Objektsegmentering og tekstdeteksjon i forelesningsvideo

Eirik Grythe

Masteroppgave
Master i Teknologi - Medieteknikk
30 ECTS
Institutt for informatikk og medieteknikk
Høgskolen i Gjøvik, 2005

Masterprogrammet i medieteknikk
har blitt kjørt i samarbeid med
Kungliga Tekniska högskolan (KTH),
Stockholm, Sverige

Institutt for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Abstract

In video surveillance situations it is often desirable to see the all the content of the video background. To do this, moving objects in foreground have to be removed. In this thesis it is used videos from lecture, and the lecturer walking in front of the blackboard is removed to make the text visible. It has been developed a binary picture set parallel to the movie that restricts which area is the blackboard and which is the person in foreground. Binary picture sets were developed using Euclidean distance to RGB colours and CbCr-components. The pictures had to be threshold to give a good separation. CbCr-components had the best segmentation effect.

The text was detected by using Euclidean distance to RGB colours, but since this didn't result in a good readability, four edge detection operators was implemented. Qualitatively Prewitt was most suitable to detect text. By adding pictures of detected text, readability was improved additionally. By saving a picture before each time the text are rubbed out, a picture set of the blackboards contents through the lecture will be generated. A picture set like this will have large utility value if you quickly want to read the blackboard contents from a lecture.

Attempts to digitalise handwritten text have been accomplished, but even after the text coherence was improved with a morphological operation, the text was not recognizable.

Since video processing is time consuming, two reduction algorithms have been developed; one to crop movies in area, and one sub sampling algorithm.

Sammendrag

I en del videoovervåkningssituasjoner ønsker man til enhver tid å se hva bakgrunnen i videoen inneholder. For å gjøre dette må man segmentere vekk objektene som beveger seg i forgrunnen. I denne oppgaven er det tatt utgangspunkt i videoer fra forelesning og foreleseren som beveger seg foran tavla har blitt fjernet slik at man til enhver tid kan se all teksten. For å gjøre dette har det blitt utviklet en binær bildeserie som går parallelt med selve filmen og avgrenser hvilket område som er tavla og hvilket område som er personen i forgrunn. Binære bildeserier ble utviklet ved å regne ut den euklidske avstanden til RGB-farger og CbCr-komponenter. For å gi en god avgrensing ble bildene tersklet. CbCr-komponentene gav den beste segmenteringen.

Teksten ble detektert ved bruk av euklidske avstand til RGB-farger, men siden dette ikke gav god lesbarhet, ble det utviklet fire kantdeteksjonsfiltre. Kvalitativt sett var Prewitt best egnet til å detektere tekst. Ved å addere flere bilder av detektert tekst, vil resultatet forbedres ytterligere. Dersom man lagrer et bilde før hver gang skrift blir pusset vekk, får man en bildeserie med tavlas innhold gjennom en forelesning. En slik bildeserie har stor nytteverdi hvis man raskt ønsker å lese tavleinnholdet fra en forelesning.

Det ble også gjort forsøk på å gjøre om håndskrift til digital skrift. Selv etter at en morfologisk operasjon hadde blitt foretatt på skifta, lot den seg ikke gjenkjenne.

Siden videobehandling krever mye prosesseringstid, har det blitt utviklet en algoritme for å beskjære filmene i areal og en algoritme for å begrense hvor mange bilder som er nyttig å behandle.

Forord

Denne masteroppgaven i medieteknikk er utført ved Institutt for informatikk og medieteknikk ved Høgskolen i Gjøvik. Formålet med oppgaven har vært å finne metoder for segmentering av objekter i bevegelse og detektere tekst i forelesningsvideo. Studenter og forelesere vil da i ettertid kunne laste ned forelesningens innhold i den visualiseringsformen de måtte ønske. Under arbeidet har jeg fått mye god litteratur og veiledning av PhD. student Andrei Ouglov og ønsker å rette en stor takk til han. Jeg vil også rette en takk til utvikleren av APIet Oxide, Øyvind Kolås. Å kunne jobbe med et nytt bilde og videbehandlings API har vært både spennende og krevende. Øyvind har vist særdeles god programmeringsinsikt og har vært til stor hjelp til å forklare tekniske muligheter og begrensinger i APIet. I tillegg vil jeg takke Rune Hjelsvold for dyktig veiledning og grundig korrektur på rapporten. Richard Galland er en fransk PhD. student som jeg vil takke for stor interesse og mye kunnskap om skriftgjenkjenning.

Innhold

Abstract . . . . . **iii**
Sammendrag . . . . . **v**
Forord . . . . . **vii**
Innhold . . . . . **ix**
1 Introduksjon **1**
1.1 Prosessering 1
1.2 Problembeskrivelse 1
1.3 Problemformulering 2
2 Videoprosessering **7**
2.1 RGB/YCbCr 7
2.2 Digital video 8
2.3 Begrepesdefinering 8
2.4 Bevegelse 9
2.5 Støy 10
2.6 Masker 12
2.6.1 Euklidsk avstand 12
2.6.2 Kromatisitet 13
2.6.3 Terskelverdier 13
2.7 Objektsegmentering 15
2.7.1 Snaking 15
2.7.2 Bevegelsesglatting 16
2.8 Deteksjon av skrift 17
2.8.1 Kantdeteksjon 17
2.8.2 Håndskrift 20
2.8.3 Skriftgjenkjenning 20
2.8.4 Morfologiske operasjoner 20
2.9 Reduksjon av data i video 21
2.9.1 Subsampling 21
2.9.2 Beskjæring av areal 22
2.10 API-Oxide 23
2.10.1 Kvalitetstap 23
3 RGB **25**
3.1 Generering av masker 25
3.1.1 Eksperimentoppsett 25
3.1.2 Resultat 28
3.1.3 Diskusjon 30
3.2 Objektsegmentering 32
3.2.1 Eksperimentoppsett 32
3.2.2 Resultat 32
3.2.3 Diskusjon 33

4 YCbCr . . . 35
4.1 Generering av masker . . . 35
4.1.1 Eksperimentoppsett . . . 35
4.1.2 Resultat . . . 36
4.1.3 Diskusjon . . . 36
4.2 Objektsegmentering . . . 38
4.2.1 Eksperimentoppsett . . . 38
4.2.2 Resultat . . . 38
4.2.3 Diskusjon . . . 41
4.3 Forbedring av masker og objektsegmentering . . . 41
4.3.1 Fylling ved deteksjon av hull . . . 43
4.3.2 Fylling ved bruk av tekstmaske . . . 44
4.3.3 Segmentering av objektkontur . . . 46
5 Objektsegmentering ved bruk av temporal medianbevegelse . . . 49
5.1 Eksperimentoppsett . . . 49
5.2 Resultat . . . 49
5.3 Diskusjon . . . 50
6 Behandling av skrift . . . 53
6.1 Kontraststrekking . . . 53
6.2 Kantdeteksjon . . . 53
6.2.1 Eksperimentoppsett . . . 53
6.2.2 Resultat . . . 54
6.2.3 Diskusjon . . . 57
6.3 Forbedring av leselighet . . . 57
6.3.1 Eksperimentoppsett . . . 57
6.3.2 Resultat . . . 58
6.3.3 Diskusjon . . . 58
6.4 Deteksjon av skriftreduksjon . . . 60
6.4.1 Eksperimentoppsett . . . 60
6.4.2 Resultat . . . 60
6.4.3 Diskusjon . . . 60
6.5 Skriftgjenkjenning . . . 61
6.5.1 Eksperimentoppsett . . . 61
6.5.2 Resultat . . . 61
6.5.3 Diskjusjon . . . 63
7 Beskjæring av video i tid og rom . . . 65
7.1 Subsampling . . . 65
7.1.1 Eksperimentoppsett . . . 65
7.1.2 Resultat . . . 67
7.1.3 Diskusjon . . . 69
7.2 Beskjæring av areal . . . 69
7.2.1 Eksperimentoppsett . . . 69
7.2.2 Resultat . . . 71
7.2.3 Diskusjon . . . 71
8 Konklusjon . . . 73
9 Videre arbeid . . . 77

9.1 Algoritmisk utvikling av masker 77
9.2 Bevegelsessegmentering 77
9.3 Skrift 78
9.4 Brukerundersøkelse 78
9.4.1 Materiale 78
9.4.2 Spørreskjema 78
Figurer **79**
Bibliografi **83**

1 Introduksjon

1.1 Prosessering

I film er vanligvis skuespillernes opptreden hovedinnholdet. I andre tilfeller er det ønskelig å bevare bakgrunnen og fjerne objektene som beveger seg i forgrunnen. Eksempler på slike situasjoner er registrering av ledige parkeringsplasser, utvikling under operasjon, overvåking av murvegger som er utsatt for tagging og filming i undervisningssammenheng. Denne oppgaven tar for seg problematikken rundt segmentering av foreleser og bevaring av skrifta på tavla i undervisningssammenheng. Selv om utviklingen er konsentrert om undervisningssituasjonen er det lagt vekt på at løsningene skal være overførbare til andre systemer. Ved å tilpasse noen verdier kan systemet brukes til segmentering, deteksjon og beskjæring til ulike typer overvåkningsvideoer med stillestående kamera.
Før teksten i bakgrunnen kan detekteres, må objektene som beveger seg i forgrunnen segmenteres vekk. I tidligere arbeid med segmentering har man forsøkt å bevare objektene i forgrunnen og fjernet bakgrunnen. Dette er mye brukt i for eksempel intelligente trafikkovervåkningssystemer. Bilene er da av interesse, mens bakgrunnen segmenteres vekk. Målet i denne oppgaven er å gå motsatt vei; segmentere vekk objekter og bevare bakgrunnen. For å kunne gjøre dette er man avhengig av en maske som gir en nøyaktig avgresning av området som skal bevares. Dersom ikke objektet i bevegelse blir fullstendig segmentert vekk, vil det skape støy når teksten i bakgrunnen skal detekteres. Når pikslene foreleseren består av er segmentert vekk, kan man når som helst trykke stopp under avspilling og se hele tavlas innhold.
Skrifta på tavla vil så bli detektert. Grunnen til at det vil være interessant å detektere skrift, er at man da kan lage funksjonalitet for å foreta tekstbaserte søk i forelesningens innhold. For å detektere tekst vil det bli generert masker ut ifra den euklidske avstanden til RGB-fargen til krittskrift. Det vil også bli testet ulike kantoperatorer. Når man detekterer skrift i bilder er man avhengig av en god representasjon for at skrifta skal bli leselig. Skrifta pusses vekk med jevne mellomrom. Ved å utvikle en løsning som detekterer når tavla pusses, kan det opprettes en bildeserie av alt innholdet som har stått på tavla i løpet av timen.
Ved å detektere når det er bevegelse i filmen og lagre disse bildene kan det genereres en ny videofilm som består av mange færre rammer og som kan gi en hurtigavspilling av forelesningen. Overnevnt videobehandling vil da kun være nødvendig å foreta på de rammene som skiller seg tilstrekkelig mye fra foregående rammer. Som regel dekker ikke tavla hele filmen. Det var derfor nyttig å beskjære filmen slik at man kun ser tavla. Dette vil også redusere filmens størrelse mye og nedlastning av resultatvideoen vil gå raskere.

1.2 Problembeskrivelse

Ved Høgskolen i Gjøvik har det blitt filmet flere forelesninger. Filmene er tilgjengelig på Internett, og de brukes i flere prosjekter ved skolen. Det er nyttig å kunne laste ned forelesningsfilmer for både studenter og forelesere. Man kan da se tilbake på hva som har blitt gjennomgått. Problemet med nedlastning av film i høy kvalitet er at det kreves

stor båndbredde. Komprimeres filmene blir skrifta på tavla uleselig. For å kunne overføre forelesningens innhold på en raskere måte har det blitt utviklet tre løsninger.
Essensen i en forelesningsvideo er det foreleseren sier og det som skrives på tavla. Foreleseren kan derfor segmenteres vekk. Skrifta vil da være det eneste som forandrer seg i bildet og den vil dukke opp etter hvert som foreleseren skriver. I øyeblikket før skrifta pusses vekk, vil det være nyttig å lagre innholdet på tavla i bilde.

- En serie bilder fra tidspunktene før tavla pusses vil kunne lastes ned hurtig og man vil raskt kunne lese hele forelesningens innhold

Når det ikke er noe bevegelse i film er det heller ikke nødvendig å ivareta mange like rammer per sekund. Ved å detektere de rammene som skiller seg tilstrekkelig mye fra foregående rammer og sette sammen disse til en ny video, vil man få

- en hurtigavspillingsvideo.

En slik video vil være redusert i størrelse siden den inneholder færre rammer per sekund og man vil raskt kunne se hele forelesningen.
Tredje mulighet er å generere

- en objektsegmentert film i normal hastighet med lyd.

Forskjellen fra denne filmen og den originale er at foreleseren er fjernet. Skrifta vil da dukke opp på tavlen etter hvert som foreleseren skriver. For å redusere størrelsen på denne filmen, vil det være en fordel å beskjære filmen etter tavla.

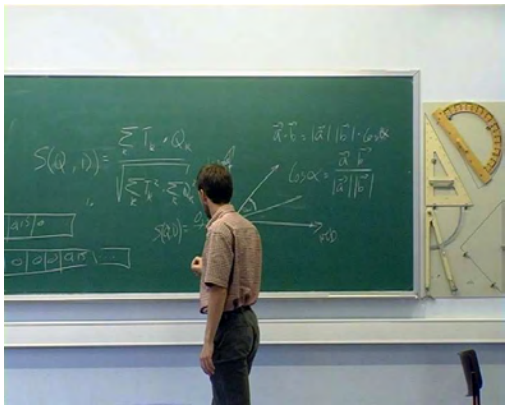
En video fra en forelesning kan deles inn i 3 lag (se figur 1 - 4). Bakgrunn, som er tavla og andre statiske objekter i rommet, skrifta som kontinuerlig legges på og strykes fra bakgrunnen og foreleseren som beveger seg i forgrunnen. Målet med denne oppgaven er å segmentere det området som viser foreleseren og samtidig bevare området som befinner seg bak personen. Videoen vil da kun inneholde den statiske bakgrunnen og skrifta som etter hvert dukker opp eller blir pusset vekk. En av utfordringene blir derfor å finne gode metoder som avgrenser området i bildet der foreleseren befinner seg og der tavla befinner seg.

1.3 Problemformulering

Ved bruk av stillestående kamera, vil et objekt i bevegelse befinne seg på forskjellig sted i bildet avhengig av tiden. Slår man sammen to bilder hvor objektet har forskjellig plassering og regner ut gjennomsnittsbildet, vil man få en uskarp gjengivelse av objektet. En slik utjevning av bevegelse over tid kalles temporal medianfiltrering. Ved å slå sammen alle bildene over et tidsintervall, vil objekter i bevegelse få en lavere representasjon i hvert enkelt punkt og utvaskingen vil forbedres ytterligere.

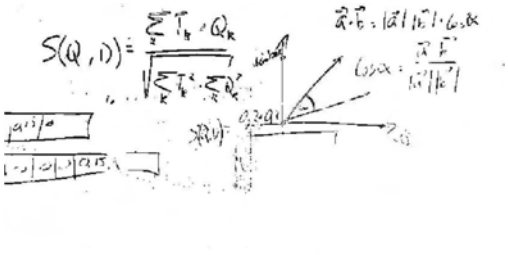
- Vil temporal medianfiltrering være en egnet metode for å fjerne objekter i bevegelse?

Euklidsk RGB-avstand er et mål for hvilken avstand fargene rød, grønn og blå har i det 3-dimensjonale fargerom. Avstanden beregnes ved å ta utgangspunkt i en valgt referansefarge og sammenligne denne med fargesammensetningen i et annet piksel. Man får da et mål for hvor ulike fargene er. Dersom fargene er ulike vil avstanden bli stor. Ved å sammenligne referansefargen med alle pikslene i bildet, kan man generere et gråtonebilde. Intensiteten til gråtonene vil da være et mål på hvor langt ifra den valgte fargen pikslene i bildet befinner seg. Er referansefargen fra et ensfarget område, vil alle

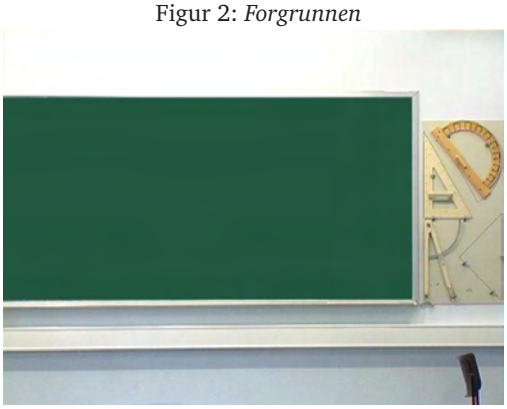


Figur 1: *Bildet kan deles inn i 3 lag*

Figur 2: *Forgrunnen*



Figur 3: *Teksten*



Figur 4: *Bakgrunnen*

pikslene i dette området få lave verdier. Ved å terskle disse verdiene, kan det genereres en sort/hvit-maske som angir hvor bildet skal oppdateres.

- Vil man få ytterligere forbedringer av objektsegmenteringen ved å kombinere bevegelsesinformasjon og fargeinformasjon?

Problemet med fargeavstand er at ensfargede områder vil ha forskjellig fargeverdi dersom belysningen er forskjellig. Dette vil i praksis si at et gråtonebilde med utgangspunkt i fargen på tavla, ikke vil inkludere hele tavla dersom tavla har ujevn belysning. Intensitetsforskjeller kan løses på flere måter. En metode vil være å utvikle en fargemaske for hvert av de ulikt belyste områdene, og slå i sammen maskene for å gi en fullstendig avgrensing. En bedre metode vil være å fjerne intensitetsforskjellene ved bruk av kromatisitet. Dette gjøres ved å addere fargekanalene rød, grønn og blå og dividere med den fargen man ønsker å finne. Man vil da få lik kromatisitet.

- Vil programmet håndtere ulikt belyste områder?

YCbCr er et fargerom som blir brukt i video. Ved å regne ut den euklidske avstanden til krominanskomponentene og terskle resultatet vil man kunne skille forgrunnen fra bakgrunnen.

- Vil bruk av CbCr-komponenter gi en adaptiv og bedre avgrensing enn RGB-verdier?

På grunn av bevegelse, intensitetsforskjeller og støy vil vanligvis ikke en binær maske gi en fullstendig markering av ønsket område. En medianfiltrering eller støyfjerning vil kunne rette mange av disse filene, men ulempen med medianfiltrering er at konturen til objekter blir mindre presis og dette vil resultere i en dårligere segmentering. Det vil derfor bli utviklet to metoder for å forbedre de binære maskene.

- Vil man få en bedre dekkende maske ved å tette små hull som ikke har blitt inkludert i maska?

Når et objekt er i bevegelse, får man forstyrrelse fra konturen til objektet etter at segmenteringen er foretatt.

- Hvordan kan forstyrrelsen fra konturen til objekter fjernes?

Etter at objektene i forgrunnen i filmen er fullstendig segmentert vekk, vil det være nyttig å kunne detektere skrift og andre mønster i bakgrunnen. Deteksjon og gjenkjennelse av mønster vil være nyttig ved for eksempel registrering av bilnummer ved landegrenser. I denne oppgaven vil det gjøres en deteksjon og forbedring av leseligheten til teksten som skrives på tavla.

- Vil kantinformasjon gi en god representasjon av skrift?
- Vil sammenslåing av bilder av detektert skrift over et tidsrom gi en forbedring av skriftgjengivelsen?
- Vil man kunne gjøre om hånskrevet tavleskrift til digital tekst ved bruk av skriftgjenkjenningsprogram?

Når det skrives på tavla øker totalverdien av hvite piksler. Pusses det vekk skrift, vil totalverdien reduseres. Ved å detektere hvor mange hvite piksler det befinner seg i ulike rammer, vil man kunne finne tidspunktet når det pusses vekk skrift. Rammene i tidspunktet når det blir pusset vekk tekst vil være nyttig å lagre slik at man i ettertid kan se alt som er skrevet på tavla i en bildeserie.

- Hvordan kan man finne tidspunktene når skrifta på tavla pusses vekk og innholdet skal lagres i bilder?

I de to siste forskningsspørsmålene legges det vekt på å utvikle metoder som vil redusere prosesseringstiden til programmet og gjøre filmene mindre slik at det går raskere å laste dem ned. Oppdateringsfrekvensen av bilder i video er høy for å kunne gi en god gjengivelse av hurtige bevegelser. Når det er lite bevegelse, er det ikke nødvendig å ha en høy oppdateringsfrekvens. Ved å detektere bilder i video som skiller seg mye fra foregående bilder, finner man de bildene som er nyttig å prosessere. Det kan så genereres en hurtigavspillingsvideo av disse rammene.

- Vil en undersamplet video bli redusert i størrelse og vil videre prosessering gå raskere?

For å forbedre ytelsen ytterligere vil det være nyttig å kunne foreta en beskjæring i rom. Standard proporsjoner til videofilmer i PAL format er 720:576. Dersom området man filmer ikke dekker hele bildet, vil det være nyttig å kunne kutte vekk utenforliggende område.

- Vil en beskåret video være redusert i størrelse?

For å oppsummere de overstående forskningsspørsmålene vil følgende to hovedemner bli besvart:

- Vil objekter i video kunne segmenteres slik at kun bakgrunnen blir synlig?
- Vil tekst i forelesningsvideo kunne detekteres og gjengis slik at den blir leselig?

2 Videoprosessering

Dette kapitlet gir en innføring i de ulike videoprosesseringene som blir brukt i forskningsarbeidet.

2.1 RGB/YCbCr

RGB er et av de mest brukte fargerom. Ved å kombinere fargene rød, grønn og blå kan man lage alle farger. RGB-fargerommet er et adaptivt fargerom. Det vil si at det trengs en lyskilde for å lage en synlig farge. I en del tilfeller er det også en alfa kanal i tillegg til de tre fargekanalene. Med denne kanalen kan man regulere luminansen.

Figur 5: *Sammensetting av fargene i de tre kanalene rød, grønn og blå danner et fargebilde [1]*

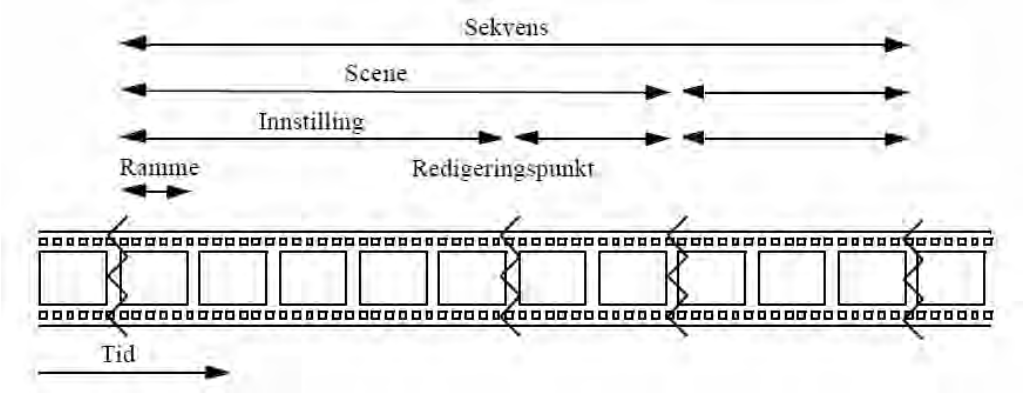
YCbCr er et fargerom som blir brukt i video. Y er luminans komponenten, Cb og Cr er krominanskomponentene. Dette fargerommet ble lagd for standard fjernsynsapparater (SDTV), men det har også blitt tatt mye i bruk i forskning. Blant annet egner fargerommet seg godt til å foreta ansiktsgjenkjenning [2] og å lage 3D-ansiktsmodeller[2]. Det viser seg også å gi en robust og nøyaktig deteksjon av forgrunn og skygger i video. [1]. Dette utnyttes til blant annet å segmentere biler i trafikkovervåkningssystemer [3, 4]. Cb- og cr-komponentene kan brukes til å skille bakgrunnen fra forgrunnen i en video. For konvertere et RGB-bilde til YCbCr-bilder brukes følgende formler [1]:

- Y = 0.299R + 0.587G + 0.114B
- Cb = -0.169R - 0.331G + 0.5B + 0.5
- Cr = 0.5R - 0.419G - 0.081B + 0.5

CMYK er et fargerom som blir brukt mye til printing. Fargerommet består av de 4 fargene cyan, magenta, yellow, key(sort). CMYK har mye til felles med RGB, men det gir et bedre utskriftsresultat. Ulempen med både CMYK og RGB er at det er vanskelig å gjenskape en farge på skjerm og i utskrift. Dette skyldes gammut-innstillinger og resulterer ofte i differanse i hudfarge.

2.2 Digital video

En video er sammensatt av en rekke etterfølgende bilder som spilles av i en bestemt hastighet. En sammenhengende film kalles en sekvens (figur 6). En ramme er det minste elementet i video. Rammene danner instillinger som igjen kan danne scener og sekvenser. En innstilling er definert som en kontinuerlig sekvens av lyd og bilde. Ifølge G. Davenport, T.G. Aguierre Smith og N. Pincever [5] er en innstilling den fundamentale komponenten i film.



Figur 6: *Filstruktur [6]*

2.3 Begrepesdefinering

En *ramme* er et bilde i en video. En video er satt sammen av et bestemt antall bilder i sekundet. I PAL-formatet[1] er oppdateringsfrekvensen på 25 bilder/sekund, NTSC[2] 29,97 bilder/sekund og i film er det 24 bilder/sekund. Etterfølgende bilder vil heretter bli omtalt som rammer.

En *bevegelsesramme* er en ramme som skiller seg en bestemt prosentandel fra en tidligere ramme. Bevegelsesrammer detekteres ved å sammenlikne alle piksler i to rammer.

En *innstilling* er en kontinuerlig film uten klipp. Filmene i denne oppgaven inneholder kun en innstilling siden kameraet startes i begynnelsen av timen og stoppes i slutten.

En *scene* inneholder flere innstillinger fra samme tid og sted.

En *sekvens* er relaterte scener med samme handlingstråd. Siden videoen som brukes i denne oppgaven ikke er sammensatt av scener og sekvenser, blir ingen av disse begrepene brukt.

En binær *maske* er et sort/hvitt-bilde. Masker kan f. eks brukes til å angi hvilke piksler i et bilde som tilhører et objekt.

