) = 0$ har de två lösningarna

$$\lambda_1 = \frac{1}{2}\left(a + d - \sqrt{(a-d)^2 + 4b^2}\right)$$
$$\lambda_2 = \frac{1}{2}\left(a + d + \sqrt{(a-d)^2 + 4b^2}\right)$$



Figur 32. De två linjerna visar bredd respektive höjd. Cirklarna är slumptal tagna från en tvådimensionell normal-fördelning. I formeln för bredd och höjd har n=2 använts. Ellipserna är nivåkurvor för mahalanobismetrikens avstånds-funktion. Den skattade kovariansmatrisen har använts som utgångspunkt för alla beräkningar.

Dessa lösningar är matrisens egenvärden, och eftersom de har en tolkning som varianser kallar vi dem för egenvarianser. De är alltså de nya varianserna i det diagonaliserande koordinatsystemet **V**.

Vi söker nu klustrets storlek. Bredd och höjd är ju inga enkelt uppmätta storheter för ett normalfördelat pixelkluster, inte som för en rektangel. Men man kan t ex definiera bredden som två gånger n standardavvikelser:

$$w = 2 \cdot n\sigma_{x_1'} = 2 \cdot n\sqrt{\sigma_{x_1'x_1'}} = 2n\sqrt{\lambda_1}$$
$$h = 2n\sqrt{\lambda_2}$$

Som bekant innehåller 3 standardavvikelser en viss andel av sannolikhetsmassan. För en endimensionell normalfördelning med medelvärde m och standardavvikelse σ är denna andel t ex $\mathrm{P}(-3\sigma < X - m < 3\sigma) = 0{,}999$. Därför kan man tänka sig att valet av just n=3 ger en generös uppskattning av bredd och höjd, medan n=2 kanske kan stämma bättre överens med vår intuitiva uppfattning om klustrets bredd och höjd.

λ_1 kommer att vara det minsta och λ_2 det största egenvärdet.
Som synes är summan av egenvarianserna samma som summan av de ursprungliga varianserna:

$$\lambda_1 + \lambda_2 = a + d$$

och totala variansen är bevarad.

Den uppmärksamme läsaren invänder kanske att uttrycket för λ_1 kan bli negativt även om varianserna *a* och *d* är positiva. För att en matris skall kunna vara en kovariansmatris räcker det inte att den är symmetrisk – den måste som tidigare påpekats även vara *positivt semidefinit*, d v s alla matrisens egenvärden måste vara icke-negativa. Med hjälp av uttrycken för egenvärdena kan vi härleda villkor på *a*, *b* och *d* för att en symmetrisk 2×2-matris skall vara positivt semidefinit:

$$\begin{cases} a \geq 0, d \geq 0 \\ ad \geq b^2 \end{cases}$$

Den diagonaliserande basen **V** erhålles genom att beräkna egenvektorerna, d v s lösningarna till

$$\boldsymbol{\Sigma}\mathbf{x} = \lambda\mathbf{x}$$

Det är så enkelt som att normera egenvektorerna och sätta in dem som kolonner i **V**:

$$\mathbf{V} = \begin{pmatrix} \frac{\mathbf{x}_1}{|\mathbf{x}_1|} & \frac{\mathbf{x}_2}{|\mathbf{x}_2|} \end{pmatrix}.$$

Vi får systemet

$$(\boldsymbol{\Sigma} - \lambda\mathbf{I})\mathbf{x} = 0 \quad \Leftrightarrow \quad \begin{pmatrix} a-\lambda & b \\ b & d-\lambda \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0\,.$$

Eftersom λ är egenvärden har ekvationssystemet icke-trivial lösning – de två raderna i systemet kommer att urarta till identiska ekvationer. För den tveksamme visar vi detta på ett enkelt sätt genom att utnyttja den karakteristiska ekvationen:

$$(a-\lambda)(d-\lambda) - b^2 = 0$$
$$\Leftrightarrow$$
$$(d-\lambda) = \frac{b^2}{(a-\lambda)}$$

Insättning följt av multiplikation med $(a-\lambda)/b$ i den undre ekvationen ger

$$\begin{pmatrix} a-\lambda & b \\ b & \frac{b^2}{a-\lambda} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0 \quad \Leftrightarrow \quad \begin{pmatrix} a-\lambda & b \\ a-\lambda & b \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0$$

Egenvektorerna blir alltså

$$\mathbf{x}_1 = (b, \lambda_1 - a)$$
$$\mathbf{x}_2 = (b, \lambda_2 - a)$$

Enligt teorin är egenvektorerna ortogonala eftersom $\boldsymbol{\Sigma}$ var symmetrisk. För att enklare se att de faktiskt är ortogonala kan vi utnyttja relationen

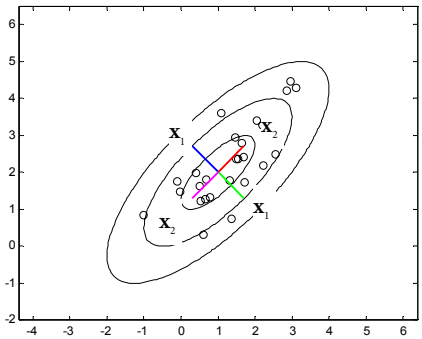
$$\lambda_1 + \lambda_2 = a + d$$

för att uttrycka båda egenvektorerna i det minsta egenvärdet.

$$\mathbf{x}_1 = (b, \lambda_1 - a)$$
$$\mathbf{x}_2 = (b, d - \lambda_1)$$

Nu ser vi enkelt att de är ortogonala:

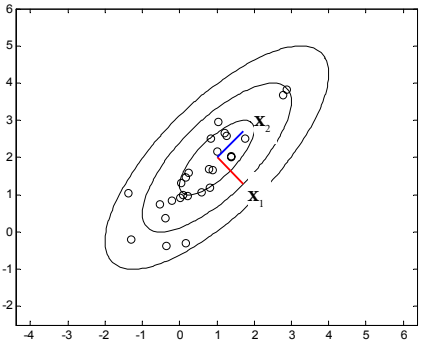
$$\mathbf{x}_1 \cdot \mathbf{x}_2 = b^2 - (a - \lambda_1)(d - \lambda_1) = -p(\lambda_1) = 0$$



Figur 33. De fyra alternativen att välja ut två basvektorer från.

Än så länge vet vi bara att $\mathbf{x}_1$ svarar mot det minsta egenvärdet och $\mathbf{x}_2$ mot det största. Om man bildar en bas av de två vektorerna skulle den basen kunna se ut på olika sätt. Efter normering skulle $\mathbf{x}_1$ kunna vara vilken som helst av bredd-vektorerna i figur 33, och $\mathbf{x}_2$ skulle kunna vara vilken som helst av höjd-vektorerna. Hur skall vi då kunna beräkna rotationen hos segmentet?

Om vi tror att pixlarna i klustret föreställer ett ansikte är det inte så svårt. Då borde vi välja $\mathbf{x}_1$ som vänster-höger-vektor och $\mathbf{x}_2$ som upp-ner-vektor. Detta eftersom vi antar att ansiktet är avlångt. När detta väl är avklarat måste man välja tecken på vektorerna så att de får samma orientering som den ursprungliga basen. I bilderna är det ursprungliga koordinatsystemet positivt orienterat. Vi antar nu att ansiktet aldrig lutar mer än 90° från upprätt position. Då ska vi välja $\mathbf{x}_1$ så att dess x_1-koordinat i det ursprungliga systemet är positiv och $\mathbf{x}_2$ så att dess x_2-koordinat också är positiv. Figur 34 visar valet av vänster-höger-vektor och upp-ner-vektor.



Figur 34. Valet av vänster-höger-vektor $\mathbf{x}_1$ och upp-ner-vektor $\mathbf{x}_2$.

Vi har alltså före normering

$$\mathbf{x}_1 = \operatorname{sgn}(b) \cdot (b, \lambda_1 - a)$$
$$\mathbf{x}_2 = (b, \lambda_2 - a)$$

sgn(b) är signumfunktionen som har värdet 1 om b är positiv, -1 om den är negativ. $\mathbf{x}_2$ behöver inte korrigeras eftersom

$$\lambda_2 - a = \frac{1}{2}\left(d - a + \sqrt{(d-a)^2 + 4b^2}\right) > 0$$

Låt oss nu kalla rotationsvinkeln för θ. Enligt antagandet ovan gäller $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. Om man funderar lite med hjälp av figuren kommer man fram till att

$$\tan\theta = \frac{\lambda_1 - a}{b} = -\frac{b}{\lambda_2 - a}.$$

Man kan använda arctan för att lösa ut θ. Problemet med arctan är att man kan få en annan vinkel än man tänkt sig. Under de förutsättningar som vi angett ovan visar en närmare analys dock att man får rätt vinkel.

Med hjälp av tidigare samband kan *a* bytas mot *d* i ovanstående formler för tan θ om man så önskar. Men uttrycken för λ_i är ganska krångliga – kan vi inte bli av med dem? Man kan få en enklare formel som inte innehåller några rotuttryck genom att utnyttja formeln för tangens av dubbla vinkeln:

$$\tan 2\theta = \frac{2\tan\theta}{1-\tan^2\theta} = \frac{2\dfrac{\lambda_1 - a}{b}}{1-\left(\dfrac{\lambda_1 - a}{b}\right)^2} = \frac{2b}{\dfrac{b^2}{\lambda_1 - a} - (\lambda_1 - a)} =$$

$$= \frac{2b}{(\lambda_1 - d) - (\lambda_1 - a)} = \frac{2b}{a-d}$$

Här har vi utnyttjat den karakteristiska ekvationen i näst sista steget. När man löser ut θ ur den här formeln gäller det dock återigen att se upp. Man kan få en annan vinkel än man tänkt sig. En grundlig analys visar att en utväg är att använda följande uttryck:

$$\theta = \frac{1}{2}\operatorname{atan2}(-2b, d-a)$$

Man kan införa en version av arcustangens som tar två argument och returnerar en vinkel i intervallet $[-\pi, \pi]$. Denna funktion brukar i programmeringssammanhang betecknas atan2, och det är den som används i uttrycket ovan.

Vi sammanfattar ovanstående resultat, och använder nu de rätta beteckningarna i stället för *a*, *b*, *d* etc. Under antagandet att pixlarna i klustret föreställer ett ansikte får vi egenvarianser

$$\sigma_{x_1' x_1'} = \frac{1}{2}\left(\sigma_{x_1 x_1} + \sigma_{x_2 x_2} - \sqrt{\left(\sigma_{x_1 x_1} - \sigma_{x_2 x_2}\right)^2 + 4\sigma_{x_1 x_2}^2}\right)$$

$$\sigma_{x_2' x_2'} = \frac{1}{2}\left(\sigma_{x_1 x_1} + \sigma_{x_2 x_2} + \sqrt{\left(\sigma_{x_1 x_1} - \sigma_{x_2 x_2}\right)^2 + 4\sigma_{x_1 x_2}^2}\right),$$

bredd och höjd

$$w \propto \sqrt{\sigma_{x_1' x_1'}}$$

$$h \propto \sqrt{\sigma_{x_2' x_2'}}$$

samt rotationsvinkel

$$\theta = \frac{1}{2}\operatorname{atan2}\left(-2\sigma_{x_1 x_2}, \sigma_{x_2 x_2} - \sigma_{x_1 x_1}\right).$$

För den som är bekant med principalkomponentanalys kan vi dra slutsatsen att det inte var själva principalkomponenterna som var av intresse ovan. Det var ju egenvarianserna och de diagonaliserande egenvektorerna som gav oss klustrets storlek och rotation.

Likaledes kommer det att visa sig nedan att när vi gör motsvarande diagonalisering av kovariansmatrisen för ett stickprov av ansikten så kommer just egenvektorerna att spela en viktig roll, så viktig att de kallas för egenansikten (eng *eigenfaces*).

K-Means-algoritmen för segmentering

K-Means-algoritmen är mycket enkel, och används för segmentering av data. På förhand måste man veta hur många kategorier (kluster) man förväntar sig finna. Dessutom måste man ha ett startvärde för medelvärdet av punkterna i varje kluster. Algoritmen har två steg som itereras till konvergens:

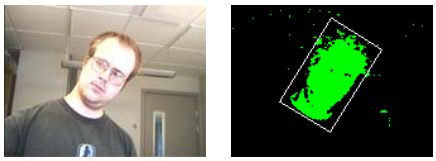
1. Tilldela varje datapunkt till det kluster vars medelvärde den ligger närmast.
2. Uppdatera medelvärdena för varje kluster.

Det finns många sätt att variera K-Means, t ex genom valet av metrik. Euklidisk metrik är kanske det vanligaste valet. Manhattan-avståndet

$$d(\mathbf{x},\boldsymbol{\mu}) = \sum_i |x_i - \mu_i|$$

skulle också kunna fungera bra. I vissa fall kan Mahalanobis-avstånd vara tänkbart.

I vårt arbete har K-Means-algoritmen använts både för segmentering av färginformationen i UV-planet (se avsnittet Färgsegmentering) och segmentering av områden i xy-planet. I det senare fallet visade det sig vara viktigt att veta exakt hur många kluster man förväntar sig finna i bilden, och att de är väl separerade. I praktiken var dessa villkor nästan aldrig uppfyllda, så algoritmen visade sig inte användbar för den typen av segmentering.



Figur 35. Det gröna området har maskats fram med hjälp av färgsegmentering. En algoritm har beräknat vilka färger som finns i bilden, och en mänsklig observatör har fått ange vilken som var hudfärg.

