Forord

BIRI - Biometric Intelligent computeR Interface, ett skritt på veien mot et intelligent interaktivt multimedia-system!

BIRI er et prosjekt delt inn i tre mindre prosjekter med doktor-stipendiat Erik Hjelmås i spissen. Disse går på ansiktssporing, ansiktsgjenkjenning og ansiktsanimasjon.

Vi skal i dette prosjektet ta for oss ansiktssporingen og gjøre dette ved hjelp av flere typer algoritmer. Denne ansiktssporingen tar utgangspunkt i bilder fra et webcamera og skal etter prosessering vite om og eventuelt hvor det finnes ansikter i bildet. Dette skal skje nærmest mulig sanntid.

Etter ansiktssporingen kan man velge å gjenkjenne gjeldende ansikt, eller eventuelt legge dette inn i en database (dette er prosjekt 2 i BIRI, ansiktsgjenkjenning).

Vi vil gjerne takke følgende personer:

- Ivar Farup - vår veileder, for et meget godt samarbeid, mye god hjelp i forbindelse med programmeringen og utarbeidingen av algoritmene.
- Erik Hjelmås - vår oppdragsgiver, for et meget godt samarbeid og den gode hjelpen med problemer rundt Linux.

Prosjektet er gjennomført av:

Tom Audun Seljeflot Anders Nygård

Innhold

Figurer

Tabeller

Kapittel 1

Innledning

1.1 Oppgaver og avgrensninger

1.1.1 Generelt

BIRI er et forsøk på å utvikle et intelligent interaktivt multimediasystem hvor brukeren kan kommunisere med datamaskinen mest mulig gjennom lyd og bilde, isteden for via mus og tastatur. De langsiktige målene med BIRI er:

- Demonstrere intelligente egenskaper ved en datamaskin.
- Et fri programvare alternativ for et biometrisk system.
- Implementere "state of the art" innen ansiktsdeteksjon og sporing, ansiktsgjenkjenning, talegjenkjenning, ansiktsanimasjon og talegenerering.

BIRI ble opprinnelig startet i 1997 under navnet ICI (Intelligent Computer Interface) [4], men dette akronymet er nå opptatt av andre, derav lever prosjektet videre under nytt navn. ICI ble født under AI-kurset høsten 1997 og gjennomført som et studentprosjekt av Henry Johansen og Christel Beate Lerøy våren 1998. Det ble skrevet og publisert en rapport, samt laget en demo-video av bevegelsesdeteksjon og ansiktsdeteksjon i ICI. Prosjektet føres nå videre i form av tre BIRI prosjekter. Disse er delt inn i henholdsvis :

- Ansiktsdeteksjon/-sporing.
- Ansiktsgjenkjenning.
- Ansiktsanimasjon og talegenerering.

Det finnes hittil ingen åpen kildekode tilgjengelig innenfor ansiktssporing. Behovet vil i fremtiden øke innen dette området. Spesielt siden teknologien begynner å bli så rask at den takler sanntidsvideo. Ansiktssporing/-deteksjon er nødvendige det første steg for å gjøre ansiktsgjenkjenning (BIRI 2). Eksempler på kommende behov:

- "Intelligent" GUI, bruker lyd og bilde i stedet for tastatur og mus (funksjonshemmede).
- Indeksering av video sanntid.
- Dikteringssystemer.
- Spill/Underholdningsystem for brukeren.
- Styring av kamera (for eksempel holde fokus på et ansikt, selv om det beveger seg i bildet).
- Adgangskontroll basert på ansiktsgjenkjenning.
- Politiarbeid - finne personer som ligner.

1.1.2 Vår oppgave med avgrensninger

Oppgaven er å lage en fungerende ansiktssporer applikasjon. Applikasjonen skal kunne spore ansikter ved hjelp av to typer algoritmer. Det skal være lett å utvide applikasjonen med flere algoritmer og funksjoner. Den skal kunne syes/utvikles sammen med BIRI 2 og BIRI 3.

Algoritmene skal helst være så raske at ansikts deteksjonen skjer nærmest mulig sanntid. Det vil si det ikke bør gå noe særlig tregere enn ett bilde i sekundet. Brukeren skal kunne velge hvilken algoritme han/hun vil bruke ved hjelp av menyvalg.

Vi har satt som begrensning at ansikter i profil ikke vil bli implementert på dette tidspunkt. Ansiktet må være så nærme kamera at dette utgjør minimum 40×40 piksler. Systemet skal kunne fungere på maskiner med Linux. Kjernen må være av versjon 2.4.0 eller nyere. Dette fordi video4linux først implementeres i denne versjonen. Designet er mer eller mindre gitt og blir prioritert ned. Det er hastigheten på algoritmene og hvordan de virker som blir prioritert.

1.2 Målgruppe

Siden vi utvikler applikasjonen i åpen kildekode er målgruppen alle med interesse for ansiktssporing. Applikasjonen med kildekode vil bli lagt ut på internett tilgjengelig for alle. Spesielt vil oppdragsgiver være en pådriver til videre utvikling da dette er en del av hans doktor oppgave. Rapporten er skrevet for personer med gode kunnskaper innen data og programmering.

1.3 Formålet med oppgaven

For vår egen del valgte vi denne oppgaven fordi det var mye ukjent ved den. Begge prosjektdeltakerne hadde liten kjennskap til Linux og miljøet rundt. Vi hadde begge lyst til å lære oss linux miljøet bedre. Dessuten var mange av problemstillingene ved prosjektet interessant. Med både programmering mot

GUI i C++, koblingen mellom webcamera og en applikasjon, samt koblingen med algoritmer som opererer på bilder. Det er også moro å holde på med noe som ingen andre har gjort før deg. Sist men ikke minst er det jo litt interessant at et program, med en viss sikkerhet, kan finne ansikter i bilder. Valget av et emne som ingen av deltakerne hadde særlig forkunnskap om kan føre til tidsmangel og eventuelt et litt dårligere resultat, men deltakerne ment dette ville oppveies av mye større utbytte i form av økt kunnskap. Vi mener det er liten vits i å jobbe et halvt år med noe man kan fra før, bare for å få et bedre resultat. Det er tross alt et skole-prosjekt, og meningen er jo å lære mest mulig!

1.4 Egen bakgrunn og kompetanse

Gjennom studiene har vi fått kunnskap i objektorientert programmering, algoritmer, bildebehandling, bruk av biblioteker og noe kjennskap til linux. Siden det skulle programmeres innen C++ var jo dette godt kjent far før, det gjaldt bare å lære seg bruken av biblioteket Qt. Stort sett har vi måtte lære oss alle programmer brukt under hele prosjektet. Dette er programmer som Lyx, Latex, Emacs og bruken av cvs. Noe av grunnen til at vi valgte nettopp denne oppgaven var jo også for å lære oss mye av miljøet i Linux.

1.5 Arbeidsformer

Prosjektgruppen har jobbet jevnt og tildels mye under hele prosjektperioden. Siden dette er en oppgave med "få" linjer kode og mye hodebry har vi jobbet mye sammen om problemstillingene. Siden Tom Audun har vært opptatt på torsdager, mens Anders har jobbet med prosjektet, har Tom Audun måttet ta igjen dette siden. Dette har ført til at det også har vært en del individuell jobbing. Denne jobbingen er blitt koordinert via bruken av CVS. Det er også blitt fordelt enkelte arbeidsoppgaver og ansvar etter hvert. Gruppen har hatt jevnlig kontakt med både oppdragsgiver og veileder i form av møter, mail og ICQ.

Denne litt "åpne" arbeidsformen har fungert meget bra for en gruppe med så få deltakere. Det har ikke blitt fulgt noen generell utviklingsmodell, da det er litt vanskelig å følge standardiserte maler for utvikling under et så forskningsvinklet prosjekt. Hvis vi skal nevne noen modell må det bli en mellomting/blanding av fossefallsmodellen, iterativ modell og evolusjonær modell. I startfasen ble hovedsakelig trekkene i fossefallsmodellen fulgt, mens under kodingen/implementeringen ble en kombinasjon av iterativ- og evolusjonær modell brukt.

1.6 Organisering av rapporten

Kapittel 1 er et innledningskapittel. Her beskrives det hva oppgaven går ut på, generelt og med avgrensninger, samt målgruppen for oppgaven, vår faglige

bakgrunn og hvilke arbeidsformer vi har arbeidet under.

Kapittel 2 er resultatet av analysefasen med kravspesifikasjonen. Dette kapittelet inneholder alle krav vi har satt til applikasjonen, noe som gjenspeiler de mål vi har satt oss. Det inneholder også nærmere beskrivelse for hvilket system applikasjonen skal fungere under. Videre er det en generell beskrivelse av hvordan vi tenkte oss implementeringen av de forskjellige algoritmene.

Kapittel 3 er en kort beskrivelse av det vi gjorde under design med GUI og filstrukturer. Det meste av denne delen var gitt fra oppdragsgiver og er derfor nedprioritert av oss.

Kapittel 4 tar for seg implementeringen av systemet. Her beskrives utviklingsverktøyene før vi går mer inn på hvordan algoritmene er blitt implementert. Det brukes flere figurer og kode eksempler for å forklare mye av dette. Videre tar vi for oss sammensyingen av BIRI's tre prosjekter, problemer vi fikk underveis før det beskrives hvordan man siden kan implementere flere algoritmer til applikasjonen.

Kapittel 5 oppsummerer testfasen av prosjektet. Denne delen beskriver kort om testomgivelsene og våre testresultater.

Kapittel 6 er en diskusjon rundt hva vi har oppnådd som resultat, kritikk av oppgaven, videre arbeid og gruppas arbeid.

Konklusjonen av hva vi har oppnådd kontra det vi hadde som målsettingen tar vi for oss i kapittel 7.

I Appendixet vil alle vedleggene være samlet. Først en ordliste med forklaringer, før forprosjektet og gantskjemaet. Alle statusrapportene kommer til slutt.

Kapittel 2

Kravspesifikasjon

2.1 Overordnet krav

Det skal lages en fungerende sanntids ansiktssporer applikasjon. Denne skal motta sanntids video fra et webkamera og kunne operere på denne videoen ved hjelp av to forskjellige sporingsalgoritmer. Den skal også vise video på skjerm med ett eventuelt ansikt innrammet.

2.2 Krav til systemet

- Applikasjonen skal kunne fungere med minst to forskjellige ansiktssporer algoritmer.
- Algoritmene skal velges og konfigureres ved hjelp av menyvalg av brukeren.
- Algoritmene skal kunne fungere i samme applikasjon som Biri 2 og 3.
- Det skal være lett å utvide applikasjonen med flere algoritmer.
- Algoritmene skal kunne fungere i sanntid.
- Applikasjonen/programvaren skal utvikles i åpen kildekode.
- Applikasjonen skal være mest mulig innvariant over for lysforhold.
- Applikasjonen skal utvikles ved hjelp av C++ med biblioteket Qt.
- Det skal sendes et utklipp av ansiktet videre til Biri 2.
- Prosjektet skal være ferdig innen 23. mai 2001 klokken 1200.

2.3 Bruker beskrivelse

2.3.1 Systemets omgivelser

Systemet skal kunne brukes på alle typer PC'er med Linux som operativsystem. Det må være kompatibelt med video4linux standarden og koblet til egnet kamera. Det mest normale for en PC er å stå innendørs. Det er derfor også naturlig at det meste av testingen vil foregå der. Systemet vil derimot bli testet under flere forskjellige lyssettinger og omgivelser.