Et *objekt* består av en gruppe piksler som danner et sammenhengende område. I videoer

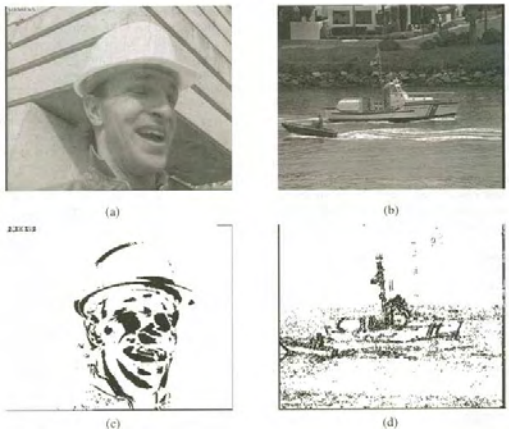
[1] PAL(Phase Alternating Line). TV-systemet som er mest brukt i Vest-Europa og Australia.
[2] NTSC(National Television System Committee). TV-systemet som er mest brukt i Amerika og Canada.

fra forelesninger blir foreleseren og tavla omtalt som objekter.

En tavlevideo kan deles inn i tre *lag*. Bakgrunnen og tavla er et lag som ikke er i bevegelse og som tilhører bakgrunnen. Skrifta er et lag som legges på tavla. Foreleseren og studentene tilhører forgrunnen som er et lag med bevegelse.

2.4 Bevegelse

Bevegelse i film vises ved at grupper av piksler har forskjellig plassering i etterfølgende rammer. Det er allerede gjort mye arbeid med å detektere bevegelse. [7, 8, 9, 10].I MPEG[3] f. eks blir det brukt blokkbaserte metoder for å ivareta bevegelse [11]. Områder av like piksler blir da behandlet som en blokk, og kun posisjonen til blokka må lagres for hver ny ramme. Siden videoene som skal behandles i denne oppgaven krever en detaljert gjengivelse av bevegelse i skrifta, må bevegelsesdeteksjonen foregå på pikselnivå. For å finne differensen mellom to bilder, vil man da måtte sammenligne alle pikslene i det ene bildet, med pikslene i det andre. Totaldifferansen kan detekteres ved å telle opp antall røde, grønne og blå piksler i hvert av bildene og sammenlikne summene. Eller man kan sammenlikne hvert piksel i de to bildene en etter en og øke differanseverdien når de er forskjellige. Sistnevnte metode vil være best til å detektere bevegelse [12, 13, 14]. Grunnen til dette er at det kan være bevegelse i bildet selv om total fargeverdi ikke er forandret. I figur 7 er bevegelse representert med sorte områder. I bildet til venstre er det stor bevegelse i ansiktet til personen, mens i bildet til høyre er det litt bevegelse i vannet og båten.



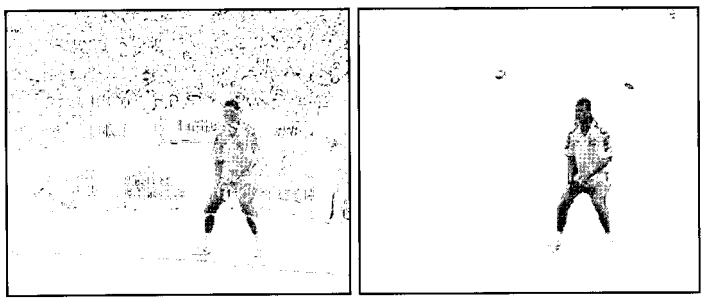
Figur 7: *(a) og (b) er originalbilder. (c) og (d) viser piksel som har vært i bevegelse ifølge en global bevegelsesdeteksjon [15].*

Store lysforandringer vil også oppfattes som bevegelse [16]. Grunnen til dette er at fargene får høyere verdier når de tilføres lys, og dette vil tolkes som forandring. I videoer fra undervisning vil derfor tilfeller som når overheaden slås på, gardinene blir flyttet eller lys bli slått på, oppfattes som global bevegelse [17]. For å begrense slike feilkilder kan man se på bevegelser over en periode.

[3]MPEG (Moving Picture Experts Group). Blokkbasert komprimering av video. MPEG er godt egnet til å redusere størrelsen til videofiler, ulempen er at små detaljer i bildet forsvinner.

2.5 Støy

Siden støy vil kunne feiltolkes som bevegelse er det viktig å kunne redusere støyen. Dette er det gjort forsøk på å begrense ved bruk av forskjellige filter [18, 19, 20, 21, 22, 23]. Figuren under viser et eksempel på hvordan et bilde ser ut før og etter støyfjerning.



Figur 8: *Støyfjerning ved bruk av MRF-bassert klassifisering [24]*

Bildet til venstre gir mye bevegelsesrepresentasjon blant publikum. Ser man på større områder av gangen, vil man kunne fjerne slik støy og få representert objekter i bevegelse bedre. En annen mye brukt metode for å fjerne støy er medianfiltrering [25, 26, 27]. Når et bilde medianfiltreres, tas det utgangspunkt i et piksel som sammenlignes med omkringliggende piksler. Er pikslet i sentrum ulikt omkringliggende piksler, vil dette forandres slik at det blir likt. Hvor mye støy medianfilteret skal fjerne, avgjøres av størrelsen på matrisen. Enkeltpiksler som skiller seg mye fra omkringliggende piksler vil som regel være støy. Uansett om det er objekter i bevegelse i bildet eller ikke, vil det hele tiden være noe støy som vil tolkes som bevegelse [28, 29, 30, 31]. Ulempen med støyfjerning er at når støypartikler fjernes, fjernes også små detaljer i bildet. Ved å studere bildene i figur 9 ser man at etter støyfjerningen har spesielt øynene til dama blir mindre skarpe. Skrifta på tavla inneholder mange små detaljer som lett kan oppfattes som støy. I denne oppgaven er det viktig at skrifta i resultatvideoen blir representert like godt som i originalvideoen.

Figur 9: *Fjerning av impulsstøy ved bruk av en 5x5 CWM glatting: (Øverst til venstre) original bilde, (Ø.H.) bilde med "salt og pepper støy", (N.V) CWM glatting med Wc = 16, (N.H.) CWM glatting med Wc=5 [15].*

2.6 Masker

Masker blir brukt til å avgrense områder. Dette kan brukes til å hente ut bestemte objekter, fjerne objekter, generere en statisk bakgrunn o.l. [32, 33, 34]. I blant annet nyhetsendinger og sendinger fra Norsk Tipping står personen i studioet foran en ensfarget bakgrunn (bluescreen/greenscreen). Bakgrunnen kan da maskeres og erstattes av en passende studiobakgrunn. Når det forekommer bevegelse, må masken oppdateres for hver ramme som skal behandles. Det skyldes at objekter i bevegelse vil ha forskjellig plassering i bildet til forskjellig tid. Masker kan genereres på flere måter. Det blir i denne oppgaven generert fargemasker og CbCr-masker.

Figur 10: *Bildet øverst til venstre viser et sort hvitt bilde av et flagg. Øverst til høyre har det blitt generert en maske av dette bildet der alle segmenter med hvit farge er maskert. I bildet nederst til høyre er det generert en maske av fargebildet nederst til venstre. Alle fargene bortsett fra grønt er maskert [15].*

2.6.1 Euklidsk avstand

For å foreta en klasifisering, kan må benytte euklidsk avstand for å finne hvilken avstand fargene i bildet har til hverandre. RGB-farger kan, som vist i figur 11, representeres i et 3-dimensjonalt fargerom. Alle farger kan lages av disse tre vektorene. Den euklidske avstanden er den avstanden to farger har til hverandre i det 3-dimensjonale fargerom. For å regne ut denne avstanden, tar man utgangspunkt i fargesammensetningen til et piksel i rammen, og regner ut hvor langt ifra dette pikslet de andre fargene i rammen befinner seg. Ut ifra resultatene man da får, kan det genereres en filtermaskering av fargeavstandene.

Euklidsk fargeavstand $= \sqrt{(\mathrm{red}_2 - \mathrm{red}_1)^2 + (\mathrm{gren}_2 - \mathrm{gren}_1)^2 + (\mathrm{blue}_2 - \mathrm{blue}_1)^2}$

Er avstanden i det 3-dimensjonale fargerom stor, vil man få en høy verdi, og dette kan visualiseres med en mørk piksel i maska. På de områder der avstanden er kort kan det settes lyse områder i maska. Jo lysere områdene er nærmere hverandre ligger fargene og jo likere er de. Intensiteten i gråtonemaskene kan selvsagt inverteres hvis dette er ønskelig.

Figur 11: *Farger illustrert som terninger [35]*

2.6.2 Kromatisitet

RGB er et adaptivt fargerom. Det vil si at verdiene til fargene øker proporsjonalt med luminansen. En ensfarget flate med ulik belysning vil derfor ha forskjellige fargeverdier. Siden alle de tre verdiene rød, grønn og blå har økt like mye i det belyste området, kan man bruke følgende formel dersom man ønsker å se bort ifra luminansforskjeller[]:

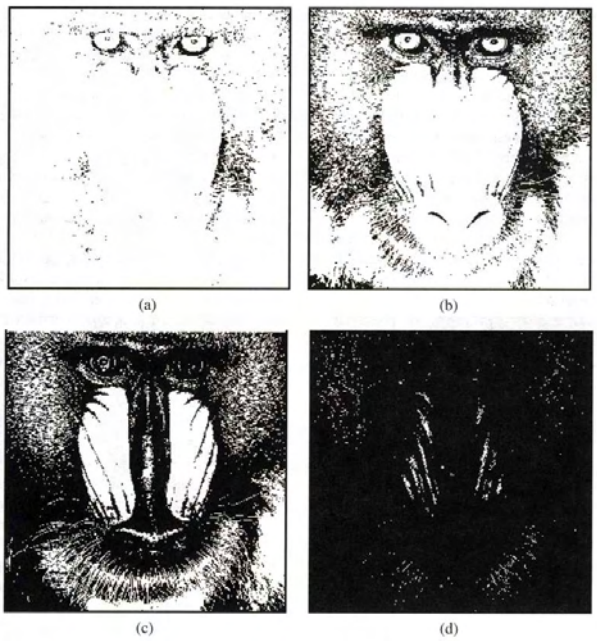
$$\mathrm{Red}_2 = \frac{\mathrm{Red}_1}{\mathrm{Red}_1+\mathrm{Green}_1+\mathrm{Blue}_1} \quad \frac{\mathrm{Green}_2}{\mathrm{Green}_1} = \frac{\mathrm{Green}_1}{\mathrm{Red}_1+\mathrm{Green}_1+\mathrm{Blue}_1} \quad = \mathrm{Blue}_2 \quad = \frac{\mathrm{Blue}_1}{\mathrm{Red}_1+\mathrm{Green}_1+\mathrm{Blue}_1}$$

Dersom man ønsker å detekere et ensfarget området ved bruk av euklidsk avstand, kommer denne formelen godt til nytte .

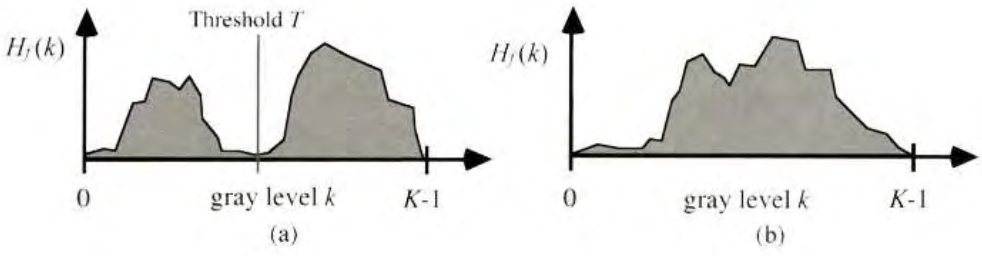
2.6.3 Terskelverdier

Terskelverdier kan utvikles på flere måter [36, 37, 38]. De kan f.eks settes manuelt eller utvikles algoritmisk. For å finne best mulig terskelverdi algoritmisk, er man avhengig av å ha en optimal verdi. I figur 12 er det vanskelig å utvikle en løsning for å finne den beste terskelverdien algoritmisk. Bilde (a) og (d) er ytergrenser som kan fjernes, men for å finne ut om terskelverdien for bilde (b) er bedre enn terskelverdien for bilde (c) trenger man noe å sammenligne med. I visse tilfeller har man et optimalt bilde som man ønsker å gjenskape [36], i andre tilfeller må man velge verdien selv.

I bilde (b) i figur 12 er ikke nederste del av ansiktet til apen synlig. I bilde (c) er også mye av bakgrunnen blitt en del av apen. For å få en god atskillelse på disse områdene, er man avhengig av et bilde med godt separerte verdier. Figur 13 viser histogramverdiene til to bilder. I bilde (b) ligger alle verdiene i et område, og det er derfor vanskelig å sette en terskelverdi. I bilde (a) derimot er det godt separerte verdier. I dette bildet kan det algoritmisk detekteres en god terskelverdi ved å finne det lokale minimumspunktet. Selv om et bilde ikke har godt separerte histogramverdier kan dette forbedres ved å foreta en kontraststrekking [39, 40]. Det som da skjer er at histogramverdiene blir trukket ut over hele spekteret i histogrammet. Er verdiene sentrert om et område, vil dette kunne gi en god forbedring og en god terskelverdi vil lettere kunne detekteres.



Figur 12: *Terskelverdiene setter en grense for hvor store områder som skal markeres [15].*



Figur 13: *Hypotetisk histogram: (a) godt separerte verdier (b) dårlig separerte verdier [15]*

2.7 Objektsegmentering

I tidligere arbeid med objektsegmentering [41, 42, 43, 44, 45] har målet vært å detektere objekter og segmentere disse. Det er altså objektets bevegelsesmønster og utseende som har blitt ivaretatt. Eksempel på systemer som bruker objektsegmentering er ITS (Intelligent Transport Systems) [46] og for eksempel ved segmentering av personer som vist i figur 14.

Figur 14: *Deteksjon av personer i video [47]*

Personene i bildet har blitt detektert og bakgrunnen blir segmentert vekk. I denne oppgaven er det ønskelige å gå motsatt vei. Objektet må detekteres og så fjernes for å kunne gi en visualisering av en fullstendig bakgrunn. Objekter detekteres for å kunne segmenteres vekk slik at bakgrunnen kan analyseres. En slik type segmentering vil være nyttig i f. eks videoovervåkning av flyplasser. Blir det forlatt bagasje, kan systemet detektere forandring i bakgrunnen i bildet og et evt. terrorangrep kan forhindres. Per dags dato ser det ikke ut til å være utviklet noen slike løsninger.

For å kunne foreta en bevegelsessegmentering må det foretas en deteksjon av objektet før selve segmentering kan utføres. Dette kan gjøres på flere måter:

- segmentering ved bevegelsesdeteksjon og analyse av temporale binære masker [48]
- segmentering ved bruk av bevegelsesanalyse av bildeforsterkning [49]
- segmentering ved bruk av MRF-modellen (Markov Random Field) [50]

I ITS (Intelligent Transport Systems) [46] som dette systemet har mest til felles med blir det gjort en bevegelsesdeteksjon og analyse av temporale binære masker. Dette bli foretatt en grov avgrensing av objekter i bevegelse ved bruk av adaptiv terskling. For å foreta segmenteringen blir piksler med lignende lysintensitet og bevegelse gruppert ved bruk av klustringsalgoritmer.

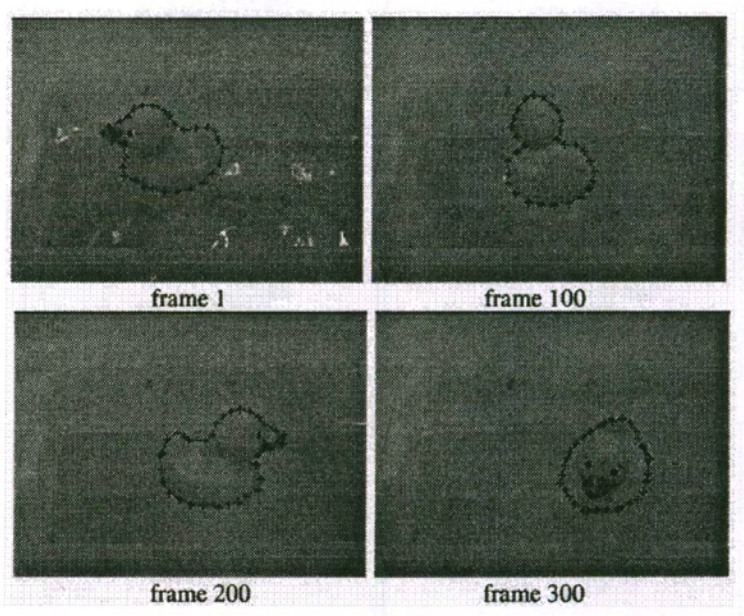
Gjenkjenning og analysere av objektene blir ofte foretatt etter at segmenteringen er foretatt. Man kan på den måten gjenkjenne biler, ansikter og andre objekter som for eksempel skrift i video.

2.7.1 Snaking

Snaking blir brukt til å detektere konturen til objekter. Verktøyet er blant annet implementer i Adobe Photoshop [4] og brukes for eksempel til å fjerne røde øyne. Ved å ta utgangspunkt i et piksel som befinner seg inne i et objekt, kan man detektere kantene som avgrenser objektet. Dette gjøres ved at man fra referansepikslet beveger seg i en retning til man finner en kant. Ved å følge denne kanten rundt hele objektet, vil man kunne markere konturen. For å følge kanten blir det benyttet en kantoperator. Man er da avhengig av at objektet som snakingalgoritmen skal avgrense er nokså ensfarget og

[4] Adobe Photoshop er et populært bildebehandlingsprogram med mye god funksjonalitet.

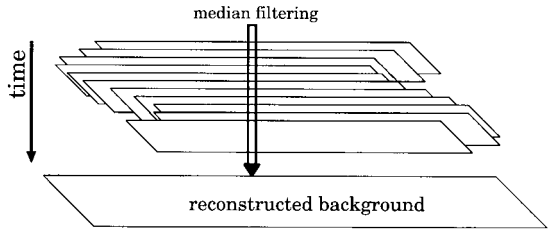
har samme intensitet. Det må også være en tydelig sammenhengende kant rundt hele objektet. Er ikke disse kravene tilfredstilt vil snakingen ekskludere deler av objektet eller inkludere områder som ikke hører med. I figuren 15 har det blitt foretatt snaking på et bilde med en and på mørk bakgrunn. Punktene i figuren viser markering som blir foretatt av snakingalgoritmen.



Figur 15: *Snaking av and i fire ulike videorammer [51]*

2.7.2 Bevegelsesglatting

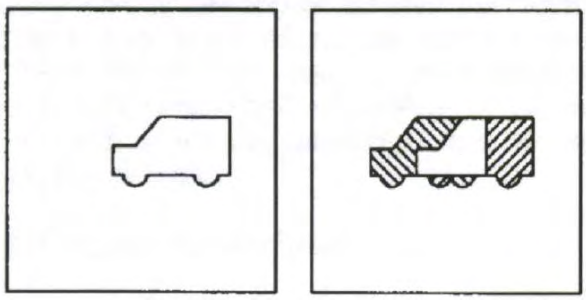
For å finne bakgrunnen i film er det gjort forsøk på å slå sammen rammer over et tidsrom [24]. Objekter som er i bevegelse vil da glattes ut over tid og bakgrunnen vil komme tydeligere fram. Dersom objektene som er i bevegelse er mye i bevegelse vil dette kunne gi en forholdsvis god segmentering av bakgrunnen. Er objektene i bildet store og de befinner seg innenfor området som representeres til enhver tid, vil det ikke være mulig å få vasket dem vekk fullstendig.



Figur 16: *Medianfiltrering for å finne bakgrunnen [24]*

Figur 17 viser gjengivelsen av en bil i to forskjellige punkter. I høyre figur har bilen beveget seg til venstre og kun i det hvite feltet har bilen en representasjon i begge punktene. Ved en sammenslåing av de to bildene vil bilen i de grå feltene kun ha halvparten

så stor representasjon som i det hvite feltet. Den varierende graden av representasjon til objekter i bevegelse kan benyttes til å foreta en utvasking over tid.



Figur 17: *Temporal medianfiltrering [52]*

2.8 Deteksjon av skrift

For å få en god representasjon av skrifta er man avhengig av at den skiller seg fra bakgrunnen på en eller annen måte og at den har en godt avgrenset kontur [53]. Dersom skriften og bakgrunnen er forholdsvis like, kan det foretas en kontraststrekking for å få et tydeligere skille. For å detektere skrift kan det tas utgangspunkt i fargen til skrifta og den euklidske avstanden kan beregnes. Man er da avhengig av at skrifta er ensfarget og har en unik belysning. En bedre måte vil derfor være å foreta en kantdeteksjon.

2.8.1 Kantdeteksjon

For å finne konturen til et objekt, må man finne kantene til objektet [54, 27]. De mest kjente kantdeteksjonsoperatorene er Robert, Sobel, Prewitt og FreiChen [55, 28, 56, 57, 58]. En kant forekommer ved at det er store forskjeller i nærliggende pikselverdier. For å gjøre en presis kantdeteksjon, er man avhengig av skarpe kanter. Forskjellene kan enten være i farge eller lysstyrke. De fire operatorene Sobel, Roberts, Prewitt og FreiChen har til felles at de foretar en 2-dimensjonal romlig gradientmåling i et bilde og forsterker store frekvensforskjeller som tilsvarer kanter. Det blir brukt to matriser til å detektere kanter. Gx detekterer horisontale kanter og Gy detekterer vertikale kanter. Når man så skal slå sammen horisontale og vertikale kanter brukes følgende formel:

$$|G| = \sqrt{Gx^2 + Gy^2} \tag{2.1}$$

Etter at kantene er detektert må bildet terskles for å begrense hvor mange kanter som skal representeres i bildet.

Sobel

Sobel operatoren egner seg best til å detektere tilnærmet absolutt gradienter i gråtonebilder [59].

-1	-2	-1
0	0	0
+1	+2	+1

Tabell 1: Gx

-1	0	+1
-2	0	+2
-1	0	+1

Tabell 2: Gy

Roberts

Roberts-operatoren har et mindre effektivt område enn de andre maskene. Dette skyldes at det står 0 de fleste stedene i matrisen. Resultatet av dette er at denne operatoren blir mindre følsom for støy.

0	0	-1
0	1	0
0	0	0

Tabell 3: Gx

-1	0	0
0	1	0
0	0	0

Tabell 4: Gy

Prewitt

Prewitt operatoren er mer følsom for vertikale og horisontale kanter enn diagonale kanter. Denne operatoren er derfor ikke så isotrop som Sobel operatoren. Isotrop vil si at operatoren har like egenskaper i alle retninger i bildet.

-1	-1	-1
0	0	0
+1	+1	+1

Tabell 5: Gx

-1	0	+1
-1	0	+1
-1	0	+1

Tabell 6: Gy

FreiChen

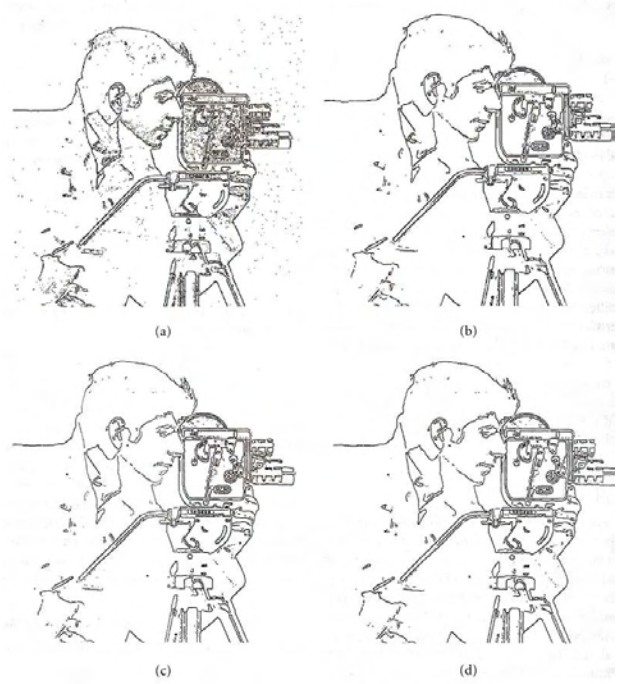
0	0	-1
$\sqrt{2}$	0	$\sqrt{2}$
0	0	-1

Tabell 7: Gx

-1	$-\sqrt{2}$	-1
0	0	0
1	$\sqrt{2}$	1

Tabell 8: Gy

Figur 18: *Orginal kamerabilde, 512x512 piksler [15]*

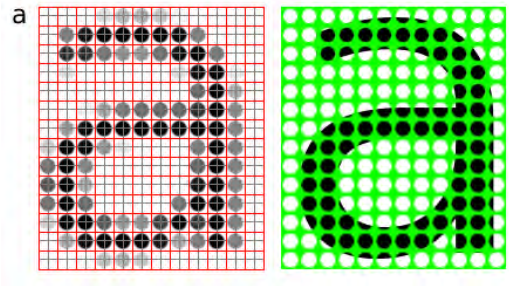


Figur 19: *Sammenligning av kantdeteksjoner ved bruk av forskjellige gradient operatorer. (a) 3x3 Roberts, (b) 3x3 Prewitt, (c) 3x3 Sobel (d) 3x3 FreiChen. [15]*

2.8.2 Håndskrift

Håndskrift setter store krav til oppløsning for at den skal kunne detekteres og bli godt leselig. Siden håndskrift inneholder mange små detaljer, vil som regel ikke en kantdeteksjon gjengi alle kantene i teksten. Teksten vil da kunne bli usammenhengende og mangler bruddstykker. For å tette slike hull, kan det brukes morfologiske operasjoner [60]. Dette skjer ved at det blir brukt en valgt matrise som utvider strekene slik at små hull tettes. Linjene blir da tykkere. Når man så etterpå fjerner små pikselrepresentasjoner, vil støy bli fjernet og mønsteret vil bli sammenhengende. Lukking av kanter fungerer bra i mange tilfeller, men siden håndskrift ikke har noe logisk mønster for hvilke kanter som skal kobles sammen, vil ikke alltid resultatet kunne forbedres.

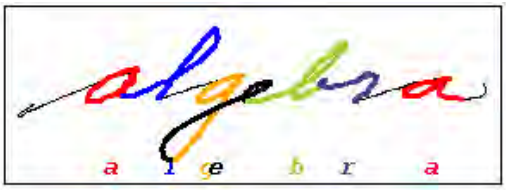
Siden oppløsningen i DV ikke er særlig god, og skrifta som blir skrevet på tavla er liten, vil det ikke være mulig å få en på langt nær så god representasjon av bokstavene som vist i figur 20.



Figur 20: *Pikselrepresentasjon av bosktaven a[61]*

2.8.3 Skriftgjenkjenning

I ”The UNIPEN Project” er det gjort mye arbeid på gjenkjenning av håndskrift. Det er utviklet en stor database som inneholder mange forskjellige typer håndskrift. Prosjektet er publisert på http://hwr.nici.kun.nl/unipen.