Färgsegmentering

Varför är färgsegmentering relevant för ansiktsdetektion? Om man kunde segmentera bilden efter färg, skulle man t ex kunna lista ut vilket av områdena som är hudfärg genom att undersöka hur väl den liknar ett ansikte. Sedan kan man använda det områdets läge som styrsignal.

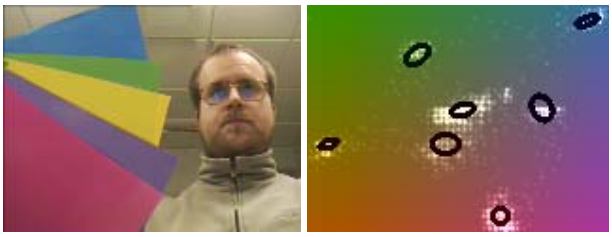
Figur 36. Bilderna visar tre snitt ur YUV-kuben.

Färginformationen i en pixel består av en vektor i något koordinatsystem som beskriver pixelns färg. Till exempel kan man använda RGB, HSV eller YUV-färgmodellen. Fördelen med HSV och YUV är att de separerar färg från ljusintensitet, så att man får en tvådimensionell färgvektor.

I vårt fall användes YUV-färgmodellen. I Figur 36 visas tre snitt ur YUV-färgkuben. I kuben går U- och V-värdena från -128 till 127, och Y varierar från 0 till 255. I den vänstra bilden är Y = 0, i mitten Y = 128 och till höger Y = 255. Som synes finns all färginformation i U och V, Y handlar bara om ljusintensitet. Koordinatsystemet i UV-

planet är valt så att enhetsvektorn i U-led pekar åt höger, och enhetsvektorn i V-led pekar neråt

För att visualisera färginformationen i en bild kan man till exempel göra ett histogram över pixlarnas UV-värden.



Figur 37. En bild och histogrammet över bildens UV-värden. De svarta ellipserna visar några färgkluster som segmenterats fram med hjälp av K-Means-algoritmen.

I Figur 37 visas en bild och dess histogram. Den högra bilden är histogrammet. Den visar en rektangel i UV-planet som omfattar de färger bilden innehåller. Återigen är koordinatsystemet i UV-planet är valt så att enhetsvektorn i U-led pekar åt höger, och enhetsvektorn i V-led pekar neråt. I bilden ovan till höger är varje pixels U- och V-värden valda så att de visar vilken färg den punkten motsvarar, och Y-värdet är en funktion av antalet pixlar som har precis den färgen.

För att reducera färginformationen i en bild till en mer lätthanterlig beskrivning, skulle man kunna anta att det finns ett litet antal distinkta färger i bilden, men att pixlarna som tillhör respektive färg har en viss slumpmässig variation. Då tenderar histogrammet att innehålla ett antal kluster. Givet en bild, önskar vi anpassa en matematisk modell till färginformationen i bilden, som låter oss klassificera pixlarna efter vilket kluster de tillhör, d v s vilken färg de har. Denna modell, och den s k EM-algoritmen som använts för att anpassa den till en viss bild, beskrivs i nästa avsnitt. Där beskrivs också hur man kan använda modellen för att klassificera pixlarna i bilden.

För att erhålla startvärden till EM-algoritmen görs en försegmentering med K-Means. Ett fixt antal kluster M antas förekomma i bilden. För att erhålla startvärden slumpas ett antal medelvärden ut genom att välja M rektangelfördelade slumptal ur mängden

$$\left\{(u,v)\in \mathsf{R}^2; u_{\min} < u < u_{\max},\ v_{\min} < v < v_{\max}\right\}.$$

$u_{\min}$, $v_{\min}$, $u_{\max}$ och $v_{\max}$ är de minsta respektive största U,V-värden som förekommer i bilden.

Ett problem med att slumpa ut startvärden är att någon kategori kan vara tom redan från början, om startvärdet råkar hamna i en del av UV-rektangeln som inte förekommer i bilden. I vår implementation genererar vi nya slumpvärden om så är fallet. Ett problem med denna

lösning är att även de nya startvärdena kan hamna på tomma delar av histogrammet. Det kan ta ganska lång tid (flera sekunder i vår implementation) innan man får fram användbara startvärden.

För att beräkna avstånd i K-Means-algoritmen användes euklidisk metrik.

Även Mahalanobis-metrik provades, men detta visade sig fungera mindre bra. Problemet med denna metrik är att om ett kluster växer, så blir enhetsellipsen för Mahalanobis-avståndet större när man uppdaterar kovariansmatrisen för klustret. Det finns alltså en tendens för ett kluster att växa mer och mer, och tränga undan de andra.

Man kan däremot utnyttja Mahalanobis-avståndet om man vet att det bara finns väl separerade kluster, och man har bra startvärden.

Efter att ha itererat K-Means erhålls startvärden för klustrens medelvärden som en följd av algoritmen. EM-algoritmen behöver även startvärden på kovariansmatriserna, och dessa kan nu beräknas för pixlarna i varje kluster.

Det har nämnts ovan att kovariansmatrisen beskriver formen på ett kluster. När K-Means konvergerat tenderar klustren att vara mer cirkulära än de är när EM-algoritmen konvergerat, eftersom euklidisk metrik använts i K-Means.

Nackdelen med K-Means är att den inte klarar av överlappande kluster, d v s färgkluster med närliggande medelvärden men olika form. Detta klarar däremot EM-algoritmen.

EM-Algoritmen för blandningar av normalfördelningar

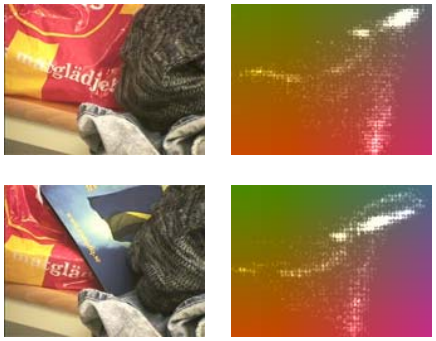
En grov översikt över den statistiska modell vi använt här är att pixlarna i varje färgkluster i histogrammet antas vara ett stickprov av en normalfördelning. Vi bara intresserade av pixlarnas U- och V-värden enligt YUV-färgmodellen, så normalfördelningarna är tvådimensionella. Det gäller att skatta parametrarna för dessa normalfördelningar. När dessa är kända, behövs en metod för att räkna ut vilken fördelning en viss pixel kommer ifrån, så att den kan klassificeras.

Precis som i förra sektionen använder vi M som beteckning på antalet klasser. Med en pixels färg menas i det här avsnittet dess koordinater (u, v) i UV-planet.

Färgen för varje pixel $\mathbf{x}_i = (u, v)$ antas tillhöra någon modellklass $\omega \in \{\omega^1, \omega^2, \ldots, \omega^M\}$. Bilden kan alltså segmenteras genom att för varje pixel skatta vilken modellklass dess färg hör till. Varje modellklass är knuten till en normalfördelning i UV-planet som anger fördelningen av färgen hos pixlarna som tillhör den klassen. Man måste i förväg ange antalet modellklasser *M*.

Efter att ha infört dessa definitioner kan läsaren gärna tänka sig att olika modellklasser helt enkelt är olika färger. Detta bygger på antagandet att olika normalfördelningar kommer att representera vad en mänsklig observatör skulle uppfatta som distinkta färger. Det kan vara lite si och så med den saken, beroende på hur väl EM-algoritmen konvergerar.

EM-algoritmen är en standardalgoritm för att uppskatta okända parametrar i en statistisk modell. I vårt fall använder vi den för att skatta en blandning av normalfördelningar, på engelska *gaussian mixtures*. Genom att anpassa ett antal normalfördelningar till färginnehållet i bilden kan man erhålla medelvärden och kovariansmatriser för de olika modellklassernas fördelningar.



Figur 38. Till vänster syns en bild och till höger dess histogram. Den stora vita klumpen uppe till höger i histogrammet visar att det finns många gråa nyanser i bilden. I det undre paret har en ny färg dykt upp i bilden: blått. Några nya kluster i det blåa området av UV-rektangeln syns nu i histogrammet.

Sedan kan man använda den anpassade modellen för att klassificera pixlarna i en ny bild efter färg. Detta sker förstås under antagande att inga nya dyker upp i bilden. Då skulle man ju behöva börja om från början med EM-algoritmen, eventuellt efter införande av ytterligare en färgkategori. Hur histogrammet förändras när en ny färg dyker upp visas i Figur 38.

EM står för *expectation maximisation*. Ordagrant översatt betyder detta väntevärdesmaximering. Man skall maximera väntevärdet av *likelihooden* för data givet de okända parametrarna med avseende på dessa parametrar. Detta förfarande itereras till konvergens. Algoritmen går alltså ut på att man itererar följande steg:

- E-steget: Beräkna väntevärdet av likelihooden för data givet nuvarande parametrar
- M-steget: Finn de parametrar som maximerar detta väntevärde.

Vi skall nu tillämpa EM-algoritmen på blandningar av normalfördelningar.

Låt $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ vara en bild, där $\mathbf{x}_i = (u_i, v_i)$ är färgen i pixel nummer *i* och *n* är antalet pixlar i bilden. Låt $\boldsymbol{\omega} = (\omega_1, \omega_2, \ldots, \omega_n)$ vara valet av modellklass för varje pixel. Låt $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_M)$ vara de okända relativa frekvenserna för varje modellklass, d v s π_i är andelen pixlar som tillhör modellklassen ω^i. Låt vidare $\boldsymbol{\Theta} = (\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2, \ldots, \boldsymbol{\Theta}_M)$ vara de obekanta parametrarna i fördelningarna för respektive modellklass. I vårt fall använder vi ju normalfördelningar, så de obekanta parametrarna för en viss modellklass ω^i är medelvärdet och

kovariansmatrisen, $\boldsymbol{\Theta}_i = (\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. Låt $\boldsymbol{\Psi} = (\boldsymbol{\pi}, \boldsymbol{\Theta})$ vara beteckningen för alla de okända parametrarna.

Låt nu **y** vara beteckning för **x** kompletterad med modellval för varje pixel. Likelihooden för **y** givet parametrarna för modellklassernas fördelningar betecknar vi $g(\mathbf{y}|\boldsymbol{\Psi})$. Färgerna antas vara oberoende stokastiska variabler så *g* är alltså en produkt. Vi kan lika gärna maximera väntevärdet av log(*g*) givet parametrarna, vilket förvandlar produkten till en summa.

Vi alltså följande steg i EM-algoritmen:
1. Låt $Q(\boldsymbol{\Psi}, \boldsymbol{\Psi}^{(t)}) = \mathsf{E}\left[\log\left(g(\mathbf{y}|\boldsymbol{\Psi})\right) \middle| \mathbf{x}, \boldsymbol{\Psi}^{(t)} \right]$.
2. Sök $\boldsymbol{\Psi} = \boldsymbol{\Psi}^{(t+1)}$ som optimerar $Q(\boldsymbol{\Psi}, \boldsymbol{\Psi}^{(t)})$.

Det visar sig att i fallet med blandningar av normalfördelningar finns det analytisk lösning av steg två, så man behöver bara iterera beräkningen av de optimala parametrarna som en direkt funktion av data.

Man får

$$Q(\boldsymbol{\Psi}, \boldsymbol{\Psi}^{(t)}) = \sum_{i=1}^{n} \sum_{m=1}^{M} w_i^m \left(\boldsymbol{\Psi}^{(t)}\right) \cdot \left(\log \pi_m + \log p\left(x_i \middle| \omega_m, \boldsymbol{\Theta}_m\right)\right),$$

där

$$w_i^m(\boldsymbol{\Psi}) = \mathsf{E}\left[\mathsf{I}\left(\omega_i = \omega^m\right) \middle| \mathbf{x}_i, \boldsymbol{\Psi} \right] = \mathsf{P}\left[\omega_i = \omega^m \middle| \mathbf{x}_i, \boldsymbol{\Psi} \right].$$

$\mathsf{I}(a)$ är indikatorfunktionen, som antar värdet 1 om villkoret *a* är uppfyllt, 0 annars.

Med Bayes formel får vi följande uttryck för *w*:

$$w_i^m(\boldsymbol{\Psi}) = \frac{\pi_m p\left(x_i \middle| \omega^m, \boldsymbol{\Theta}_m\right)}{\sum_{k=1}^{M} \pi_k p\left(x_i \middle| \omega^k, \boldsymbol{\Theta}_k\right)}$$

Uttrycken för de maximerande parametervärdena ges nedan. För enkelhetens skull låter vi w_i^m vara beteckning för $w_i^m\left(\boldsymbol{\Psi}^{(t)}\right)$.

$$\pi_m^{(t+1)} = \frac{\sum_{i=1}^{n} w_i^m}{n}$$

$$\boldsymbol{\mu}_m^{(t+1)} = \frac{1}{n\pi_m^{(t+1)}} \sum_{i=1}^{n} w_i^m \mathbf{x}$$

$$\boldsymbol{\Sigma}_m^{(t+1)} = \frac{1}{n\pi_m^{(t+1)}} \sum_{i=1}^{n} w_i^m \left(\mathbf{x} - \boldsymbol{\mu}_m^{(t+1)}\right)\left(\mathbf{x} - \boldsymbol{\mu}_m^{(t+1)}\right)^{\mathsf{T}}$$

Dessa beräkningar itereras till konvergens. EM-algoritmen konvergerar mycket långsamt, men det spelar inte så stor roll på dagens snabba datorer. I vår implementation konvergerar algoritmen på några sekunder.