2.3.2 Systemets brukere

Siden applikasjonen blir utviklet i åpen kildekode vil den bli lagt ut på internett, tilgjengelig for alle. Alle med denne type interesse er da hjertelig velkommen til å bruke applikasjonen.

2.3.3 Livssyklus

Applikasjonen vil bli utviklet ved hjelp av CVS (Concurrent Version System) slik at alle utvidelser/oppdateringer på systemet skal være lett å holde kontroll over. Vi skal igjennom to implementeringsfaser der det implementeres en algoritme hver gang. I etterkant av denne fasen vil applikasjonen/algoritmen bli testet grundig før den blir offentliggjort. Det vil være en dokumenteringsfase som går parallelt under hele prosjektet. Applikasjonen blir som nevnt tidligere utviklet i åpen kildekode slik at videreutvikling av det ferdige produktet vil være mulig for hvem som helst med interesse. Det er på grunn av dette viktig at vi som utviklere gjør kildekoden mest mulig oversiktlig. Dette for at det i fremtiden lett skal kunne legges til flere algoritmer til applikasjonen.

2.3.4 Ytelse

Applikasjonen/algoritmene skal kunne kjøres mest mulig i sanntid. Dette forutsetter selvfølgelig en relativt kraftig maskin og at programmeringsspråket de utvikles i er raskt/effektivt. Derfor valgte vi C++ som kompilerer ned til maskinkode. Ytelsen vil også være avgjørende i forhold til valg av algoritmer.

2.3.5 Begrensninger

Vi har satt som begrensning at ansikter i profil ikke vil bli implementert på dette tidspunkt. Ansiktet må være så nærme kamera at dette utgjør minimum 40×40 piksler. Systemet skal kunne fungere på maskiner med Linux. Kjernen må være av versjon 2.4.0 eller nyere. Dette fordi video4linux først implementeres i denne versjonen. Designet er mer eller mindre gitt og blir lavt prioritert. Det er hastigheten på algoritmene og hvordan de virker som blir prioritert.

2.4 Funksjonell spesifikasjon

2.4.1 Klassifisering og preprosessering

En funksjon som skal ta inn en kandidat til ansikt (QImage), fra de nedenstående algoritmene, å skanne bildet for å sjekke om det er noe ansikt der. Returnerer eventuelle koordinater. Også denne skal ha en meny for konfigurering av for eksempel terskel, feilrate og øvre/nedre størrelse på ansikt.

2.4.2 Algoritme 1: Bruteforce

Dette er en algoritmen som er langt i fra sanntid og opererer på hele bildet. Denne tar inn et bilde fra kameraet og skanner over hele bildet ved hjelp av GrayProsessingImage::multiScaleScanForTemplate(). Denne funksjonen kaller klassifiseringsalgoritmen for hvert piksel i bildet, skalerer så ned bildet med en faktor 1.2 og kaller klassifiseringsfunksjonen igjen. Slik fortsetter den inntil bildet er mindre enn 20×20 piksler, da returnerer den en verdi som beskriver hvor likt et ansikt det "beste" området i bildet er og hvor det er.

Denne algoritmen var i utgangspunktet ikke planlagt, men ble likevel implementert. Dette for å vise at det vi jobbet med faktisk var mulig. Det er mye mer motiverende for gruppemedlemmene å se ting i praksis enn bare å lese om det!

2.4.3 Algoritme 2: Bevegelse

Ved menyvalg skal man kunne velge bevegelsesalgoritmen. Denne skal bruke bevegelse til å finne/spore ansikter. I forbindelse med bevegelse er det nødvendig med en type terskling og filtrering slik at kun bevegelser av betydning blir oppdaget. Ved aktivisering av algoritmen skal denne kjøres. Når algoritmen har funnet et mulig objekt sendes dette videre for klassifisering. Dersom det er et ansikt, skal dette sendes videre til Biri2 (gjenkjenning). Det må også være en meny for konfigurering av algoritmen. Her skal det være mulig å sette parametere som terskelverdier osv.

2.4.4 Algoritme 3: Farge

Ved menyvalg skal man kunne velge algoritme 3. Denne har farger den skal arbeide ut ifra. Det er i denne forbindelse viktig å være ute etter hudfarge. Ved funn av mulig objekt skal dette sendes videre til klassifisering/preprosessering. Dersom objektet da viser seg å være et ansikt skal dette sendes videre til Biri 2 (gjenkjenning). Det må også her være en meny for konfigurering av algoritmen (fargeavstander etc.).

Kapittel 3

Design

3.1 GUI

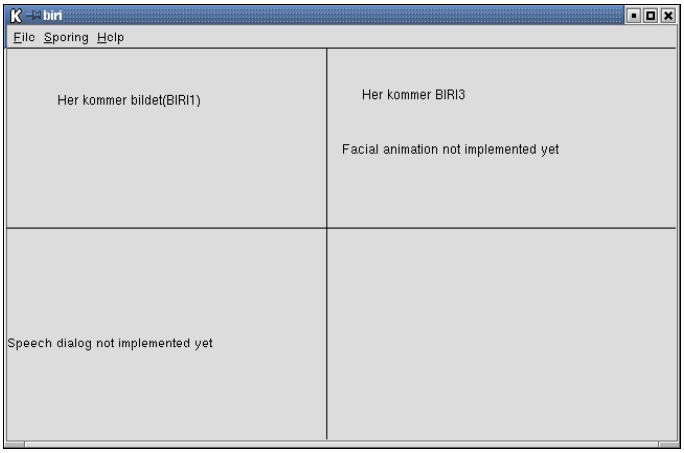
Rammene rundt designet er stort sett blitt gitt av oppdragsgiver og derfor planla vi ikke med mer enn tre dager til design fasen. Denne fasen var planlagt tidlig i prosjektet etter "vanlig" fossefallsmodell. Det viste seg tidlig at denne delen av oppgaven var avhengig av at algoritmen var ferdig implementert da designet i hovedsak skulle ta seg av styringen av parametere til hver av algoritmene. Fasen ble flyttet til vi var ferdig med implementeringen av hver algoritme for da viste vi hvilken parametere vi skulle ha med.

Av oppdragsgiver fikk vi en ferdig hovedapplikasjon (figur 3.1) med inndelte områder til hver av Biri-prosjektene. Her hadde vi allerede fått inndelt område med utskrift av bilde fra kamera.

I møte med oppdragsgiver ble det besluttet at det skulle brukes enkle dialogbokser for konfigurering av valgt algoritme. Det skulle også være en "apply" knapp for aktivering av gitte parametere. Det ble derfor laget menyvalget "Sporing" med undervalgene "Start sporing", "Stop sporing", "Bruteforce..." (figur 3.2), "Bevegelse..." (fig. 3.3), "Farge..." (fig. 3.4) og "Bevegelse og Farger..." (fig. 3.5).

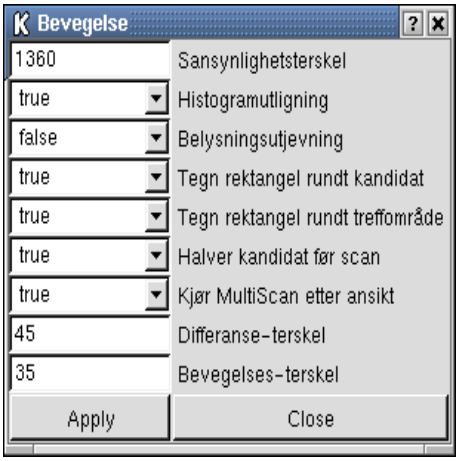
Start sporing er et valg der bruteforce algoritmen starter som "default", hvis ingen andre algoritmer er konfigurert. Med "Stop sporing" kan man stoppe alle typene algoritmer. De andre valgene fører til dialogbokser for hver av algoritmene. Ved valg av "apply" knappen starter gjeldende algoritme med gitte parametere. Dialogboksen vil også forsvinne og prosesserte bilder vil vises i hovedapplikasjonen. Ved valget "close" lukker dialogboksen seg uten å gjøre annet.

For hver av algoritmene markerer disse søkeområde og treff dersom dette er valgt som parametere. Dette markeres med stiplede linjer (svart og hvit) rundt søkeområdet der algoritmen søker, samt rundt eventuelle treff innenfor søkeområdet.



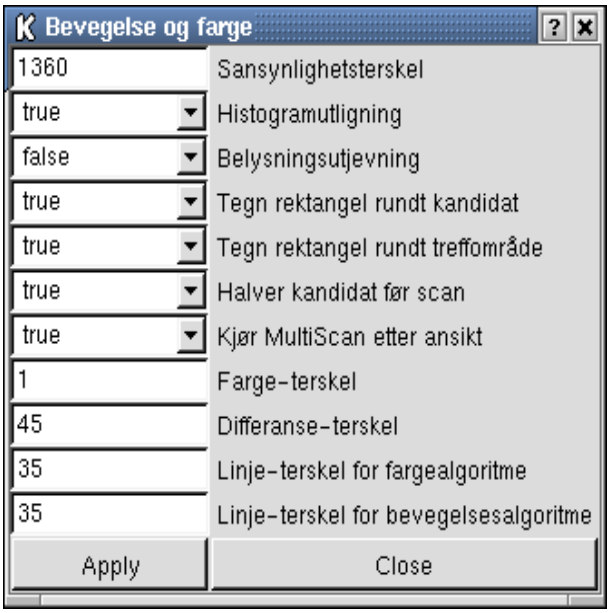
Figur 3.1: Hovedvinduet i Biri applikasjonen.

Figur 3.2: Dialogboks ved valg av "Bruteforce..."



Figur 3.3: Dialogboks for konfigurering ved valg av bevegelsesalgoritme.

Figur 3.4: Dialogboks for konfigurering ved valg av fargealgoritme.



Figur 3.5: Dialogboks ved valg av "Bevegelse og farger...", her vises alle mulige parametere.

3.2 Hva gjør de forskjellige parameterene?

Sannsynlighetsterskel

Denne parameteren beskriver hvor stor avstanden fra template bildet til gjeldende ramme/område kan være for at det skal aksepteres som et ansiktstreff.

Histogramutligning

Denne parameteren er en boolean, true eller false. Dersom man setter denne til true vil det bli kjørt histogramutligning på kandidaten man mottar fra algoritmene, samt på templaten.

Belysningsutjevning

Denne parameteren er en boolean. Dersom man velger denne til true vil det bli foretatt belysningsutjevning på kandidaten man mottar fra algoritmene og på templaten. Dette kan gjøres dersom bildene man mottar i fra kamera er tatt under dårlige lysforhold (spesielt skjeve). Man prøver da å utligne lysvariansen i bildet så langt det lar seg gjøre. Dette vil også føre til at svært mørke eller lyse bilder også blir bedre.

Tegn rektangel rundt kandidat

Denne parameteren er en boolean. Ved valg av true vil det bli tegnet opp et rektangel rundt kandidaten som blir sendt med til denne funksjonen fra hver av algoritmene. Linjen som blir tegner opp inneholder annenhver 3 svarte og 3 hvite piksler.

Tegn rektangel rundt treffområde

Denne parameteren er en boolean. Ved valg av true vil det bli tegnet opp et rektangel rundt treffområde som blir sendt med til denne funksjonen fra L2Metric. Linjene som blir tegnet opp inneholder også her annenhver 3 svarte og 3 hvite piksler.