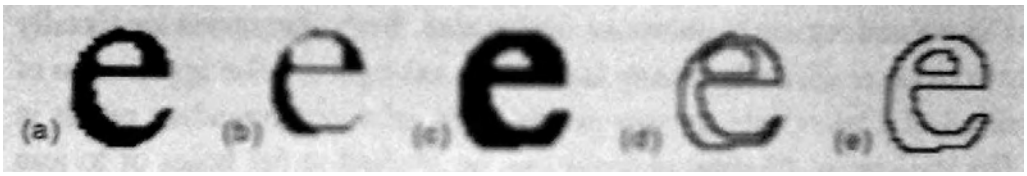
Til skannere medfølger det programvare som konverterer håndskrit til digital skrift. Resultatet er svært varierende, og det settes store krav til at håndskrifta er pen og nøyaktig og at ordene ikke er skrevet i kursiv. Teksten som skrives på tavla, tilfredsstiller sjeldent noen av disse kravene, men det vil være spennende å se resultatet av en skriftgjenkjenning.

2.8.4 Morfologiske operasjoner

Morfologiske operasjoner brukes vanligvis til å fjerne støy, men kan også brukes til forbedring av tekst. Når det foretas deteksjon av skrift blir nødvendigvis ikke 100% av skrifta detektert. Det vil da mangle bruddstykker i enkelte bokstaver. For oss mennesker er det som regel enkelt å skjønne sammenhengen i bokstavene, men for et skriftgjenkjen-

ningsprogram kan små bruddstykker gjøre at skriften ikke lar seg gjenkjenne. Slike bruddstykker kan tettes ved bruk av morfologiske operasjoner. Det finnes i utgangspunktet 2 grunnleggende morfologiske operasjoner, dilation(utvidelse) og erosjon. Figur 21 viser resultatet av dilation og erosjon. Avhengig av hva den morfologiske operasjonene skal gjøre kan det brukes ulike matriser. Selve operasjonen er ingen matrise, men operasjonen jobber med såkalte strukturelementer som er matriser. Vanligvis er matrisene forhåndsdefinert (kule, diamant, skive, kvadrat, linje), men de kan også defineres av brukeren. Stukturelementene er filtermasker S(i,j) med størrelsen k1 x k2 hvor koeffisientene har binære verdier [62, 60].
For å tette de små bruddstykkene som mangler i tekst kan det utføres en closing. Dette er en kombinasjon av de to operasjonene. Det utføres da først en dilation etterfulgt av en erosjon. Dette vil gi den effekten at pikslene blir addert slik at strekene i teksta blir tykkere og mer sammenhengende. Erosjonen fører til at pikslene blir fjernet slik at bokstavene får tilbake normal tykkelse, men strekene er mer sammenhengende. Closing brukes blant annet til å fjerne støy i små områder, koble sammen nærliggende områder, utjevne grenser.

Opening er det motsatte av closing. Men utfører da først erosjon etterfulgt av dilation. Strekene blir da gjort tynnere før de så får igjen normal størrelse. Opening brukes blant annet til å utgjenvne grenser, gjøre smale avgrensninger tydeligere og fjerne støy.



Figur 21: *(a) orginalbilde, (b) etter en iterasjon erosjon, (c) etter en iterasjon dilation, (d) forskjellen mellom (a) og (b) som viser piksler som er fjernet av erosjon, (e) forskjellen mellom (a) og (c) som viser piksler som er lagt til med dilation. [62]*

2.9 Reduksjon av data i video

Digital video tar svært mye plass og er derfor krevende å laste ned fra Internett. Figur 22 viser hvilke krav som skal til for å overføre forskjellige typer video.

2.9.1 Subsampling

Siden forelesningsfilmene er i DV-kvalitet er de tidkrevende å laste ned for studenter og forelesere i ettertid. Videofilemen må derfor reduseres i størrelse før de kan overføres. Dessuten tar videobehandling tid og krever mye prosessorkraft. Før den krevende videoprosesseringen starter, kan det foretas beskjæringer for å redusere datamengden som må behandles. En teknikk for å foreta bevegelsesrammeuttrekning er presentert av Tonomura, Akutsu, Otsuji og Sadakata [63]. Rammer blir her trukket ut med et jevnt intervall. Ulempen med denne metoden er at den ikke tar hensyn til hvor mye bevegelse som forkommer. Zhang, Low og Smoliar [64] har utviklet en bedre metode der det foretas en sammenlikningsbasert uttrekking ved bruk av histogrambasert analyseteknikk.

Avi-filmer i PAL har en oppdateringsfrekvens på 25 bilder i sekund. Grunnen til den høye frekvensen, er for å gi en god gjengivelse av raske bevegelse og unngå flimring. Når det er lite bevegelse i filmen, er det ikke nødvendig å oppdatert bildet så ofte. Siden

TABLE 1 Digital video application requirements

Application	Bitrate Req.	Distortion Req.	Transmission Req.	Computational Req.	Standards Req.
Network video on demand	1.5 Mbps 10 Mbps	High medium	Internet 100 Mbps LAN	MPEG-1 MPEG-2	MPEG-1 MPEG-2 MPEG-7
Video phone	64 Kbps	High distortion	ISDN p × 64	H.261 encoder H.261 decoder	H.261
Desktop multimedia video CDROM	1.5 Mbps	High distortion to medium	PC channel	MPEG-1 decoder	MPEG-1 MPEG-2 MPEG-7
Desktop LAN videoconference	10 Mbps	Medium distortion	Fast ethernet 100 Mbps	Hardware decoders	MPEG-2, H.261
Desktop WAN videoconference	1.5 Mbps	High distortion	Ethernet	Hardware decoders	MPEG-1, MPEG-4, H.263
Desktop dial-up videoconference	64 Kbps	Very high distortion	POTS and internet	Software decoder	MPEG-4, H.263
Digital satellite television	10 Mbps	Low distortion	Fixed service satellites	MPEG-2 decoder	MPEG-2
HDTV	20 Mbps	Low distortion	12-MHz terrestrial link	MPEG-2 decoder	MPEG-2
DVD	20 Mbps	Low distortion	PC channel	MPEG-2 decoder	MPEG-2

Figur 22: *Krav til ulike typer videoformat [15]*

antall bilder per sekund er konstant, tar ikke avi-formatet hensyn til hvor mye bevegelse som forekommer. Ved å sammenligne rammene kan man finne ut når det er bevegelse i filmen. Er det stor forskjell mellom to etterfølgende rammer, har det som regel vært et sceneskift i filmen [17, 65].
Når små objekter i video er i bevegelse, er forskjellen mellom to etterfølgende rammer liten. Forskjellen blir da først synlig etter litt tid. Ved å sammenligne en ramme med rammer utover i filmen til det er ønsket differanse blir overgått, kan man detektere rammer som viser bevegelse i filmen. Gunsel og Tekalp [37] beregner en differanseverdi mellom gjeldende ramme k og de N foregående rammene. Dette blir gjort ved å sammenlikne fargehistogrammet til ramme k og det gjennomsnittlige fargehistogrammet til de N foregående. I denne oppgaven blir fargehistogrammet til ramme k sammenlignet med N etterfølgende rammer til ønsket differanse overstiges. Av rammene som blir beskrevet som bevegelsesrammer, vil det genereres en ny video av, og det er kun disse rammene som senere er nyttig å behandle. Hvis det genereres en ny film som kun består av rammene med bevegelse, vil hastigheten til handlingsforløpet i videoen variere avhengig av hvor mye bevegelse som forekommer. Å forandre hastigheten på handlingsforløpet vil i få tilfeller være en ønsket løsning. Det vil derfor måtte opprettes en matrise som har pekere til bevegelsesrammene.

2.9.2 Beskjæring av areal

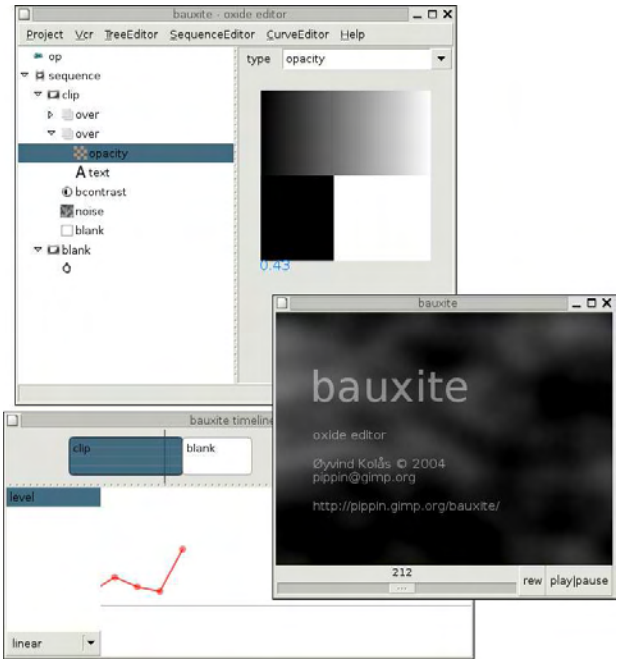
Beskjæring av bilder foretas vanligvis manuelt. Å foreta en avgrensning manuelt er den enkleste metoden også i video. Skal beskjæringen derimot foretas automatisk, er man avhengig av en maske eller referansepunkter som avgrenser området som skal ivaretas. Dersom det er filmet med et stillestående kamera, holder det å generere en maske til første ramme i filmen, og beskjære alle andre rammene med denne maska som mal. I

mange tilfeller vil en beskjæring redusere en film mye i størrelse. Ta for eksempel figur 26. Tavla utgjør i denne videoen under 1/3 av hele bildet. Etter beskjæring vil den videre behandlingen av filmen og nedlasting kunne foretas på en 1/3 av tiden.

2.10 API-Oxide

Øyvind Kolås som er ansatt på HiG, har utviklet et rammeverk for videoprosessering kalt Aluminium. Dette ligger tilgjengelig på http://pippin.gimp.org. Rammeverket inneholder funksjonalitet for behandling av bilder og video. Oxide er en komposisjonering og animasjonsmotor som er bygd i gggl. Gggl er ferdigutviklede bibliotekfiler for bilde- og videobehandling. APIet Aluminium er linuxbasert og er skrevet i C-kode.

Bauxite er det grafiske brukergrensesnittet til Oxide 23. Når C-koden og xml-filene er ferdig utviklet, kan disse åpnes i Bauxite og man kan gjøre en tilpassning av terskelverdier.



Figur 23: *Baxite [61]*

2.10.1 Kvalitetstap

Når det genereres videoer i Oxide brukes det en MPEG-kodek. Problemet med MPEG er at videofilene blir komprimert for hver gang det gjøres en prosessering på filmen. Siden denne komprimeringen ikke er tapsfri, vil filmene bli dårligere og dårligere for hver gang de behandles. Det finnes ingen metode for å generere tapsfri video i rammeverket, men dette problemet kan løses på to måter:

- det kan genereres en png-bildestrøm
- eller all videobehandlinga kan foretas i en og samme algoritme før videofila genereres

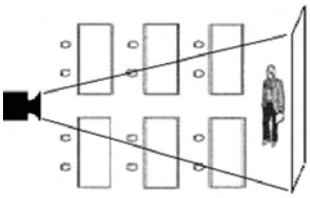
Ved å lagre rammene i videoen midlertidig i bilder i png format vil man beholde originalkvaliteten. Dette skyldes at png er en tapsfri komprimeringsform. Figur 24 viser hvordan kvaliteten reduseres etter flere iterasjoner i JPEG [66]. Kvalitetstapet vil bli

på samme måte i MPEG siden begge metodene er blokkbaserte.

Figur 24: *Bildet viser hvordan kvaliteten reduseres for hver gang bildet regenereres i JPEG kvalitet. Kvalitetsreduksjonen synes spesielt godt i røde og grønne områder [61]*

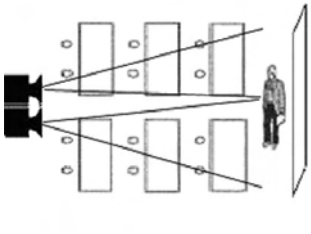
3 RGB

For å få en god skriftrepresentasjon, må filmene som behandles være i DV-kvalitet. I dette prosjektet settes det et krav til at filmene er tatt opp med et stillestående kamera, slik at de har en statisk bakgrunn. Det er tatt utgangspunkt i 3 filmer (figur 26, 28, 29) som er opptak gjort av forelesninger ved skolen. Filmene har ulik belysning og kameraene gjengir tavla i forskjellig størrelse. Ulik belysning gir mange utfordringer og programmet som skal utvikles er nødt til å kunne håndtere dette.

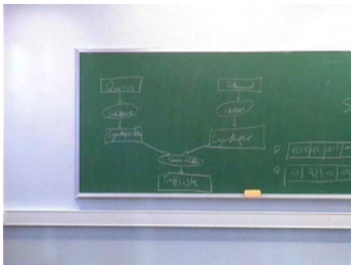


Figur 25: *Filming med et kamera[67]*

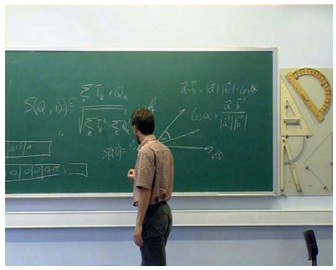
Figur 26: *Hele tavla*



Figur 27: *Filming med to kameraer[67]*



Figur 28: *Venstre*



Figur 29: *Høyre*

Første videofilm jeg tok utgangspunkt i, er en oversiktsfilm av hele tavla (figur 26). Kameraet står bakerst i klasserommet og lesbarheten til skrifta er ikke særlig god. For få bedre leselighet, blir skrifta mye tydeligere dersom det brukes to kameraer som filmer hver sin halvdel av tavla. Disse to tavlevideoene vil avslutningsvis måtte spleises sammen igjen for å få sammenhengende tekst [67].

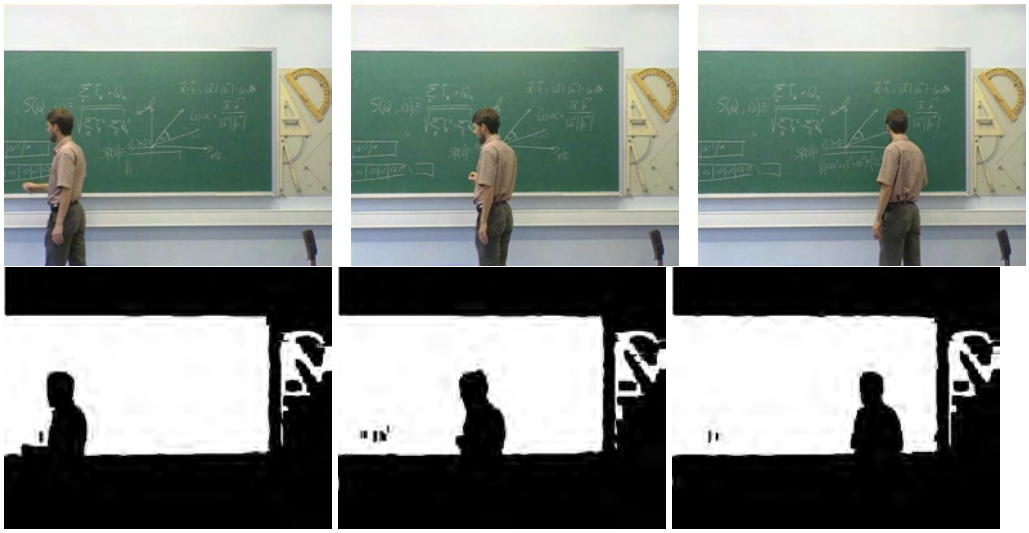
For å teste om det var mulig å digitalisere håndskrift i video lagde jeg en ny film. Tavla dekker i denne filmen hele skjermbildet og skrifta får dermed i høyere oppløsning.

3.1 Generering av masker

3.1.1 Eksperimentoppsett

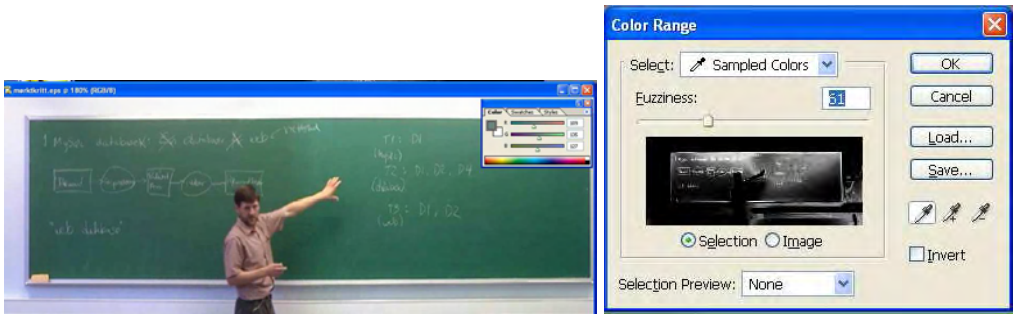
Binære masker blir i denne oppgaven brukt til skille tavla og foreleser. I figuren 30 har det blitt generert masker til tre rammer. Hvite felt viser hvilket område som skal oppdateres. Siden foreleseren er i stadig bevegelse vil masken måtte oppdateres for hver ramme som

skal behandles. Blir ikke masken oppdatert, vil personen bevege seg ut av det mørklagte området han opprinnelig stod inne i og han vil bli en del av det hvite tavleområdet ikke blir segmentert vekk. For å foreta objektsegmentering blir nemlig kun det hvite området i hver maske oppdatert, mens det sorte området forblir uforandret.

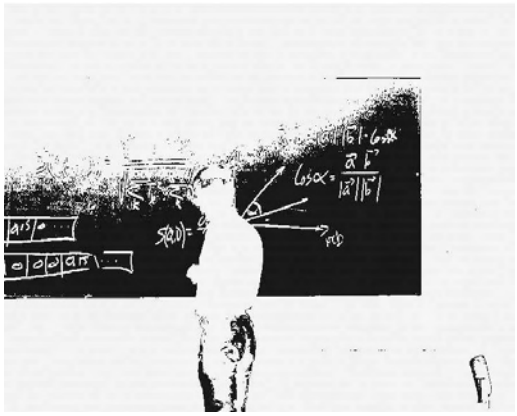


Figur 30: Rammer med tilhørende masker

Når det skal genereres en maske, kan det tas utgangspunkt i fargesammensetningen i et piksel. Slike fargemålinger kan f. eks gjøres i Adobe Photoshop (figur 3.1.1). Siden det er tavla som skal bevares i videoen, må det tas en måling innenfor tavleområdet. Målingen gir RGB-verdier fra 0 til 255. Ved å sammenligne de målte verdiene med verdiene i alle andre piksler i bildet, kan man få generert en maske. Utregningen er beskrevet i kapitel 2.6.1. Fargeverdiene innenfor tavla vil ha en liten avstand siden disse ligger nær den grønnfargen det ble tatt utgangspunkt i. Til høyre i figur 3.1.1 kan man se hvilke fargeavstander resten av pikselen i bildet har til den målte verdien. Resultatet kan lagres i et gråtonebilde. Punkter som har fargeverdier som ligger tett ved hverandre får en lav gråtoneverdi og dermed et mørkt område i bildet. For å sette et krav til hvor høye gråtoneverdiene skal være for at de skal inkluderes i maska, må det settes en terskelverdi. Alle punkter som er mørkere enn en valgt grense settes helt sorte, resten settes lyse. Dette er vist i figur 32.

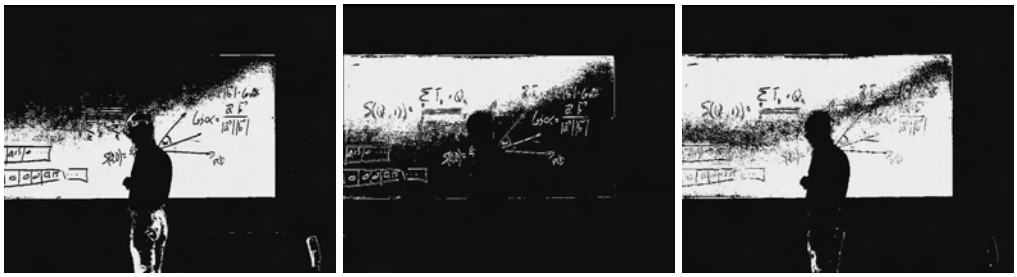


Figur 31: *Fargermålinger i Adobe Photoshop som gir gode overlappende masker*



Figur 32: *Kun Delvis markering av tavleområdet*

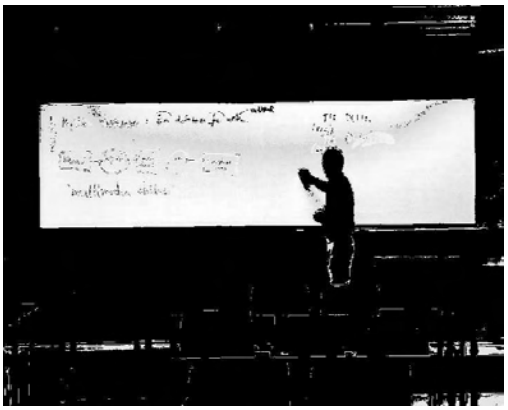
For at maska skal bli bedre visuelt synlig på hvitt ark, blir alle masker videre i rapporten invertert. Det vil si at pikslene som er hvite settes sorte og omvendt. I figur 32 har maska en stor mangel; kun deler av tavleområdet er inkludert. Det øvre tavleområdet som ikke inngår i maska vil derfor ikke bli oppdatert. Grunnen til at ikke øver del av tavla har blitt inkludert i maska, skyldes belysningen over tavla. Lyset gjør at grønnfargen i det opplyste området har høyere fargeverdier og dermed en større fargeavstand. Dette er vanskelig å se i bildene, men det får stort utslag i maskene. For å få en fullstendig maske må det foretas en ny fargemåling i det opplyste området og de to maskene må settes sammen. I disse videoene er det hovedsakelig en lyskilde som påvirker maskene. I tilfeller der det er flere lyskilder må det utvikles en maske for hver av dem og alle maskene må slås sammen.



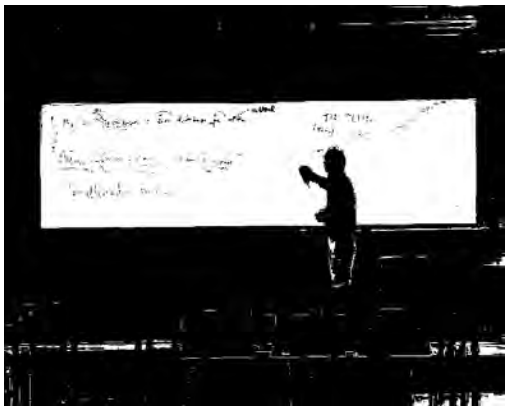
Figur 33: *Sammenslåing av masker av ulikt belyste områder*

For å sette sammen maskene må terskelverdiene tilpasses. Bildet til høyre i figur 33 viser en maske der terskelverdiene ikke er tilpasset. Terskelverdiene er en grense for hvor langt ifra de målte verdiene resten av fargene i bildet kan ligge. Ved å sette et lavere terskelkrav, vil altså det hvite området i figur 33 utvides. Det er viktig å være klar over at siden det er tatt utgangspunkt i to forskjellige fargeverdier vil også terskelverdiene være forskjellige. Setter man en for lavt terskelkrav, vil det bli inkludert store områder som ikke er tavla. Buksa til læreren har en ganske lik grønnfarge som tavla, men det er viktig at terskelverdien avgrenser disse to områdene.

I figur 34 og 35 det tatt utgangspunkt i grønnfarge for belyst og ikke belyst tavleområde, og terskelverdiene er tilpasset slik at maskene gir en god overlapping, uten at det blir mye representasjon fra området rundt. Forskjellen på bildene er at det ene er i gråtoner, mens det andre er i sort/hvitt. Gråtonebildet gir, i tillegg til maskeringen, et mål på hvor godt fargene passer til referansefargene. Dette ble benyttet under objektsegmentering i neste kapittel. Gråtoneskalaen ble brukt til å angi hvor mange prosent av det maskerte området skulle oppdateres. Dette viste seg å gi en dårligere oppdatering enn sort/hvit bildet og gråtonebildene blir derfor ikke brukt. Gråtonebilder er vist i figur 34 og resultatet etter segmentering er vist i diagrammet i figur 51.



Figur 34: *Gråtonemaske*



Figur 35: *Sort/hvitt-maske*

3.1.2 Resultat

Figur 36: *Det markerte tavleområdet vises i hvitt. Det er foretatt to fargemålinger og den euklidsk fargeavstand pikslene i bildet har til disse fargene, ble beregnet. De to resulterende bildene ble så slått sammen og overlappingen er justert med to terskelverdier. Det binære bildet ble til slutt invertert.*

Videoen som skal gjøre en kontinuerlig maskering av tavla, mens personen beveger seg, inneholder en bildeserie med lignende binære bilder.

Romlig medianfiltrering

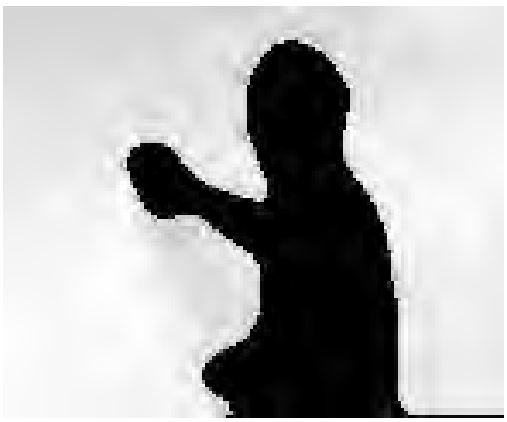
For å fjerne små hull og støy ble maskene medianfiltrert. Det ble brukt matriser av forskjellig størrelse, men et naborom på 3x3 piksler(figur 37) gav best resultat etter objektsegmentering.

Figur 37: *Romlig medianfiltrering av figur 34 med en 3x3 matrise.*

Figur 38: *Romlig medianfiltrering av figur 34 med en 9x9 matrise.*

I figur 34 er det markert mange små områder som ikke tilhører tavlen. Dette må betraktes som støy. Medianfiltrering egner seg godt til å fjerne små støypartikler, men store støyområder vil bli utvidet istedenfor å bli fjernet (sammenlign området øverst i figur 34 og 37).

Figur 39: *Masken inkluderer noen små områder inne i personens kontur. Dette vil bli støy etter segmentering*



Figur 40: *Blir masken medianfiltrert vil større deler av personen bli inkludert i masken og dette vil skape mer forstyrrelse*

Teksten blir ikke inkludert når man tar utgangspunkt i tavlefargen (figur 34). Slike tynne tekstlinjer egner medianfiltrering seg godt til å inkludere, men medianfiltrering kan ikke benyttes for å fjerne støy i masker i denne oppgaven. Grunnen til dette er at man får en mindre nøyaktig avgrensing av personen i bildet. Når objektsegmentering skal foretas, vil det fra masken i figur 39 bli litt støy fra markering inne i personens kontur. Etter at medianfiltreringen er foretatt (figur 40), blir omrisset mindre nøyaktig og man

får en økt feilrepresentasjon etter segmentering (se resultat i figur 51).