För en grundligare genomgång av teorin för blandningar av normalfördelningar rekommenderas (Titterington, 1985). En introduktion till EM-algoritmen för just färgsegmentering finns i (Lindgren, 2002).

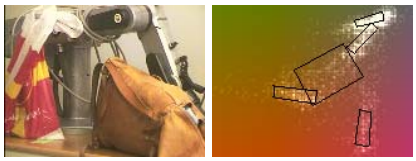
Vi skall nu beskriva hur man kan segmentera en bild efter att EM-algoritmen har konvergerat. Som en biprodukt av algoritmen får man ju värdena på

$$w_i^m(\Psi) = \mathrm{P}\left[\omega_i = \omega^m \,|\, \mathbf{x}_i, \Psi\right]$$

Eftersom dessa anger sannolikheterna att en viss pixel tillhör modellklass nummer *m* (givet pixelns färg och normalfördelningarnas parametrar) kan vi använda dem för att segmentera bilden.

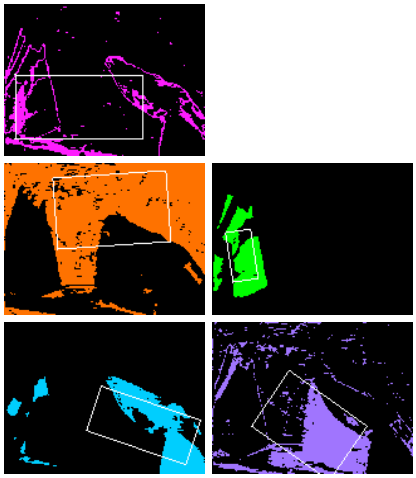
Till exempel kan vi bestämma oss för att en pixel tillhör den modellklass som ger högst värde på w_i^m. Hur detta ser ut visas i Figur 40.

Alternativt kan vi välja ut en viss färg och maska fram de pixlar som har en viss minsta sannolikhet att tillhöra den färgen. Denna minsta sannolikhet kallar vi för sannolikhetströskeln. Hur det kan se ut visas i Figur 41 och Figur 42. Den vita rektangeln i varje bild har beräknats med metoden från avsnittet Approximation av ett 2D-klusters form. Eftersom de områden vi maskat fram inte är särskilt väl samlade stämmer inte antagandena som gjordes i det avsnittet, och rektanglarna blir missvisande. Tanken är att det som maskats fram skall föreställa ett ansikte som i Figur 35. Då överensstämmer rektangeln bättre med klustrets form.

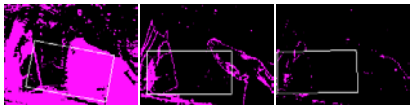


Figur 39. Till vänster syns en bild och till höger dess histogram. Rektanglarna visar vilka normalfördelningar EM-algoritmen har hittat. Som synes kan de överlappa varandra. Antalet modellklasser M *har varit lika med 5.*

Figur 40. Till höger syns en segmentering där varje pixel har tilldelats den kategori som den med störst sannolikhet tillhör.



Figur 41. Bilderna visar hur man kan maska ut olika färger. Sannolikhetströskeln har varit 50%.



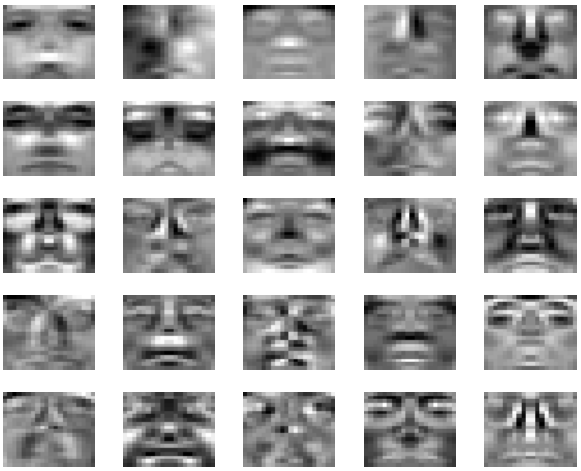
Figur 42. Här visas hur maskningen påverkas när sannolikhetströskeln varieras. Från vänster till höger har den varit 1%, 50% och 99%.

PCA

PCA är en förkortning för principal component analysis, principalkomponentanalys på svenska, och är en metod för datareduktion av mångdimensionella data. Idén är att ta fram ett antal ortogonala vektorer som beskriver så mycket varians inom datan som möjligt, om datan projiceras på dem. Projektion av datan på dessa vektorer kallas principalkomponenter och de kan beräknas som egenvektorerna till en kovariansmatris. För mer information om principalkomponentanalys, se (Johnson, 1992).

Vi gjorde PCA på ett antal bilder av ansikten sedda framifrån. Det här är en ofta använd metod och därför har egenvektorernas riktningar för ansiktsbilder fått ett eget namn, egenansikten (*eigenfaces* på

engelska) eftersom de ofta påminner om ansikten. Figuren nedan visar ett exempel på hur de 25 som svarar mot de största egenvärdena kan se ut. Det största egenvärdet hör till egenansiktet längst upp till vänster, och därefter avtar de radvis.



Figur 43. De 25 egenansikten som tillhör de 25 största egenvärdena, framtagna med träningsbilder från UMISTs ansiktsdatabas. Man kan notera ett ansiktslikt utseende och att alla är symmetriska eller antisymmetriska kring sin vertikala mittlinje. Symmetriegenskaperna förklaras med att alla ansikten i databasen speglats och tagits med i träningsmängden.

När vi beräknade egenansiktena till vår ansiktsföljare använde vi oss av 102 ansiktsbilder av 20 olika personer från UMISTs ansiktsdatabas (Graham, 1998). För att bli av med hår och annat som kan variera väldigt individuellt beskar vi bilderna så att de bara omfattade ögon, näsa och mun. Eftersom ansikten är symmetriska kompletterade vi dessutom träningsmängden med varje ansikte speglat i sin vertikala mittlinje. Det är detta som har lett till att alla egenansiktena har ett symmetriskt eller antisymmetriskt utseende.

Vi har tidigare nämnt att PCA bygger på att man bestämmer egenvektorerna till kovariansmatrisen. Nu skall vi ge en formell definition. Låt **X** vara den mångdimensionella stokastiska variabeln som bilderna betraktas som observationer av. Låt riktningen på den första principalkomponenten vara $\mathbf{w}_1$. Då ges denna vektor av

$$\mathbf{w}_1 = \arg\max_{\|\mathbf{w}_1=1\|} \mathrm{E}\left[\left(\mathbf{w}^{\mathrm{T}}\mathbf{X}\right)^2\right]$$

$\mathbf{w}_1$ är alltså den vektor i vars riktning **X** varierar allra mest. Det visar sig att lösningen på det här optimeringsproblemet är att välja $\mathbf{w}_1$ lika med den egenvektor till kovariansmatrisen för **X** som har det största egenvärdet.

Om de k-1 första principalkomponenterna bestämts så bestäms den k:te som principalkomponenten på residualen

$$\mathbf{w}_k = \arg\max_{\|\mathbf{w}_k=1\|} \mathrm{E}\left[\left(\mathbf{w}^{\mathrm{T}}\left(\mathbf{X} - \sum_{i=1}^{k-1}\mathbf{w}_i\mathbf{w}_i^{\mathrm{T}}\mathbf{X}\right)\right)^2\right]$$

Det visar sig att $\mathbf{w}_k$ skall väljas lika med den egenvektor till kovariansmatrisen för **X** som har det *k*:te största egenvärdet.

För att göra PCA krävs det att man har tillgång till ett antal träningsbilder, i vårt fall ansiktsbilderna. För dessa bilder skattas en kovariansmatris, denna brukar betecknas $\mathbf{\Sigma}$. Kovariansmatrisen beskriver hur varje pixel i de olika bilderna tenderar att samvariera linjärt med varje annan pixel. Om kovariansen t ex har ett högt positivt värde mellan två pixlar innebär det att om intensiteten i den ena ökar så brukar intensiteten i den andra också öka. Ett negativt värde betyder att om den ena ökar så brukar den andra minska. En pixels kovarians med sig själv är dess varians.

För att kunna skatta $\mathbf{\Sigma}$ behövs en medelvärdesbild, en sådan brukar skrivas $\boldsymbol{\mu}$ och skattas så här:

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{k=1}^{N}\mathbf{x_k}$$

Här betecknar $\mathbf{x}_k$ en bild och N är antalet bilder. Den naturligaste representationen av en bild är en m×n matris, men när kovariansmatrisen skattas måste de olika bilderna vara i form av kolonnvektorer. Detta är inget problem utan åstadkommes genom att kolonnstapla bildmatriserna, d v s lägga den andra kolonnen under den första osv. För att illustrera låt $\mathbf{A}$ vara en matris

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

då är dess kolonnstapling

$$\begin{pmatrix} a \\ c \\ b \\ d \end{pmatrix}$$

På så sätt omformas bilderna till mn×1 kolonnmatriser (och samma görs med $\boldsymbol{\mu}$). Sedan skattas kovariansmatrisen enligt

$$\mathbf{\Sigma} = \frac{1}{N-1}\sum_{k=1}^{N}\left(\mathbf{x_k} - \boldsymbol{\mu}\right)\left(\mathbf{x_k} - \boldsymbol{\mu}\right)^T$$

Resultatet av detta är en mn×mn matris. Om man vill kan man bortse från faktorn framför summan. Den förändrar varken matrisens egenvärden eller egenvektorer.

De vektorer som beskriver den största variansen hos datan är de egenvektorer till $\mathbf{\Sigma}$ som svarar mot de största egenvärdena. Alla egenvektorerna tillsammans bildar en ortogonal bas för bildrummet, men om man väljer ut enbart ett antal av dem med de största egenvärdena så erhålles ett linjärt underrum som beskriver bilder som liknar dem som ingår i träningsdatan.

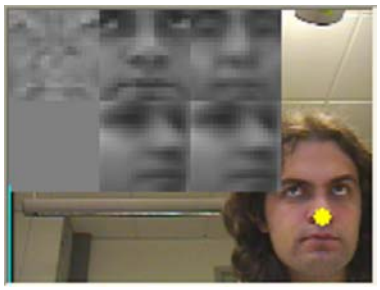
Ifall egenvärdena är sorterade med det största först, (d_1, d_2, d_3, ...), så anger

$$\frac{\sum_{i=1}^{k} d_i}{\sum_{i=1}^{mn} d_i}$$

hur stor andel de k starkaste egenvektorerna beskriver av den totala variationen. Man bör välja så många så att åtminstone 90% av den totala variationen beskrivs.

Att beräkna dessa egenvärden och egenvektorer är ett lämpligt arbete för MATLAB och görs också enkelt däri.

PCA-baserad face finding



Figur 44. I den övre raden ansiktsbilder visas **Y**-$\mathbf{Y}_f$, **Y**+**M** *och* $\mathbf{Y}_f$+**M**. *Den undre raden visar en egenansiktsbas byggd på semiprofilbilder, som för tillfället inte användes i programmet. Pricken visar centrum för den rektangel i bilden som använts för att generera* **Y**.

Figur 45. Här visas från vänster till höger **Y**-$\mathbf{Y}_f$, **Y**+**M** *och* $\mathbf{Y}_f$+**M**. *Som synes föreställer området egentligen inte alls ett ansikte, och skillnadsbilden längst till vänster blir ganska stor.*

HUR AVGÖR MAN VAD SOM LIKNAR ETT ANSIKTE?

Det underrum som spänns upp av de utvalda egenansiktena, dvs de vektorer som kan beskrivas med en linjärkombination av dem, utgör nu mängden av alla bilder som i någon mån liknar träningsansiktena. Avståndet mellan en bild och detta underrum är alltså ett mått på hur lik den är träningsansiktena. Om det var tillräckligt många och olika träningsansikten säger detta mått förhoppningsvis också hur lik bilden är ett ansikte i allmänhet.

De här beräkningarna görs på följande vis. Låt de starkaste egenvektorerna (de med störst egenvärden, i vårt fall var de omkring 20 st) utgöra kolonnnerna i matrisen **W**, så att $\mathbf{W} = (\mathbf{w}_1\ \mathbf{w}_2\ \ldots\ \mathbf{w}_n)$ där $\mathbf{w}_i$ är egenvektorerna. Vidare är **X** bilden vi vill analysera och **M** är medelansiktesbilden. Beräkna skillnaden mellan bilden och medelansiktet.

$$\mathbf{Y} = \mathbf{X} - \mathbf{M}$$

Beräkna längderna av denna vektors projektioner på egenvektorerna. Detta är också dess koordinater i ansiktsrymden med egenansiktena som bas. (I detta koordinatsystem ligger medelansiktet i origo).

$$\mathbf{W}_{\text{test}} = \mathbf{W}^{\mathsf{T}}\mathbf{Y}$$

Beräkna koordinaterna för projektionen av **Y** i hela bildrummet.

$$\mathbf{Y}_f = \mathbf{W}\mathbf{W}_{\text{test}}$$

Avståndet till ansiktsrymden är nu

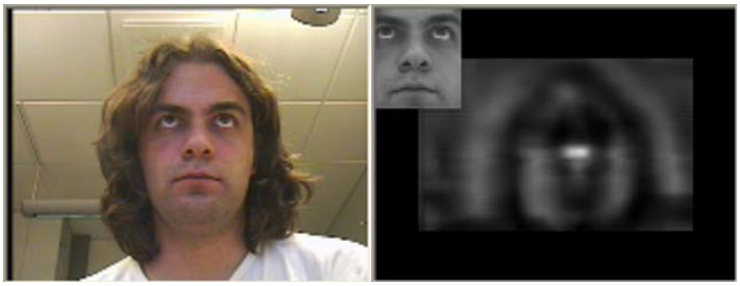
$$\left\| \mathbf{Y} - \mathbf{Y}_f \right\|$$

Denna term kommer härmed att benämnas dffs, distance from face space. Här kan man välja mellan olika metriker, men föreslagsvis används euklidisk metrik.