Halver kandidat før scan

Denne parameteren er en boolean. Dersom true blir valgt, halverer man kandidatområdet for at sammenlikningen mot templaten skal gå fortere. Dette er det samme som om templaten skulle vært dobbelt så stor. Man antar da at det ikke finnes så små ansikter som 20×20.

Kjør MultiScan etter ansikt

Denne parameteren er en boolean. Dersom denne er false vil ikke multiScaleScanForTemplate eller L2Metric bli kjørt. Man vil da altså ikke finne ansikter

i bildet, men bare gjøre selve begrensningen eller ingen begrensning dersom bruteforce er valgt.

Farge-terskel

Her kan man sette hvor stor avstanden d skal være. Dersom denne terskelen er lav vil kun hudfarge etter vår gaussiske modell bli akseptert som hudfarge. Mens dersom den er stor vil flere piksler bli godkjent som hudfarge etter hvor like de er vår modell.

Differanse-terskel

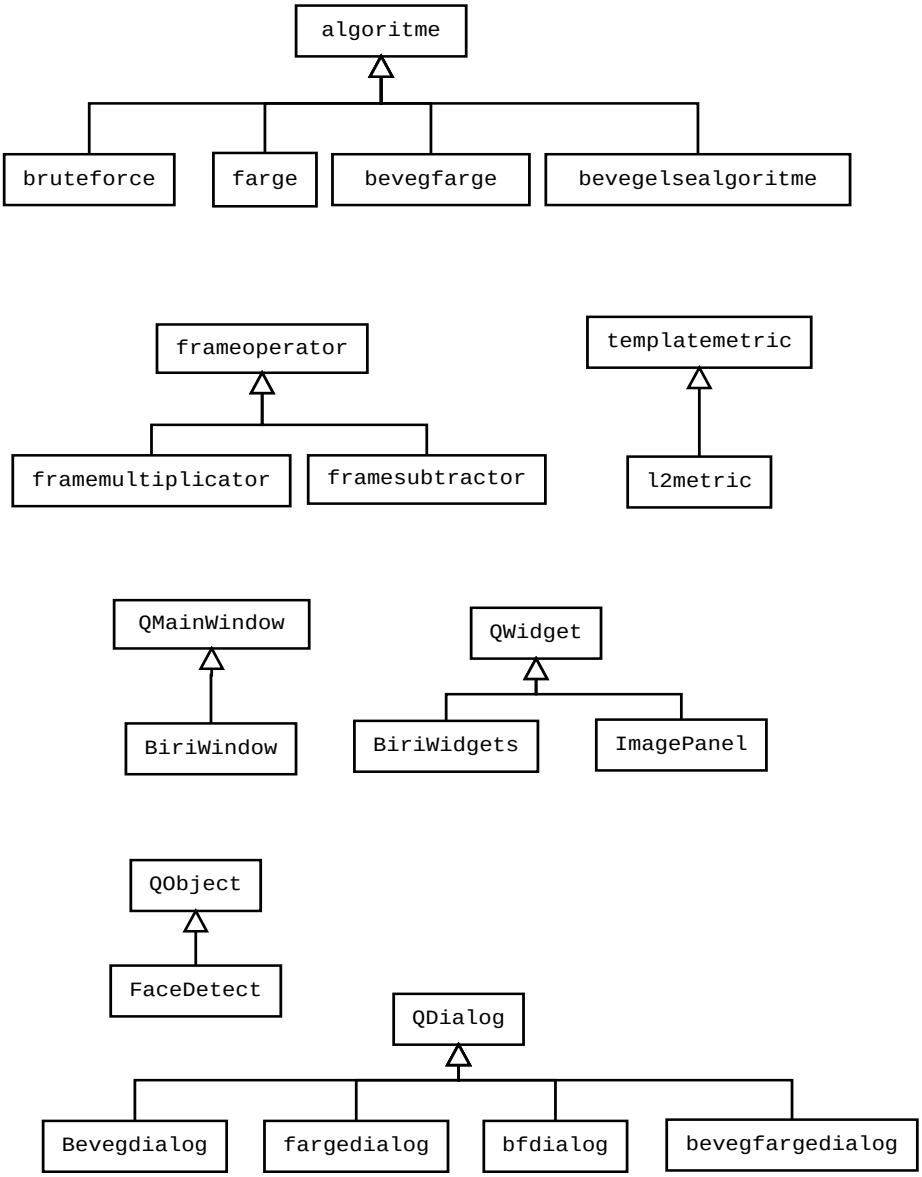
I likhet med farge-terskelen har vi også en liknende terskel for bevegelse. Denne terskelen er for å bestemme hvor stor avstanden mellom gjeldende piksel og tilsvarende piksel i forrige bilde skal være for å bli akseptert som bevegelse.

Linje-terskel for fargealgoritmen og bevegelsesalgoritmen

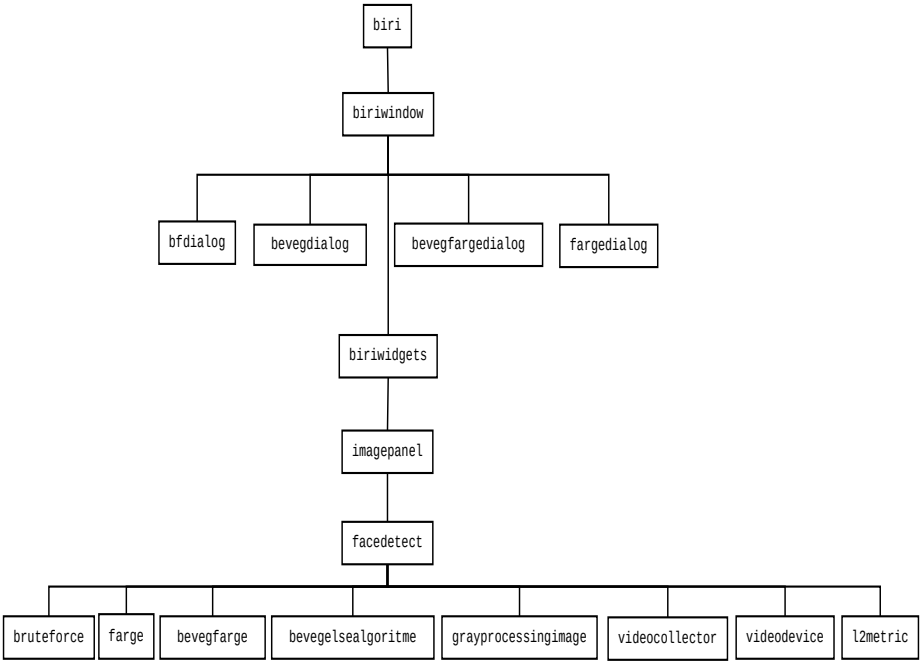
Etter at man har gjort om et bilde til GrayProsessingImage med kun svarte eller hvite piksler ut i fra farge- eller bevegelse- terskelen, teller man opp antall svarte piksler pr. linje. Dersom en linje inneholder flere svarte piksler enn linje-terskelen vil denne linjen markeres som begynnelsen på bevegelsen/fargen. Ut i fra denne markeringen av start på bevegelse/farge sendes dette som et QRect() til funksjonen som tegner opp kandidat for det videre søket av multiScaleScanForTemplate.

3.3 Filstruktur og avhengigheter

Som vist i figur 3.6 arver alle algoritmene (bruteforce, farge, bevegfarge og bevegelsealgoritme) i fra klassen algoritme. Dette er gjort for å lettere kunne implementere nye algoritmer (se kapittel 4). I figur 3.7 vises avhengighetene mellom hver av klassene og hvem som eier hvem. Vi ser her at biriwindow eier alle dialogboksene. I dialogboksene setter man parametere til hver av algoritmene noe som gjør at vi må sende dem gjennom hele widgets hierarkiet da disse parameterene er private.



Figur 3.6: Klasse hierarki



Figur 3.7: Objekt hierarki. Viser hvilke objektutviklingsverktøyer som eies av hvilke.

Kapittel 4

Implementering

4.1 Valg av utviklingsverktøy

En av forutsetningene til dette prosjektet var at det skule utvikles basert på "fri programvare" også kjent som "åpne kildekode". Dette vil si at kildekoden skal være fritt tilgjengelig for alle, og fritt for videreutvikling av andre. Det mest hensiktsmessige operativsystemet ved utvikling av åpen kildekode er som kjent linux og ble derfor valgt. I linux er videobehandling litt spesielt da dette først kommer i sene utgaver av operativsystemet. Dette da under navnet video4linux som kommer første gang i kjerne-2.4.0.

Ett annet krav fra oppdragsgiver var at applikasjonen skulle utvikles ved hjelp av Qt-biblioteket for C++. Dette biblioteket baserer seg på ferdige klasser og objekter spesielt innenfor GUI, som lett kan implementeres. Her finnes det også en bildeklasse som tar hånd om bilder som kommer i fra kamera. Ved valg av programmeringsspråk ble det spesielt lagt vekt på at applikasjonen skulle være mest mulig i stand til å kjøres i sanntid. C++ er godt egnet til dette siden språket kompileres ned til maskinkode.

All programmeringen er basert på Objektorientert programmering siden språket, C++ , og særlig Qt-biblioteket er basert på dette.

Kodingen er gjort i Emacs, og på cvs. Rapportskrivingen er gjort i Lyx, også denne har vi brukt cvs på.

4.2 Hvordan hente inn bilde fra kamera?

Her ser vi en av mange fordeler ved å jobbe med åpen kildekode. Oppdragsgiver fant nemlig to ferdige klasser på internett som tok hånd om koblingen mot kamera. Disse to klassene heter CVideoDevice og CVideoCollector, de stammer fra en webkamera-applikasjon som heter "CamStream" [2].

CVideoCollector skanner over maskinen etter tilgjengelige kameraer. Ut i fra denne informasjonen opprettes et CVideoDevice objekt. Herfra blir det sendt et signal hver gang kameraet har et nytt bilde inne i bufferen. Dette signalet er

```
......
vcollect = new CVideoCollector;
video = vcollect->GetVideoDevice(0);
connect(video, SIGNAL(deviceNotify()), this, SLOT(process()));
video->Open(2);
video->EnableRGB(1);
video->SetSize(320,240);
......
```

Tabell 4.1: Slik kobles det mot kameraet.

Figur 4.1: Gjennomsnittsansiktet/Templaten vi sammenlikner opp i mot.

koblet mot processs() - funksjonen i FaceDetector (se tabell 4.1), slik at hver gang et nytt bilde kommer inn fra kameraet blir det behandlet ut fra valgte algoritmer og vist til skjerm.

4.3 Klassifiseringsalgoritme (L2Metric)

Denne algoritmen er gitt fra veileder og oppdragsgiver. Den inneholder i tillegg til selve klassifiseringsfunksjonen også to forprosesseringsrutiner, en for histogramutligning og en for belysningsutjevning. Klassifiseringsfunksjonen sammenligner en ramme på 20×20 piksler med template bildet (figur 4.1). Dette gjøres ved å måle den Euklidske avstanden mellom testrammen og templaten:

$$d = \sum_{i,j} (I_{ij} - T_{ij})^2$$

hvor I_{ij} er pikselverdien til testrammen i punktet (i,j) og T_{ij} er pikselverdien til det samme punktet i templaten [1]. Templaten er et bilde som er gjennomsnittet av et testsett med ca. 2600 bilder.