Kromatisitet

I tidligere beskrevet metode blir det tatt utgangspunkt i en fargeverdi for hvert av de ulike belyste tavleområdene og terskelverdiene må tilpasses for å få en maske som dekker hele tavla (figur 33). Dette er en lite adaptiv metode som krever mange finjusteringer for å fungere bra. Ved å bruke kromatisitet som forklart i kapittel 2.6.2,vil man ikke behøve å ta hensyn til intensitetsforskjeller. I figur 41 er det tatt utgangspunkt i figur 29 generert et nytt bilde. Ved å kun foreta en fargemåling av tavlen, beregne den euklidske avstanden og terskle resultatet, vil man få generert en maske som dekker hele tavlen (figur 42).

Figur 41: *Ensfargede områder som tavlen har i dette bildet lik kromatisitet*

Figur 42: *I bildet er det kun tatt utgangspunkt i en fargeverdi innenfor tavlen. Som man ser blir nå hele tavleområdet inkludert selv om det opprinnelig hadde ulik belysning*

3.1.3 Diskusjon

Maske

Det binære resultatbildet av fargeavstanden gir et godt skille mellom tavla og personen. At mye av buksa til foreleseren blir representert, skyldes at fargen på buksa hans er svært lik grønnfargen på tavla. Siden buksa befinner seg under tavlehøyde, vil ikke dette påvirke tavlas innhold. Problemet med maska er at noe av håret til foreleseren blir markert. Når segmenteringen skal foretas er det nemlig meget viktig at maska gir en eksakt avgrensning av objektet i bevegelse. Selv om bare små områder av håret hans er synlig, vil dette virke forstyrrende siden det er det eneste som beveger seg i filmen etter segmenteringen, og det vil skape problemer når teksten skal detekteres.
Tavla er sammensatt av to elementer

- selve tavla
- og skrifta som legges som et lag på tavla

I resultatet er det tatt utgangspunkt i grønnfargen på tavla, og det er sett bort ifra skrifta. Grunnen til dette er at skrifta inneholder mange fargenyanser mellom grønn tavlefarge og kritthvit skrift og dette problemet viser seg å gi så mye støy at det får en negativ virkning. For å få markert all skrifta, må det på samme måte som for tavla, utvikles to nye masker. En for kritt i normalt belyst område og en for kritt i belyst område. For å

få inkludert all skrifta med forskjellige nyanser må det settes et lavt krav til terskelverdi, slik at store varianser blir inkludert. Problemet som da oppstår at man får mye feilrepresentasjon i maskene (se figur 43 og 44)

Figur 43: *Krittmaske for normalt belyst tavleområde*

Figur 44: *Krittmaske for kritt i belyst område*

Skriftmaskene viser seg å inneholde mye støy. Slår man sammen skriftmaskene og den kombinerte tavlemaska, får man en totalt en mye dårligere avgrensning enn om man bare bruker skriftmaskene. Krittmaskene ble derfor ikke brukt for å fjerne objekter i bevegelse. Å detektere skrift krever mye arbeid. Hvordan dette kan gjøres, er beskrevet nærmere i kapittel 6.
Selv om markeringen av tekst ikke blir benyttet i resultatet, betyr ikke det at teksten ikke blir oppdatert etter hvert som foreleseren skriver. Siden det genereres en tilhørende maske til hver ny ramme som skal behandles, vil den nye maska oppdateres både når det bli skrevet ny tekst og når tekst pusses vekk.

Terskelverdier

Det optimale ville vært at programmet algoritmisk fant de beste terskelverdiene. Grunnen til at dette ikke har blitt gjort, er at i de 3 videoen som har blitt brukt befinner foreleseren bildet til enhver tid. Det kan derfor ikke hentes et tomt kalibreringsbilde som den algoritmiske tilpassingen kan forsøke å oppnå.
For å generere maske som håndterer ulikt belyst område, må det i dette tilfellet utvikles følgende verdier:

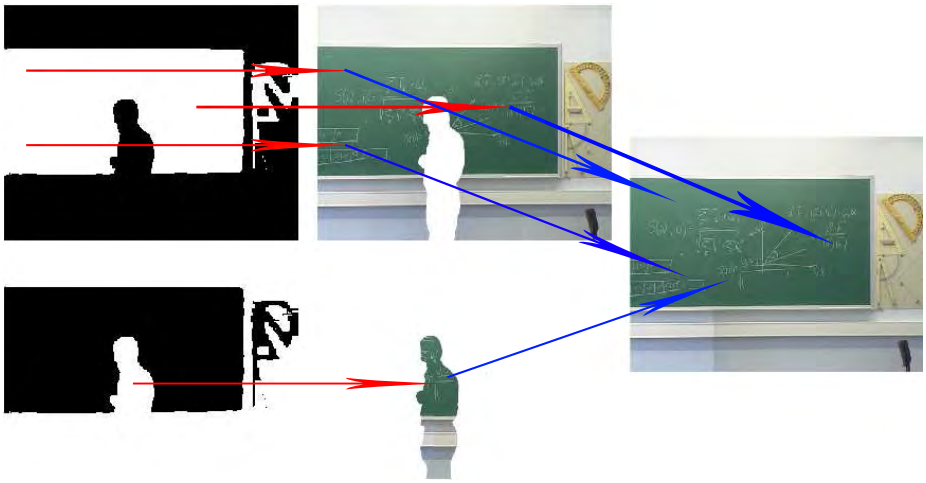
1. rgb-verdier for tavlefarge i normal og opplyst område
2. terskelverdier for de to maskene

Ved å justere punkt 2, får man maskene i figur 33 til å overlappe hverandre passelig mye uten at det kommer for mye representasjon av andre elementer. Har nemlig foreleseren klær som er lik grønnfargen på tavla, vil han bli en del av tavla hvis denne verdien er satt for høy. Klesfargen hans vil tolkes som tavlefarge. Selv om dette ikke har noen stor påvirkning på tavla vil det legge seg over skrifta på tavla.
At programmet krever mange terskelverdier er negativt. Ved bruk på forskjellige videoer, vil disse verdiene måtte tilpasses, og det vil derfor kreves unødig forarbeid.

Ved bruk av kromatisitet blir tavlen likt belyst og man vil da kun måtte bruke en RGB-verdi og en terskelverdi.

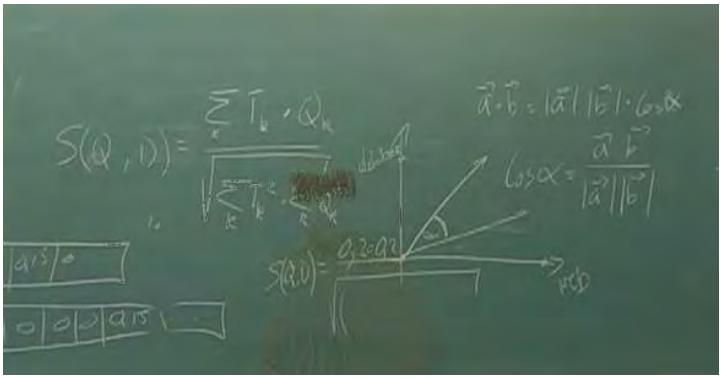
3.2 Objektsegmentering

3.2.1 Eksperimentoppsett



Figur 45: *Øverste rad: Området som er hvitt i maska viser hvilke piksler som skal hentes fra det nye bildet. I det sorte området befinner personen seg. For å få segmentert vekk personen må disse pikslene hentes fra en tidligere ramme hvor foreleseren ikke befinner seg i denne posisjonen. Nederste rad: Området som ikke ble oppdatert hentes fra en tidligere ramme. De røde pilene viser hvilke områder i bildene som er maskert. De blå pilene viser hvordan disse to bildene kan sys sammen til et segmentert bilde.*

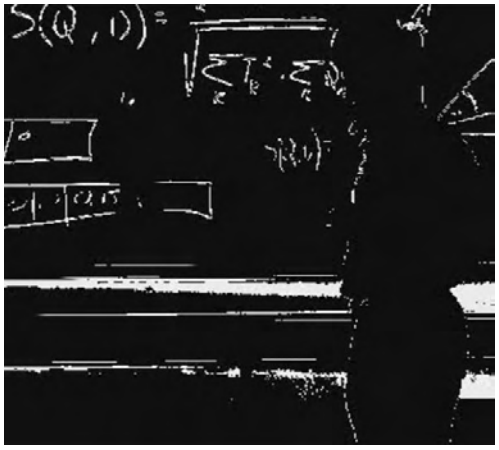
3.2.2 Resultat



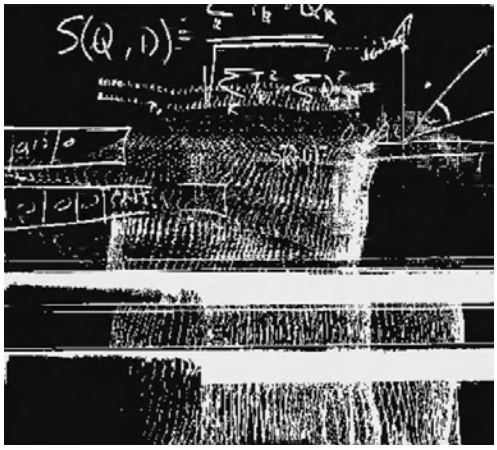
Figur 46: *Objektsegmentering ved bruk av sort/hvitt-maske generert ved bruk av fargeavstand*

3.2.3 Diskusjon

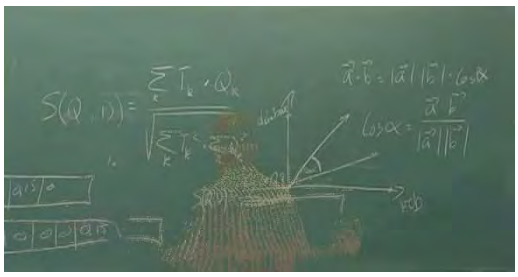
Selv om de genererte fargene ser ut til å gi en god begrensning av personen, blir noe av konturen til foreleseren med i resultatvideoen. Selv om han havner bak teksten, er det forstyrrende å se han i bakgrunnen. For å visualisere konturforstyrrelse har det blitt tatt utgangspunkt i tekstfarge og generert to sort/hvit-masker (figur 47 og figur 47). Denne forstyrrelsen har det bli gjort forsøk på å fjerne i kapittel 4.3.



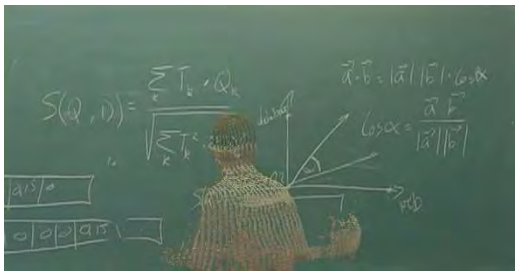
Figur 47: *Visualisering av objektkontur*



Figur 48: *Objektkontur over tid*

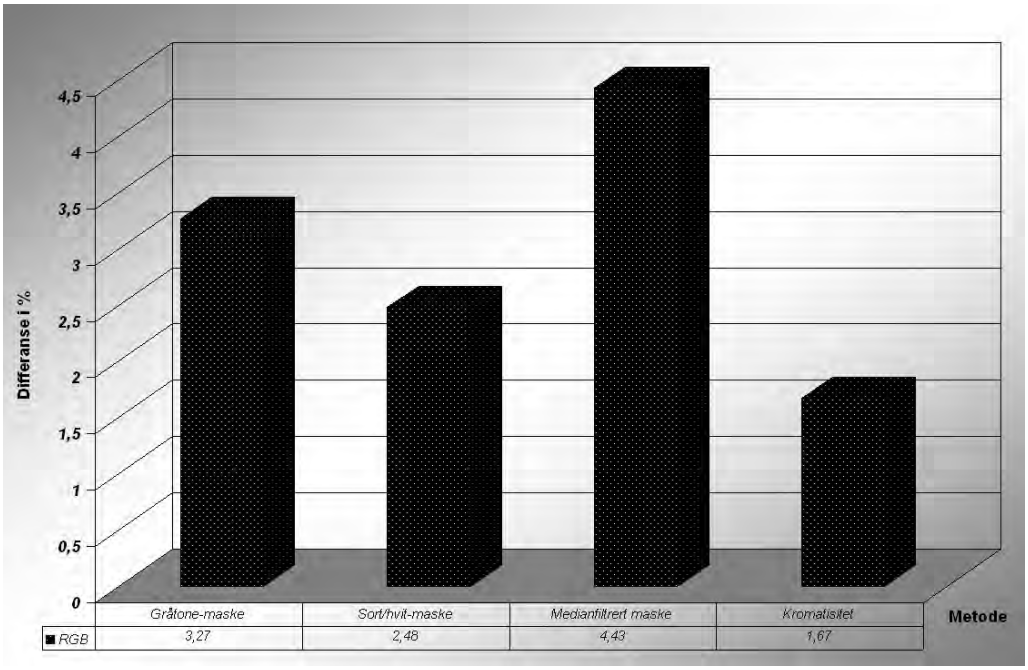


Figur 49: *Objektsegmetnering ved bruk av gråtone-maske*



Figur 50: *Objektsegmentering ved bruk av median-filtrert maske*

I resultatvideoene ser man tydelig at det blir mye forstyrrelse av omrisset til personen ved bruk av den medianfiltrerte maska. En konvertering av maska fra gråtoner til fullstendig svart/hvitt viste seg og gi en klarere avgresning og dermed mindre støy i resultatvideoen.



Figur 51: *Sort/hvit-masken gir minst støy av objektet i bevegelse. Bruk av gråtonemaske gir noe mer støy, mens en medianfiltrert maske egner seg dårligst til segmentering.*

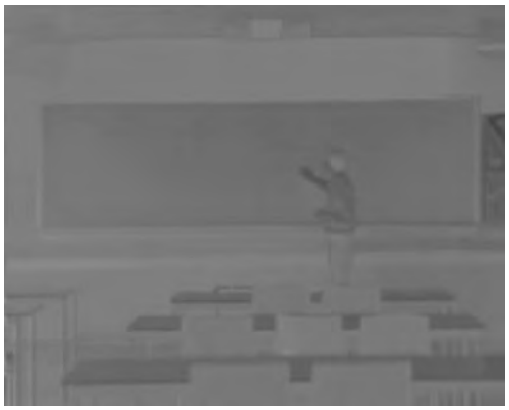
4 YCbCr

4.1 Generering av masker

4.1.1 Eksperimentoppsett

Ved bruk av RGB-verdiene, kan man regne ut Cb- og Cr-komponentene som forklart i 2.1. For hvert enkelt piksel må det beregnes en tilhørende verdi. Generering av bilder av Cb- og Cr-komponentene til figur 52 gav resultatet vist under.

Figur 52: *Originalbilde*



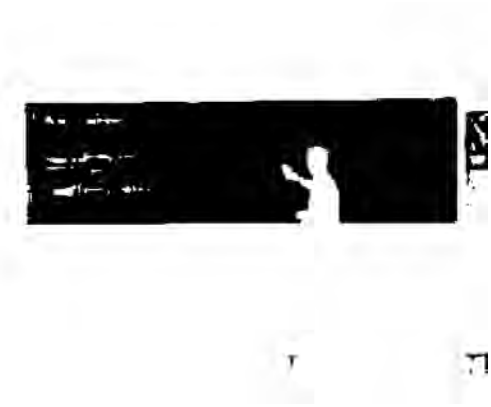
Figur 53: *Cb-komponent*

Figur 54: *Cr-komponent*

Ved å addere gråtonene i hver av bildene, får man resultatet vist i figur 55. Dette bildet må, på samme måte som fargemaska, terskles for å gi et godt resultat. Siden tavla er det mørkeste området i figur 55 må det detekteres en terskelverdi som avgrenser tavla. Denne verdien vil variere avhengig av fargen på det som skal terskles.

4.1.2 Resultat

Figur 55: *Addering av Cb- og Cr-komponentene*



Figur 56: *Tersklet og invertert CbCr-maske*

For å gi en god avgrensing måtte bilde 55 terskles. Ved å sette alle fargeverdier som er mørkere enn en valgt grense helt hvite og resten sort, blir bildet tersklet og invertert. Resultatvideoen ligger på http://studweb.hig.no/980141/videofiler.rar

4.1.3 Diskusjon

En CbCr maske gir en god avgrensning av området og det viser seg å bli mindre støy enn ved bruk av fargemaske når objektsegmenteringen foretas i neste kapittel.

Figur 57: *Segmentering med RGB-mask*

Figur 58: *Segmentering med CbCr-mask*

CbCr metoden er både enklere og krever færre terskelverdier enn fargedeteksjon og er derfor å foretrekke. Som man ser av figur 56 så blir ikke skrifta inkludert i maska. CbCr-komponentene tar heller ikke hensyn til fargene i bildet. Dette er grunnen til at linjalen til høyre i bildet også blir markert. Terskelverdien for intensitet varierer noe fra film til film. For å foreta en god segmentering, krever programmet en tilpassing av denne verdien. Terskelverdien ble detektert ved å skrive maske til skjerm og justering av verdiene ble gjort for å få best mulig resultat.

Terskelverdien på 94% gir et godt resultat i denne videoen. I de to andre filmene som ble brukt varierer denne verdien med +/- 2%. I figur 59 og 60 er terskelen forskjøvet 2% i hver retning. Figur 60 ser ut til å gjøre en mer nøyaktig avgrensning enn det som

Figur 59: *Piksler med over 92% intensitet er hvit*

Figur 60: *Piksler med over 96% intensitet er hvit*

er vist i resultatet, men dette skulle vise seg å være feil. Settes verdien så høy blir mye av konturen til personen med i bildet og man får mye støy etter at objektsegmentering er foretatt. Resultatene over er fra en ikke beskåret video.

Komprimering

Resultatet av de adderte CbCr-bildene ble dårligere når dette blir gjort på en beskåret video. Figur 62 viser maska etter beskjæring. Denne maska inneholder mange flere feil enn den maska før beskjæring. Grunnen til dette når det genereres nye videoer i rammeverket blir filmene MPEG komprimert som forklart i kapittel 2.10.1.

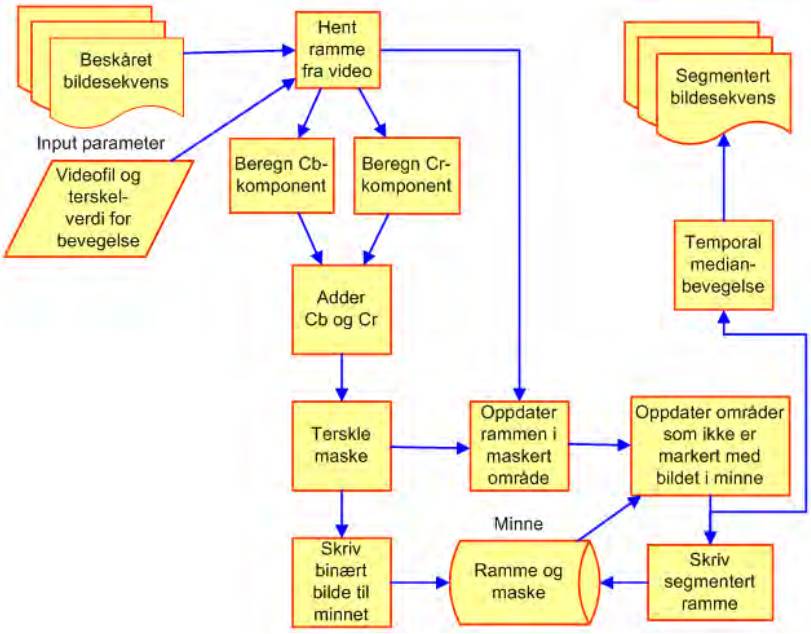
Figur 61: *Addert Cb- og Cr-komponenter etter beskjæring av video*

Figur 62: *Terskelverdien måtte tilpasses slik at piksler med over 96% intensitet blir hvite*

4.2 Objektsegmentering

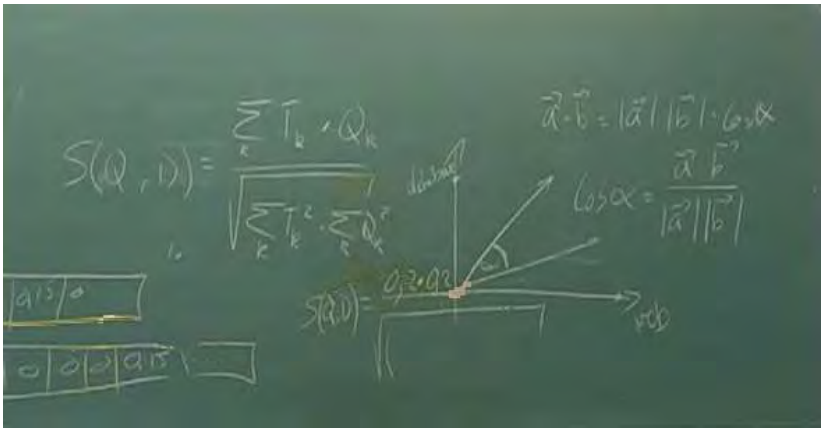
4.2.1 Eksperimentoppsett

Objektsegmenteringen ved bruk av CbCr-maske ble foretatt på samme måte som for RGB-maske (se forklaring i kapittel 3.2.1).



Figur 63: *Flytskjema for objektsegmentering*

4.2.2 Resultat

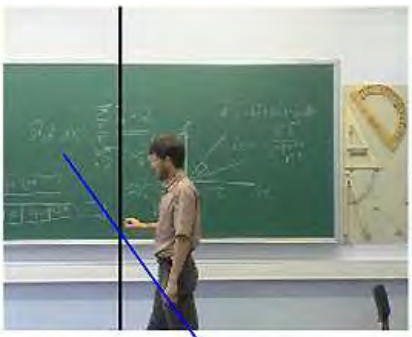


Figur 64: *Objektsegmentering ved bruk av tersklet CbCr-maske*

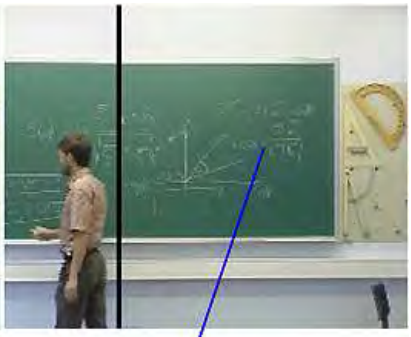
I videofilene ser man tydelig at bruk av CbCr-maske gir best resultat. Fargemasker gir en del støy fra konturen. Videofiler for segmentering ved bruk av fargemasker:
http://studweb.hig.no/980141/videofiler.rar
Segmentering ved bruk av CbCrmasker: http://studweb.hig.no/980141/videofiler.rar

Sammenligning

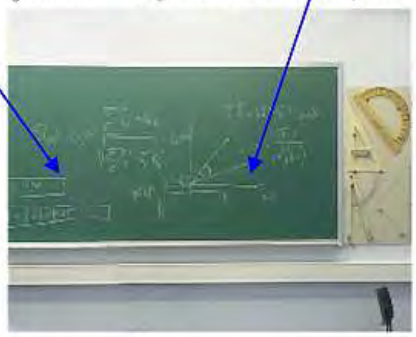
I bildene er det vanskelig å se hvilket resultat som er best. Det ble derfor generert et tilnærmet lik optimalt segmentert bilde for å kunne sammenligne resultatene. Måten dette ble gjort på var at det ble tatt utgangspunkt i to rammer med 2 sekunders mellomrom. Bildene er fra omtrent samme tidspunkt som figur 46 og 64. Det er derfor ingen forandring i tekst som vil kunne påvirke sammenligningen. I bildene beveger personen seg raskt fra høyre til venstre. Ved å finne en vertikal linje som personen passerer kan det av de to objektfrie delene genereres et nytt bilde. Se figur 65.