Efter att ha experimenterat en del insåg vi att avståndet av bildens projektion på underrummet till medelansiktet också var viktigt att ta hänsyn till. Ifall en bild skulle ligga nära ansiktsrummet, men om t ex en av projektionerna på PCA-baserna har ett avstånd till medelansiktet som är mycket större än vad variationen var för träningsansiktenas projektioner så är det troligt att bilden trots allt inte föreställer ett ansikte.

Medan dffs är ett avstånd till ansiktsrummet så är avståndet från $\mathbf{W}_{test}$ till origo i ansiktsrymden (dvs medelansiktet) ett avstånd i ansiktsrymden, ett distance in face space, och förkortas härmed difs. I och med att egenvärdena anger varianserna längs projektionsriktningarna och att origo är medelvärde så är det enkelt att ta fram en mahalanobis-difs.

Med dffs och difs konstruerades ett nytt mått på hur ansiktslik en bild är. För att kunna göra en mahalanobis-dffs berräknades medelvärde och varians för träningsansiktenas dffs. Summan av dessa två mahalanobisavstånd kallade vi för det kombinerade avståndet. Här var dock ett problem. Alla träningsansiktenas dffs var av en viss storleksordning, större än noll. Detta innebär att avståndens medelvärde blev större än noll och därmed kommer en mahalanobis-dffs att tala om avståndet till detta medelvärde. Om en bild faktiskt skulle ha ett dffs nära noll så skulle dess mahalanobis-dffs bli större än den för en bild som låg på samma dffs som träningsansiktena. Med andra ord så skulle bilder med litet dffs bestraffas.



Figur 46. För att generera bilden till höger har vi låtit en rektangel av konstant storlek svepa över bilden och för varje läge beräknat det kombinerade avståndet från rektangelns innehåll till medelansiktet. Den infällda ansiktsbilden visar vilken position som gav den ljusa fläcken i mitten. Ju ljusare punkt, desto mindre kombinerat avstånd.

För att hitta ett ansikte i en bild m h a vårt kombinerade avstånd låter man ett sökfönster svepa över hela bilden och räknar ut det kombinerade avståndet för varje position. Dessutom måste alla olika storlekar på sökfönstret provas. Om det kombinerade avståndet är tillräckligt litet någonstans kan man anta att det finns ett ansikte på den punkten med dimensioner givna av sökfönstrets storlek.

Att hitta ett ansikte på det här viset är alltså tidskrävande och kan inte göras i realtid. Om sökfönstrets storlek fixeras, kan en plottning av

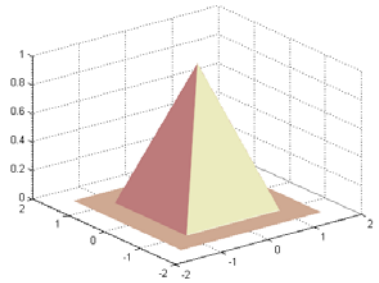
hur mycket områdena kring varje punkt i en bild liknar ett ansikte med sökfönstrets storlek göras. Bilden som då uppkommer brukar kallas för ett moln, eng *cloud*.

Omskalning av bilder

Säg att vi har en rektangel i en bild, given av parametrarna bredd w, höjd h, mittpunkt (x, y) och rotationsvinkel θ. Egenansiktena är genererade utifrån träningsbilder med en viss bredd w_0 och höjd h_0. Vi vill kunna jämföra innehållet i rektangeln med träningsbilderna genom att beräkna något av de avstånd som diskuterats ovan.

Det allra första vi måste göra är att skala och rotera pixlarna i rektangeln till en buffer som har dimension $w_0 \times h_0$. Ett enkelt sätt att göra detta är att införa ett koordinatsystem i rektangeln där enhetsvektorn i x-led är rektangelns ena kortsida och den i y-led är ena långsidan. Vi antar då att innehållet i rektangeln föreställer ett ansikte, och att det inte är roterat mer än 90° från upprätt position.

Sedan faltar man innehållet i rektangeln med en kärna av lämplig storlek och lägger resultatet i buffern. En pyramidformad kärna som i figur 47 ger bra resultat, man får då god utjämning. Om man använder en sådan kärna kallas förfarandet bilinjär medelvärdesbildning, eng *bilinear averaging*.



Figur 47. Pyramidens ekvation är $f(x,y) = \begin{cases} \min(1-|x|, 1-|y|) & \max(|x|,|y|) < 1 \\ 0 & f.ö. \end{cases}$

PCA-baserad ansiktsföljning

Hur kan man använda PCA-basen för att följa ett ansikte i bilden? Antag att vi i en bild har hittat ett ansikte, och att vi har en rektangel som innehåller ansiktet. Vi kan utnyttja denna rektangel för att hitta ansiktets nya läge i nästa bild i videoströmmen. Förutsättningen är att det nya läget inte är alltför långt från det förra.

Ett enkelt sätt att minimera en storhet som beror på många parametrar är att använda en steepest descent algoritm. Den är girig och finner bara ett lokalt minimum i närheten av en viss punkt. I vårt fall önskar vi finna en ny rektangel vars innehåll minimerar det kombinerade avstånd som infördes i förra avsnittet. Rektangeln definieras av 5 parametrar – bredd w, höjd h, mittpunkt (x, y) och rotationsvinkel θ.

Först beräknas det kombinerade avståndet för innehållet i rektangeln som vi fick med oss från förra bilden. Vi betecknar detta

$$d(\mathbf{r}),$$

där $\mathbf{r} = (x, y, w, h, \theta)$ är en vektor som innehåller parametrarna som beskriver rektangeln. Vi kommer nedan att beteckna komponenterna i $\mathbf{r}$ med $r_1 = x, r_2 = y$ etc.

Sedan beräknar vi en approximation av gradienten för avståndsfunktionen:

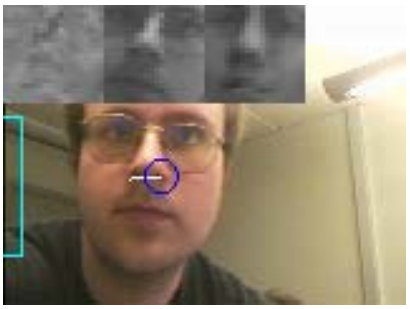
$$\nabla d(\mathbf{r}) = \sum_{i=1}^{5} \frac{\partial d}{\partial r_i}(\mathbf{r}) \cdot \mathbf{e}_i \approx \sum_{i=1}^{5} \frac{d(\mathbf{r} + \Delta r_i \mathbf{e}_i) - d(\mathbf{r})}{\Delta r_i} \mathbf{e}_i$$

Från flerdimensionell analys vet vi att gradienten av en differentierbar funktion pekar i den riktning funktionen växer snabbast. I gradientens motsatta riktning avtar funktionen snabbast. Vi hoppas nu att det finns ett lokalt minimum i närheten av $\mathbf{r}$ och att vår approximation av gradienten skall ge oss rätt riktning för att snabbast komma ner till minimumet. Om man går för långt kan man missa minimumet, och det kan därför vara lämpligt att prova gå några olika långa steg i gradientens motsatta riktning och se vilket som ger störst minskning av *d*.

Vi sammanfattar algoritmen:

1. beräkna en approximation av gradienten
   $\nabla d(\mathbf{r}_k) \approx \sum_{i=1}^{5} \frac{d(\mathbf{r}_k + \Delta r_i \mathbf{e}_i) - d(\mathbf{r}_k)}{\Delta r_i} \mathbf{e}_i$
2. beräkna $d(\mathbf{r}_k - \varepsilon \cdot \nabla d(\mathbf{r}_k))$
3. om detta värde är mindre än $d(\mathbf{r}_k)$, låt $\mathbf{r}_{k+1} = \mathbf{r}_k - \varepsilon \cdot \nabla d(\mathbf{r}_k)$

Om ingen förbättring erhölls i steg 3, kan det antingen betyda att ansiktet inte har rört sig, eller att ansiktsföljningen har misslyckats. I steg 1 har man möjlighet att välja olika steglängder Δr_i för de olika parametrarna vid beräkningen av gradienten. Man kan t ex välja hur många pixlar större bredden skall vara och hur många radianer större rotationsvinkeln skall vara.
De tre stegen kan itereras flera gånger per bild, om processorkraften räcker till. Det är nämligen mycket möjligt att man inte når fram till minimum efter bara ett steg.



Figur 48. Cirkeln visar var steepest-descent-algoritmen just nu befinner sig. Det vita strecket visar vilken riktning gradienten pekade när algoritmen tog förra steget. Arvid rörde huvudet åt vänster i bilden när den togs.

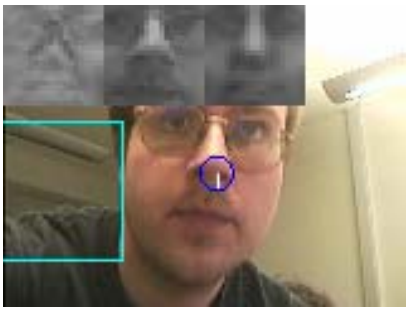
Låt oss nu diskutera några variationer på ovanstående algoritm. I steg 1 kan man välja en annan formel för den numeriska approximationen av gradienten som är mindre bruskänslig, t ex

$$\nabla d(\mathbf{r}_k) \approx \sum_{i=1}^{5} \frac{d(\mathbf{r} + \Delta r_i \mathbf{e}_i) - d(\mathbf{r} - \Delta r_i \mathbf{e}_i)}{2\Delta r_i} \mathbf{e}_i$$

Steg 2 kan utföras för flera olika värden på ε. Vid beräkning av de nya parametervärdena i steg 3 väljer man det av ε-värdena som gav störst minskning.
Man kan även låta epsilon vara en vektor, och ha olika vikter för de olika komponenterna i $\mathbf{r}$. Då har vi skalärprodukt mellan $\boldsymbol{\varepsilon}$ och gradienten:

$$\mathbf{r}_k - \boldsymbol{\varepsilon} \cdot \nabla d(\mathbf{r}_k)$$

Gradientens belopp blir stort om några av derivatorna är stora i punkten $\mathbf{r}_k$. Man kan gissa att om gradienten är stor kan man våga



Figur 49. Arvid rörde huvudet nedåt när den här bilden togs.

göra ett stort steg, därför att vi ännu inte nått en punkt där gropen börjar plana ut. Om man inte håller med om detta kan man normera gradienten och bara utnyttja dess riktning:

$$\mathbf{r}_k - \boldsymbol{\varepsilon} \cdot \frac{\nabla d(\mathbf{r}_k)}{\left|\nabla d(\mathbf{r}_k)\right|}.$$

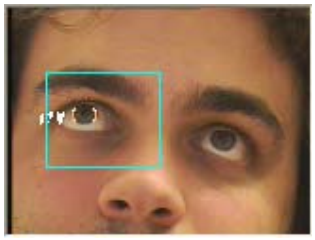
Då beror steget bara på $\boldsymbol{\varepsilon}$.

Övriga algoritmer

Förutom de algoritmer som använts i den slutliga produkten, har en mängd av olika mer eller mindre välgrundade idéer provats och förkastats. En del av de mer lovande algoritmerna presenteras nedan.

Cirkelfinnare

Förhoppningen med följande algoritm var att den skulle kunna hitta regnbågshinnan om ögat är synligt i bilden. Den bygger på en derivationsoperator, och algoritmen är därför bruskänslig. Derivationsoperatorer brukar man använda för kantdetektion, och så är fallet även här.



Figur 50. Bilden visar resultatet av en uttömmande sökning efter cirklar i det rektangulära sökfönstret. Cirkeln kring regnbågshinnan hade faktiskt högst värde på cirkelmåttet. Dessutom har ett antal mindre cirklar funnits till vänster i sökfönstret.

Antag att kanten mellan regnbågshinnan och ögonvitan är cirkulär i bilden. För att finna denna söker vi efter cirklar i bilden, där medelvärdet av den radiella derivatan av intensiteten är stor – d v s vi vill detektera ”runda kanter”. Nedan diskuterar vi hur man kan göra en diskret approximation av detta medelvärde.

Antag att vi har en gråskalebild vars intensitet i en viss punkt ges av $I(x, y)$. Följande integral är ett mått på intensiteten utmed en cirkel med radie r.

$$\int_{\theta=0}^{2\pi} I\left(x_0 + r\cos\theta, y_0 + r\sin\theta\right) d\theta$$

Vi har alltså parametriserat cirkeln:

$$\begin{cases} x(\theta) = x_0 + r\cos\theta \\ y(\theta) = y_0 + r\sin\theta \end{cases}$$

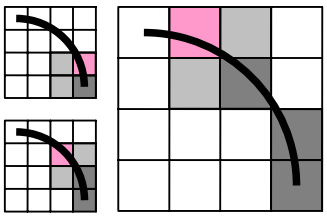
Integralen är i själva verket sånär som på division med cirkelns omkrets lika med medelvärdet av intensiteten utmed cirkeln.