4.4 multiScaleScanForTemplate()

GrayProcessingImage er en klasse utviklet av veildeder i forbindelse med klassifiseringsalgoritmene. Den arver fra QImage. Når et objekt av denne typen opprettes blir bildet som sendes med konvertert til et gråtonebilde. Denne klassen inneholder blandt annet funksjonen multiScaleScanForTemplate().

Den tar inn et QImage som er et potensiellt område for ansikt (kandidat). Videre kjører den en dobbel løkke hvor den innerste kaller klassifiseringsfunksjonen, L2Metric, for hvert piksel i kandidaten. Den ytre løkka skalerer ned kandidaten med en faktor 1.2 , inntil kandidaten er mindre enn 20×20 piksler. Dette for å lete etter ansikter i forskjellige størrelser.

Det blir til enhver tid lagret to verdier, en for å beskrive hvor god likheten mellom gjennomsnitts ansiktet og gjeldende ramme er, og en for hvor i bildet rammen befinner seg. De to verdiene som lagres tilhører hele tiden den rammen som har best likhet, dette paret returneres når funksjonen er ferdig.

Antall sammenlikninger algoritmen gjør før reskalering er ca.:

$$(320 \times 240) \times (20 \times 20) = 34560000$$

Dette kan effektiviseres drastisk dersom det blir gjort reskalering på kandidaten først. Hvis man for eksempel reskalerer kandidaten til det halve, vil det indirekte si at man øker den nedre grensen for ansiktsstørrelse fra 20×20 til 40×40 piksler. Dette vil føre til vel fire ganger færre sammenlikninger bare ved første skanning. (Før første reskalering i løkka blir gjort).

4.5 Bruteforce algoritmen - ingen begrensning.

Denne "algoritmen" har ingen stor oppgave. Den tar inn et bilde fra kameraet, kjører så GrayProcessingImage::multiScaleScanForTemplate() på dette bildet.

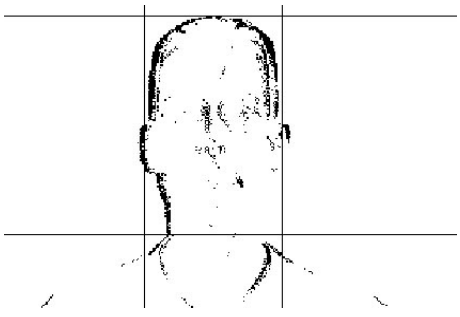
4.6 Hvordan detektere bevegelse i bilder?

Programmet tar inn to bilder i rekkefølge fra kameraet og kopierer disse inn i midlertidige variable. Vi har da "bilde1" og "bilde2". For å finne bevegelse i bildet tar vi ("blide1" - "bilde2") for å se på forskjellen mellom disse. De to bildene lagres som ett GrayProsessingImage for å konvertere til gråtone bilde.

Der det har vært endringer fra "bilde1" til "bilde2" vil dette vises som svarte piksler i et nytt GrayProsessingImage "bilde3" (se tabell 4.2). Dersom det ikke er endringer mellom de to bildene vil "bilde3" bli helt hvit. For å kunne bestemme om det er en stor nok bevegelse eller ikke, teller vi opp antall svarte piksler pr. linje (se figur 4.2 og tabell 4.3). Dette gjøres i fra topp mot bunn, fra bunn mot topp, fra venstre mot høyre og i fra høyre mot venstre. Dersom det er flere svarte piksler enn bestemt ut i fra "Linjeterskelen" pr. linje registreres denne linjen som starten på bevegelsen. Da dette gjøres i fra hver side av bildet vil vi sitte igjen med ett rektangel der det er en viss konsentrasjon av bevegelse. Ett utklipp av dette området sendes da med som parameter til multiScaleScanForTemplate-funksjonen.

```
QRect BevegelseAlgoritme::operator()(QImage image1, QImage
image2)
{
GrayProcessingImage tmpimage1 = image1.copy();
GrayProcessingImage tmpimage2 = image2.copy();
GrayProcessingImage image = image1.copy();
uchar *bits = image.bits();
uchar *tmpbits1 = tmpimage1.bits();
uchar *tmpbits2 = tmpimage2.bits();
for(int i=0; i < 320; i++) for(int j=0; j < 240; j++)
  {
  if(abs((int)*tmpbits1-*tmpbits2) >differanseTerskel )
    *bits = 0;
  else
    *bits = 255;
  bits++; tmpbits1++; tmpbits2++;
  }
........
```

Tabell 4.2: Hvordan vi gjør bevegelses deteksjon.



Figur 4.2: Opptelling av svarte piksler pr. linje begrenser søkeområdet for ansiktsdeteksjon.

```
//Ser etter hudfarge/bevegelse fra høyre mot venstre
linje=0;
while(xright>0 && linjeterskel>linje) {
   for(y=0;y<239;y++)
   {
      if(image.pixelIndex(xright,y)==0)
        {
        linje++;
        }
      }
   xright--;
   }
```

Tabell 4.3: Eksempel på hvordan vi finner større konsentrasjoner av bevegelse eller farger, dette gjøres fra alle kanter. (se figur 4.2)

4.7 Hvordan finne hudfarge i bilder?

Etter studier er det funnet ut at hudfarge best kan skilles ut i det normaliserte rg-fargerommet [5]. Dette er fordi man ved bruken av rg farger eliminerer problemer i forhold til lysvariansen i bildene. For å finne rg farger gjøres dette ved:

$$r = \frac{R}{R+G+B} \quad og \quad g = \frac{G}{R+G+B}$$

Vi får da at en farge x i bildet har matrisen :

$$x = \begin{pmatrix} r \\ g \end{pmatrix}$$

For å kunne bestemme om fargen er hudfarge, trenger vi en gaussisk modell å sammenlikne i mot. Dette ble gjort ved at vi samlet en del bilder av ren hudfarge tatt med vårt webcamera. Deretter ble det kjørt et MathLab-script som vi fikk av oppdragsgiver på disse bildene. Av dette scriptet fikk vi da middelverdien og kovariansen :

$$\mu = \begin{pmatrix} \bar{r} \\ \bar{g} \end{pmatrix} \quad og \quad \Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix}$$

Av dette kan vi så finne den gaussiske modell bestemt av middelverdien μ og kovariansen Σ, for dernest å finne avstanden fra hver piksel-farge til denne. Avstanden d finner vi ved:

$$d = \sqrt{\left((x-\mu)^t \, \Sigma^{-1} \, (x-\mu) \right)}$$

For å bestemme om pikselen så har hudfarge eller ikke gjøres dette ved en sammenlikning mellom terskelen "Farge-terskel" og avstanden *d*. Dersom *d* er

Figur 4.3: Ved avgrensning med farge og treff områder.

mindre enn denne terskelen blir pikselen godkjent som hudfarge (se kode tabell 4.4).

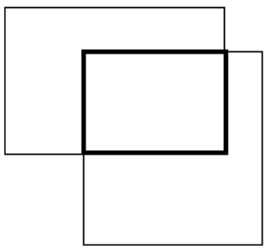
Piksler som blir godkjent som hudfarge settes lik svart (får verdien 0), mens hvit (verdien 255) viss ikke. For å finne større områder med hudfarge der det kan være aktuelt å søke etter ansikt begrenses dette likt som for bevegelse. Her teller vi da opp antall svarte piksler pr linje (se figur 4.2). Dette gjøres i fra topp mot bunn, fra bunn mot topp, fra venstre mot høyre og i fra høyre mot venstre. Dersom det er flere svarte piksler enn bestemt ut i fra "Linjeterskelen" pr linje registreres denne linjen som starten på bevegelsen. Da dette gjøres i fra hver side av bildet vil vi sitte igjen med ett rektangel der det er en viss konsentrasjon av hudfarge (figur 4.3 og tabell 4.3). Ett utklipp av dette området sendes da med som parameter til klassifiserings funksjonen.

4.8 Hvordan kombinere bevegelse-/farge- deteksjon?

Kombinasjonen av bevegelse- og fargedeteksjon er implementert slik at det må finnes både bevegelse og hudfarge for at noe skal returneres. Den tar inn to bilder og kjører fargedeteksjon på dem. Hvis den ikke detekterer hudfarge i det heletatt blir det returnert et QRect(0,0,0,0) (et rektangel med øverste venstre hjørne i punktet (0,0) og bredde 0 og høyde 0). Om den derimot detekterer hudfarge blir det også kjørt en bevegelsesdeteksjon på de to bildene. På samme måte som med farge så blir det returnert et QRect(0,0,0,0) dersom det ikke detekteres bevegelse. Blir det detektert både bevegelse og hudfarge blir det returnert et QRect som er snittet av farge og bevegelse (se figur 4.4)

```
QRect Farge::operator()(QImage image1, QImage image2)
{
GrayProcessingImage image = image1.copy();
for(int i=0; i < 320; i++) //Går gjennom alle pixlene i bildet
  for(int j=1; j < 240; j++)
  {
  QColor px = image1.pixel(i,j);
  green = px.green();
  red = px.red();
  blue = px.blue();
  rred = (red/(red+green+blue)); //Regner om til normalisete rg
farger
  ggreen = (green/(red+green+blue));
  //Finner avstanden d fra (i,j) til gaussisk modell
  // bestemt av middelverdien og kovariansematrisa
  d=sqrt((((rred-middeldir)*(dbb*(rred-middeldir)
-dab*(ggreen-middeldig)))+((ggreen-middeldig)*
(-dba*(rred-middeldir)+daa*(ggreen-middeldig))))*
(1/(daa*dbb)-(dba*dab)));
  if(d < fargeterskel)
    {
    //dersom avstanden er mindre enn satt i fargedialog
    image.setPixel(i,j,0); //setter pikel til svart
    }
    else
      {
      image.setPixel(i,j,255); // eller setter pixel til hvit
      }
    }
.........
```

Tabell 4.4: Hvordan vi gjør farge deteksjonen.



Figur 4.4: Ved bevegelse- og fargedeteksjon taes snittet av de to.

4.9 Implementering av tråder

For å kunne aksessere menyer og annen GUI samtidig som sporingen kjører, var det nødvendig å implementere QThread klassen.

FaceDetect-klassen arver fra QThread. Man må derfor definere den abstrakte virtuelle funksjonen run() i FaceDetect. Denne funksjonen funksjonen kjører all prosesseringen av bildene. Dette er implementert ved at den går i en uendelig løkke, nederst i løkka venter tråden på et nytt bilde. Når dette bildet er klart i kameraet blir det sendt et signal til process()-sloten. Denne vekker da opp igjen tråden. Slik fungerer det for hvert bilde.

4.10 Sammensyingen av Biri 1, 2 og 3

Siden det kun var biri1 og biri2 som var ferdige med sine deler til den planlagte sammensyingen i slutten av prosjektet, ble det til at kun disse ble syd sammen. Vår oppgave var jo i Biri som helhet å spore ansikter i sanntid for siden å sende dette utklippet videre til biri2. Biri 2 skulle så gjenkjenne dette utklippet med sin applikasjon.

Biri 2 hadde laget en "frittstående" applikasjon som ikke var avhengig av Biri-hovedvinduet. De lastet inn bilde som skulle gjenkjennes i fra fil slik at de i utgangspunktet ikke skulle være avhengige av oss dersom noe skulle gå galt.