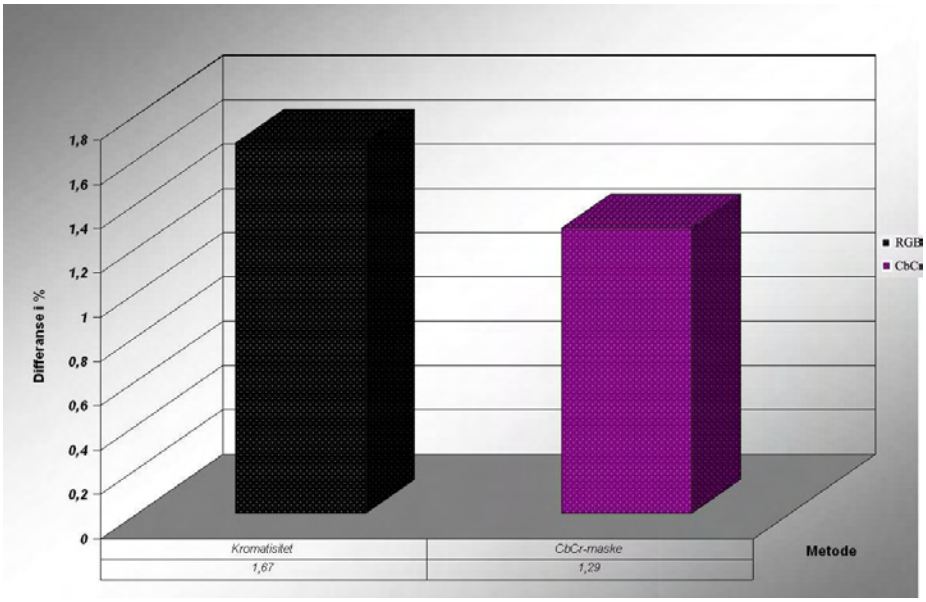
Figur 6.6: *Første sammenligningsramme*



Figur 6.7: *Andre ramme (2 sekunder senere)*



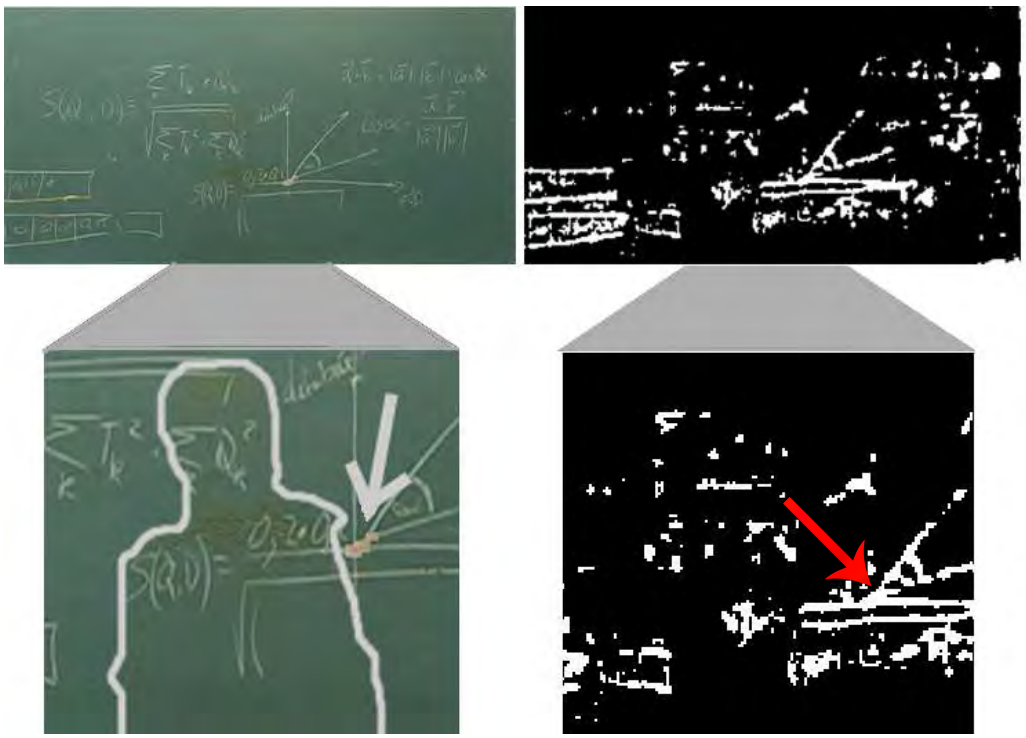
Figur 6.8: *Bildet er sammensatt området til venstre for den sorte streken i figur 6.6 og høyre del av figur 6.7*



Figur 65: *Søylene viser hvor mange prosent av objektene som ikke har blitt segmentert vekk. Som man kan se er CbCr-maskene bedre egnet enn RGB-maskene når det skal foretas segmentering*

4.2.3 Diskusjon

Pilen i figur 66 viser et området som ikke har blitt oppdatert. Dette skyldes at det er hull i maska. Skjortefargen har blitt ivaretatt i dette området. Så lenge maska ikke dekker dette området, vil det ikke bli oppdatert så lenge det er mye skrift i dette området. For å fjerne denne feilkilden må hullene i maska tettes. Å tette hullene ved å øke terskelverdiene i maskene, vil ikke gi noen god løsning. Som omtalt i kapittel 4.1.3 vil maskene inkludere ytterkanten til personen og man får mer støy. Maskene vil derfor måtte tettes ved å gå igjennom og fylle hullene. Hvordan dette kan gjøres er omtalt i neste kapittel. Andre forstyrrelse i maska er at innenfor omrisset av personen synes konturen til personen svakt. Dette skyldes at litt av ytterkanten til personen blir inkludert i maskene. For hver ny ramme som skrives til videofil blir også dette omrisset inkludert. Det vil derfor utgjøre en synlig mengde støy i området personen befinner seg. I kapittel 4.3.3 blir det gjort forsøk på å utvide arealet til personen slik at konturen av han ikke blir inkludert i maska.



Figur 66: *Den røde pilen i masken til høyre viser et område som ikke er inkludert pga. mye tekst. Dette resulterer i at deler av personen ikke blir segmentert vekk i dette området(se grå pil i bildet til venstre). Det grå omrisset rundt personen viser hvor det fortsatt er noe i forstyrrelse i bildet av personkonturen.*

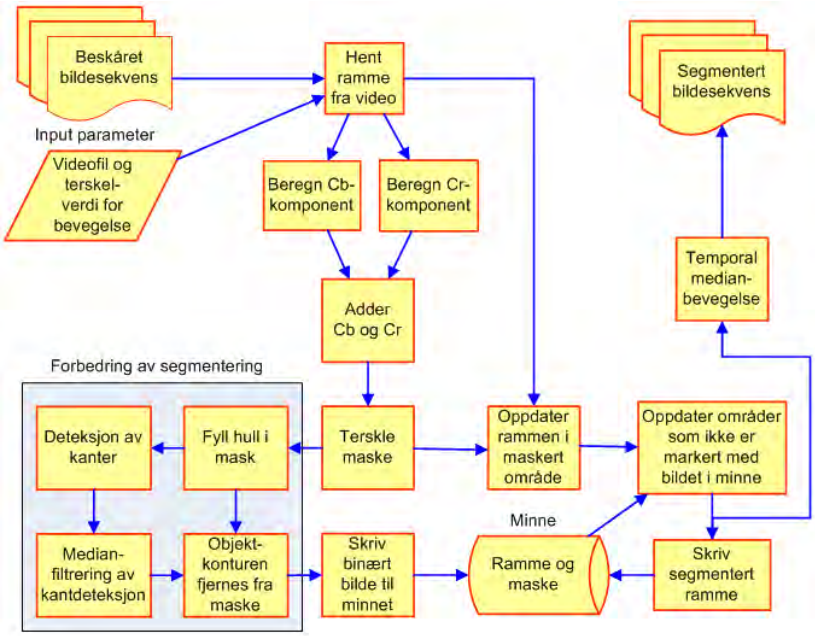
4.3 Forbedring av masker og objektsegmentering

Som vist i figur 65 gir CbCr-masken den beste segmenteringen av objekter i bevegelse. Den blir derfor kun gjort forsøk på å forbedre CbCr-masken og ikke RGB-maskene.
Grunnen til at det fortsatt er litt representasjon av objekter i bevegelse etter segmenteringen skyldes to ting:

- alle områder i tavlen blir ikke oppdatert og

- deler av konturen til objektet i bevegelse blir synlig

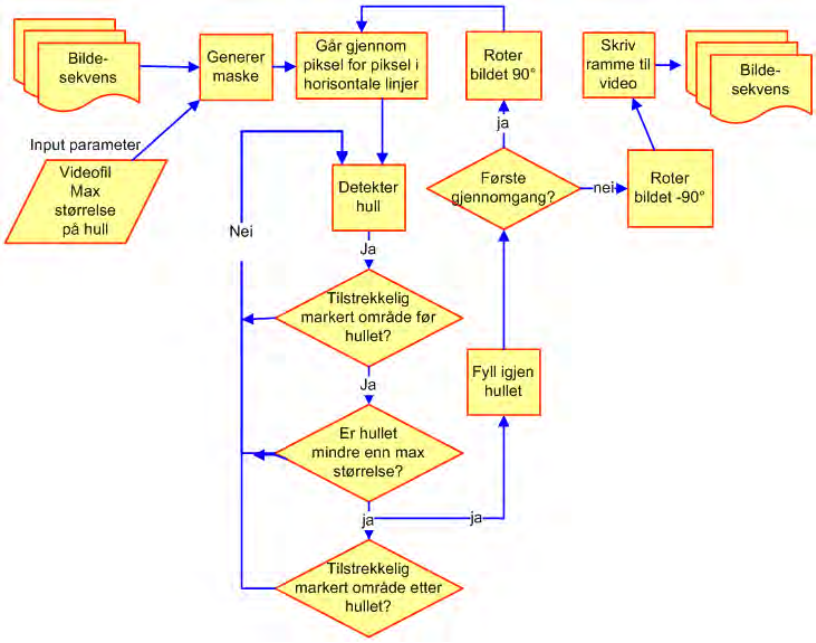
For at alle områder skal oppdateres, vil det i dette kapittelet bli tettet hull i maskene. For å fjerne konturen, vil det bli gjort forsøk på å utvide maskene slik at de ikke inkluderer noe av omrisset til personen.



Figur 67: *Flytskjema for segmentering etter forbedring av maskering*

4.3.1 Fylling ved deteksjon av hull

Eksperimentoppsett



Figur 68: *Flytskjema for fylling av hull*

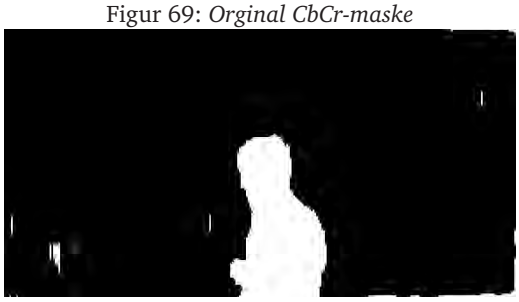
For å tette hullene i maska, blir pikslet i bildet først traversert horisontalt. Så lenge pikslet ikke inngår i det markerte området i maska skjer det ingen forandring. Når det derimot er maskert område, telles det opp en kontrollvariabel som sjekker om det har vært tilstrekkelig mye detektert område før et evt. hull. Det sjekkes så om hullet er mindre enn grensen og at området etter hullet er større enn grensen. Er det markerte området før og etter hullet tilstrekkelig stort og hullet lite nok, blir det fylt igjen. Grunnen til at størrelsen på området før og etter hullet sjekkes, er fordi det er ikke ønskelig å tette hull som etterfølges av en tynn vertikal linje. Grensen er maksimum størrelse på hullet og minimum størrelse på området på siden av hullet. Ved å sette grensen som en input parameter, kan programmet brukes til å tette hull av ønsket størrelse.

Resultat

I figur 70 er bildet traversert horisontalt og grensen for maks størrelse til hull og min størrelse til omkringliggende områder er satt til 16 piksler.

Figur 69: *Orginal CbCr-maske*

Figur 70: *Horisontal tetting av hull*



Figur 71: *Vertical tetting av hull med utgangspunkt i figur 69*

Figur 72: *Kombinasjon av horisontal og vertikal fylling*

Diskusjon

Variabelen i dette programmet er hvor stort kravet til hull i maska skal være. I dette tilfellet er verdien satt til 16 piksler. Settes denne verdien lavere, vil hull som er større enn valgt verdi ikke tettes (se figur 73). Settes verdien vil store områder fylles igjen. Dette kan man tydelig se i linjalen til høyre i figur 74. Hvis denne verdien settes altfor høy, vil personen tolkes som et hull i bildet, og maska vil bli ubrukelig.

Figur 73: *Horisontal fylling av hull som er mindre enn 10 piksler*

Figur 74: *Horisontal fylling av hull som er mindre enn 20 piksler*

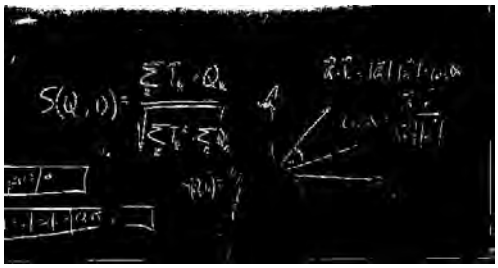
4.3.2 Fylling ved bruk av tekstmaske

Eksperimentoppsett

Det som ikke er markert i denne masken, er skrifta. I figur 69 er skriftkonturen som ikke er markert, noe tykkere enn i opprinnelig. For å kompensere for dette kan det utvikles en skriftmaske som medianfiltreres med en stor matrise. Ved å addere skriftmaskene til

figur 69 vil man få tettet noen av hullene.

Resultat



Figur 75: *Deteksjon av tekst ved bruk av fargeavstand*

Figur 76: *Tekstmaska subtrahert fra figur 69. Teksten fyller da noen av hullene i maska*

Diskusjon

Siden hullene etter skrifta er store, ble det valgt en stor medianmatrise på 8x8 for å tette disse hullene. Skrifta blir da en god del tykkere enn den opprinnelig er. Ved å sammenligne bildene i figur 77, ser man at masken er forbedret, men resultatet er ikke på langt nær så godt som i etter fylling av hull.

Figur 77: *Venstre: Orginal CbCr-maske. Midten: Etter subtraksjon av tekstmaske. Høyre: Etter horisontal og vertikal fylling*

4.3.3 Segmentering av objektkontur

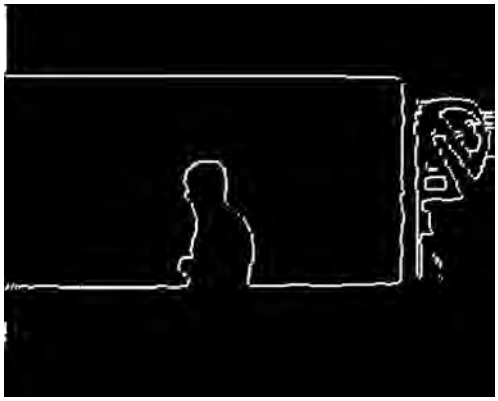
Eksperimentoppsett

Grunnen til at det blir konturen i den segmenterte videoen er at ytterkanten til personen ikke blir markert. Det er derfor ønskelig å fjerne det maskerte området under den røde streken i figur 78.

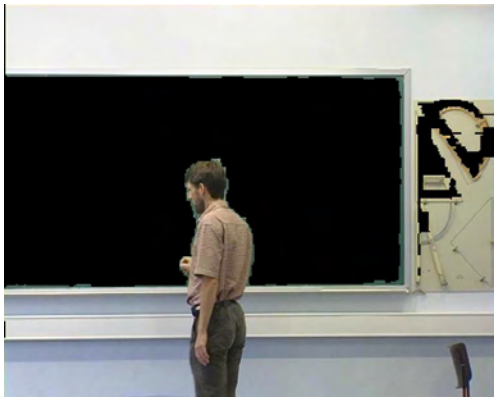
Figur 78: *Det røde omrisset skal subtraheres fra maska*

Dette kan gjøres på flere måter. I kapittel 7.2 blir det detektert et tilstrekkelig stort markert område. Dette vil være gjennomførbart i dette tilfellet, men det kreves en del terskelverdier. En bedre metode vil derfor være å foreta en kantdeteksjon etter at hullene i masken er tettet. Ved å ta utgangspunkt i figur 72 og foreta en Prewitt kantdeteksjon, vil konturen til personen markeres. Denne konturen kan så medianfiltreres for å utvides. Ved å addere figur 72 med figur 79, vil personens omriss bli større.

Resultat



Figur 79: *Objektkontur*



Figur 80: *Maskeringen av tavlen ekskluderer hele personen og har en god avgrensing til omrisset*

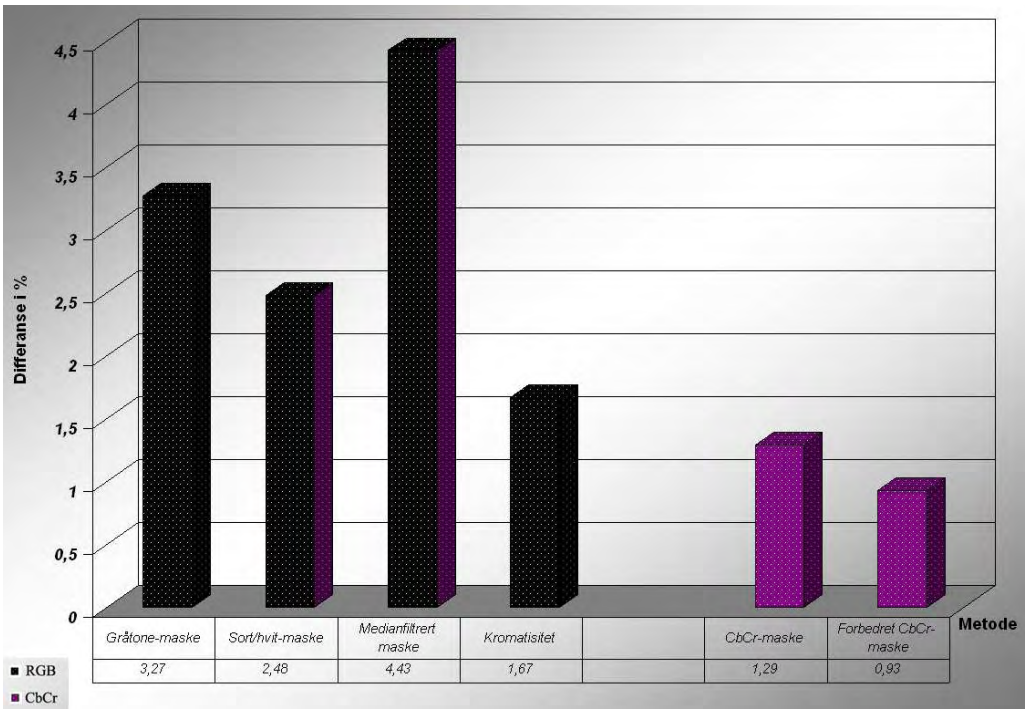
Figur 81: *Forstyrrelse etter at CbCr-masken var forbedret med fylling av hull og fjerning av kontur*

Diskusjon

Som man ser av figur 80 er avgrensingen av personen litt utenfor personens kontur. Resultatet i figur 82 viser at segmenteringen er forbedret.

Romlig medianfiltrering

Medianfiltrereing er egnet til å fjerne støy. Små hvite prikker vil tolkes som støy og derfor kunne fjernes. For at også større områder skulle fjernes ble det valgt en matrise på 8x8 piksler. Som man ser av figur 83 er noe av støyen fjernet, men algoritmen for å tette hull gir bedre resultat enn denne metoden også.



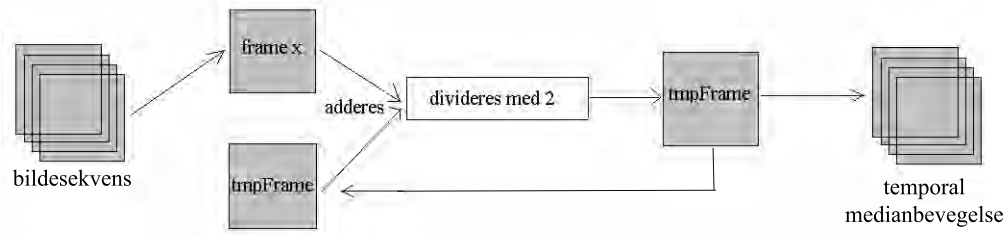
Figur 82: *Søylen til høyre viser segmenteringsgraden etter at forbedring er foretatt*

Figur 83: *Medianfiltrering med en 8x8 maske av figur 4.3.2*

5 Objektsegmentering ved bruk av temporal medianbevegelse

5.1 Eksperimentoppsett

For å glatte ut bevegelse må snittverdien til flere rammer ivaretas. Ved å gå gjennom to rammer piksel for piksel og legge gjennomsnittsverdien i et nytt bilde får man et bilde med middelverdi. Dette snittbildet kan så sammenliknes med neste ramme i videoen og de nye snittverdiene legges i snittbildet (figur 84). Snittbildet må skrives til videofil mellom hver utregning.

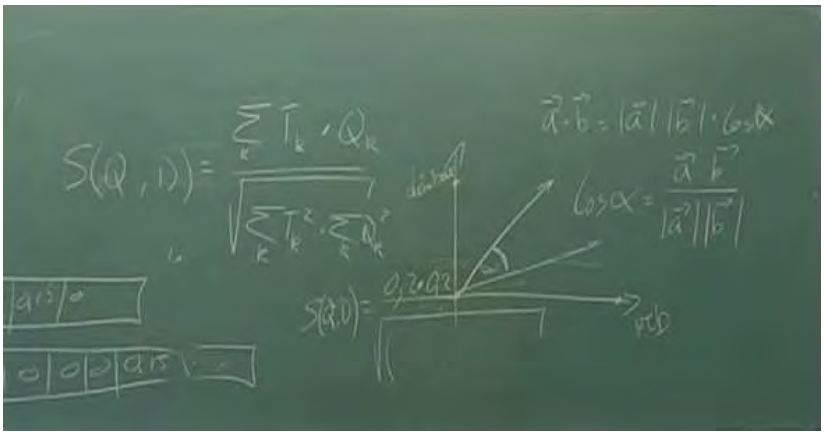


Figur 84: *Gjennomsnittsbilde*

5.2 Resultat

Temporal medianfiltrering fører til at gjengivelsen av foreleseren blir svakere jo raskere han beveger seg. Dette skyldes at rammer over et tidsrom blir midlet. Foreleseren vil ha en viss representasjon i bildet uansett hvilke tidsrom midlingen foregår over. Er han mye i bevegelse forsvinner han omtrent helt, står han i ro kommer han tydeligere frem igjen (se fig. 85). I seg selv er ikke denne metoden en tilfredsstillende løsning på segmenteringen, men kombinert med andre metoder vil denne metoden kunne gi en forbedring av segmenteringen. Resultatet ligger tilgjengelig på http://studweb.hig.no/980141/videofiler.rar

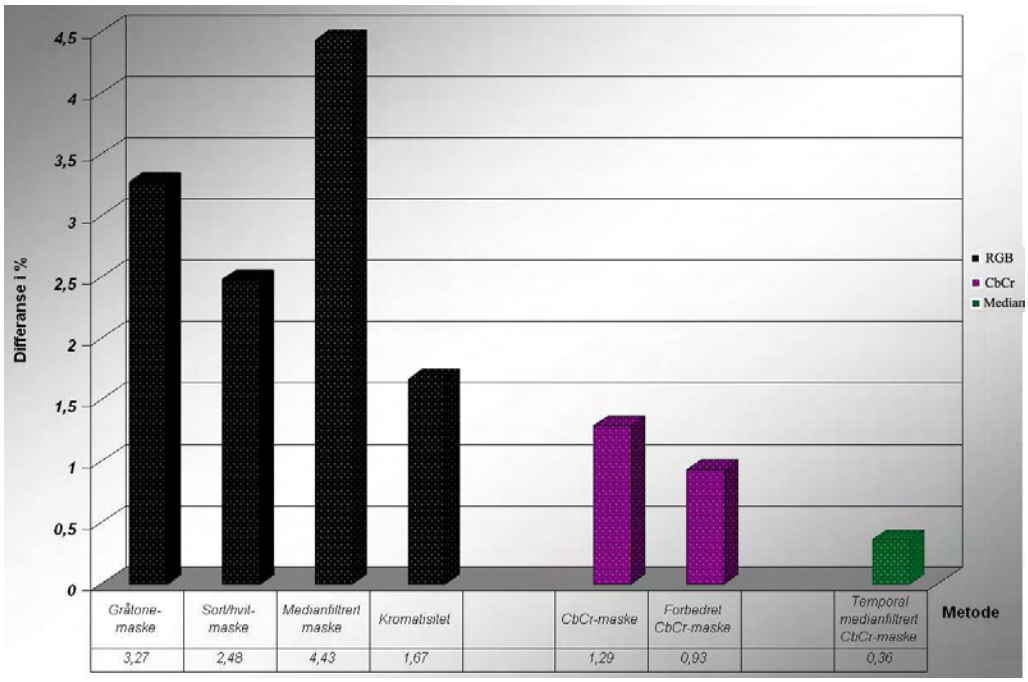
Figur 85: *Medianbevegelse*



Figur 86: *Medianbevegelse etter objektsegmentering med CbCr-maske*

5.3 Diskusjon

Det er viktig å være klar over at dette er en verdi som er tilpasset denne videoen og den vil måtte tilpasses ved behandling av andre videoer. Dersom en annen film av en forelesning er filmet nærmere tavla, vil foreleseren blir større i bildet og hans bevegelser vil få mer å si for den totale bevegelsen i bildet. Konsekvensene av dette er at det blir tatt vare på flere bilder og prosesseringen ikke vil gå like raskt.



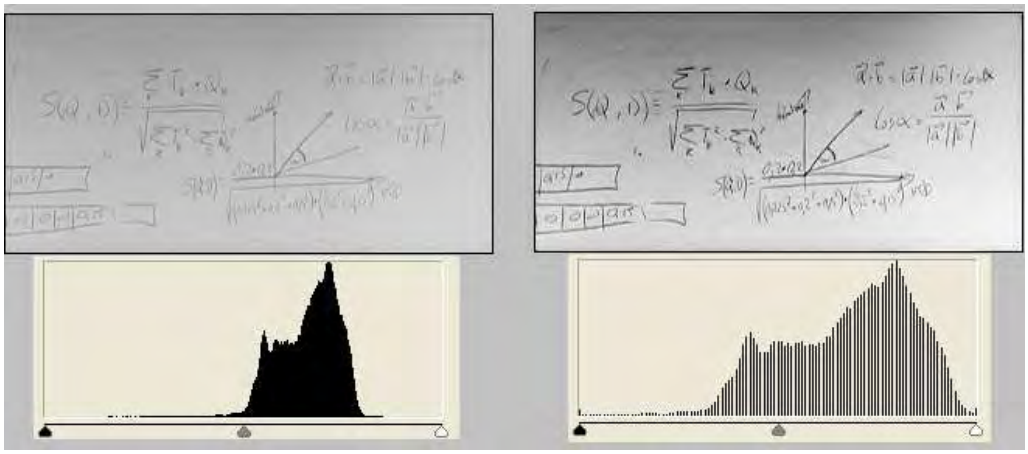
Figur 87: *Diagrammene viser segmenteringsgraden til de ulike metodene. RGB-maskene gir mest støy etter objektsegmentering. Siden CbCr-maskene var den beste metoden ble det kjørt en temporal medianfiltrering av denne resultatvideoen. Resultatet er vist i grafen til høyre.*

6 Behandling av skrift

Når objektene i forgrunnen i video er segmentert vekk, kan skrifta detekteres. Under utvikling av masker viste det seg at generering av fargemasker med utgangspunkt i krittfargen, ikke gav noe god gjengivelse av teksten (kapittel 3.1.3). I dette kapittelet vil det gjøres forsøk på å forbedre dette resultatet og det vil bli implementert fire kantdeteksjonsalgoritmer.

6.1 Kontraststrekking

Grønnfargen på tavla og krittfargen skiller seg lite fra hverandre. For å forbedre dette kan det foretas en kontraststrekking. Siden svart skrift på hvit bakgrunn er behagelig å se på, konverteres rammene fra tavle- og krittfarger i RGB til gråtoner. Kontraststrekkingen gjøres ved at histogramverdiene blir trukket ut over gråtonespekteret fra 0 til 255. Av figur 88 ser man at skrifta i det kontraststrekte bildet blir tydeligere. Disse verdiene kan forbedres ytterligere ved å fjerne alle de små søylene til venstre i histogrammet for å få stukket bildet mer. For å bevare informasjonen som ligger i gråtoneverdiene, settes det



Figur 88: *Kontraststrekking*

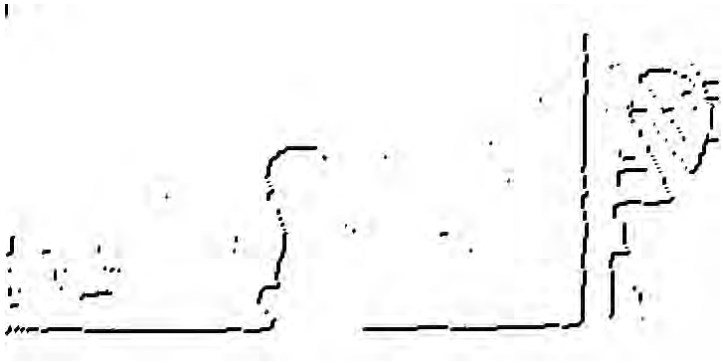
ikke noen terskelverdi for hva som skal tolkes som sort og hvitt før etter sammenslåingen av tekstmasker er gjort.