Om vi deriverar integralen partiellt med avseende på r så erhåller vi ett mått på hur snabbt intensiteten ändras i radiell led. Vi antar att man i denna integral kan kasta om integrations- och derivationsordning.

$$\frac{\partial}{\partial r}\int_{\theta=0}^{2\pi} I(x,y)\,d\theta = \int_{\theta=0}^{2\pi} \frac{\partial}{\partial r} I(x,y)\,d\theta =$$
$$= \int_{\theta=0}^{2\pi} \left(\cos\theta \cdot I_x(x,y) + \sin\theta \cdot I_y(x,y)\right) d\theta$$

Rent praktiskt tittar vi bara på första, fjärde, femte och åttonde oktanten i ett kartesiskt koordinatsystem med origo i (x_0, y_0). Anledningen är att delar av regnbågshinnan kan vara dold av ögonlocken.

För att kunna använda detta i praktiken måste vi diskretisera integralen samt derivatorna I_x och I_y ovan.



Figur 51. Bilden visar tre steg i en Bresenham-liknande algoritm som används för att stega fram utmed en cirkel i första kvadranten.

En Bresenham-liknande algoritm med beslutsvariabel används i vår implementation för att stega fram utmed en cirkel i första oktanten. Cirkelns delar i övriga oktanter erhålles genom spegling i koordinataxlarna. Beslutskriteriet går ut på att man beräknar $\Delta = (x_0 + x)^2 + (y_0 + y)^2 - r^2$ för tre kandidatpunkter (upp, upp vänster samt vänster) och väljer den som ger lägst värde på Δ.

För varje steg beräknar vi integrandens värde i den punkten och lägger det till en summa som approximerar integralen.

Antag att punkten har koordinaterna (x, y) räknat med cirkelns medelpunkt som origo. För att beräkna $\cos\theta$ används x/r, och $\sin\theta = y/r$. I_x approximeras av $I(x+\Delta x, y) - I(x, y)$. På samma sätt får vi $I_y \approx I(x, y+\Delta y) - I(x, y)$.

Vi försökte införa en hel del heuristik i algoritmen, men trots detta blev den aldrig särskilt bra på att finna just regnbågshinnan, utan fann cirklar på alla möjliga ställen där den radiella derivatan råkade vara stor. Problemet är att medelvärdet vi använder oss av kan vara högt även i områden som en mänsklig betraktare inte skulle anse innehålla någon cirkel.

Blockvis analys

Grundidén i det här avsnittet är att beräkna binära representationer av bilden med avsevärt lägre upplösning. Sedan beräknas snittet av dessa för att få fram ett intressant område. I detta avsnitt diskuteras hur detta skulle kunna leda till en ansiktsfinnare.

Låt oss införa begränsningarna att bilden måste innehålla precis ett ansikte, och att bakgrunden och de klädesplagg som syns i bilden måste vara enfärgade och omönstrade. Då kan man tänka sig att man genom att kombinera en rörelsedetektor och en detaljfinnare skulle kunna finna det ansiktet.

Det är helt enkelt fråga om att för varje block om t ex $n \times n$ pixlar i den ursprungliga bilden beräkna något värde från mängden {0, 1} som lagras i en ny matris J_k. Detta måste göras minst två gånger med k=1 och k=2 för att vi skall kunna bilda snittet av dessa två matriser. Vi definierar snittet $J_1 \bigcap J_2$ som

$$J_1 \bigcap J_2 (x, y) = \begin{cases} 0 & J_1(x, y) \neq J_2(x, y) \\ 1 & J_1(x, y) = J_2(x, y) \end{cases}$$

Vi skall nu ge exempel på en rörelsedetektor J_1 och en detaljfinnare J_2. I båda fallen undersöks om en tillräckligt stor andel av pixlarna i blocket uppfyller ett visst villkor.

Låt I_{m-1} vara föregående bild i videoströmmen och I_m vara den nuvarande. Ett sätt att detektera om något rört sig i bilden är att undersöka skillnaden i intensitet mellan bilderna. Vi definierar skillnadsbilden

$$\Delta(x,y) = \left| I_m(x,y) - I_{m-1}(x,y) \right|.$$

Detta är en enkel derivationsoperator i tiden.

Om ingenting rört sig framför kameran blir $\Delta(x,y)$ då bara brus. Därför är det naturligt att tröskla skillnadsbilden. Låt

$$\delta(x,y) = \begin{cases} 0 & \Delta(x,y) < \theta \\ 1 & \Delta(x,y) \geq \theta \end{cases}$$

Vi bestämmer oss nu för att ett block innehåller rörelse om en viss andel r av pixlarna i blocket uppfyllde trösklingsvillkoret.

$$J_1(x,y) = \begin{cases} 0 & \sum_{j=0}^{n-1}\sum_{i=0}^{n-1} \delta(x+j, y+i) < rn^2 \\ 1 & f.ö. \end{cases}$$

På liknande sätt inför vi en detaljfinnare J_2 genom att byta ut funktionen Δ ovan mot summan av två faltningar med Sobel-faltningskärnorna G_x och G_y.

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \quad G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}.$$

Vi väljer nu

$$\Delta(x,y) = \left| G_x * I(x,y) \right| + \left| G_y * I(x,y) \right|$$

Detta är en enkel derivationsoperator i rummet. I själva verket är det en approximation av beloppet på gradienten i punkten (x, y).

$$\left|\nabla I\right| = \sqrt{I_x^2 + I_y^2} \approx \left|I_x\right| + \left|I_y\right|$$

Sobel-faltningskärnorna ger mindre bruskänsliga derivationsoperatorer, men vi trösklar ändå resultatet för att kunna använda samma formler som för J_1. Ett litet problem i sammanhanget är att om ursprungliga bilden är av storlek $n \times n$ så är resultatet av faltningen $(n-1) \times (n-1)$. En lösning på problemet är att även utnyttja en 1 pixel tjock ram av pixlar utanför blocket vid faltningen.

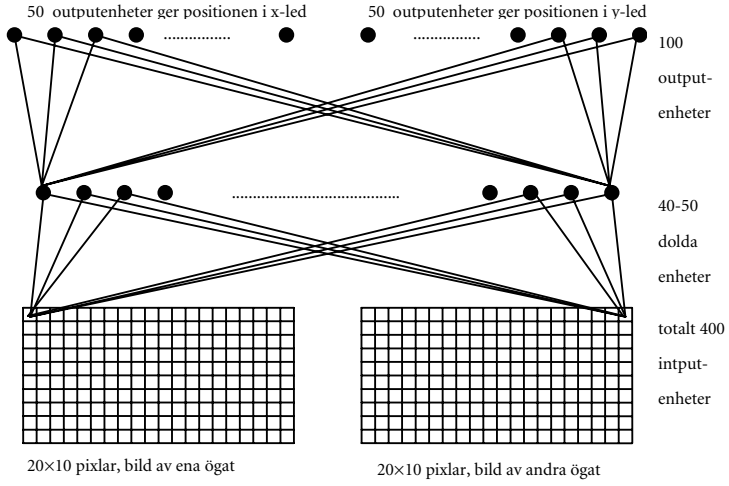
Detaljfinnaren är avsedd att användas för att ta bort de områden som rört sig men som inte innehöll mycket detaljer, t ex de delar av kläderna som syns i bilden.

Med de hårda begränsningarna av bildens innehåll som infördes ovan verkade detta fungera så länge ansiktet rör sig. En komplikation är att personens hår genererar höga värden på båda derivationsoperatorerna ovan. Om bakgrunden är mer komplicerad eller det finns risk för att en hand eller ett annat rörligt föremål dyker upp i bilden, kommer snittet av J_1 och J_2 med säkerhet inte att ge ett område som bara innehåller ansiktet.

Neurala nätverk

Många metoder för att finna ansikten använder sig av neurala nätverk och de har också med framgång använts till ögonstyrning. Vi gjorde en del tester med neurala nätverk. De beskrivs kortfattat här för fullständighetens skull och för att möjligtvis ge uppslag till någon som ska arbeta inom ett område som liknar vårt. Teori och praktiska detaljer angående neural nätverk kommer inte att gås igenom här.

I en artikel, som vi hittade på nätet under vår marknadsundersökning, beskrev Rainer Stiefelhagen, Jie Yang och Alex Waibel, vid universitet i Karlsruhe, hur de använt neurala nätverk för blickföljning (Stiefelhagen, 1997). Figur 52 visar arkitekturen för deras nätverk.



Figur 52. Ett neuralt nätverk som använts för ögonföljning.

De hade valt att koda målkoordinaterna som normalfördelningar. För varje område på skärmen det skulle vara möjligt att urskilja i x- eller yled fanns en utsignalsnod. Sedan tränades nätet att ge ifrån sig diskreta normalfördelningar med väntevärde som angav den korrekta koordinaten i x- och y-led. Tanken med denna kodning var att det skulle bli enbart kontinuerliga förändringar i utsignalen, oavsett hur insignalen ändrades. Tänk på hur det ser ut om istället utsignalerna

kodats med en enda etta på den rätta koordinaten och nollor överallt annars. Utsignalen skulle då vara en spikfunktion. Om en liten förändring i insignalen ska ge upphov till en ny fokuspunkt så måste spiken flyttas till en annan position. Därmed blir skillnaden mellan de två utsignalerna mycket stor. Med den här typen av normalfördelningskodning undviker man detta problem.

Enligt deras egen artikel hade de uppnått mycket goda resultat och en hög noggrannhet på 1.3-1.8 grader beroende på om nätverket var upptränat för en specifik person eller om det var upptränat för flera olika personer. Det är alltså möjligt att detta är en metod att experimentera med för någon som vill ta vid efter oss och fortsätta med ögonstyrning.

Inspirerade av denna metod och dess resultat och en del andra artiklar provade vi om det var möjligt att använda neurala nätverk för att hitta nästippen i en bild där det fanns ett ansikte. Vi använde två nät och kodade koordinaterna med normalfördelningar. De tester som gjordes var i och för sig inte så omfattande men resultatet var en aning avskräckande. Efter en hel del träning ansåg nätverken att nästippen låg på samma ställe i nästan alla bilder som visades för det. Självklart är det möjligt att vi angrep problemet på fel sätt i det här fallet och att det skulle vara möjligt att lyckas bättre med neurala nätverk.

Ett ganska lyckat resultat uppnåddes dock med neurala nätverk. Vi provade, för att underlätta segmentering, att träna ett nätverk att avgöra om en bild innehöll en eller två fyllda ellipser. I sin bild hade båda ellipserna samma färg och var dessutom en aning utsuddade längs sina konturer. (Detta kom av att ellipsbilderna genererades i en storlek och sedan förminskades till 15×15 mha interpolation). Att ellipserna var lite utsuddade kan ha gjort det lättare för nätverket. Här kodades nu målsignalerna på vanligt vis, med en utsignalsnod. En nolla betydde en ellips och en etta två stycken. Detta var en uppgift som nätverket faktiskt blev mycket bra på. Dess felfrekvens på träningsmängden och slumpmässigt genererade tester var låg. Om alternativen att det också kunde finnas inga eller tre ellipser i bilden infördes, så försämrades emellertid nätets prestanda i våra tester. Det här resultatet var förstås inget vi sedan använde, men det visar åtminstone i någon mån på att neurala nätverk kan fungera.

Ovan nämnda experiment utfördes mha MATLABS neural network toolbox, vilken är mycket praktisk och lättanvänd. Med den går det tex enkelt att byta mellan olika träningsalgoritmer.
Om man sedan skulle vilja använda ett nätverk som är upptränat i MATLAB borde det gå ganska smidigt att exportera de olika lagrens vikter och implementera ett neuralt nätverks aktiveringsalgoritm i C++. (På så vis undviker man implementerandet av träningsalgoritmerna, vilka är mera komplicerade än aktiveringsalgoritmen).

Appendix 2 – MATLAB-introduktion

Här ges en kort introduktion. Det finns också ett bra hjälpsystem i MATLAB. Om man skriver help får man upp ett antal områden man kan söka vidare på. Ifall man skriver `help funktion` får man information om funktionen ifråga. I avsnittet om egenansikten finns ett exempel som visar hur man beräknar egenansikten för ett antal sparade bilder.

MATRISER OCH VARIABLER

Den vanligaste datastrukturen i MATLAB är matrisen. Några exempel på hur man kan skapa matriser:

```
A = [1 2 3; 3 4 5; 6 7 8; 9 10 11];
k = 10;
B = k*[1 3 5; 2 4 6; 7 9 11; 8 10 12];
I = eye(3);       % enhetsmatrisen av ordning 3
Z = zeros(4,3);   % 4×3 matris med bara nollor
E = ones(3);      % 3×3 matris med bara ettor
D = diag([1 2 3]);% diagonalmatris
x = 0:0.1:10;     % x = [0 0.1 0.2 ... 10]
y = sqrt(x);      % y = [0 sqrt(0.1) sqrt(0.2) ... sqrt(10)]
i = 10:-1:1;      % i = [10 9 8 ... 2 1]
```

Här markerar semikolonen inuti hakparenteserna ny rad i matrisen. Semikolon sist på en rad betyder att resultatet av operationen inte ska skrivas ut i kommandofönstret.

INDEXERING AV MATRISER

Index i matlab börjar alltid med ett och inte med noll som i de flesta andra programmeringsspråk.