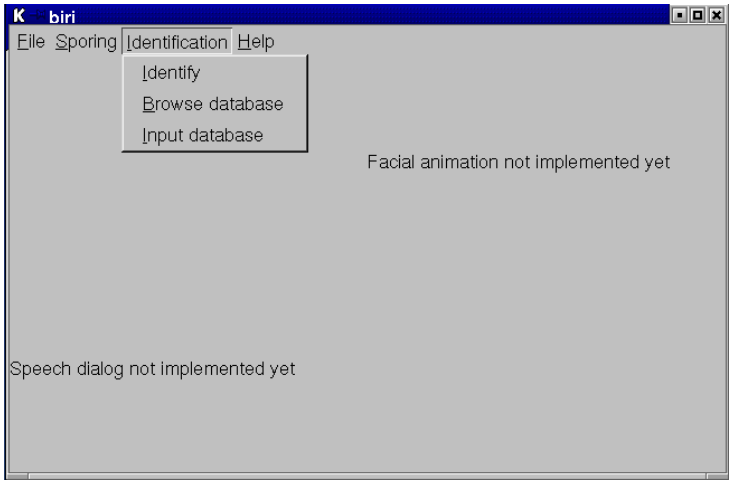
Ved sammensyingen av disse to applikasjonene lagde vi i fellesskap et nytt menyvalg i Biri-hovedvinduet som startet biri2's applikasjon (fig. 4.5). Dette menyvalget, "Identification ", fikk undervalgene "Identify", "Browse database" og "Input database". Ved valg av "Identify" blir treffområdet direkte sendt videre til deres hovedapplikasjon, der man har muligheten til å gjenkjenne dette opp mot deres database. Når man velger "Browse database" får man muligheten til å utforske hele databasen. "Input database" legger inn siste treffområde inn i databasen.

4.11 Problemer underveis

Begge gruppedeltakerne keventuellejøpte inn private USB webkameraer til bruk under prosjektet. Det viste seg at det var en feil med usb driverne i 2.4.0 kjernen, noe som gjorde at vi brukte mye tid til å få utviklingsmiljøet til å fungere skikkelig. Løsningen ble kjerne 2.4.1 da det i denne versjonen var fikset. Det er også brukt mere tid på å lære selve operativsystemet enn det vi trodde for å kunne installere programvaren vi bruker.

Siden vi ikke kunne noen av de programmene vi skulle bruke har dette også tatt ekstra med tid. Programmer som Emacs, Lyx, Latex, KDE2, linux, CVS, osv. måtte læres før vi kunne begynne.

Under hele prosjektperioden har QPainter klassen som følger med i Qt biblioteket vært et problem. Vi får ikke denne klassen til å tegne opp igjen bildene fullt ut. Den tegner bare opp igjen deler av bildene.



Figur 4.5: Slik er menyvalgene etter sammensyingen.

Tråder har også vært et problem vi har brukt mye tid på i siste del av prosjektperioden. Det er helt nødvendig å få prosesseringa som skjer i multiScaleScanForTemplate() inn i en egen tråd, da det ellers vil være meget vanskelig å aksessere menyer osv. Vi fikk til tråder helt på slutten av perioden, etter at vi egentlig hadde tatt en beslutning om å gå over kun til rapporten.

4.12 Implementering en nye/andre algoritmer

Som det går frem av kravspesifikasjonen, var det viktig at vi bygde opp koden så modulær som mulig. Dette for at det skal være lett å jobbe videre med den, da den er åpen kildekode.

Vi har oppfylt dette kravet meget godt ved å implementere algoritmene ved hjelp av funksjonsklasser. Det vil si at vi opprettet en basis algoritme-klasse som alle sporingsalgoritmer arver fra. Denne klassen inneholder en abstrakt virtuell funksjon som overloader operatoren: (), man kan med andre ord ikke opprette et objekt av denne klassen. Alle klassene som arver fra denne må da definere denne operator funksjonen. Det er inne i denne funksjonen bevegelsesdeteksjonen, fargedeteksjonen eller eventuelle nye algoritmer kodes (tabell 4.4). På denne måten kan en funksjon som tar inn en peker til basisklassen som parameter, like gjerne ta inn en peker til en klasse som arver fra denne.

Skal man utvide applikasjonen med en ny algoritme, må man bare opprette en ny klasse som arver fra basisklassen og og implementere overloadingen av operatoren: (). I tillegg må man legge til et valg for denne i menyen, samt en tilhørende slot som starter den. Mye av dette er stort sett bare å klippe og lime fra tidligere algoritmer. Med andre ord er det stort sett kodingen av den overloadede operatoren som krever tankebruk og tid.

Kapittel 5

Testing

Systemet er utviklet og testet under Linux med kjerne 2.4.1 og KDE 2.0/2.1. Kameraene som har blitt brukt er Terratec TerraCam USB og Philips USB disse er begge fargekameraer. Testingen er foretatt under forskjellige lysforhold både innendørs og noe utendørs for alle algoritmene.

Hver av algoritmene er blitt testet etter hvert som de ble ferdig implementert, slik det også fremgår av Gantskjemaet.

Etter at alle algoritmene og GUI med parameterstyringen var ferdig gjorde vi en bedre testfase for hver av algoritmene. Her var det mye fokus på hastigheten til hver algoritme og med hvilke parametere de forskjellige fungerer best med under "normale" forhold. "Normale" forhold betegner vi som at det står et webcamera ved en datamaskin der en person sitter foran og styrer programmet.

Generelt for alle algoritmene har vi at de går betydelig raskere uten belysningsutjevning (preprosessering 2).

Vi skulle gjerne hatt en uke til for å teste ut algoritmene grundigere. Det viste seg å ta mye mer tid enn forventet å teste dem, da man må prøve veldig mange kombinasjoner av terskler for å få optimale resultater under forskjellige forhold.

5.1 Bruteforce algoritmen

Denne søker i hele bildet etter ansikter i forskjellige størrelser, derfor var mye av fokusen her på antall treff og at det hele fungerte. Denne algoritmen er treg siden den ikke har noen som helst begrensning og det tar ca. 30 sekunder mellom hvert bilde på en 1GHz pc.. Det viste seg allerede her at klassifiseringsalgoritmen gav mye svakere resultater her, enn den gjorde på testsettet med bilder den var utviklet mot. Selv om det finnes et meget lett plassert ansikt i bildet kan den gjerne finne områder på veggen, genseren eller også svær ofte kaffetrakteren i bakgrunnen som gir bedre treff enn ansiktet.

5.2 Bevegelses algoritmen

Denne algoritmen begrenser søkeområdet for multiScaleScanForTemplate og L2Metric til der det er bevegelse i bildet. Det varierer hvor mye tid denne algoritmen bruker siden søkeområdet kan variere etter hvor mye bevegelse det er i bildet. Dersom det er bevegelse i hele, eller bevegelse i ytterkantene av bildet, vil denne bruke like lang tid som Bruteforce. Noe av bakgrunns "bevegelsene" som forårsakes av støy i kameraet kan ignoreres ved justering av differanseterskelen. På samme måte kan man ignorere en del små bakgrunnsbevegelser ved å justere linjeterskelen. Det kan ofte være vanskelig å registrere bevegelse i hakepartiet, hvis man ikke bruker en høyhalset genser e.l.

Denne algoritmen fungerer best innendørs hvor det er relativt rolig bakgrunn. Utendørs er det ofte mange ting i bakgrunnen som beveger seg i vinden.

5.3 Farge algoritmen

Denne algoritmen begrenser søkeområdet til der det er hudfarge i bildet. Det varierer hvor mye tid denne algoritmen bruker siden søkeområdet kan variere etter hvor mye hudfarge det er i bildet. Dersom hudfarge dekker hele, eller finnes i alle ytterkantene på bildet, vil også denne bruke like lang tid som Bruteforce.

Vår fargemodell et tatt ut i fra vanlig "norsk" hudfarge med kun variasjoner i lysstyrke. Resultatet av "colorscript" ligger som variable i filen inc.h og kan lett endres. Algoritmen er ikke blitt testet opp mot ansikter fra nasjonaliteter med litt mer enn 3 uker sol i året.

Dersom farger i bildet er noen lunde lik hudfarge kan dette påvirke slik at søkeområdet blir større. For eksempel kan en hudfarget genser bli tatt ut som kandidat. Vi hadde på grupperommet for eksempel litt problemer med at en skillevegg var gammelrosa, dette førte til at den ble med i søkeområdet. Store deler av disse problemene kan nok bli borte hvis man har kameraer med bedre kvalitet enn de helt billigste webkameraene. Da man i de tilfellene kan justere ned fargeterskelen og dermed eliminere objekter som er helt i "ytterkant" av hva som godkjennes som hudfarge.

Alt i alt er nok dette den algoritmen som imponerte mest, den holder på kandidaten svært stabilt og ved normale forhold begrenset den søket nesten til det optimale.

5.4 Bevegelse og farge algoritmen

Denne algoritmen ble det svært liten tid til å teste ut, da den egentlig ikke inngikk i oppgaven. Vi fant allikevel ut at den var rimelig effektiv i mange tilfeller utendørs. Vi testet den i sola med en god bris som førte til mye bevegelse på grunn av blafring i parasollen. Denne blafringen førte til store lysforskjeller da bakgrunnen vekselvis ble skyggelagt og lyslagt. Dette ble eliminert da også hudfarge var nødvendig.

Kapittel 6

Diskusjon

6.1 Resultat

Målet vårt har hele tiden vært å lage en sanntids ansiks-sporer ut i fra de kravene vi satte i Kravspesifikasjonen. Dette mener vi selv å ha greid over vår forventning. Målsettingen med ett bilde pr. sekund er nådd, ved optimale lys- og bakgrunnsforhold. Alle andre krav er også oppfylt. Det er i tillegg til kravene også implementert en algoritme ekstra (kombinasjon farge og bevegelse). Brute-Force algoritmen var egentlig heller ikke planlagt i utgangspunktet.

6.2 Alternativer og valg underveis

Det første vi tok en beslutning på var å utsette implementeringen og planleggingen av GUI. Dette gjorde vi da vi ikke viste hvilke eller hvor mange parametere dialogboksene skulle inneholde. GUI er hele tiden blitt nedprioritert både fra oss og oppdragsgiver. Derfor er det blitt en ganske enkel løsning. For eksempel ville det vært mye bedre å lage en dialogboks som inneholder ark-faner (tabsheets) for hver av dialogene.

Vi så også tidlig at vi hadde et problem med QPainter klassen. Det ble brukt noen dager i et desperat forsøk på å løse dette problemet. Da dette mislyktes tok vi en beslutning om å "overse" det.

Det ble litt for lite rapportskriving i fra starten, noe som vi har angret på siden. Dette har ført til litt lange dager den siste tiden.

6.3 Kritikk av oppgaven

Oppdragsgiver hadde en tendens til å komme med nye krav til applikasjonen for hvert møte vi hadde med han. Dette er vel et problem som oftere dukker opp i denne type forskningsprosjekter, hvor "veien blir til mens du går". Bortsett fra dette synes alle prosjektmedlemmene at oppgaven har gått knirkefritt.

6.4 Videre arbeid og nye prosjekter

Som nevnt er oppdragsgiver en pådriver til videreutvikling av denne applikasjonen siden han holder på med sin dr. oppgave innen emnet. Ved videre arbeid har vi laget applikasjonen enkel for å implementere flere algoritmer som begrenser søkeområdet for multiScaleScanForTemplate og L2Metric. Det er også enkelt å legge til nye menyer med parameterstyring til en eventuell ny algoritme.