6.2 Kantdeteksjon

6.2.1 Eksperimentoppsett

Sobel, Roberts, Prewitt og FreiChen operatorer har blitt utviklet som forklart i kapittel 2.8.1. Da det ble foretatt en deteksjon med Sobel operatoren, viste det seg at kun kanter som avgrenset nedre-høyre del av objekter ble detektert. Grunnen til dette er at når bildet traverseres fra venstre mot høyre og topp til bunn vil matrisene i tabell 1 og 2 detekterer nedre-venstre kanter i objekter. I figur 89 er det foretatt en kantdeteksjon hvor av en ramme med foreleseren og man kan se at det er mange kanter som fortsatt

ikke er markert.



Figur 89: *Sobel kantdeteksjon ved bruk av to matriser*

For å få en fullstendig kantdeteksjon må matrisene derfor speiles om 0-punktsaksene. De nye Sobel operatorene blir da som vist under:

+1	+2	+1
0	0	0
-1	-2	-1

Tabell 9: *Gx (invers matrise av tabell 1)*

+1	0	-1
+2	0	-2
+1	0	-1

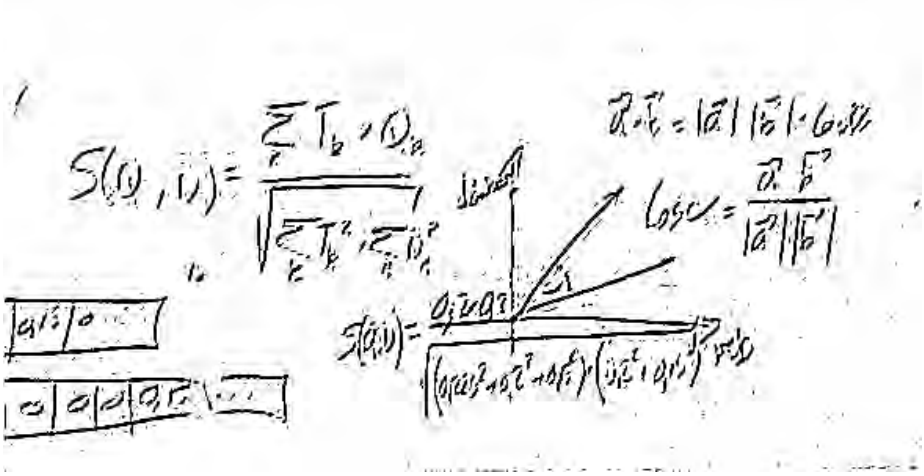
Tabell 10: *Gy(invers matrise av tabell 2)*

Dette må gjøres på samme måte for Prewitt, Roberts og FreiChen operatoren. Det vil altså måtte brukes 4 matriser i hvert tilfelle for at alle kantene i bilde skal kunne detekteres ved kun en travesering.

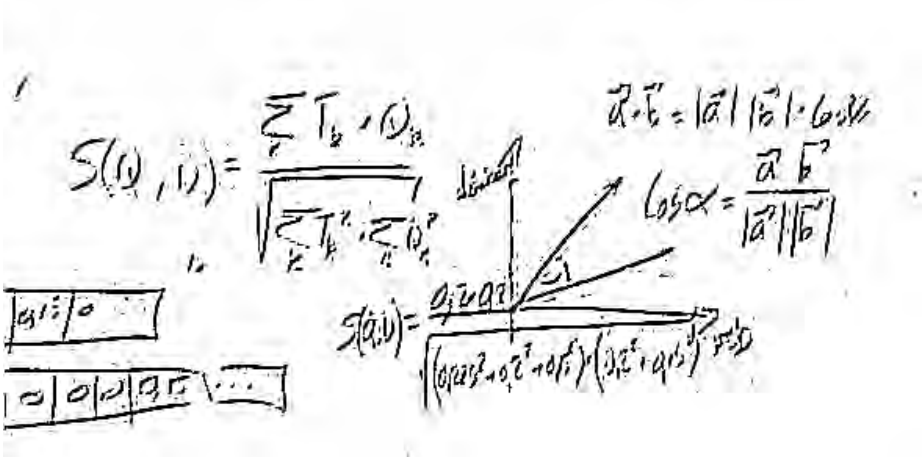
Legger man merke til hvilke kanter som er detektert i personen i figur 89, kan man få inntrykk av at det ikke stemmer at det kun blir markert nedre-høyre kanter ved bruk av matrise 1 og 2. Grunnen til at øvre og venstre del av personen er markert, skyldes at personen utgjør en avslutningen av tavleobjektet. Bildet traverseres fra venstre mot høyre og topp til bunn. Når man starter fra venstre og detekterer tavle, vil det i forandringen fra tavle til person blir satt en kant som markerer avslutningen på tavlen. Dette vil da være en høyre avgrensingen av tavleobjektet.

6.2.2 Resultat

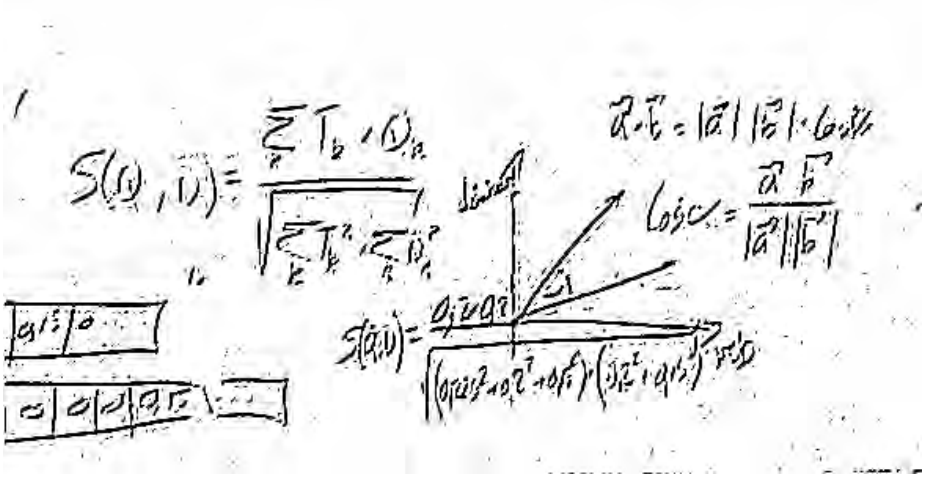
For å kunne foreta en kvalitativ analyse av hvilke av de 4 operatorene som gir best resultat, ble det tatt et lite utsnitt.



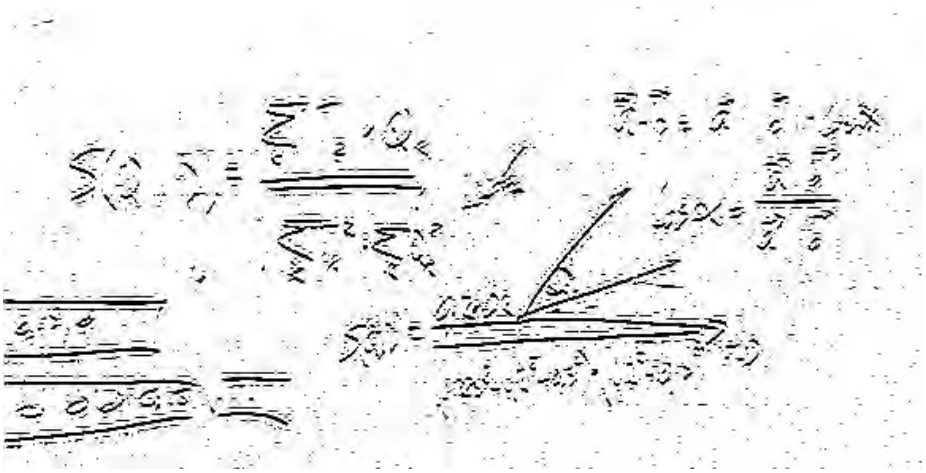
Figur 90: *3x3 Sobel*



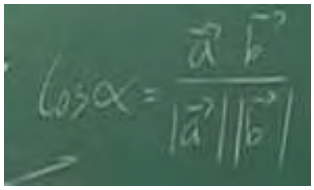
Figur 91: *3x3 Prewitt*



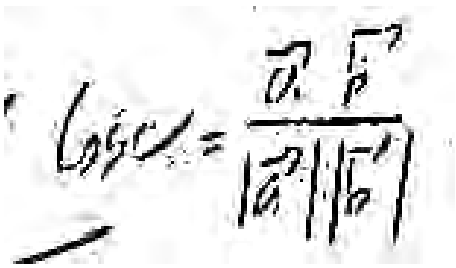
Figur 92: *3x3 FreiChen*



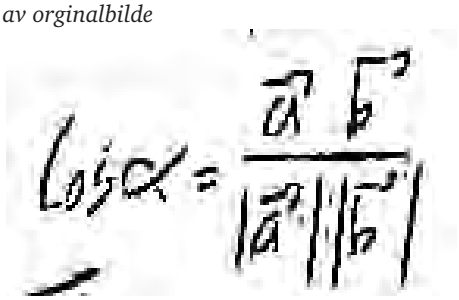
Figur 93: *3x3 Robert*



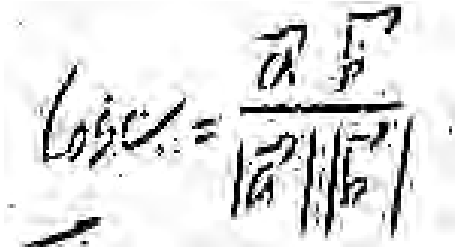
Figur 94: *Snitt av orginalbilde*



Figur 95: *Analyse av 3x3 Sobel-operator*



Figur 96: *Analyse av 3x3 Prewitt-operator*

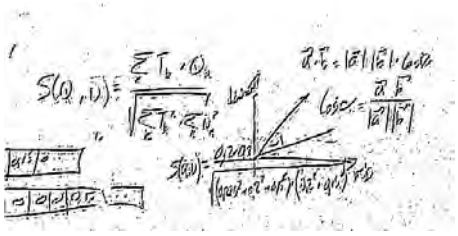


Figur 97: *Analyse av 3x3 FreiChen-operator*

Figur 98: *Analyse av 3x3 Roberts-operator*

6.2.3 Diskusjon

Prewitt, Sobel og FreiChen operatorene gir et forholdsvis godt leselig resultat. Ved bruk av alle disse tre operatorene, detekteres det en del støy pga. konturen i tavla. Ved å øke terskelverdien for kanter, vil man få fjernet støy, men settes verdien for høy vil også svak skrift forsvinne. Det vil derfor være en fordel å sette en forholdsvis lav terskelverdi og heller fjerne støyen i bildet ved bruk av et støyfilter.

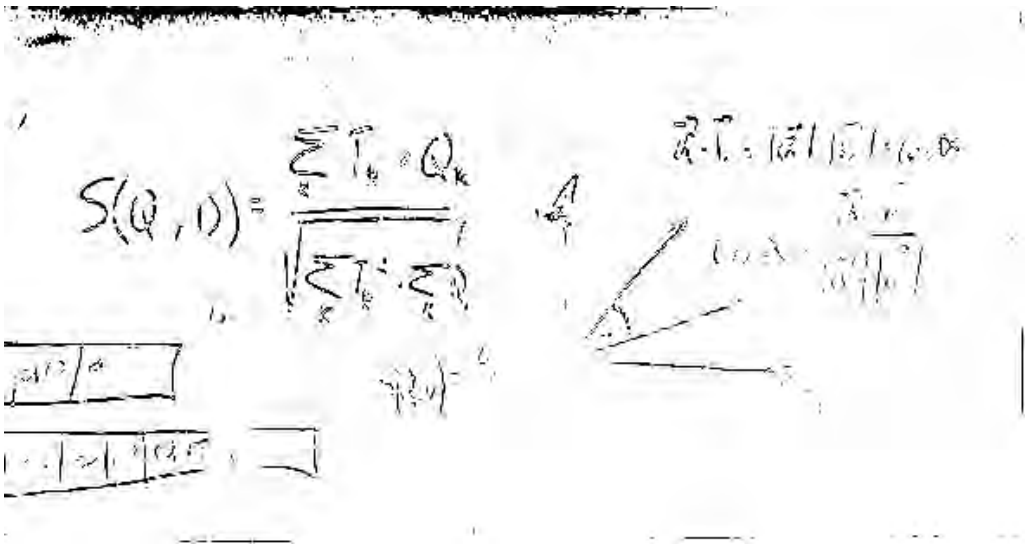


Figur 99: *Prewitt med lav terskelverdi (0.5)*

Figur 100: *Prewitt med høy terskelverdi (4.0)*

6.3 Forbedring av leselighet

6.3.1 Eksperimentoppsett

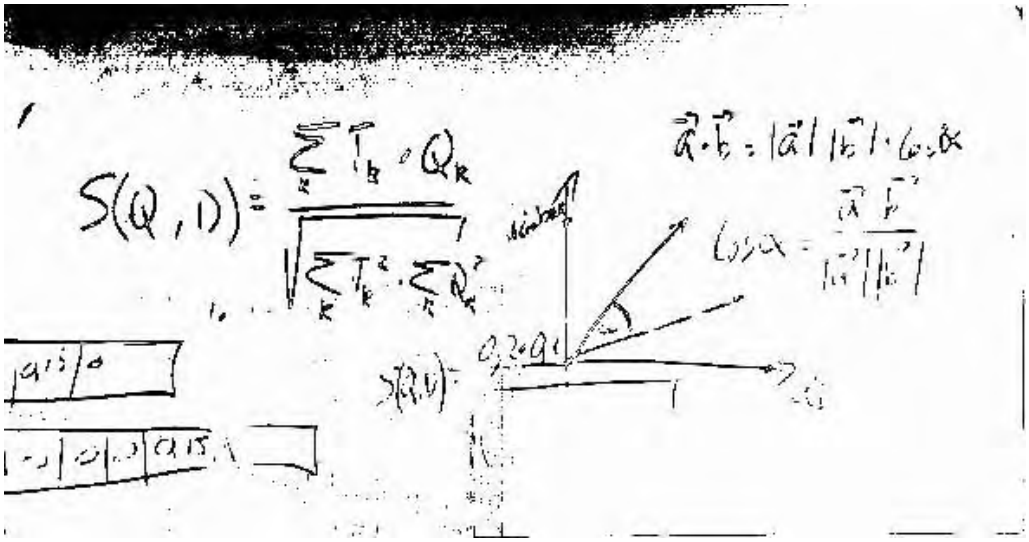


Figur 101: *Detektert kritt ved bruk av euklidsk fargeavstand*

Figur 101 viser resultatet etter at den euklidske avstanden med utgangspunkt i RGB krittfarge er beregnet og resultatet er tersklet. Hvordan dette kan gjøres er forklart i kapittel 3.1.1. Som man ser mangler store deler av teksten. Grunnen til dette er at det er mye representasjon av grønnfargen fra tavla i den hvite krittskrifta. Når man generer en maske for krittfargen, blir bare rundt 70 til 80% av all skrifta representert. Hvilket område av skrifta som blir maskert, varierer litt fra ramme til ramme. Ved å addere flere rammer med detektert tekst, vil man derfor få representert en høyere prosentandel av skrifta. Jo flere rammer man legger sammen, jo tydeligere blir skrifta. Ulempen er at de små representasjonene av støy som forekommer også blir addert og utgjør etter litt tid

områder som forstyrrer skrifta. Siden støy forekommer på forskjellig sted til ulik tid, vil det ikke få en like høy representasjon som skrifta i hvert punkt. Ved å legger sammen alle verdiene i et bestemt punkt i f. eks 90 masker, vil dette pikslet få en høy totalverdi dersom det er skrift. I et område som ikke inneholder skrift vil dette pikslet være markert i enkelte masker pga. støy, men totalverdi for representasjon vil være mye lavere enn for skrift. Det kan da settes en terskelverdi som tilsier at pikslet må være markert i minst f. eks 70 av 90 tilfeller for at det skal vises som skrift. For å ivareta totalverdien, kan overnevnte beregning fortas i hvert piksel i maskene og totalverdien kan legges inn i et midlertidig bilde.
Skrifta legges som et nytt lag på bakgrunnen, og den forandres når læreren pusser tavla eller korrigere og utbedrer det han skriver. Skrifta må derfor representeres dynamisk slik at forandringer blir ivaretatt. For å gjøre dette må bare en viss mengde av de foregående rammene legges i totalmaska. Overnevnte algoritmen fungerer altså bra fra ramme nummer 0 til 90, men hva gjør man så når man skal finne total krittverdi for ramme 1 til 91? Å beregne alle maskene på nytt for å finne totalverdi for hver ny ramme som kommer etter ramme nr 90 vil være svært krevende. For å gjøre denne oppdateringen, ble det utviklet en ringbuffer.

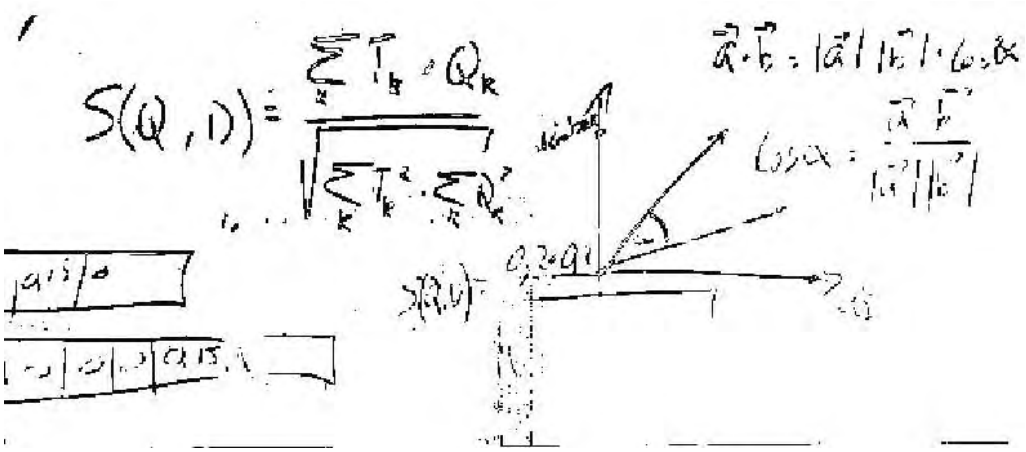
6.3.2 Resultat



Figur 102: *Addisjon av krittmasker over tid uten eliminering av støy*

6.3.3 Diskusjon

Skriftgjengivelsen vil ikke kunne bli like god som i den opprinnelige videoen fordi det blir foretatt MPEG-kompresjon for hver gang videoen behandles (se forklaring i kapittel 2.10). Dersom videoen skal gjennomgå flere behandlinger og det for hver gang blir foretatt en komprimering, vil kvalitetstapet blir synlig i den resulterende teksten. Utvikling av funksjonalitet for å lagre ukomprimert video i APIet er under arbeid.
Tekstdeteksjon med utgangspunkt i en RGB-farge blir ikke like god som ved Prewitt-kantdeteksjon. Grunnen til at det i dette forsøket allikevel er brukt RGB, er at denne masken har større forbedringspotensiale. Resultatet av en forbedret av en ramme med



Figur 103: *Addisjon av krittmasker over tid med eliminering av støy*

detektert tekst med Prewitt-operatoren ville blitt bedre, men man ville ikke fått et like godt bilde av forbedringen. Og det er nettopp forbedringen som skal visualiseres i dette avsnittet.

Sammensetting av videofiler

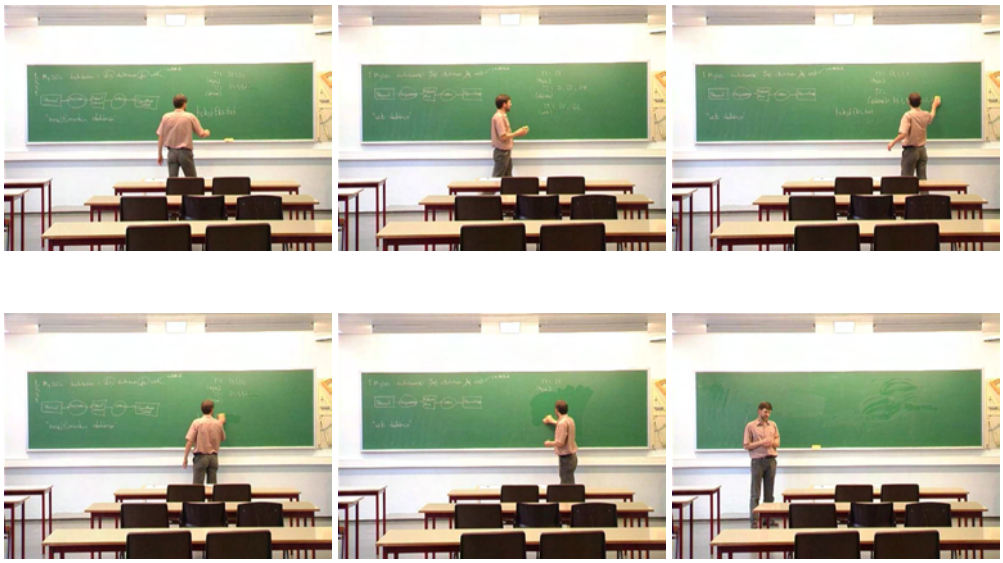
For å få en tydelig gjengivelse av teksten på tavla, har det i dette kapittelet kun blitt tatt utgangspunkt i de to filmene som gjengir hver sin halvdel av tavla (figur 28 og 29). Skriftoppløsningen i videoen av hele tavleområdet (figur 26) er ikke på langt nær så god som oppløsningen i disse filmene. Problemet med å bruke to filmer, er at når tekstreduksjon blir detektert og det skal foretas en gjengivelse av hele tavla må det genereres et sammensatt bilde. Et rammenummer i filmer fra forskjellige kameraer vil ikke angi samme tidspunkt dersom filmingen ikke er startet helt likt. Enkleste måten å løse dette problemet på, er å slå sammen de to filmene til en panorama video som er synkronisert på lyd. Dette ble gjort av Andrei Ouglov [67].

6.4 Deteksjon av skriftreduksjon

6.4.1 Eksperimentoppsett

Etter at objekter i forgrunnen er fjernet, vil det eneste som varierer i video være hvor mye tekst som befinner seg på tavla. I et sort/hvitt invertert bilde av tavla, vil totalverdien av antall sorte piksler (skrift) øke når foreleseren skriver på tavla, og den vil reduseres når tekst pusses vekk. I øyeblikket rett før det foretas skriftreduksjon vil det være nyttig å lagre det som står på talva i et bilde. Bilder av tidspunktet før tavla pusses vil da kunne oppsummere alt som har stått på talva i videoen.
Måten dette har blitt gjort på, er at hver ramme i videoen sammenlignes med en ramme fram i tid. Hvis prosentvis reduksjon av tekst overgår en terskelverdi blir det lagret et bilde av teksten før reduksjonen. Grunnen til at det må settes et krav til reduksjon er at totalt antall kantdetekterte piksler i bildet vil varierer noe fra bilde til bilde pga. støy.

6.4.2 Resultat



Figur 104: Rammene i øverste rad er fra øyeblikket rett før det blir foretatt tavlepuss. Disse rammene blir lagret i en bildeserie som skal gjengi alt innholdet på tavla i løpet av forelesningen. Rammene i nederste rad viser at det foretas tavlepuss i de tre tilfellene. Bildene er lagd for å demonstrere når det foretas tavlepuss og objektsegmentering er derfor ikke foretatt

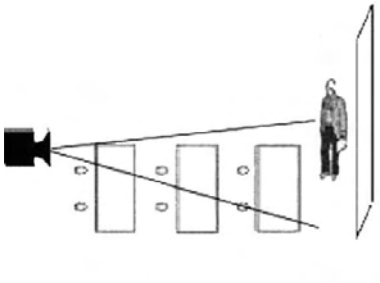
6.4.3 Diskusjon

Ved å studere video av detektert tekst på http://studweb.hig.no/980141/videofiler.rar, kan man se at det kommer litt representasjon av personen selv etter at objektsegmentering er foretatt. Når teksten kantdetekteres vil altså de delene av personen som ikke er segmentert vekk utgjøre en liten forsyrrelse i teksta. Siden denne forsyrrelsen varierer litt med tiden, vil dette kunne oppfattes som en tekstreduskjon. For at objektforsyrrelse ikke skal resultere i falske skriftredusjoner, kan det settes et minimumskrav til hvor mye skrift som må fjernes for at et av teksten skal lagres.

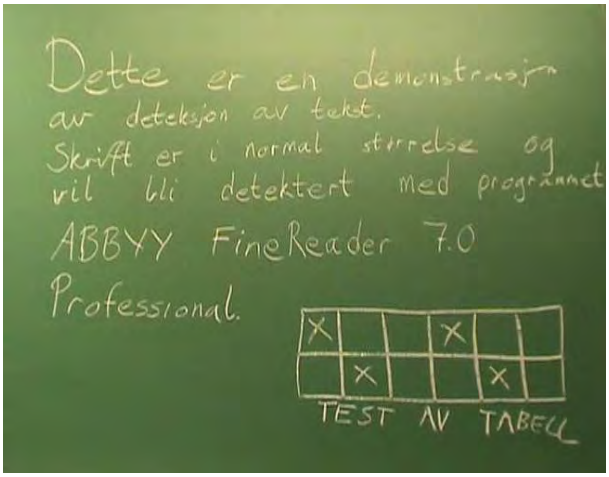
6.5 Skriftgjenkjenning

6.5.1 Eksperimentoppsett

Ved bruk av høyre og venstre video er skrifta tydelig gjengitt, men problemet er at den ikke er skrevet pent nok til at den lar seg digitalisere. For å teste muligheten for skriftgjenkjenning ble det derfor lagd en ny video. Kameraet ble satt så nærme tavla at tavla dekket hele bildet. Skrifta som ble skrevet var i normal størrelse og i forholdsvis gjennomsnittlig skjønnskriftkvalitet(figur 106). Dette er forhold som bør kunne ligge til rette i en forelesningsvideo.