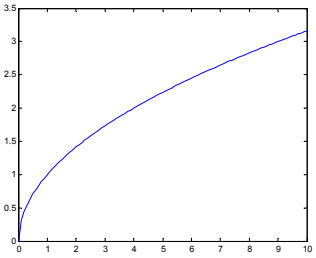
```
A(1,1);     % elementet på första raden och
            % första kolonnen i A
A(:,1);     % första kolonnnen i A
A(2,:);     % andra raden i A
A(1:2,2);   % första och andra raden i A
A(:,end);   % sista kolonnnen i A
size(A,1)   % antalet rader i A
size(A,2)   % antalet kolonner i A
size(A)     % returnerar både antalet rader och
            % kolonner
A(:);       % kolonnstapling av A
```

NÅGRA RÄKNEOPERATIONER

```
A+B;  % addition (kräver att matriserna
      % är av samma ordning).
A-B;  % subtraktion (kräver att matriserna
      % är av samma ordning).
```

```
A*B;  % matrismultiplikation
A.*B; % elementvis multiplikation (kräver att
      % matriserna är av samma ordning).
A./B; % elementvis division (kräver att
      % matriserna är av samma ordning).
A^2;  % matrispotens
A.^2  % elementvis potens
A^-1; % inversen till A
A';   % hermitesk transponering (lika med vanlig
      % transponering för reella matriser)
A.';  % vanlig transponering

conv2(A,M); % 2d-faltning av A och M
cos(A);     % cosinus av elementen i A
cov(A);     % skattar kovariansmatris för
            % flerdimensionell stokastisk
            % variabel, där varje rad i A
            % betraktas som en observation
[V,D]=eig(A);% egenvektorer och egenvärden till A
            % hamnar i V och D
exp(A);     % exponentialfunktionen av elementen
            % i A
inv(A);     % inversen till A
log(A);     % naturliga logaritmen av elementen
            % i A
max(A);     % returnerar största elementet från
            % varje kolonnn i A
mean(A);    % medelvärdena för varje kolonn i A
min(A);     % returnerar minsta elementet från
            % varje kolonnn i A
plot(x,y);  % plotta varje element i y mot
            % motsvarande i x
reshape(A,m,n);% omformar matrisen A så att den
            % blir m×n, förutsätter att A har mn
            % element
sin(A);     % sinus av elementen i A
sqrt(A);    % kvadratroten ur elementen i A
```



Figur 53. Resultatet av MATLAB-kommandot plot(x,y);

Om man tex vill plotta funktionen $\sqrt{x}$ i intervallet [0,10] kan man göra så här:

```
x = 0:0.1:10;
y = sqrt(x);
plot(x,y);
```

IF- OCH ELSESATSER

```
if x~=y              % x skild från y
      % kommandon...
elseif x==y & y==1   % & betyder och
      % kommandon...
elseif x>y | y <= 0  % | betyder eller
      % kommandon...
else
      % kommandon...
end
```

FOR-LOOP

```
for k=1:K   % k går från 1 till K
      % kommandon...
end

for k=[1 1 2 3 1 4 8 [K:-1:1]]% är också
            % tillåtet
      % kommandon...
end
```

WHILE-LOOP

```
k=0;
while k<10
      % kommandon...
      k=k+1;
end
```

CELL-ARRAY

En annan datastruktur som kan vara användbar är en cell-array. En sådan är en array av matriser som kan ha olika storlek, den indexeras med {index}. Ett exempel på en cell-array:

```
matrices{1}=[1 2 3; 3 4 5; 6 7 8];
matrices{2}=[1 1 1; 3 4 5];
matrices{3}=1;
matrices{1};      % returnerar den första
                  % matrisen
```

Kommandot size fungerar även på en cell-array.

BILDER

I matlab lagras en gråniväbild som en matris.

```
image(im)   %visa bilden im, dess elementmåste
            % vara mellan 0 (svart) och 255
            % (vitt)
imagesc(im) % skalar om elementen så att det
            % minsta blir 0 och det största 255,
            % innan bilden visas
im = imread('face.tif','tif');% ladda bilden med
            % filnamn face.tif och formatet till
            % tif till im
imwrite(im,'face2.jpg','jpg');% spara im med
            % filnamnet face2.jpg i formatet jpg
im2 = imresize(im,[20 20],'bilinear');% gör om
            % bilden im till 20×20 med bilinjär
            % interpolation
im2 = imresize(im,[20 20],'bicubic');% gör om
            % bilden im till 20×20 med bikubisk
            % interpolation
im2 = fliplr(im); % vänd bilden i horisontell led
```

Om man vill filtrera en bild kan man göra så här:

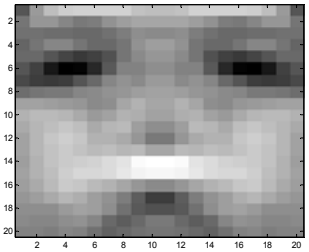
```
A = [-1 -2 -1;0 0 0;1 2 1];% sobelmask
im2 = conv2(im,A);      % 2d-faltning
```

Egenansikten

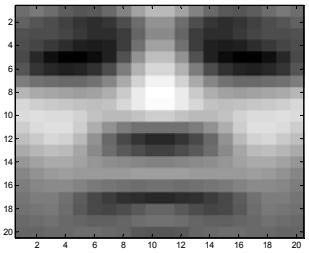
Här följer ett exempel på hur man kan göra egenansikten i matlab. Vi förutsätter att alla träningsbilderna är lika stora och har filnamnen img1.jpg, img2.jpg osv.

```
datamat = [];
for k = 1:nbrOfImages % nbrOfImages får tyvärr
                      % hårdkodas
      tempim =
      imread(['img',num2str(k),'.jpg'],'jpg');

      datamat = [datamat; tempim(:)'];
      tempim = fliplr(tempim);% om vi vet att
                      % bilderna föreställer något
                      % som antas vara symmetriskt
                      % kring vertikal axeln, som tex
                      % ansikten framifrån, så kan vi
                      % ta med speglingen
                      % i vertikala axeln också
      datamat = [datamat; tempim(:)'];
end
datamat = double(datamat);% konvertera till
                      % double, cov kräver nämligen
                      % det
covmat = cov(datamat);% beräkna kovariansmatrisen
[V,D] = eig(covmat);% V får covmats egenvektorer
                      % som kolonner och D dess
                      % egenvärden på diagonalen
m = size(tempim,1);% tempim kommer referera till
                      % sista bilden i sekvensen, men
                      % de är ju alla lika stora
n = size(tempim,2);
bas1 = V(:,end);  % ta ut sista kolonnen ur V
im = reshape(bas1, m, n); % omforma bas1 till m×n
meanim = mean(datamat);
meanim = reshape(meanim,m,n);
figure(1)             % nästa figur att rita i är
                      % figur 1
imagesc(im)
colormap(gray);   % bilden ska visas i gråskalor
figure(2)             % nästa figur att rita i är
                      % figur 2
imagesc(meanim)
colormap(gray);
```



Figur 54. Resultatet av Matlab-kommandot imagesc(im); Nämligen det första egenansiktet. Att det är symmetriskt beror på för varje ansikte i träningsmängden har även dess spegling i vertikala axeln tagits med.



Figur 55. Resultatet av Matlab-*kommandot imagesc(meanim), nämligen medelansiktet. Att det är symmetriskt beror på att för varje ansikte i träningsmängden har även dess spelging i vertikala axeln tagits med.*

Matrisen datamat tilldelas en tom matris på första raden. Tredje raden laddar in bild nr k i ordningen. På fjärde raden byggs datamat på radvis med datan från varje bild. Kommandot cov skapar en kovariansmatris för datan i datamat och eig beräknar dess egenvärden och egenvektorer. I matlab behöver man alltså inte bry sig om att själv beräkna medelvärdesbilden för att skatta kovariansmatrisen. Efter detta finns egenvärdena på diagonalen i V, som är en diagonalmatris, och egenvektorerna finns som kolonnner i D. Kommandot eig i matlab sorterar också egenvärdena i stigande ordning, så att de största egenvärdena ligger längst ner åt höger i V. Egenvektorerna sorteras på motsvarande vis.

Egenansiktena är nu i form av kolonnvektorer så om man vill titta på dem måste man göra en invers kolonnstapling och återforma dem till m×n matriser. Kommandot `reshape` omformar en matris till en annan med formatet m×n, (detta förutsätter förstås att matrisen har mn element). Kommandot `mean` beräknar medelvärdet kolonnvis för `datamat`, så det som returneras av det är alltså en radmatris.

M-FILER

Kommandon kan skrivas in i kommandofönstret, men det är också möjligt att skriva dem i en texteditor så att man får ett riktigt program. Detta kan göras på två sätt. Man kan skriva in allt man vill göra i en textfil och spara som .m. Det andra alternativet är att skapa en funktion. Textfilen måste då börja med en deklaration av funktionen, och kan tex se ut så här:

```
function w = foo(x,y,z)
w = x^2+y^2+z^2;
```

Denna sparas sedan som foo.m.
När man vill exekvera en m-fil så skriver man bara dess namn, om det inte är en funktion för då måste man skicka med argument också. Skillnaden mellan en funktion och en vanlig m-fil är att alla variabler som allokeras inuti funktionen inte kan användas utanför den och försvinner när den exekverat färdigt. I en vanlig m-fil kommer allting att finnas kvar i minnet, också efter anropet.
Exempelfunktionen kan tex anropas så här:

```
a = foo(1,2,3);
```

SPARA OCH LADDA DATA

```
save data A B      % spara variablerna A och B i
                   % en fil som kommer heta
                   % data.mat
save data          % sparar alla variabler i
                   % arbetsminnet i filen data.mat
load data          % ladda in de variabler som
                   % finns sparade i data.mat
```

Appendix 3 – Klass- och sekvensdiagram

CArvidMickeController

Interfacet mellan det grafiska gränssnittet och våra experimentella funktioner.

CFaceFinder

Använder sig av färg, rörelse och PCA för att hitta ansikten.

CBlockMatrix

Denna klass är en sorts matris som delar in en bild i olika block. Varje block refererar till ett område i bilden och har ett värde som säger hur intressant detta område är.

CSegmentedBlockMatrix

Här har varje block ett heltalsvärde som anger vilket segment det tillhör. Värdet -1 talar om att det inte tillhör något segment.

CColorBlocks

Här kan varje block tillhöra flera olika segment.

CMotionHint

Beräknar blockvisa differensbilder för att åstadkomma rörelsedetektion enligt avsnitt Blockvis analys i kapitlet Övriga algoritmer.

CMatrixMath

Ett interface för matrisoperationer.

CMatrixMath2x2

Implementerar CMatrixMath för 2×2 matriser.

CStatisticsArray

Innehåller en statistisk beskrivning över data som kategoriserats i olika klasser. Datan själv finns inte med, utan enbart den statistiska beskrivningen i form av kovariansmatriser, medelvärden och sannolikheter. Datan antas här vara tvådimensionell, därav ärvningen av CMatrixMath2x2.

CKmeansArray

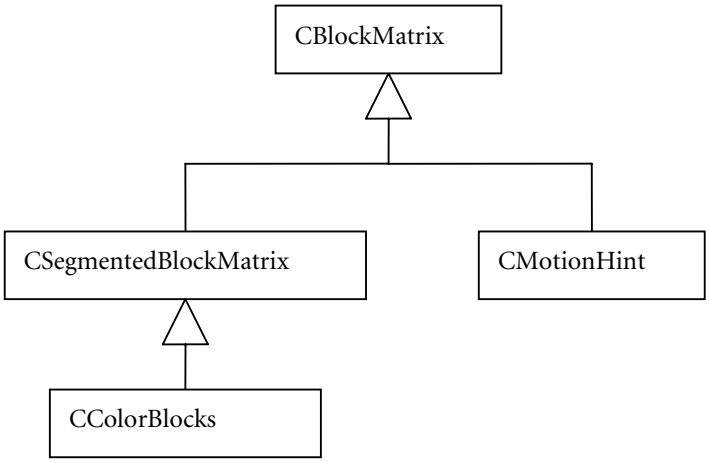
Denna klass kan utföra kmeans-algoritmen, samt innehåller den statistiska beskrivningen av de olika klasserna. Datapunkter kan läggas till efterhand för att sedan låta algoritmen skatta en statistisk beskrivning.

CMixturesArray

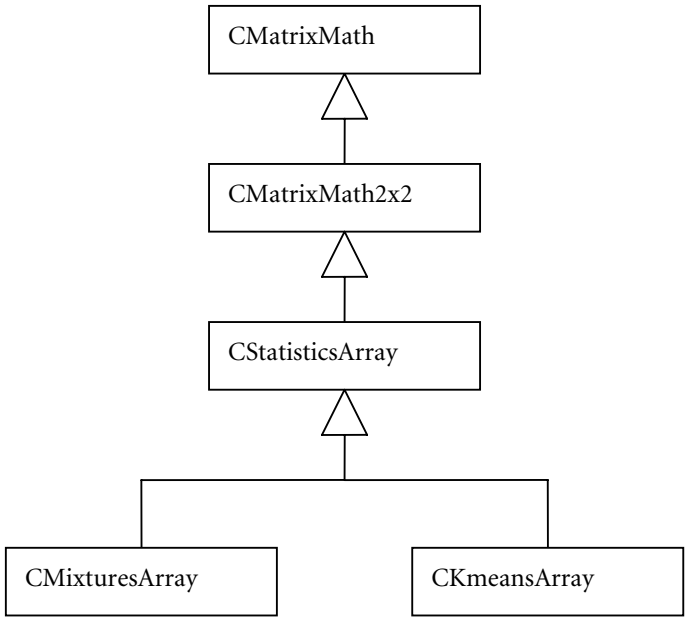
Kan göra EM-algoritmen och innehåller den statistiska beskrivningen av de olika klasserna. Datapunkter kan läggas till efterhand för att sedan låta algoritmen skatta en statistisk beskrivning.