Et forslag til videre arbeid hadde vært å implementere flere typer klassifiseringsalgoritmer[1] for å se om andre er bedre i enkelte situasjoner. Dette kunne bli gjort slik at man kunne velge hvilken klassifiseringsalgoritme man ville kjøre ut i fra menyvalg.

Det er også høyst aktuelt å implementere flere typer algoritmer for å begrense søkeområdet for klassifiseringsalgoritmene.

6.5 Evaluering av gruppas arbeid

6.5.1 Innledning

Prosjekt deltakerne har lagt ned en betydelig arbeidsmengde i dette prosjektet. Prosjektstart var den 26. januar, men mye av arbeidet med å skaffe seg oversikt over prosjektet var allerede i gang før dette. Vi har hatt mange møter med veileder og oppdragsgiver både planlagte og litt mer uoffisielle da vi har lurt på ting. Vi har jobbet med prosjektet stort sett hver uke, hovedsakelig onsdag til fredag da undervisning har blitt lagt til før i uken. De fleste fagene våre ble avsluttet før påske slik at vi har jobbet hele uker etter det. Vi har prøvd å styre unna helgejobbing.

Siden dette er et pågående forskningsprosjekt, var det vanskelig å beregne tidsbruken korrekt. Mye av arbeidet har gått på å sette seg inn i algoritmer for siden å implementere disse. Noe som kan være både tidkrevende og vanskelig.

Vi har til vår store glede klart å følge Gantskjemaet etter planen, med noen forsinkelser da det var som hektigst i andre fag. Disse forsinkelsene ble siden tatt igjen av iherdig jobbing under og etter påsken.

6.5.2 Organisering

Vi jobbet for det meste sammen i begynnelsen av prosjektet til vi begge begynte å få kontroll over Qt og videre arbeidsoppgaver. Etter hvert er arbeidsoppgaver blitt fordelt etter evner og interesser. Alle beslutninger er blitt tatt i fellesskap, viktigere beslutninger også sammen med oppdragsgiver og/eller veileder.

6.5.3 Fordeling av arbeidet

Siden vi ikke har hatt tid til å jobbe sammen om alt har arbeid blitt fordelt etter evner og interesser. Arbeidet, både koding og rapportskriving, har blitt koordinert ved bruk av CVS.

6.5.4 Prosjekt som arbeidsform

Dette har vært en veldig nyttig å lærerik arbeidsform siden ingen av prosjektdeltakerne har tidligere gjennomført et så stort prosjekt. Vi har kun fått kjennskap til teorien igjennom faget systemutvikling.

Vi har styrt arbeidsmengden selv etter det vi først planla i forprosjektet. Vi følte underveis at vi lå langt etter, noe som vi egentlig aldri gjorde. Det var vel en kombinasjon med mye arbeid innen andre fag som fikk oss til å tro det. Arbeidet ble fort hentet inn igjen etter påske da alle våre fag ble avsluttet.

Siden det er første gangen vi arbeider i prosjekt har vi dratt god nytte av vår veileder og hans hjelp under hele prosjektet.

6.5.5 Subjektiv opplevelse av prosjektet

Vi synes prosjektet har vært særdeles lærerikt på mange måter. For det første har vi lært å kjenne hele miljøet i og rundt Linux. For det andre har vi fått erfare hvordan det er å jobbe i et forskningsprosjekt. Ikke minst har det å jobbe under et større prosjekt vært lærerikt der vi har måttet ta hensyn til andre prosjekter for at det hele skulle kunne syes sammen til slutt.

Selv om det å sette seg inn i nye programmer og et nytt operativsystem har tatt mye tid [3], føler vi at det har gått veldig greit. Vi valgte jo også denne oppgaven for å lære oss disse tingene. Det har også vært moro å jobbe med Qt og bibliotekbruken opp mot C++.

Humøret har til tider vært noe anspent ved jobbingen med QPainter. En liten periode gav også implementeringen av tråder oss noe hodebry.

Kapittel 7

Konklusjon

Prosjektgruppen har utviklet en fungerende ansiktssporingsapplikasjon. Ved riktig valg av algoritmer og parametere fungerer den også i sanntid.

Selve klassifiseringsalgoritmen viser det seg at ikke fungerer riktig så bra i praksis som den gjør på testsettet den ble utviklet mot. Dette gjør at applikasjonen ikke alltid finner ansiktet som er i bildet, eller at den av og til finner andre ting som tydeligvis ligner mer på et ansikt enn selve ansiktet. Vi nevnte tidligere at denne algoritmen er det ikke prosjektgruppa som har utviklet, og kan derfor ikke gjøre noe med det. Hovedtyngden i gruppas oppgave var å utvikle/implementere algoritmer som begrenser søket etter ansikt. Dette er løst nesten så godt som det lar seg gjøre hvis lysforhold og forstyrrelser ikke er alt for vanskelige. Spesielt algoritmen som går på farger er svært effektiv dersom det ikke er for mange forstyrrelser med hudfarge i bakgrunnen.

Resultatet er alt i alt bedre enn hva gruppas medlemmer hadde forventet før prosjektstart. Da ingen av oss hadde spesielt gode forkunnskaper om bildebehandling, og ingen av oss hadde hørt om teorier rund ansiktsdeteksjon.

Tillegg A

Definisjoner av ulike ord og utrykk

Biometrisk system:
: Et system som baserer seg på biometriske egenskaper, det vil si biologiske egenskaper som identifiserer mennesket.

BIRI:
: Biometric Intelligent computeR Interface

Biri1:
: Ansiktssporing/-deteksjon.

Biri2:
: Ansiktsgjenkjenning av bilde sendt fra Biri1.

Biri3:
: Ansiktsanimasjon.

C++:
: Et objektorientert programmeringsspråk.

CVS:
: Concurrent Version System, et program som hjelper deg med å holde orden på versioner av filer.

Kandidat:
: Et område i bildet som har potensiale til å inneholde et ansikt.

Linux :
: Et operativsystem basert på åpen kildekode og Unix.

Qt:
: Et GUI-bibliotek for C++ programmeringsspråket.

Signal:
: En funksjonalitet i Qt-biblioteket som gjør det enkelt å koble sammen diverse hendelser med funksjoner.

Slot:
: En spesiell måte å lage funksjoner på i Qt, forskjellen på disse funksjonene

og vanlige funksjoner i C++ er at sloter kan trigges ved hjelp av signaler.

Template:
: Gjennomsnittsbilde for søking etter ansikt (figur 4.1).

Terskling:
: En satt grense for følsomhet, trigges dersom grensen overskrides.

Video4linux:
: En standard for å motta og behandle bilder i Linux.

Tillegg B

Forprosjekt

B.1 Mål og rammer

B.1.1 Bakgrunn

BIRI er et forsøk på å utvikle et intelligent interaktivt multimediasystem hvor brukeren kan kommunisere med datamaskinen mest mulig gjennom lyd og bilde, isteden for via mus og tastatur. De langsiktige målene med BIRI er:

- Demonstrere intelligente egenskaper ved en datamaskin
- Et fri programvare alternativ for et biometrisk system
- Implementere "state of the art" innen ansiktsdeteksjon og sporing, ansiktsgjenkjenning, talegjenkjenning, ansiktsanimasjon og talegenerering.

BIRI ble opprinnelig startet i 1997 under navnet ICI (Intelligent Computer Interface), men dette akronymet er nå opptatt av andre, derav lever prosjektet videre under nytt navn. ICI ble født under AI-kurset høsten 1997 og gjennomført som et studentprosjekt av Henry Johansen og Christel Beate Lerøy våren 1998. Det ble skrevet og publisert en rapport, samt laget en demovideo av bevegelsesdeteksjon og ansiktsdeteksjon i ICI. Prosjektet føres nå videre i form av tre BIRI prosjekter. Disse er delt inn i henholdsvis :

1. Ansiktsdeteksjon/sporing
2. Ansiktsgjenkjenning
3. Ansiktsanimasjon og talegenerering

Det finnes hittil ingen åpen kildekode tilgjengelig innenfor ansiktssporing. Behovet vil i fremtiden øke innen dette området. Spesielt siden teknologien begynner å bli så rask at den takler sanntidsvideo. Ansiktssporing/deteksjon er nødvendige det første steg for å gjøre ansiktsgjenkjenning (BIRI 2).
Eksempler på kommende behov :

- Intelligent GUI, bruke lyd og bilde i stedet for tastatur og mus (funksjonshemmede).
- Indeksering av video sanntid.
- Dikteringssystemer
- Spill/Underholdningsystem for brukeren
- Styring av kamera (for eksempel holde fokus på et ansikt, selv om det beveger seg i bildet).
- Adgangskontroll basert på ansiktsgjenkjenning.

B.1.2 Overordnet prosjektmål

Å lage to algoritmer for ansiktssporing som skal kunne fungere i samme applikasjon som Biri 2 og 3.

B.1.3 Prosjektmål

Å lage en fungerende sanntids - ansiktssporer applikasjon. Den skal kunne benytte seg av to forskjellige typer algoritmer. Algoritmene skal kunne velges fra menyvalg av brukeren.

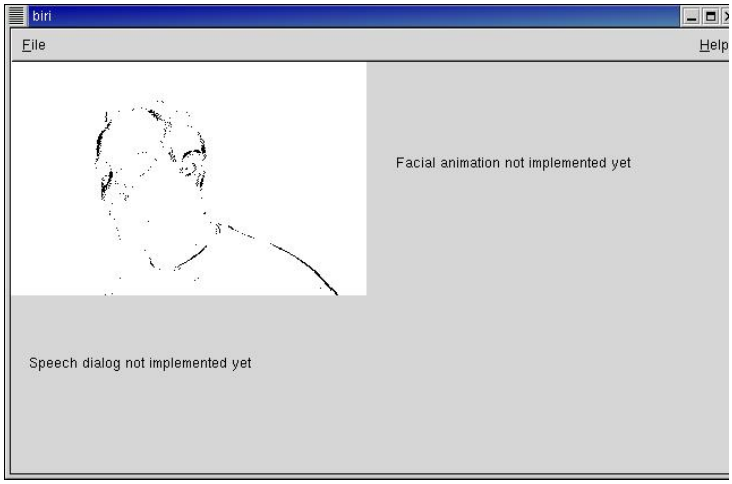
B.1.4 Rammer

Det er investert i en relativt kraftig PC og ett Web-kamera for bruk i alle tre BIRI -prosjektene. Det er også avsatt ett grupperom til disposisjon for de tre prosjektene. Prosjektet skal gjennomføres i perioden 26.01.01 til 23.05.01. (Gruppemedlemmer har også gjort store private investeringer, hovedsakelig på grunn av prosjektet)

B.2 Omfang

B.2.1 Beskrivelse

Oppgaven er å lage en fungerende ansiktssporer applikasjon. Applikasjonen skal kunne spore ansiktet ved hjelp av to typer algoritmer. (For eksempel ved hjelp av bevegelse eller farger). Det skal være lett å utvide applikasjonen med flere algoritmer og funksjoner. Den skal kunne syes/utvikles sammen med BIRI 2 og BIRI 3. Algoritmene skal hels være så raske at ansikts deteksjonen skjer nærmest mulig sanntid. Det vil si at det ikke bør gå noe særlig tregere enn ett bilde i sekundet. Brukeren skal kunne velge hvilken algoritme han/hun vil bruke ved hjelp av menyvalg. (Eksempel på GUI se fig. B.1)



Figur B.1: Eksempel på GUI

B.2.2 Oppgaveomfang og avgrensning

Vi har målsatt oss å finne ut av to forskjellige algoritmer for ansiktssporing/deteksjon. Denne oppgaven er tidligere ikke løst på verdensbasis i åpen kode. Derfor er målsettingen om sanntid ett kritisk punkt for hver av algoritmene. På grunn av prosjektets relativt korte varighet så må vi dessverre ta den forutsetningen at objektet/ansiktet er rett foran kameraet (dvs. ikke bilder i fra siden/profil).