Figur 105: *Kamera som kun filmer tavleområdet*

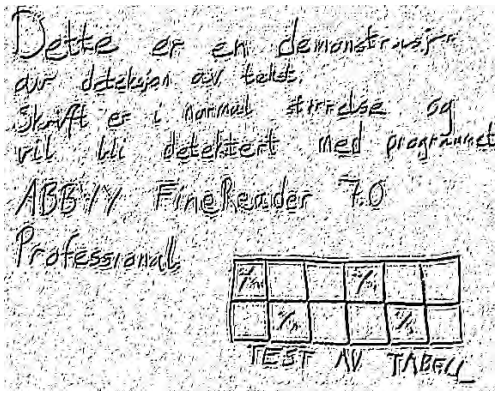


Figur 106: *Tekst med høy oppløsning og forholdsvis gjenkjennelige bokstaver*

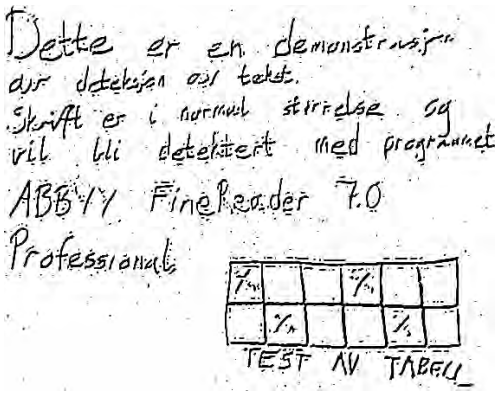
Resultatet av deteksjon av skriftreduksjon gir en serie med bilder. For å foreta en skriftgjenkjenning ble programmet ”ABBYY FineReader Professional 7.0” brukt. En testversjon av dette programmet følger vanligvis med skannere. Grunnen til at nettopp dette programmet ble valgt framfor andre tekstgjenkjenningsprogrammet, er at ABBYY er et av de mest brukte OCR programmene. Når det ble foretatt en skriftgjenkjenning på teksten i figur 108 ble ikke noen av ordene gjenkjent. Noe av grunnen til dette er at det ikke er sammenhengende tekst. Morfologiske operasjoner blir brukt til å linke linjer som inneholder små hull. Ved bruk av en standard morfologisk colosing i matlab blir figur 109 og 110 generert.

6.5.2 Resultat

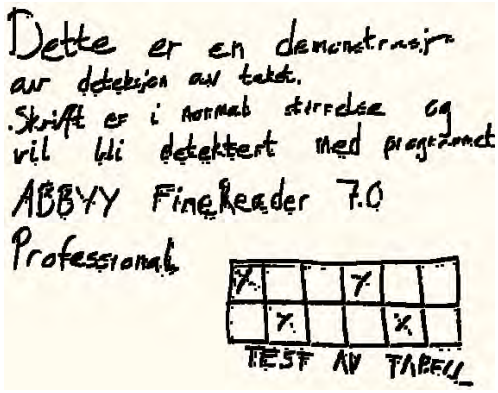
Prewitt kantdeteksjon gir en god del representasjon kontur i tavla. Ved å sette en høyre terskelverdi, vil dette kunne forbedres. Slik bildet er nå, ser det nokså utklart ut, og det er derfor foretatt en støyfjerning (figur 108).



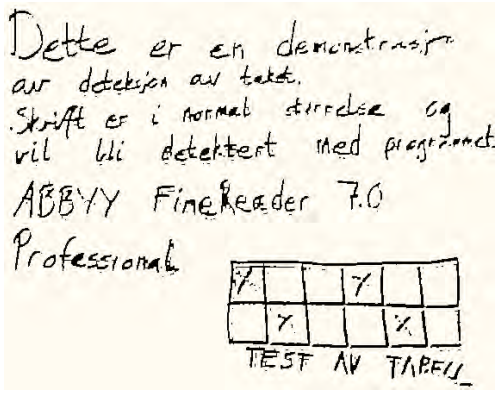
Figur 107: *Prewitt deteksjon av stor skrift*



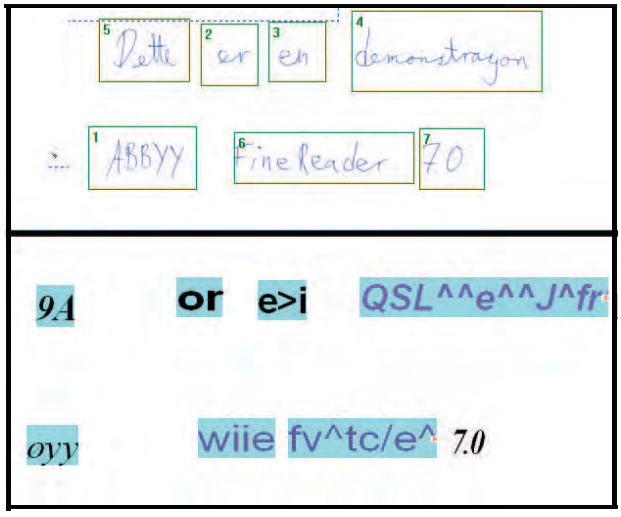
Figur 108: *Reduksjon av støy*



Figur 109: *Morfologisk operasjon (dilation) for å få sammenhengende tekst*



Figur 110: *Morfologisk operasjon (erosjon) for å få tilbake normal skrifttykkelse*



Figur 111: *Den håndskrevne teksten i øverste halvdel av bildet viser tekst som er hentet fra tavlen. Kvadratene er en markering av ord som foretas i ABBYY FineReader. I nederste område vises teksten etter at det er foretatt en tekstgjenkjenning med ABBYY. Som man kan se blir kun tallet 7.0 gjenkjent.*

6.5.3 Diskjusjon

Selv om skrifta i figur 110 er forholdsvis godt leselig for det menneskelige øyet, lar den seg ikke gjenkjenne av programmet ABBYY FineReader. Som man ser av figur 111 er sifrene 7.0 de eneste tallene som blir gjenkjent. For å finne ut om dette skyldes for lav oppløsning i bildet prøvde jeg å skanne inn håndskrevet tekst og foreta en gjenkjenning. Resultatet ble ikke forbedret. Det viser seg derfor at programmet er best egnet til å gjenkjenne maskinskrift. Håndskrift varierer svært mye fra person til person og for å foreta en god gjenkjenning må det utvikles et mer robust program.

7 Beskjæring av video i tid og rom

Beskjæringsalgoritmene er utviklet for å begrense videofilenes fysiske størrelse slik at overføring og videre videoprosessering skal gå raskere. Bruk av standard videokomprimering som f. eks MPEG vil redusere kvaliteten på skrifta. Dette er ikke ønskelig, og MPEG-komprimering kan derfor ikke benyttes. For å redusere størrelsen og samtidig bevare den originale DV-kvaliteten i filmene, har det blitt utviklet to nye algoritmer. Første algoritme baserer seg på hvor mye bevegelse som forekommer i filmen. Kun de rammene som skiller seg fra foregående rammer blir lagret i ny videofil. I andre metode blir videobildet beskåret i kantene. Kun området der tavla befinner seg er nyttig å lagre i den beskjærte filmen.

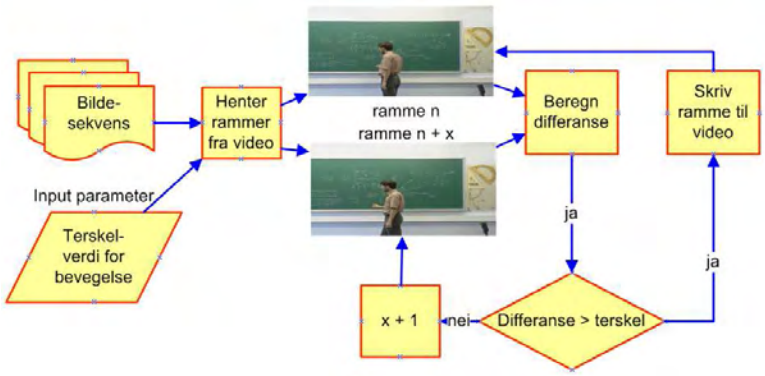
I utviklingen av disse algoritmene ble videofila av hele tavleområdet(fig. 26) og av høyre tavleområde(fig. 29) benyttet. Beskjæringsalgoritmen vil da måtte håndtere:

- ulik belysning
- at objekter i bevegelse nærme kameraet, vil utgjøre større en bevegelsesdifferanse, enn objekter langt bak i bildet

Ulik belysning er et problem når det skal utvikles maske som avgrenser området som skal beskjæres. Personens størrelse i bildet er avgjørende for hvor store utslag bevegelsene hans/hennes vil gi. Når det skal foretas bevegelsessegmentering, vil forholdsvis små bevegelser i videoen av høyre tavlehalvdel gi mye større totalt bevegelsesutslag enn identiske bevegelser i filmen av hele tavla.

7.1 Subsampling

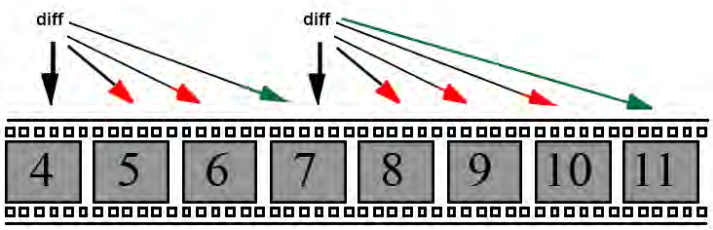
7.1.1 Eksperimentoppsett



Figur 112: *Flytskjema for segmentering av bevegelsesrammer*

Når man skal finne den totale differansen mellom to fargebilder, må piksel for piksel i de to bildene sammenlignes. Hvert piksel i et RGB-bilde består av tre fargekanaler; rød, grønn og blå og en valgfri lysintensitetskanal. Ved å sammenligne piksler med samme plassering i de to bildene, kan man detektere bevegelse i bildet. I piksler der det har vært

bevegelse, vil fargeverdiene være forandret. Ved å addere differanseverdien for hvert piksel, finner man den totale differansen mellom de to rammene. Alle rammene vil som regel ha en liten differanse selv om det ikke forekommer synlig bevegelse. Dette kan skyldes støy og små bevegelser. For at ikke alle rammer skal bli angitt som bevegelsesrammer må det derfor settes en grense. Denne terskelverdien/grensen avgjør hvor stor differansen må være for at en ramme skal markeres som bevegelsesramme. Ved å sammenligne to etterfølgende rammer finner man kun brå forandringer. Denne formen for sammenligning egner seg derfor best til å finne sceneskift. Normale menneskelige bevegelse krever litt tid før de gir et merkbart utslag. Ved å sammenligne første ramme med rammer utover i filmen inntil differanseverdien overstiger en valgt terskelverdi, vil dette gi en bedre representasjon av bevegelsesrammer.
Eksempel 1: (Figur 113). Ramme 4 sammenlignes med ramme 5, 6 og 7. Differansen mellom 4 og 5, og mellom 4 og 6 er ikke større enn terskelverdien. Derimot mellom ramme 4 og 7 er differansen stor nok. Ramme 7 blir da marker som en differanseramme. Ramme 7 må så sammenlignes med ramme 8, 9, 10 og 11.



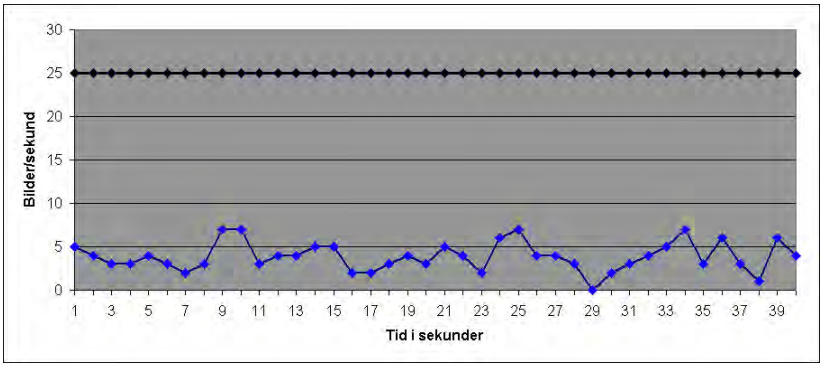
Figur 113: *Sammenligningsbassert bevegelsesrammeuttrekking*

Av disse bevegelsesrammene kan det genereres en ny video. Denne nye videoen vil inneholde alle rammene som det er nødvendig å foreta prosessering på. Siden den nye filmen inneholder færre rammer enn den opprinnelige, men avspillingshastigheten er den samme, vil filmen gi en hurtigavspilling av forelesningen. Tidspunkt da det ikke skjer noe i filmen klippes vekk, men når det er mye bevegelse får man en god representasjon.
Er det ønskelig å generere en bevegelsessegmentert video i normal hastighet, må man lagre samme bevegelsesramme inntil en ny detekteres.
For å tilpasse terskelverdien for differansen kan det være greit å skrive ut differansen til skjerm. Et eksempel på slike verdier er vist i figur 117. Siden det i dette tilfellet ikke er nødvendig å gjengi bevegelsesdetaljer, er terskelen satt så høyt at det kun blir ivaretatt 3 bilder i løpet av de første 25 rammene. Terskelverdien for bevegelse sendes med som en parameter til programmet slik at beskjæringsalgoritmen kan brukes på alle slags filmer.

7.1.2 Resultat

Terskelverdien for hvor stor differansen mellom to rammer må være, ble satt til 3%. Rammer som har en differanse på mer enn 3%, blir markert som bevegelsesrammer. Figur 116 viser hvor godt bevegelsen blir representert og figur 114 viser hvor mange piksler som har vært bevegelse.

Figur 114: *Sorte områder viser hvor det har vært bevegelse. Bildet tilsvarer en differanse på 3 prosent. Det vil altså bli generert en ny bevegelsesramme når det har forekommet så mye bevegelse som vist her.*

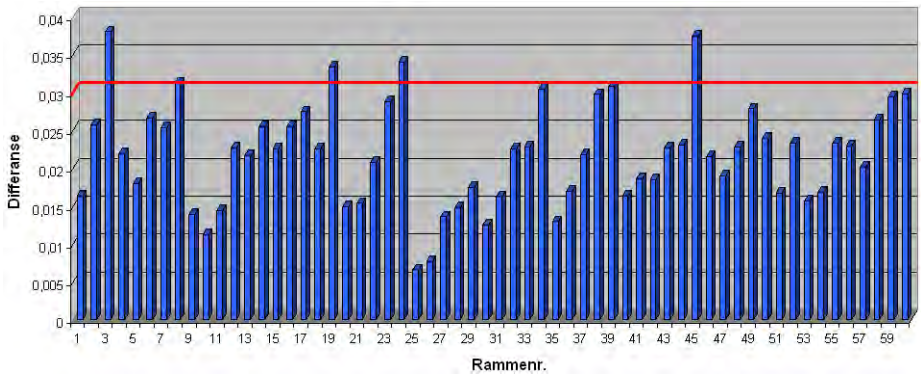


Figur 115: *Sort graf: standard bildefrekvens i digital video. Blå graf: frekvens for rammer med differanse på mer enn 3%*

Terskelen er tilpasset videoer der det er objekter i forholdsvis rolige bevegelse. Verdien viser seg å gi et godt resultat i mine tre testvideoer, men skal programmet brukes på filmer med mye bevegelse, må terskelen økes. Den nye videoen som blir generert, er satt sammen av de detekterte bevegelsesrammene. Frekvensen for hvor mange bevegelsesrammer som blir representert i sekundet, varierer etter hvor mye bevegelse det er i videoen. Settes terskelverdien for bevegelse til 3% vil det bli detektert ca 3-7 rammer/sekund når læreren skriver på tavla og beveger armen.

Når personen i resultatvideoen er i bevegelse, får han en trinnvis bevegelse. Dersom han står i ro, er det ikke mulig å se at oppdateringsfrekvensen på nye rammer er satt ned. Reduserer man kravet til bevegelse, vil den trinnvise bevegelsen utjevnes, fordi det blir generert flere bevegelsesrammer. Ulempen er at det blir en hyppigere deteksjon av bevegelsesrammer, og videoen vil bli større.

Figur 116: *Eksempel på hvor godt bevegelse blir representert etter bevegelsessegmentering med en terskelverdi på 3 prosent*



Figur 117: *Prosentvis differanse i rammene i video av høyre tavleområde 29. Alle rammene med en høyere differanse enn 3 % blir lagret i ny video*

7.1.3 Diskusjon

Differanseverdien på 3% viste seg å gi en stor begrensning av rammer, samtidig som bevegelse gjengis relativt godt. Selv om personen går fram og tilbake i bildet, blir det lite utslag. Forskjellen mellom to etterfølgende rammer er altså liten, men sammenligner man derimot bevegelse over tid vil dette gi en bedre bevegelsesdeteksjon og større utslag. Siden mange etterfølgende rammer er like, blir beskjæringen god selv med en lav terskelverdi. Terskelverdien viser seg å gi et godt resultat i alle de 3 videoene som blir brukt i denne oppgaven.

Figur 118: *Differanse på 1 prosent*

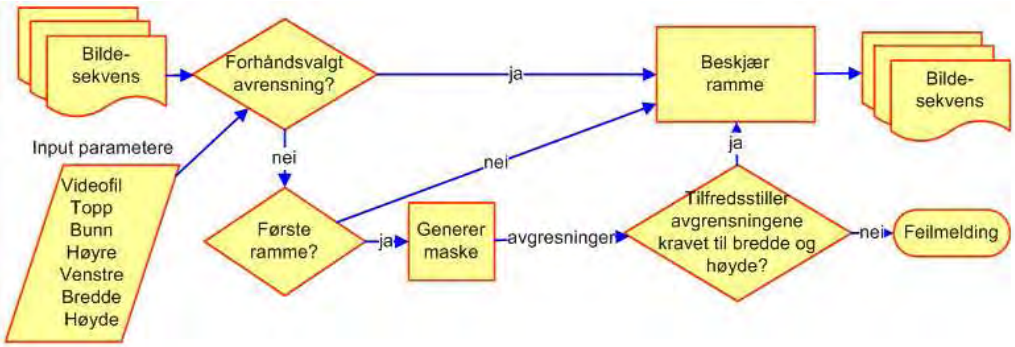
Figur 119: *Differanse på 5 prosent*

Dersom man ønsker å generere en bevegelsessegmentert video i normal avspillingshastighet, kan ikke denne videoen inneholder færre rammer enn den opprinnelige. Grunnen til dette er at avspillingshastigheten på antall rammer per sekund er konstant. Filmen må derfor settes sammen av like bevegelsesrammer inntil nye detekteres. Når jeg angav ønsket differanseverdi, plukket jeg ut en terskelverdi ut ifra hvor mye foreleseren beveget seg. Det er en vurderingssak hvor mye bevegelse som må forekomme for at prosessering skal foretas. Men konsekvensene av denne verdien er ikke kritiske. Er differansen satt for høyt, vil det tas vare på for få rammer slik at teksten kommer ujevnt fram på skjermen, er den for lav vil prosesseringen gå saktere. Som man ser av figur 117 er det liten forskjell på rammene. Dette vil gi en ganske nøyaktig representasjon av bevegelse. Selv om det er like rammer er reduksjonen i antall rammer bra. I løpet av 15 sekunder blir det tatt vare på 41 rammer. Dette tilsvarer mellom 2 og 3 rammer i sekundet. Sammenliknet med at avi består av 25 bilder/sekund vil dette utgjøre en reduksjon på rundt 90%. Effekten av dette er at videre behandling av videoen vil gå mye raskere.

7.2 Beskjæring av areal

7.2.1 Eksperimentoppsett

For å beskjære video, trenger man en avgrensning av et område. Dette kan enten gjøres ved at brukeren angir topp, bunn, høyre og venstre begrensning til et areal eller begrensingen kan foretas algoritmisk. Den algoritmiske metoden genererer en maskering av første ramme i videoen. Hvordan man genererer en maske er forklart i kapittel 3.1. Om man bruker en CbCr- eller fargemaske har ingen betydning. Så lenge filmen som skal beskjæres inneholder et nokså ensfarget område, kan man bruke masker til å avgrense



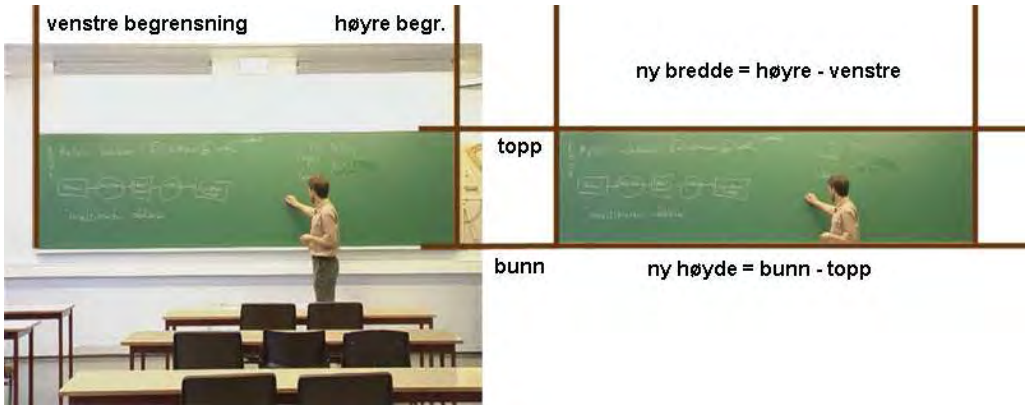
Figur 120: *Flytskjema for beskjæring av video*

alt fra tavler, murvegger til overvåkningsområder. Figur 121 avgrenser tavleområdet, men det er også mye støy i bildet. Dette skyldes at verdiene for hva som skal tolkes som tavle er satt lave. Grunnen til dette er at programmet skal kunne markere tavler i forelesningsvideoer med ulik belysning, uten at det trengs å gjøres noen justeringer. Bieffekten av dette er at fargeverdier som ligger i nærheten av tavlefargen, også blir inkludert i maska. At det kommer mye støy fra omkring liggende områder med forholdsvis lik farge tar beskjæringsalgoritmen hensyn til. Så lenge man har et grovt omriss av området som skal beskjæres, vil dette området la seg detektere.

Figur 121: *Tavlemaske genererte med standard tavlefarge og en lav terskelverdi*

Ved eksekvering kjøres kan brukeren angi to parametere med minimumskrav til bredde og høyde på arealet som skal beskjæres. Algoritmen som beskjærer videoen går så gjennom hver enkelt horisontale linje og sjekker hvor stor prosentandel som er hvite piksler. Første linje med et antall hvite piksler overgår minimumskravet til bredde, markeres som toppen av tavla. Koden går så linje for linje nedover og når den første linje som er mindre enn minimumskravet detekteres, markeres bunnen av tavle i gjeldende linje minus 1. På samme måte går koden på nytt gjennom bildet og sjekker hvor stor prosentandel av de vertikale linjene som er hvite. Man får da avgrenset høyre og venstre side av tavla. I figur 122 viser hvordan punktene topp, bunn, høyre og venstre avgrenser tavla.

Siden kravet til nøyaktighet i det maskerte området er satt lavt, må det også sjekkes om det markerte området i maska utgjør en tilstrekkelig stor del av bildet. Grunnen til dette er at et grønt avlang område i overkant eller underkant av tavla, vil kunne mistolkes som tavla og da vil topp og bunn markere dette området og ikke er tavla. Setter man et



Figur 122: *Avgrensning av området som skal skrives til ny videofil*

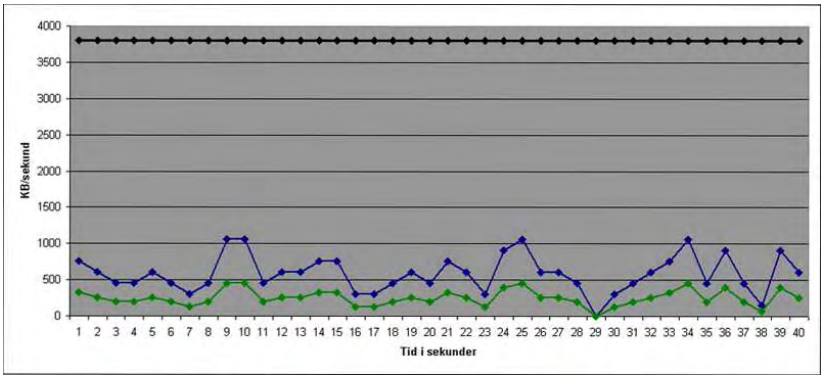
krav til at det avgrensede området skal utgjøre minst arealet av minimumskravene brukeren sendte med, forsikrer man seg om at riktig område blir markert. Er det markerte området mindre enn minimumshøyde, går koden videre nedover i bildet for å finne et høyere markert område.
Når man så har fått avgrenset det området man ønsker, kan dette området, i hver enkelt ramme, skrives til en ny videofil. Denne nye fila vil være redusert i visuell og fysisk størrelse, men vil ha samme informasjonsverdien for brukeren siden den gjengir alt innholdet på tavla.

7.2.2 Resultat

Figur 123: *Videofil etter beskjæring i areal*

7.2.3 Diskusjon

Denne algoritmen er utviklet for å være robust slik at den krever lite forarbeid for å avgrense et område. Det eneste den trenger er ei maske som gjør en grov avgrensning av området. At maska har hull og markerer områder som ikke inngår i maska, er ikke kritisk. Ved å terskle en CbCr maske får man skilt forgrunnen fra bakgrunnen i video. Vil man benytte en fargemaske, er det allerede lagt inn en generell tavlefarge som gir en avgrensning av tavle og noe støy i bildet. RGB-fargeverdien kan forandres avhengig av hvilken farge det er på området som skal beskjæres. Dersom man setter et minimumskrav til høyde og bredde kan man forsikre seg om at området som blir markert er tilstrekkelig stort. Hvor stor del av bredden og høyden til bilde området må utgjøre, vil variere fra film til film. I alle videofilene befinner foreleseren seg i bildet ved start. Hadde han



Figur 124: *Sort graf: datastrøm i digital video.*
Blå graf: datastrøm etter bevegelsessegmentering.
Grønn graf: datastrøm etter bevegelsessegmentering og beskjæring av video.

ikke gjort det ville det vært enklere å kalibrere med et tomt bilde. Når personen som ikke inngår i det maskerte området står foran tavla, vil han utgjøre et stort hull. Studer figur 121. Idet man finner lengden til vertikale linjer og man kommer til foreleseren, vil det området som er markert utgjøre en minimal del av totalhøyden i bildet. Dette området vil som regel være mye mindre enn det brukeren har spesifisert som minimumskrav til tavlehøyde. Dette skyldes jo at det foreløpig er et objekt i forgrunnen. Kjøres denne bevegelsessegmenteringen etter at objektsegmenteringen er foretatt, vil situasjonen være en annen. Siden hensikten er å beskjære videoen før den videre prosesseringen foretas, må det tas hensyn til at det er gap i maska. Ved å redusere kravet til høyde til 20% av det brukeren har spesifisert, vil programmet inkludere området over og til høyre for foreleseren og resultatet blir at høyre og venstre markering av tavla blir satt riktig. Det at objekter i forgrunnen dekker for maska er en begrensing for denne algoritmen. Utgjør objektet en stor del av området som skal beskjæres vil ikke algoritmen detektere et tilstrekkelig stort område og beskjæringen vil ikke kunne foretas. For å forhindre dette må:

- kravet til markert området settes lavere
- brukeren selv definere beskjæringspunkter eller
- rammen som videoen beskjæres etter kan ikke inneholde store objekter

8 Konklusjon

Vil temporal medianfiltrering være en egnet metode for å fjerne objekter i bevegelse?