CPCA

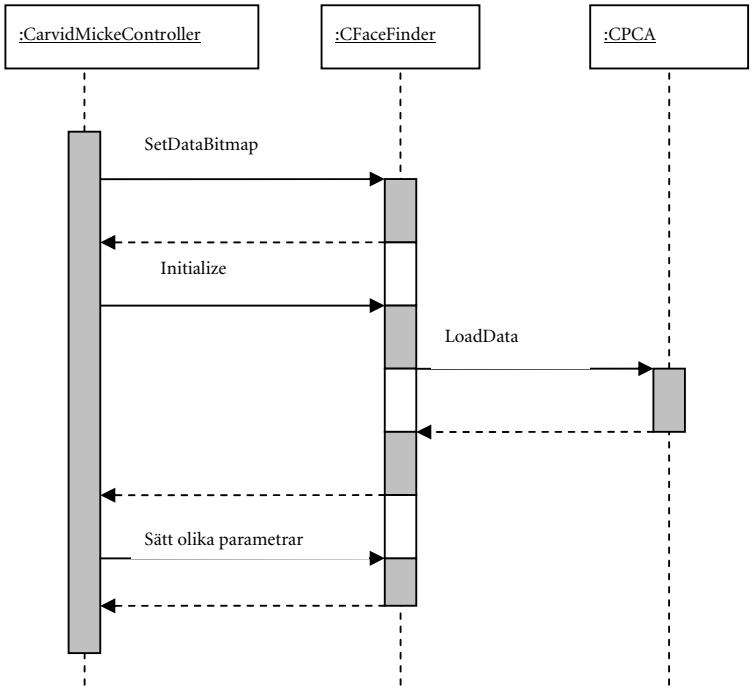
Denna klass utför själv ingen PCA. Den laddar PCA-data från fil och kan utföra beräkningar som har med PCA-matchning att göra. T ex kan den transformera ett bildområde till ett givet format och beräkna dffs för en bild.



Figur 56. Ärvningsstrukturen för de klasser som ärver av CBlockMatrix.

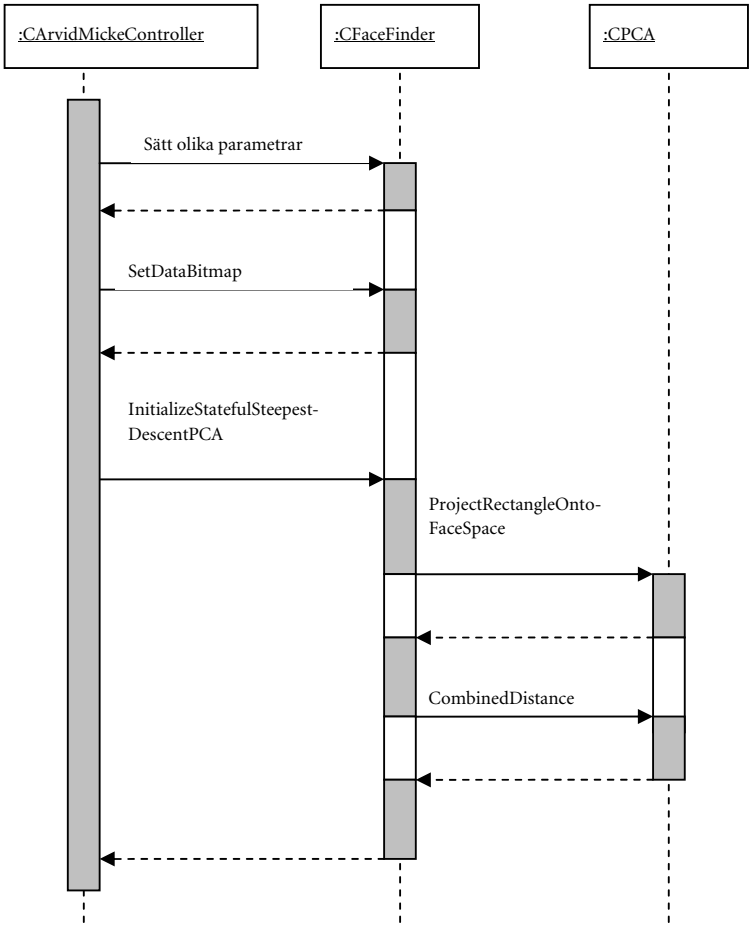


Figur 57. Ärvningsstrukturen för de klasser som ärver av CMatrixMath.



Figur 58. Initialisering av FaceFindern.
Pilen med texten ”sätt olika parametrar” ska visa att publika datamedlemmar i FaceFindern sätts utan funktionsanrop.

Figur 59. När musknappen tryckts ned väljs en ny rektangel att leta efter ett ansikte att följa i Följningsalgoritmen initialiseras. Figuren visar inte exakt hur anropen går till. Både ProjectRectangleOntoFaceSpace och CombinedDistance anropas i själva verket flera gånger. Pilen med texten "sätt olika parametrar" ska visa att publika datamedlemmar i FaceFindern sätts utan funktionsanrop.



Appendix 4 – UMIST ansiktsdatabas

Vid konstruktionen av de egenansikten som använts för testning av våra algoritmer användes en ansiktsdatabas som tillverkats vid University of Manchester Institute of Science and Technology.

Det som gjorde just denna ansiktsdatabas så intressant för oss är att den innehåller bilder på ansikten tagna från olika vinklar, och alltså även inkluderar semiprofil- och profilbilder.

Denna återfinns på webben:
http://images.ee.umist.ac.uk/danny/database.html.

Villkoren för att få använda denna ansiktsdatabas återges här på engelska. Texten nedan är hämtad ordagrant och utan att ändra formatering från (Graham, 1998):

UMIST grants the right to use the face database with the following restrictions:

- **Only images of subject 1a may be published (and then only with written permission). This is not out of vanity, it is for legal reasons.**
- **The author must reference at least one of the original papers from this site. Preferentially:**
    - Characterizing Virtual Eigensignatures for General Purpose Face Recognition
      Daniel B Graham and Nigel M Allinson.
      (in) Face Recognition: From Theory to Applications ,
      NATO ASI Series F, Computer and Systems Sciences, Vol. 163.
      H. Wechsler, P. J. Phillips, V. Bruce, F. Fogelman-Soulie and T. S. Huang (eds), pp 446-456, 1998.

- **You must notify Daniel Graham or Nigel Allinson when you wish if you intend to use the database.**

Vi har kontaktat Daniel Graham på den angivna email-adressen och erhållit hans tillåtelse att använda ansiktsdatabasen. Vår rapport uppfyller även de övriga villkoren ovan.

Appendix 5 – Kameran

Under utvecklingsarbetet användes en *zoom-pan-tilt* kamera av märket Sony EVI-D31. *Zoom* innebär att kamerans optiska zoom kan styras från fjärrkontroll eller dator. *Pan-tilt* betyder att själva kameran sitter på en ställning som med hjälp av elektriska motorer kan vridas kring två axlar. Orsaken till att vi använde en sådan kamera var att vi hoppades kunna utnyttja dessa funktioner för att åstadkomma ögonföljning. Vi antog att för att få en tillräckligt högupplöst bild för att genomföra dessa beräkningar krävdes att kameran kan zooma in på ögat och följa dess rörelse ifall användaren flyttar huvudet.

I detta appendix beskrivs en rad tekniska detaljer ur specifikationerna för kameran, samt en klass vi skrev i Microsoft Visual C++ för att styra kamerans rörelse.

Kameraklassen

För att öka abstraktionsnivån på bildbehandlingsklassernas kommunikation med kameran, skapades speciella klasser för detta. Den mest grundläggande kameraklassen, CameraIO, kan läsa och skriva på COM-porten. Kamerans VISCA-interface är nämligen kopplat till en av datorns COM-portar. För mer information om protokollet och tillgängliga kommandon, se (Sony Corporation, 1999).

Först beskrivs här hur kameratråden är implementerad. Sedan görs en jämförelse med specifikationerna i (Sony Corporation, 1999) för hur kommunikationen skall gå till, för att bekräfta att implementationen uppfyller dessa krav. Dessutom argumenteras för att trådarnas synkronisering fungerar väl.

Protokollet som används för att kommunicera med kameran från datorn kallas VISCA. För att ange hexadecimala värden används nedan beteckningen 0x. T ex anges 255 med 0xFF.

Kommunikation i VISCA-protokollet börjar med en header som anger kommandots mål och ursprung. Man kan nämligen ha flera kameror kopplade i serie på en enda COM-port. Sedan följer själva meddelandet följt av 0xFF för att avsluta kommunikationen. Kommandots mål anges som 0xZX, där Z är 8 plus avsändarens adress, och X är målets adress. Datorn har adress 0. Om X är 8, betyder det att meddelandet skall broadcastas, d v s gå ut till alla kameror. Kamerorna får sin adress genom att datorn broadcastar kommandot AddressSet, 0x88 30 01 FF.

Varje kamera kan lagra kommandon i minnesenheter som kallas sockets. Varje kamera har två sådana och kan följaktligen buffra två kommandon. Ifall ett meddelande har skickats till en kamera kan ett andra meddelande skickas efter att det första har ACKats, men medan det fortfarande exekveras, tack vare sockets. Men om ett tredje meddelande skickas innan någon av de två är färdigexekverade blir det fel och ett felmeddelande skickas från kameran.

CameraIO startar två trådar när dess konstruktor anropas. Den ena tråden, InputThread, läser kontinuerligt en byte i taget på COM-porten. Den sover tills någonting finns att läsa. Så fort den har läst 0xFF antar den att den fått in ett helt meddelande, eftersom specifikationerna anger att varje meddelande är 0xFF-terminerat. Beroende på vilken typ av kommando det rör sig om, signalerar den antingen en CEvent ackEvent eller errorEvent. Om det är svar på en fråga, kallas dessutom en virtuell funktion `HandleReply`. Då kan man i en subklass implementera önskat beteende för att ta hand om informationen. Av samma anledning kallas en virtuell funktion `HandleError` om ett felmeddelande tas emot. Virtuella funktioner kan vara långsamma, så kanske borde en funktionspekare (eng *callback-funktion*) användas istället om det visar sig för långsamt.

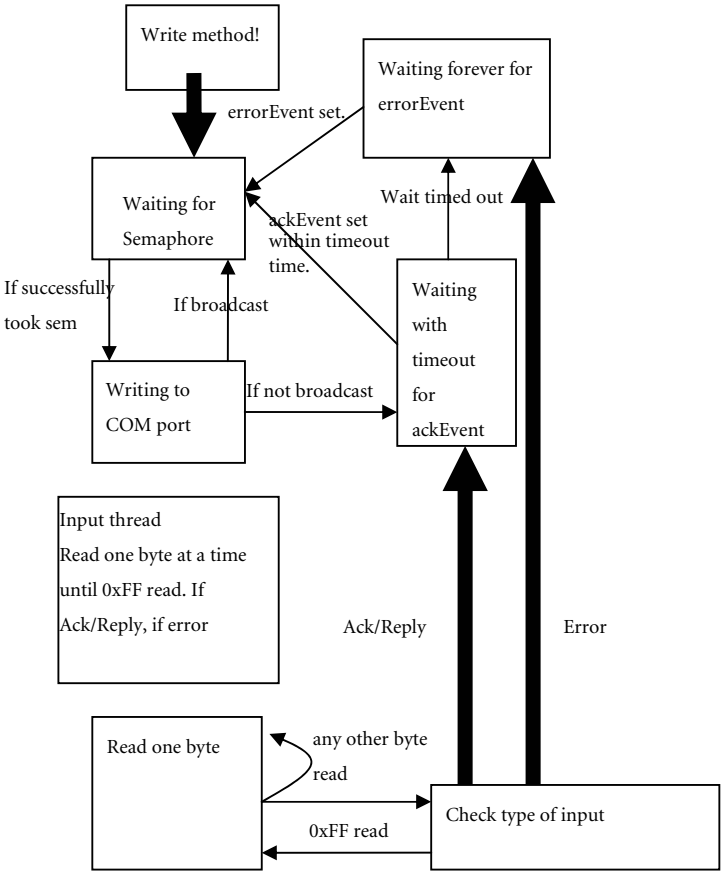
Meddelande	**Hexadecimal representation**	**CEvent**	**Virtuell funktion**
ACK	Z0 4Y FF	ackEvent	-
Command completion	Z0 5Y FF	-	-
Information return	Z0 50 … FF	ackEvent	HandleReply
Address set	Z0 38 FF	-	-
Felmeddelanden			
Syntax error	Z0 60 02 FF	errorEvent	HandleError
Command buffer full	Z0 60 03 FF	errorEvent	HandleError
Command cancel	Z0 60 04 FF	errorEvent	HandleError
No sockets	Z0 60 05 FF	errorEvent	HandleError
Command not executable	Z0 60 41 FF	errorEvent	HandleError
Broadcast			
AddressSet	88 30 0X FF	-	-
IF_Clear	88 01 00 01 FF	-	-

I tabellen anger Z avsändare och X anger vilken adress som ska sättas med samma format som specificerades ovan. Y anger vilken socket som utförde det kommando som ACKade eller avslutades.