B.2.3 Utviklingsmiljøet

Applikasjonen skal utvikles ved hjelp av programmeringsspråket C++ og Qt biblioteket, i Linux. Vi skal bruke Emacs som kodeeditor. Det er altså åpen kode/GNU - lisensiert. Lyx vil bli brukt som rapporteringsverktøy. Kravspesifiseringen vil bli skrevet i Latex, dette for å få bedre forståelse for funksjonaliteten i tidligere nevnte Lyx. Siden vi skal utnytte Video4Linux må vi kjøre kjerne versjon 2.4.0 da det kun er denne som gir støtte for dette.

B.3 Organisering

B.3.1 Ansvarsforhold

Da vi bare er to prosjektmedlemmer har vi valgt ikke å velge prosjektleder da vi mener dette blir litt vel byråkratisk for en så liten gruppe. Alle avgjørelser av betydning og eventuelt konflikter blir løst i samarbeid innad i gruppen, om nødvendig også med veileders objektive syn. Hver av prosjektdeltakerne har selv ansvaret for å sette seg inn i/lære relevant stoff i samsvar med tidsforløpet av

prosjektet. Dvs. sette seg inn i Qt etter planlagt aktivitet jfr. gantskjema og i tillegg tilegne seg kunnskap rundt aktuelle algoritmer.

Øvrige roller (veileder/oppdragsgiver)

Veileder for prosjektet:
Ivar Farup, tlf. 61 13 52 27
Oppdragsgiver:
Erik Hjelmås, tlf. 61 13 52 20

B.4 Planlegging, oppfølging og rapportering

B.4.1 Hovedinndeling av prosjektet

- Analysefase
- Implementeringsfase
- Testfase
- Sette sammen BIRI fase
- Dokumenteringsfase

Analysefasen består i hovedsak av kravspesifisering og tilegning av teoretisk kunnskap/tidligere forskning innen ansiktsdeteksjon. Ut fra tilegnet kunnskap velges to ulike algoritmer for ansiktsdeteksjon/sporing. Implementeringsfasen innebærer først å fordype seg i algoritmene som er valgt, for så å kode hver av disse (implementere dem i applikasjonen) . Det vil etter dette komme en testfase der algoritmene testes av både gruppen og veileder. Algoritmene/applikasjonen skal da testes i flere miljøer/lyssettinger. Etter algoritmene er ferdig implementert og testet vil det komme en fase der alle tre Biri prosjektene skal syes sammen. Dokumenteringsfasen vil i hovedsak bestå av rapportskriving. Denne fasen varer mer eller mindre under hele prosjektet som dere ser ut i fra gantskjemaet. Det er ikke valgt noen spesiell utviklingsmodell, da prosjektet er nærmere et forskningsprosjekt enn et systemutviklingsprosjekt. Modellen som blir fulgt er en hybrid mellom fossefallsmodellen og prototyping/evolusjonær modell.

B.4.2 Krav til statusmøter

Status møter skal avholdes hver tredje uke for å oppsummere og sikre fremgangen i prosjektet. Det skal skrives statusrapporter før hvert møte. Ved skriving av Statusrapport skal HIG standard brukes. Det skal legges mest vekt på status i forhold til planlagte aktiviteter og fremdriften i prosjektet. Begge deltakerne i prosjektet skal ha anledning til å komme med innspill og erfaringer.

B.4.3 Krav til beslutningspunkt

Etter kravspesifiseringen tidlig i prosjektet, har vi lagt inn et beslutningspunkt der vi tar en endelig avgjørelse angående hvilke to algoritmer vi skal velge. Dette gjøres i fellesskap med veileder og eventuelt oppdragsgiver på et møte.

Det vil bli tatt stilling til bruken av CVS både innad i gruppen og sammen med de andre i et møte like etter prosjektstart. Veileder og oppdragsgiver for alle gruppene vil være til stedet under møtet.

B.5 Kvalitetssikring

B.5.1 Organisering av kvalitetssikring

Det vil være ett tett samarbeid mellom prosjektgruppen, veileder og oppdragsgiver under hele prosjektet. Derfor er det ikke satt av spesielle tidspunkt for veiledermøter da disse blir etter behov. Fortløpende prototyper vil bli testet både av veileder og gruppen for å sikre algoritmenes kvalitet. Det vil i etterkant av implementeringen av hver algoritme være en testfase der algoritmene testes ut i forskjellige miljøer (forskjellige lysretninger/forhold/bakgrunn etc.). Det vil bli tatt sikkerhetskopi av kildekode og dokumentasjon hver uke. Dette vil bli gjort i form av kopiering til CD-R/CD-RW og i tillegg lagt på eget område på loke.hig.no. Hvis det blir gjort store endringer, er det også åpent for å ta sikkerhetskopi oftere. For å holde kontrollen med versjonene vi utvikler, blir CVS (Concurrent Versions System) brukt. Det er hittil ikke tatt noen avgjørelse på om CVS skal brukes i sammen med de to andre gruppene. Dette vil bli avgjort på et møte sammen med Biri 1 og 2 helt i starten av prosjektet. Dersom det blir beslutte å bruke CVS i sammen med de andre må det utarbeides en egnet politikk for bruken av CVS (f.eks ikke lov å legge inn ukompilerbar kode). Etter som vi bruker CVS trengs det ikke utarbeides noe plan for navngiving av filer. CVS vil også brukes for å holde orden på dokumentasjonsversjoner.

Hver av prosjekt deltakerne vil avsette tid tilsvarende 2 hele dager pr. uke for jobbing med prosjektet. Dette vil i hovedsak skje ved siden av studiene på kveldstid og i helger. Det vil bli en opptrapping etter påske da flere fag avsluttes til da. Dersom permisjon skal innvilges må dette avtales minst en uke i forveien og selvfølgelig jobbes inn enten i forkant eller etterkant.

B.5.2 Kvalitetssikring av kritiske suksessfaktorer

Karv til at applikasjonen skal fungere i sanntid er tidligere blitt nevnt som et kritisk punkt. Det er dette som gjør prosjektet til en suksess eller ikke. Det er derfor meget viktig at vi setter oss godt inn i de forskjellige typer algoritmer før de endelige blir valgt. Det vil i denne perioden være nødvendig med gode råd fra veileder. Siden vi i dette prosjektet har meget god tilgang på veileder er han i seg selv en del av kvalitetssikringen. Det vil bli skrevet statusrapporter til ca hver tredje uke under hele prosjektperioden. I etterkant av disse vil det bli

avholdt møter både med veileder og oppdragsgiver, der disse kan komme med tilbakemeldinger.

B.6 Fremdriftsplan

Se Gantskjerma (Tillegg H).

Tillegg C

Statusrapport 1, 19.02.01

C.1 Status

C.1.1 Planlegging/fremdriftsplan

Planlagte oppgaver og bruk av tid i følge Gantskjemaet :
- Kravsprk, 6 dager.
- Valg av Algoritmer, beslutningspunkt.
- Danne utviklingsmiljø, 5 dager.
- Design av applikasjon, 3 dager.
- Sette seg inn i teori rundt Algoritme 1, 13 dager.
- Tilegne seg kunnskap om Qt, 20 dager.

C.1.2 Organisering av ansvarsområder

Arbeidsoppgaver er ikke akkurat blitt fordelt, men falt naturlig etter interesse og evner. Tom A. har jobbet mye med å danne utviklingsmiljøet siden han har erfaring fra dette. Anders har derimot gjort kravspek. Begge har jobbet med kodingen og brukt cvs for å koordinere den foreløpige koden.

C.1.3 Klargjøring av problemstillingen/systemering

Etter møte med veileder 14.02 fikk vi en åpenbaring på hvordan applikasjonen skulle fungere samt hvordan vi egentlig skal kode den. Etter dette møtet ble det implementert en algoritme som søker i hele bildet etter ansikter, såkalt "bruteforce". Denne implementeringen fikk oss til å skjønne mye av tidligere kode og gjør det derfor litt lettere videre.

C.1.4 Rapportskriving

Vi har ennå ikke begynt med hovedrapporten. Dette blir iverksatt en av de nærmeste dagene.

C.2 Totalstatus

Vi har skaffet oss et mye bedre innblikk i det vi skal gjøre fremover, samtidig som vi er godt i gang med jobbingen. Rutinene fungerer godt.

C.3 Problemer/løsninger

De helt store problemene er vi ennå ikke kommet opp i!

C.4 Avsluttede oppgave

- Utviklingsmiljøet er ferdig etablert både på SW og HW siden.
- Kravspesifikasjonen er ferdig nedskrevet.

C.5 Oppgaver under arbeid

Implementeringen av "Bruteforce" algoritmen er ferdig, men trenger en hel del forbedringer og utprøvinger slik at vi når best resultat. Samtidig med denne implementeringen tilegner vi oss kunnskap om Qt og den første algoritmen etter planen.

C.6 Tidsfrister/overskridelser

Tidsfristene er stort sett blitt overholdt. Møtet angående valg av algoritmer er derimot ikke blitt avholdt (1,5 uke senere). Det er en allmenn forståelse for at en algoritme for bevegelsesdeteksjon blir implementert samt en for deteksjon ved hjelp av farger.

Det er ikke gjort noen vurdering på designsiden da det foreligger et ferdig design for hovedapplikasjonen. Det designet vi må ta standpunkt til baserer seg på valg av algoritmer og parametere. Disse har vi på dette tidspunkt ikke oversikt over og har derfor valgt å utsette dette designet.

C.7 Motivasjon

Motivasjonen er helt på topp og kunne ikke vært bedre.

C.8 Veileders rolle

Samarbeid med veileder og oppdragsgiver fungerer ut fra vårt synspunkt meget bra.

Tillegg D

Statusrapport 2, 19.03.01

D.1 Status

D.1.1 Planlegging/fremdriftsplan

Planlagte oppgaver og bruk av tid i følge Gantskjemaet :
- Kode algoritme 1, 16 dager.
- Testing av algoritme 1, 3 dager.

D.1.2 Organisering av ansvarsområder

På grunn av gruppens størrelse har vi litt flytende ansvarsområder. Ansvar blir fordelt ettersom det trengs.

D.1.3 Klargjøring av problemstillingen/systemering

Det skal en av de nærmeste dagene avtales møte med oppdragsgiver og veileder. Her skal det gjennomgås videre detaljert beskrivelse av algoritme 1. Trolig bevegelsesdeteksjon.

D.1.4 Rapportskriving

Vi har begynt med rapporten.

D.2 Totalstatus

Vi ligger for tiden litt bak skjema, dette på grunn av store prosjekter i andre fag som har tatt mer tid enn forventet.

D.3 Problemer/løsninger

Vi har store problemer med å få QPainter klassen til å tegne opp bildet riktig. Vi har derfor bestemt oss for å gå videre og se på dette senere ved ledige stunder. Uten denne beslutningen er vi redd det vil bli lite tid til resten av oppgaven.

D.4 Avsluttede oppgave

- Utviklingsmiljøet er ferdig etablert både på SW og HW siden.
- Kravspesifikasjonen er ferdig nedskrevet.
- Implementert bruteforce algoritmen.

D.5 Oppgaver under arbeid

Algoritme 1 skal snarest påbegynnes.

D.6 Tidsfrister/overskridelser

Som nevnt i punkt 2 er det noen overskridelser. Vi har ennå ikke påbegynt "kode algoritme 1" eller "teste algoritme 1". Dette som nevnt litt på grunn av mye å gjøre i andre fag samt problemer med QPainter klassen. Implementeringen av "bruteforce algoritmen" er da heller ikke medberegnet i Gantskjemaet.

D.7 Motivasjon

Motivasjonen kunne vært bedre. Dette på grunn av mye hodebry med Qt og QPainter.

D.8 Veileders rolle

Samarbeid med veileder og oppdragsgiver fungerer ut fra vårt synspunkt meget bra.

Tillegg E

Statusrapport 3, 18.04.01

E.1 Status

E.1.1 Planlegging/fremdriftsplan

Planlagte oppgaver og bruk av tid i følge Gantskjemaet :
- Kode algoritme 2.
- Testing av algoritme 2.
- Rapportskriving.

E.1.2 Organisering av ansvarsområder

På grunn av gruppens størrelse har vi litt flytende ansvarsområder. Ansvar blir fordelt ettersom det trengs. Anders er nå blitt tildelt ansvar for GUI, mens Tom Audun har "tråder" som sitt.

E.1.3 Klargjøring av problemstillingen/systemering

Det skal en av de nærmeste dagene avholdes møte med oppdragsgiver og veileder. Her skal det gjennomgås videre detaljert beskrivelse av algoritme 2. Trolig fargedeteksjon.

E.1.4 Rapportskriving

Vi har begynt med rapporten, men ligger langt etter. Har derimot gode notater i våre logger.

E.2 Totalstatus

Vi ligger for tiden litt bak skjema, dette på grunn av store prosjekter i andre fag som har tatt mer tid enn forventet. Påsken kom også med planlagt ferie. Bruteforce algoritmen var heller ikke planlagt i Gantskjemaet.

E.3 Problemer/løsninger

Vi har ennå ikke løst problemet med QPainter klassen for å tegne opp bildet riktig. Dette gjøres i ledige stunder. Har også noe problemer med GUI og tråder noe som skal løses førstkommende uke.

E.4 Avsluttede oppgaver

- Utviklingsmiljøet er ferdig etablert både på SW og HW siden.
- Kravspesifikasjonen er ferdig nedskrevet.
- Implementert bruteforce algoritmen.
- Algoritme 1 er ferdig implementert.

E.5 Oppgaver under arbeid

Algoritme 2 skal snarest påbegynnes, mens GUI og tråder er under utvikling.

E.6 Tidsfrister/overskridelser

På grunn av tidligere forsinkelser nevnt i forrige statusrapport er vi også nå litt på etterskudd.

E.7 Motivasjon

Motivasjonen er på topp etter en lang og fortjent Påskeferie.

E.8 Veileders rolle

Samarbeid med veileder og oppdragsgiver fungerer ut fra vårt synspunkt meget bra. Trolig har også veileder løst problemet vårt med tråder.

Tillegg F

Statusrapport 4, 27.04.01

F.1 Status

F.1.1 Planlegging/fremdriftsplan

Planlagte oppgaver og bruk av tid i følge Gantskjemaet :
- Kode algoritme 2.
- Testing av algoritme 2.
- Grundigere testing av algoritmer, 5 dager.
- Sammensying av BIRI, 3,5 dager.
- Rapportskriving.

F.1.2 Organisering av ansvarsområder

Siden forrige møte har Anders hatt ansvar for GUI mens Tom A. har bukt tiden til tråder. Videre i prosjektet vil oppgaver bli fordelt etter arbeidsmengde og interesser.

F.1.3 Klargjøring av problemstillingen/systemering

Det vil samtidig med statusmøte 4 bli avholdt møte med oppdragsgiver og veileder for videre fremgang i prosjektet. På dette møtet vil vi se på kravspesifikasjonen for å finne eventuelle mangler. Det vil også bli tatt opp med oppdragsgiver eventuelle forbedringer eller endringer.

F.1.4 Rapportskriving

Vi har ikke kommet videre med rapportskriving siden sist. Det vil si at vi ligger langt etter og må ta igjen dette nå.

F.2 Totalstatus

På tidligere statusmøter har vi vært forsinket med ca. to uker i forhold til Gantskjemaet. Disse forsinkelsene er nå tatt igjen som følge av iherdig jobbing. Algoritme to ble ferdig kodet 26.04.01, akkurat som planlagt. Nå gjenstår det bare testing av de to algoritmene for å finne de beste parameterene. Da det oppsto en feil under oppdatering av cvs'en må tråder implementeres på nytt. Rapporten samt dokumentering av koden vil bli første prioritet fremover.

F.3 Problemer/løsninger

Vi har ennå ikke løst problemet med QPainter klassen for å tegne opp bildet riktig. Vi har også brukt mye tid på feilsøking som det siden har vist seg at var en feil i den spesielle versjonen av Qt.

F.4 Avsluttede oppgave

- Utviklingsmiljøet er ferdig etablert både på SW og HW siden.
- Kravspesifikasjonen er ferdig nedskrevet.
- Implementert bruteforce algoritmen.
- Algoritme 1 er ferdig implementert.
- Algoritme 2 er ferdig implementert.
- GUI er ferdig implementert.

F.5 Oppgaver under arbeid

En grundigere testing av de to algoritmene og parameterstyringen vil bli gjort de nærmeste dagene. Tråder vil bli implementert på nytt.

F.6 Tidsfrister/overskridelser

Vi ligger nå ajour med Gantskjemaet. Som tidligere nevnt er vi noe forsinket med rapportskrivingen.

F.7 Motivasjon

Da vi har jobbet inn tapt tid, er motivasjonen igjen på topp.

F.8 Veileders rolle

Samarbeid med veileder og oppdragsgiver fungerer ut fra vårt synspunkt meget bra.

Tillegg G

Kildekode

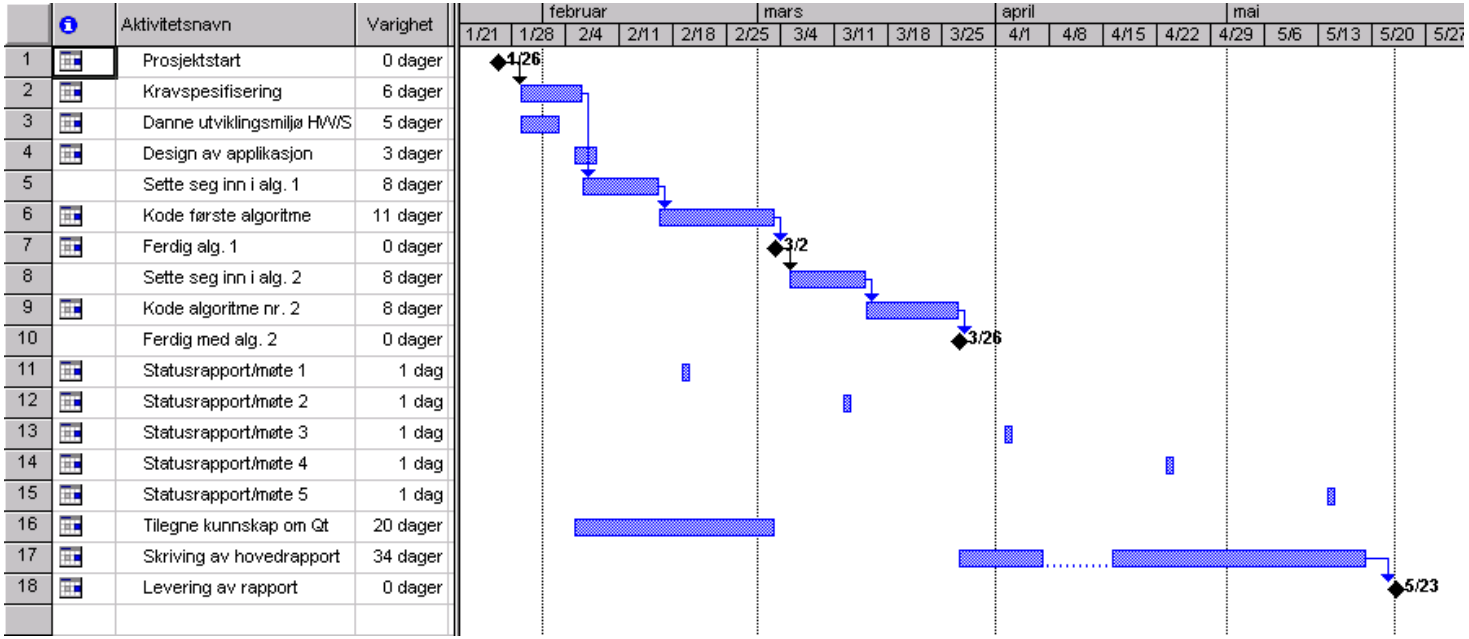
Hele kildekoden til prosjektet, ligger vedlagt på cd. Denne cd'en inneholder :

- Kildekoden for BIRI (ferdig sammensydd med BIRI 2).
- Hele rapporten i pdf-format.

Om noen skulle behøve tilgang på cvs'en til koden, kan dette muligens la seg ved nærmere avtale med Erik Hjelmås.

Tillegg H

Gantskjema



	Aktivitetsnavn	Varighet
1	Prosjektstart	0 dager
2	Kravspesifisering	6 dager
3	Danne utviklingsmiljø HW/S	5 dager
4	Design av applikasjon	3 dager
5	Sette seg inn i alg. 1	8 dager
6	Kode første algoritme	11 dager
7	Ferdig alg. 1	0 dager
8	Sette seg inn i alg. 2	8 dager
9	Kode algoritme nr. 2	8 dager
10	Ferdig med alg. 2	0 dager
11	Statusrapport/møte 1	1 dag
12	Statusrapport/møte 2	1 dag
13	Statusrapport/møte 3	1 dag
14	Statusrapport/møte 4	1 dag
15	Statusrapport/møte 5	1 dag
16	Tilegne kunnskap om Qt	20 dager
17	Skriving av hovedrapport	34 dager
18	Levering av rapport	0 dager

Figur H.1: Gantskjema.

Bibliografi

[1] Ivar Farup og Erik Hjelmås. Experimental Comparison of Face/Non-Face Classifiers. Høgskolen i Gjøvik's rapportserie, nr.1 2001.

[2] http://www.smcc.demon.nl/camstream/. Sist besøkt 21-05-2001.

[3] Matthias Kalle Dalheimer. Programming with Qt. First Edition, May 1999. ISBN: 1-56592-588-2.

[4] Christel Beate Lerøy og Henry Johansen. ICI'98 Intelligent Computer Interface. Hovedpprosjekt ved HIG, 15.mai 1998.

[5] Jie Yang og Alex Waibel. A Real-Time Face Tracker. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.