Skrivning sker från den andra tråden, OutputThread, endast om någon ber om det, nämligen som följd av att en metod `Write(BYTE* pSendBuf, int nSendBufSize)` anropas. När

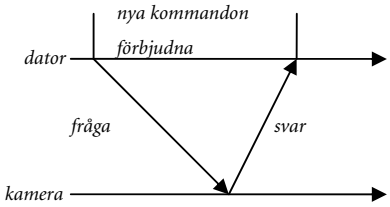
Write anropas, lagras en kopia av kommandot som pSendBuf pekar på i en kö, och en semafor ökas. OutputThread ligger och väntar på att ta semaforen. Sen skriver den kommandot på COM-porten, och väntar ett visst antal millisekunder på att ackEvent skall bli satt. Om så inte sker inom timeout-intervallet, väntar den istället på att errorEvent skall bli satt. Detta funkar eftersom varje icke-broadcast kommando returnerar antingen ACK, Information return eller något felmeddelande.

En tråd måste ges av en static funktion. För att trådarna skall kunna använda private och protected variabler från en instans av CCameraIO, ligger deras implementationer inuti varsin publik funktion, som anropas ifrån trådens statiska funktion. T ex görs i konstruktorn CCameraIO() anropet
`AfxBeginThread(InputThread,this).`
InputThread gör i sin tur
`return ((CCameraIO *)pParam)->RunInput().`



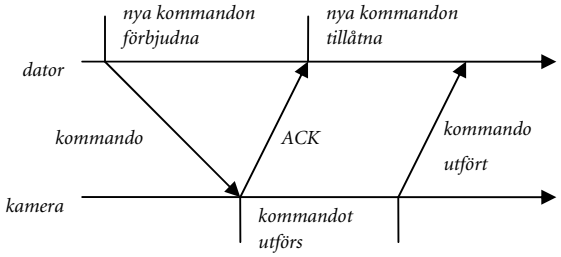
Figur 60

Här följer några figurer som visar hur kommunikation med kameran skall gå till enligt (Sony Corporation, 1999), följt av en diskussion om hur CCameraIO hanterar dessa.



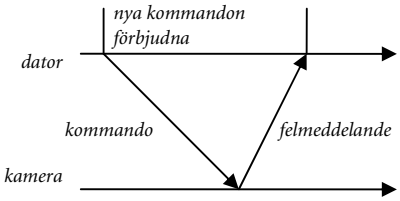
Figur 61

Enligt figuren är nya kommandon förbjudna efter att man har skickat en fråga och innan man fått svar. Implementationen uppfyller detta eftersom output-tråden väntar på ACK eller svar innan den skickar nästa kommando.



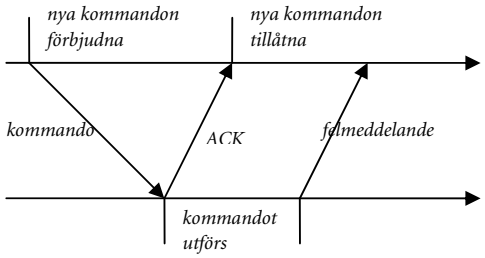
Figur 62

Det är inte tillåtet att skicka ett nytt kommando förrän ett ACK-meddelande har talat om att det föregående mottagits av kameran. Input-tråden ignorerar kommando utfört meddelandet, och det finns ingen mekanism för en observatör utanför input-tråden att få reda på om detta meddelande kommit in. Detta kan behöva ändras i en framtida version av kameraklassen.

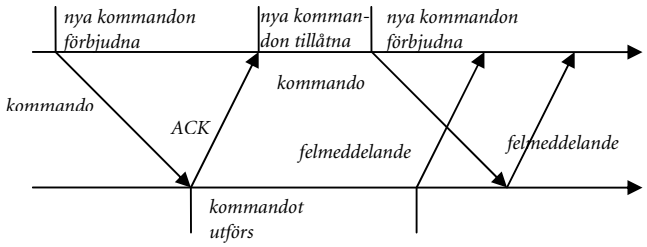


Figur 63

Om inget ACK kommer in, väntar output-tråden istället tills ett felmeddelande kommer in. Inga nya kommandon skickas alltså i det förbjudna intervallet.

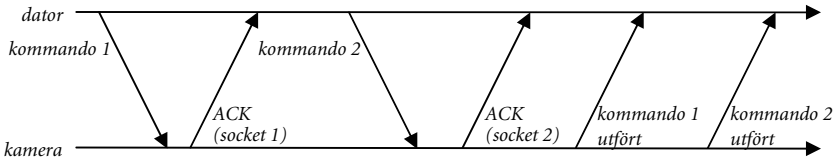


Figur 64



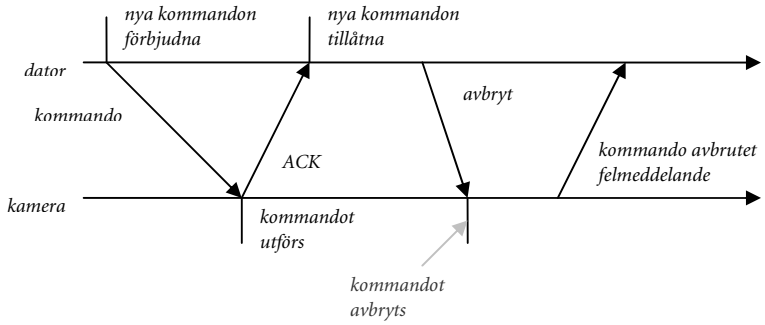
Figur 65

Om ett felmeddelande kommer under kommandots utförande, påverkas inte output-tråden. Input-tråden känner inte till sammanhanget, utan sätter errorEvent trots att den fått ACK på förra kommandot. Emellertid väntar inte output-tråden på något event eftersom den är i IDLE tillståndet. Vad händer om ett annat kommando körs enligt Figur 65? Även om errorEvent är satt, så kommer output-tråden att i första hand vänta på ACK, och sen på felmeddelande. När timeout-tiden gått kommer errorEvent att vara satt (två gånger), och output-tråden får fortsätta.



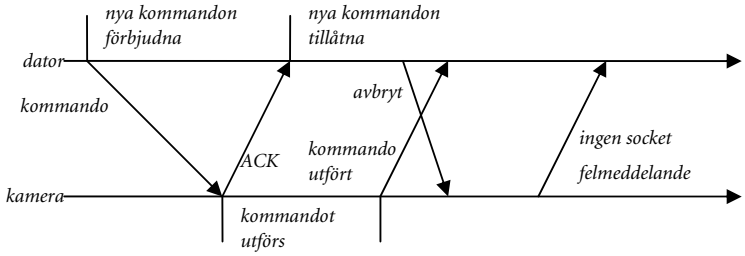
Figur 66

Två kommandon kan skickas utan problem. Men om man skickar ett tredje, innan de förra är färdiga, så får man felmeddelandet ”command buffer full” eller ”no sockets”. Detta problem hanteras inte alls av den nuvarande implementationen, vilket kan behöva åtgärdas i en senare version.



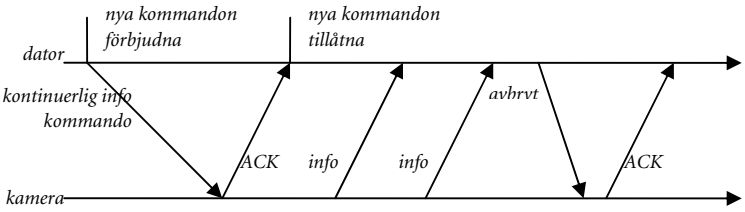
Figur 67

Om ett kommando avbryts under sin exekvering av ”cancel” kommandot, så väntar output-tråden på ackEvent, får inget ACK inom timeout-tiden och upptäcker då via errorEvent att den fått felmeddelandet ”kommando avbrutet”. Att avbryta kommandon är alltså ingen bra idé, för det blockerar output-tråden i ett antal millisekunder som är lika långt som timeout-tiden.



Figur 68

Detta scenario behandlas på samma sätt som i Figur 65. Output-tråden väntar på ACK tills timeout-tiden gått, får det inte, och går vidare när den upptäcker att den fått ett fel. Detta gör förstås input-tråden långsammare.



Figur 69

Detta scenario beskrivs i (Sony Corporation, 1999), men författarna har inte i denna källa sett något kommando som utför detta beteende. Alla frågekommandon som listas i (Sony Corporation, 1999) returnerar svar på frågan omedelbart, och bara en gång. Det verkar som om man måste implementera kontinuerliga frågekommandon i mjukvara genom att skicka samma fråga upprepade gånger i en av trådarna. Det krävs också någon mekanism för att den som ställde frågan skall kunna erhålla svaret kontinuerligt. Detta beteende finns

dock inte implementerat i den nuvarande versionen av kameraklassen.

Referenser

1. Blom, G. (1984). *Sannolikhetsteori med tillämpningar*, Lund.
2. Blom, G. & Holmquist, B. (1998). *Statistikteori med tillämpningar*, Lund.
3. Breidegard, B, Jönsson, B (1999). *En Minimeter till Emma*. URL: www.certec.lth.se/minimetern.
4. Breidegard, B (2001). *The Minimeter Maxi for People with Disabilities – A General Interface for Computer Control and Feedback*. ÖGAI-Journal, Jahrgang 2001, Nr. 2 (Österreichische Gesellschaft für Artificial Intelligence).
5. Breidegard, B (2002). *Minimetern*. Tidskriften Medikament, Nr 8, sid 100-103, 2002.
6. Graham, D. B. *The UMIST Face Database* (1998). University of Manchester Institute of Science and Technology. URL: http://images.ee.umist.ac.uk/danny/database.html
7. Graham, D. B. och Allinson, N. M. (1998). *Characterizing Virtual Eigensignatures for General Purpose Face Recognition*. Wechsler, H., Phillips, P. J., Bruce, V., Fogelman-Soulie, F. & Huang, T. S. (eds.) *Face Recognition: From Theory to Applications, NATO ASI Series F, Computer and Systems Sciences*, Vol. 163., sid 446-456, 1998.
8. Johnson, R. A. & Wichern, D. W. (1992). *Applied Multivariate Statistical Analysis*. Prentice-Hall. ISBN 0 13 041807 2.
9. Jönsson, B., Philipson, L., Svensk, A. (1998). *Vad vi lärt oss av Isaac*. Certec, LTH, ISSN 1101-9956. URL: www.certec.lth.se/dok/vadvi/index.html.
10. Lindgren, F. *Image Modelling and Estimation, A Statistical Approach* (2002). Kan erhållas från avdelningen för matematisk statistik vid matematikcentrum LU/LTH, Lund.
11. Sony Corporation (1999). *EVI-D30/D31 Command List (Ver 1.21)*. URL: http://bssc.sel.sony.com/Professional/docs/manuals/evid30commandlist1-21.pdf.
12. Sparr, G. (1995). *Linjär algebra.* Studentlitteratur.
13. Stiefelhagen, R., Yang, J. & Waibel, A (1997). *ISL Eye Gaze Tracking*. Karlsruhe Universität. URL: http://www.is.cs.cmu.edu/mie/eyegaze.html.
14. Titterington, D. M., Smith, A. F. M. & Makov, U. E (1985). *Statistical Analysis of Finite Mixture Distributions*. John Wiley & Sons. ISBN 0 471 90763 4.

Certecs rapporter

Ett urval av rapporter från CERTEC

Lindholm, A
Haptisk skattjakt
Examensarbete, Certecrapport 2:2003
http://www.certec.lth.se/dok/haptiskskattjakt/

Sjöström, C
Non-Visual Haptic Interaction Design
Doktorsavhandling, Certec 2:2002
http://www.certec.lth.se/doc/hapticinteraction/

Mandre, E
Vårdmiljö eller Lärandemiljö?
Om personer med autism inom vuxenpsykiatrin
Doktorsavhandling, Certec 1:2002
http://www.certec.lth.se/dok/franvardmiljotill/

Anderberg, M
Rörelsehinder och Toalettdesign
Examensarbete, Certecrapport 1:2002
http://www.certec.lth.se/dok/rorelsehinderoch/

Gulin, A
Certec-Brandtalaren
Examensarbete, Certecrapport 3:2001
http://www.certec.lth.se/dok/certecbrandtalaren/

Metoder från statistisk bildanalys har använts för att implementera mjukvara för huvudföljning. Indata är videoströmmen från en kamera som kopplas till en persondator. Tillämpningsområdet är hjälpmedel som låter rörelsehindrade personer interagera med en persondator. Lösningen har provats tillsammans med en person med CP-skada.

Den här rapporten hittar du också på internet:
http://www.certec.lth.se/dok/algoritmerforhuvudstyrning/

Certec
Avdelningen för
rehabiliteringsteknik,
Inst för designvetenskaper,
Lunds tekniska högskola

Certec, LTH
Box 118
221 00 Lund

Sölvegatan 26
223 62 Lund

046 222 46 95

046 222 44 31

certec@certec.lth.se

http://www.certec.lth.se

Certec är en forsknings- och utbildningsenhet inom Institutionen för designvetenskaper vid Lunds tekniska högskola. Främst via Internet gör vi en stark satsning på ”Certecmärkt” information.

Vi är ca 20 anställda och har en årsomsättning på omkring 10 miljoner kronor. Basanslag kommer huvudsakligen från Region Skåne och LTH. Projektmedel får vi t ex från KK-stiftelsen och ett flertal andra bidragsgivare.

EXAMENSARBETE CERTEC, LTH NUMMER 3:2003
ISRN CERTEC-ER-03/3-SE

JUNI 2003

Mikael Aili, Arvid Nilsson

Algoritmer för huvudstyrning av dator