Når objekter i bevegelse medianfiltreres over tid, vil man få man en svak gjengivelse av objektene. Desto raskere objektene beveger seg, desto mindre representasjon får det i hver posisjon og desto mer utydelig blir det. Når objektet befinner seg i ro, vil det etter hvert gjengis tydeligere. Hvor fort dette skjer, er avhengig av hvor langt tidsrommet medianfiltreringen foregår over. Så lenge objekter befinner seg i bildet, vil det til enhver tid gi en viss representasjon. Denne metoden gir vil derfor ikke kunne fjerne objekter fullstendig.

Vil man få ytterligere forbedringer av objektsegmenteringen ved å kombinere bevegelsesinformasjon og fargeinformasjon?

Resultatet blir forbedret betraktelig ved bruk av fargeinformasjon. Masker som ble utviklet ved bruk av CbCr-komponentene i bildet gav en bedre løsning enn den euklidske fargeavstanden.

Vil programmet håndtere ulikt belyste områder?

Når området som skal markeres har ulik belysning eller farge, vil fargeavstandsmasker måtte slås sammen for å dekke de ulikt belyste områdene. Ved å justere terskelverdiene for store fargeavstander som skal aksepteres, kan man få en god overlapping og dermed dekket hele området. Denne metoden krever at det tas fargeprøver i de ulike belyste områdene og at man tilpasser terskelverdien slik at man får en god overlapping, men samtidig liten representasjon av utenforliggende områder. Ulik belysning påvirker ikke CbCr-maskene i synlig grad.

Vil bruk av CbCr-komponenter gi en adaptiv og bedre avgrensing enn RGB-verdier?

Når man genererer masker ved bruk av CbCr-komponentene, trenger man ikke å foreta fargemålinger. Bakgrunnen skilles fra forgrunnen og man må kun velge en terskelverdi for hvor mye av forgrunnen som skal inkluderes i maska. Dersom maska dekker hele personen vil personen fjernes fullstendig. Denne metoden er raskere å tilpasse enn fargemasker og avgrensningen blir bedre og løsningen er mer adaptiv.

Vil man få en bedre dekkende maske ved å tette små hull som ikke har blitt inkludert i maska?

Hull kan tettes ved bruk av medianfiltrering og ved å addere maskene med tilhørende tekstmaske. Ingen av disse metodene vil ga en fullstendig tetting av hull. Den utviklede algoritmen derimot, gir en fullstendig tetting ved valg av riktig hullstørrelse. Ved kombinasjon av horisontal og vertikal tetting gir den forbedrede masken en 100% markering av tavleområdet.

Hvordan kan forstyrrelsen fra konturen til objekter fjernes?

Ved å detektere kanter i den fullstendige masken finner man konturen av foreleseren. Siden det er ønskelig å utvidde konturen kan kantene medianfiltreres med en 8x8 matrise. Den kantdetekterte konturen blir da en tykk strek som går utenfor selve avgrensin-

gen til personen. Ved å trekke ifra denne streken ifra masken, vil objektets kontur bli utviddet.

Vil kantinformasjon gi en god representasjon av skrift?

Siden det i den hvite krittskrifta er mye representasjon av grønnfargen i tavla er det problematisk å få en tydelig avgresning. Prewitt, Sobel og FreiChen operatoren gir en forholdsvis god deteksjon av skrift. Robert gir en nokså utydelig gjengivelse. Problemet med håndskrift er at den ofte er utydelig, en oppløsningen på 720x576 er forholdsvis lav og forholdene må derfor legges til rette for å få en tydelig skriftgjengivelse.

Vil sammenslåing av bilder av detektert skrift over et tidsrom gi en forbedring av skriftgjengivelsen?

Ved å legge sammen flere tekstmasker, vil resultatet forbedres fordi mange av delene som mangler i teksten blir fylt igjen. Problemet da er at også støy fra flere bilder blir addert. Ved å sette et minimumskrav til forekomst i hvert piksel kan denne støy filtreres vekk.

Vil det la seg gjøre å digitalisere håndskifta på tavla ved bruk av skriftgjenkjenningsprogram?

Problemet med skrift er at stekene ikke alltid er sammenhengende og skriftgjenkjenning vil derfor ikke kunne foretas. Ved å foreta morfologisk sammensetting av linjer får man tettet noen slike gap, men man får også noe mer feil. Grunnen til dette er at håndskrifta til forskjellige personer varierer mye og det blir derfor tettet kanter som ikke skal tettes. Skriftgjenkjenning av håndskrift ved bruk av ABBYY FineReader gir et dårlig resultat. Siden håndskrifta på tavla ofte er slurvete og ikke skrevet fullstendig horisontalt vil det ikke la seg gjøre å detektere den med dette programmet.

Hvordan kan man finne tidspunktene når skrifta på tavla pusses vekk og innholdet skal lagres i bilder?

Skrifta er det eneste som beveger seg i filmen etter objektsegmenteringen er foretatt. Når tavla fulles med tekst, øker totalmengden av hvite piksler. Når det pusses vekk tekst, reduseres mengden raskt. Ved å detektere når det skjer en reduksjon finner man tidspunktet for når teksten skal lagres.

Vil en bevegelsessegmentert video være redusert i størrelse og vil videre prosessering gå raskere?

En hurtigavspillingsvideo vil være godt redusert i størrelse. Hvor mye den er begrenser avhenger av hvilke krav det settes til bevegelse og hvor mye bevegelse som forkommer i filmen (figur 115).

Vil en beskåret video være redusert i størrelse?

Dersom arealet av det maskerte området ikke dekker hele bildet, vil man kunne beskjære videoen og filstørrelsen blir redusert proporsjonalt med beskjæringen. Ved å sende med parametere til hvordan videoen skal beskjæres, kan dette programmet ha en stor nytteverdi i videoredigering.

Oppsummeringsspørsmål

Vil objekter i video kunne segmenteres slik at kun bakgrunnen blir synlig?

Hvor godt objekter blir segmentert avhenger av hvor nøyaktig maskene er som avgrenser objektet. Ved sammenligning av RGB- og CbCr-masker viser det seg at CbCr-masker gir best segmentering. Kun 1,3 % av objekter er fortsatt synlig etter segmentering ved bruk

av CbCr-maske. Ved å gjøre en temporal medianfiltrering kan man glatte ut noe av denne forstyrrelsen og resultatet vil forbedres ytterligere. Etter temporal medianbevegelse er kun 0,87 % av objektet fortsatt synlig. Denne verdien varierer noe avhengig av om foreleseren står i ro eller beveger seg mye.

Vil tekst i forelesningsvideo kunne detekteres og gjengis slik at den blir leselig?

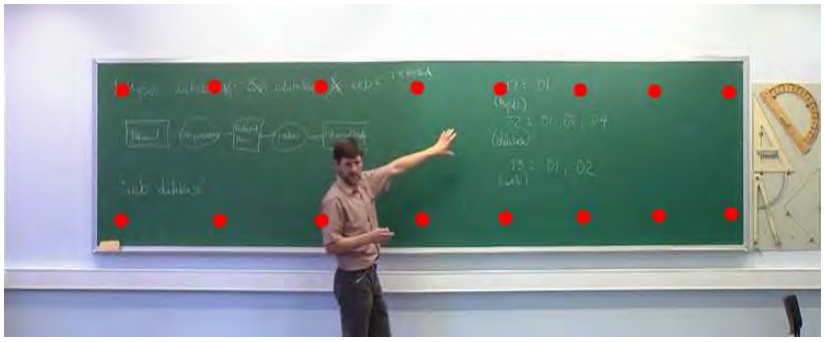
Sobel-, Prewitt- og FreiChen-operatoren er alle anvendelige til å detektere tekst. Robert derimot gir en dårlig gjengivelse. Med en kvalitativ analyse kan man se at Prewitt gir den beste gjengivelsen. Ved å slå sammen flere rammer med detektert tekst vil resultatet forbedres betraktelig. Dette skyldes at ikke all teksten blir detektert i hvert bilde og ved å slå sammen flere bilder får man en bedre gjengivelse.

Morfologiske operasjoner kan brukes til å lage sammenhengende tekst i de tilfeller det mangler små bruddstykker. Ulempen med morfologiske operatorer er at kanter i tekst som ikke hører sammen kan bli linket.

9 Videre arbeid

9.1 Algoritmisk utvikling av masker

Gjennomført metoden for å generere masker ut ifra den euklidske fargeavstanden er gjennomførbar, men ulempen er at den krever 2 målinger og tilpassing av 2 terskelverdier. En bedre metode vil være å detektere fargene algoritmisk ved å foreta fargemålinger innenfor sentrum av bildet (figur 125).



Figur 125: *Punktvise målinger*

Når man så har funnet de ulike opplyste områdene kan terskelverdiene tilpasses algoritmisk. For å gjøre dette trenger man en grov maske eller en geometrisk form algoritmen skal prøve å oppnå. Utvikling av adaptiv terskling er tidligere gjort og det ble derfor ikke lagt stor vekt på dette [36].

9.2 Bevegelsessegmentering

Algoritmen har følgende funksjonalitet:

- generering av hurtigavspillingsvideo som kun består av de detekterte bevegelsesrammene.
- generering av bevegelsessegmentert video i normal hastighet. Samme bevegelsesramme gjentas inntil en ny bevegelsesramme detekteres.

Etter at det er foretatt objektsegmentering og skriftdeteksjon på hurtigavspillingsvideoen, kan det være ønskelig gå tilbake til normal avspillingshastighet. For å gjøre dette må det videreutvikles en algoritme som lagrer rammenumrene til rammene som ble detektert i bevegelsessegmenteringen. Disse rammenumrene må lagres og kan da brukes til å sette sammen en film i normal hastighet slik det er forklart i punkt 2.
Terskel for hvor stor differanseverdien mellom to rammer skal være, angis av brukeren. Det kan her utvikles en tilleggsfunksjon hvor brukeren angir et minimumskrav til antall rammer per sekund. Hvis det ikke forekommer noe bevegelse på en god stund eller brukeren har valgt en for høy terskelverdi, kan for eksempel de tre rammene med mest bevegelse i løpet er sist sekund ivaretas. Denne funksjonaliteten vil ha mest nytteverdi i det tilfellet der brukeren har valgt en for høy terskelverdi. Det andre tilfelle vil jo være

at det ikke forekommer bevegelse, og det er nettopp de rammene denne funksjonen skal segmentere vekk.

9.3 Skrift

Skriftgjenkjenning av håndskrift har det blitt gjort mye arbeid med. Siden håndskrift er personavhengig, må det utvikles et eget mønster for hver enkelt person hvis gjenkjenningen skal bli god. Ved å lage en database som inneholder håndskrifta til mange personer vil man være i stand til å gjenkjenne mange former for håndskrift. Dette kan gjøre ved å lagre en håndskrevet og maskinskrevet utgave av en tekst. Utfordringer med tekst er at den som regel ikke er helt rett og ikke følger en linje nøyaktig. Skriver man i noe kursiv form, vil også dette gi en utfordring. Hver enkelt bokstav må tilpasses en normalskrift før gjenkjenningen kan foretas.
Når det blir foretatt tekstreduksjon lagres teksten i en serie med bilder. Disse bildene ble manuelt åpnet i ABBYY og tekstgjenkjenning ble foretatt. Det vil være en fordel å utvikle en algoritme som foretar dette gjenkjenningen automatisk på de detekterte bildene.

9.4 Brukerundersøkelse

9.4.1 Materiale

Videofiler:

- Etter beskjæring i tid og rom
- Objektsegmentering ved bruk av RGB-verdier
- Objektsegmentering ved bruk av CbCr
- Leselighet av skrift

  Bildeserie av skrifta på tavla og beskjæringsgrafer.

9.4.2 Spørreskjema

Vurderingen foretas ved å sette en verdi fra 1 til 6 (6 er best!). Skriv gjerne også kommentarer.

- Hvordan vil du vurdere nytteverdien av beskjæring i tid og rom?
- Hvor god synes du objektsegmenteringen er ved bruk av
  fargemaske?
  CbCr?
- Hvor god er leseligheten til skrifta?
- Er dette et program som kan være nyttig i undervisningssammenheng?

Figurer

1 *Bildet kan deles inn i 3 lag* . . . 3
2 *Forgrunnen* . . . 3
3 *Teksten* . . . 3
4 *Bakgrunnen* . . . 3
5 Sammensetting av fargekanalene rød, grønn og blå[1] . . . 7
6 *Filstruktur [6]* . . . 8
7 Global bevegelsesdeteksjon [15] . . . 9
8 *Støyfjerning ved bruk av MRF-bassert klassifisering [24]* . . . 10
9 Fjerning av impulsstøy[15] . . . 11
10 Maskering[15] . . . 12
11 *Farger illustrert som terninger [35]* . . . 13
12 Terskelverdier[15] . . . 14
13 Godt separerte histogramverdier [15] . . . 14
14 *Deteksjon av personer i video [47]* . . . 15
15 *Snaking av and i fire ulike videorammer [51]* . . . 16
16 *Medianfiltrering for å finne bakgrunnen [24]* . . . 16
17 *Temporal medianfiltrering [52]* . . . 17
18 *Orginal kamerabilde, 512x512 piksler [15]* . . . 19
19 Roberts, Prewitt, Sobel og FreiChen-operatoren[15] . . . 19
20 *Pikselrepresentasjon av bosktaven a[61]* . . . 20
21 Morfologisk erosjon og utvidelse [62] . . . 21
22 *Krav til ulike typer videoformat [15]* . . . 22
23 *Baxite [61]* . . . 23
24 Kvalitetstap i JPEG-bilder[61] . . . 24
25 *Filming med et kamera[67]* . . . 25
26 *Hele tavla* . . . 25
27 *Filming med to kameraer[67]* . . . 25
28 *Venstre* . . . 25
29 *Høyre* . . . 25
30 Rammer med tilhørende masker . . . 26
31 *Fargermålinger i Adobe Photoshop som gir gode overlappende masker* . . . 26
32 *Kun Delvis markering av tavleområdet* . . . 27
33 *Sammenslåing av masker av ulikt belyste områder* . . . 27
34 *Gråtonemaske* . . . 28
35 *Sort/hvitt-maske* . . . 28
36 Markering av tavleområde . . . 28
37 *Romlig medianfiltrering av figur 34 med en 3x3 matrise.* . . . 29
38 *Romlig medianfiltrering av figur 34 med en 9x9 matrise.* . . . 29
39 Personkontur . . . 29
40 Medianfiltrert mask . . . 29

41 Kromatisitet . . . 30
42 Maskering av bildet med lik kromatisitet . . . 30
43 *Krittmaske for normalt belyst tavleområde* . . . 31
44 *Krittmaske for kritt i belyst område* . . . 31
45 Objektsegmentering med bruk av binære masker . . . 32
46 Objektsegmentering ved bruk av sort/hvitt-maske . . . 32
47 *Visualisering av objektkontur* . . . 33
48 *Objektkontur over tid* . . . 33
49 *Objektsegmetnering ved bruk av gråtonemaske* . . . 33
50 *Objektsegmentering ved bruk av medianfiltrert maske* . . . 33
51 Sammenlikning av RGB-masker . . . 34
52 *Originalbilde* . . . 35
53 *Cb-komponent* . . . 35
54 *Cr-komponent* . . . 35
55 *Addering av Cb- og Cr-komponentene* . . . 36
56 *Tersklet og invertert CbCr-maske* . . . 36
57 *Segmentering med RGB-mask* . . . 36
58 *Segmentering med CbCr-mask* . . . 36
59 *Piksler med over 92% intensitet er hvit* . . . 37
60 *Piksler med over 96% intensitet er hvit* . . . 37
61 *Addert Cb- og Cr-komponenter etter beskjæring av video* . . . 37
62 *Terskelverdien måtte tilpasses slik at piksler med over 96% intensitet blir hvite* 37
63 *Flytskjema for objektsegmentering* . . . 38
64 *Objektsegmentering ved bruk av tersklet CbCr-maske* . . . 38
65 Sammenlikning av segmenteringsgrad til RGB- og CbCr-masker . . . 40
66 Forstyrrelse i maske . . . 41
67 *Flytskjema for segmentering etter forbedring av maskering* . . . 42
68 *Flytskjema for fylling av hull* . . . 43
69 *Orginal CbCr-maske* . . . 44
70 *Horisontal tetting av hull* . . . 44
71 *Vertical tetting av hull med utgangspunkt i figur 69* . . . 44
72 *Kombinasjon av horisontal og vertikal fylling* . . . 44
73 *Horisontal fylling av hull som er mindre enn 10 piksler* . . . 44
74 *Horisontal fylling av hull som er mindre enn 20 piksler* . . . 44
75 *Deteksjon av tekst ved bruk av fargeavstand* . . . 45
76 *Tekstmaska subtrahert fra figur 69. Teksten fyller da noen av hullene i maska* 45
77 Sammenligning av tetting med CbCr- og tekstmaske . . . 45
78 *Det røde omrisset skal subtraheres fra maska* . . . 46
79 *Objektkontur* . . . 47
80 Utvidelse av objektmarkering . . . 47
81 Segmentering etter forbedring av masker . . . 47
82 *Søylen til høyre viser segmenteringsgraden etter at forbedring er foretatt* . . . 48
83 *Medianfiltrering med en 8x8 maske av figur 4.3.2* . . . 48
84 *Gjennomsnittsbilde* . . . 49
85 *Medianbevegelse* . . . 49
86 *Medianbevegelse etter objektsegmentering med CbCr-maske* . . . 50

87 Sammenligning av utviklede metoder . . . 51
88 *Kontraststrekking* . . . 53
89 *Sobel kantdeteksjon ved bruk av to matriser* . . . 54
90 *3x3 Sobel* . . . 55
91 *3x3 Prewitt* . . . 55
92 *3x3 FreiChen* . . . 55
93 *3x3 Robert* . . . 56
94 *Snitt av orginalbilde* . . . 56
95 *Analyse av 3x3 Sobel-operator* . . . 56
96 *Analyse av 3x3 Prewitt-operator* . . . 56
97 *Analyse av 3x3 FreiChen-operator* . . . 56
98 *Analyse av 3x3 Roberts-operator* . . . 56
99 *Prewitt med lav terskelverdi (0.5)* . . . 57
100 *Prewitt med høy terskelverdi (4.0)* . . . 57
101 *Detektert kritt ved bruk av euklidsk fargeavstand* . . . 57
102 *Addisjon av krittmasker over tid uten eliminering av støy* . . . 58
103 *Addisjon av krittmasker over tid med eliminering av støy* . . . 59
104 Deteksjon av tidspunktene når skrifta på tavlen fjernes . . . 60
105 *Kamera som kun filmer tavleområdet* . . . 61
106 *Tekst med høy oppløsning og forholdsvis gjenkjennelige bokstaver* . . . 61
107 *Prewitt deteksjon av stor skrift* . . . 62
108 *Reduksjon av støy* . . . 62
109 *Morfologisk operasjon (dilation) for å få sammenhengende tekst* . . . 62
110 *Morfologisk operasjon (erosjon) for å få tilbake normal skrifttykkelse* . . . 62
111 OCR skriftgjenkjenning . . . 62
112 *Flytskjema for segmentering av bevegelsesrammer* . . . 65
113 *Sammenligningsbassert bevegelsesrammeuttrekking* . . . 66
114 Visualisering av bevegelse . . . 67
115 Reduksjon av bildefrekvens . . . 67
116 Rammer med bevegelsesdifferanse på mer enn 3 prosent . . . 68
117 Diagram som viser hyppigheten på bevegelsesrammer . . . 68
118 *Differanse på 1 prosent* . . . 69
119 *Differanse på 5 prosent* . . . 69
120 *Flytskjema for beskjæring av video* . . . 70
121 *Tavlemaske genererte med standard tavlefarge og en lav terskelverdi* . . . 70
122 *Avgrensning av området som skal skrives til ny videofil* . . . 71
123 *Videofil etter beskjæring i areal* . . . 71
124 Diagram for datastøm i video etter reduskjon . . . 72
125 *Punktvise målinger* . . . 77

Bibliografi

[1] Wikipedia. Wikipedia ”the free encyclopedia” (ycbcr). *Wikimedia Foundation Inc. URL: http://wikimediafoundation.org/wiki/Home*, 2005.

[2] Kyung-Yung Choi and K. Takaya. Facial feature extraction from a video sequence using independent component analysis (ica). *Communications, Computers and signal Processing*, 1.

[3] H. Hasegawa S. Nagumo and N. Okamoto. Extraction of forward vehicles by front-mounted camera using brightness information. *IEEE*, 2003.

[4] M.C. Huang and S.H. Yen. A real-time and color-based computer vision for traffic monitoring system. *IEEE International Conference and Multimedia and Expo(ICME)*, 2004.

[5] Aguierre Smith T.G. Davenport, G. and N. Pincever. Cinematic primitives for multimedia. *IEEE Computer Graphics & Applications*, pages 67–74, 1991.

[6] Hetland M. Videoverktøy integrert i world wide web. *Hovedoppgave utført ved IDI, NTNU*, 1996.

[7] J. Konrad and F. Dufaux. Improved global motion estimation forn3. *Tech. Rep. MPEG97/-M3096, ISO/IECJTC1/SC29/WG11*, 1998.

[8] J. Konrad and E. Dubois. Bayesian estimation of motion vector fields. *IEEE Trans. Pattern Anal. Machine Intell. 14,*, pages 910–927, 1992.

[9] J. Konrad M. Chahineand. Estimation and compensation of accelerated motion for temporal sequence interpolation. *Signal Prow, Image Commun. 7*, pages 503–527, 1995.

[10] T. Aach and A. Kaup. Bayesian algorithms for adaptive change detection in images sequences using markov random fields. *Signal Process. Image Commun. 7*, pages 147–160, 1995.

[11] Daniel F. DeMenthon Ryan C.Jones and David S. Doermann. Building mosaics from video using mpeg motion vectors. Master’s thesis, University of Maryland, 1999.

[12] C. Stiller. Object-based estimation of dense motion fields. *IEEE Trans. Image Process. 6*, pages 234–250, 1997.

[13] I. Sezan and eds. R. Lagendijk. Motion analysis and image sequence processing. *Kluwer, Boston, MA*, 1993.

[14] A. Caplier F. Luthon and M. Lievin. Spatiotemporal mrf approach with application to motion detection and lip segmentation in video sequences. *Signal Process. 76*, pages 61–80, 1999.

[15] Al Bovik. Handbook of image and video processing. *Academic Press, USA*, 2000.

[16] Christoph Stiller and Janusz Konrad. Estimation motion in image sequences. *IEEE Transactions on circuits and systems for video technology*, pages 1147–1167, 1999.

[17] Xingquan Zhu. Insightvideo. *Dep. of Computer Science*, page 41, 2004.

[18] HongJiang Zhang. Content-based video browsing and retrieval. *CRC Press LLC*, pages 255–277, 1999.

[19] I. Pitas and A. Venetsanopoulos. Edge detectors based on order statistics. *IEEE Trans. Pattern Analysis and Machine Intelligence, PAMI-8*, July 1986.

[20] D. Lau and G. R. Arce. Edge detector using weighted median filter. *Tech. Rep. 97-05-15, Department of Computer and Electrical Engineering, University of Delaware*, 1997.

[21] R. Hardie and Arce G.R. Ranking in rf and its use in multivariate image estimation. *IEEE Transactions on Circuits Syst. for Video Technol. 1*, June 1991.

[22] D. Karakos P. E. Trahanias and A. N. Venetsanopoulos. Directional processing of color images: Theory and experimental result. *IEEE Transactions on Image Processings*, June 1996.

[23] D. L. Paredes and G. R. Arce. Stack filters, stack smoothers, and mirror threshold decomposition. *IEEE Transactions on Signal Processing 47*, (10), Oct. 1999.

[24] Borko Furth and Oge Marques, editors. *Video Databases, Design an applications*. CRC PRESS, 2th edition, 2003.

[25] G. R. Arce. A general weighted median filter structure admitting real-valued weights. *IEEE Transactions on Signal Processing, SP-46,*, Des. 1998.

[26] P. H. L. Yin J. Astola and Y. Neuvo. Vector median filters. *Proceedings of the IEEE 78*, April 1990.

[27] Maria Petrou & Panagiota Bosdogianni. *Image Processing: The Fundamentals*. John Wiley & Sons Ltd, 1999.

[28] T. S. Huang A. C. Bovik and D. C. Munson Jr. The effect of median filtering on edge estimation and detection. *IEEE Trans. Pattern Anal. Machine Intell. PAMI-9*, pages 181–194, 1987.

[29] N. Himayat V. Koivunen and S. Kassam. Nonlinear filtering techniques for multivariate images - design and robustness characterization. *Signal Processing 57*, Feb. 1997.

[30] V. Koivunen. Nonlinear filtering of multivariate images under robust error criterion. *IEEE Transactions on Image Processing 5*, June 1996.

[31] I. Pitas and P. Tsakalides. Multivariate ordering in color image filtering. *IEEE Transactions on Circuits Syst. for Video Technol. 1*, Feb.1991.

[32] M. W. Marcellin and T. R. Fischer. Trellis coded quantization of memoryless and gauss-markov sources. *IEEE Trans. Commun. 38*, pages 82–93, 1990.

[33] R. R. Coifman and M. V. Wickerhauser. Entropy based algorithms for best basis selection. *IEEE Trans. Inf. Theory 32*, pages 712–718, 1992.

[34] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Trans. Circuits Syst. Video Technol. 6*, pages 243–250, 1996.

[35] TIFF imaging Solutions Image Maker. Introducing the colorcube: 3d color puzzle. *Spittin' Image Software, Inc*, 2000.

[36] B. Neil H. Nariman, M. Alireza. Automatic thresholding for change detection in digital video. *in Proc. SPIE 4067*, pages 133–1452, 2000.

[37] J. Astola 0. Yli-Harja and Y. Neuvo. Analysis of the properties of median and weighted median filters using threshold logic and stack filter representation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, (38), 1991.

[38] E. Coyle J. Fitch and N. Gallagher. Median filtering by threshold decomposition. *IEEE Trans. Acoustics, Speech and Signal Processing, ASSP-32*, Dec. 1984.

[39] William K. Pratt. *Digital Image Prosessing*. John Wiley & Sons,Inc., 2001.

[40] Liang-Tien Chia Haoran Yi, Deepu Rajan. A new motion histogram to index motion content in video segments. *Elsevier B.V.*, 2004.

[41] Thomas B. Moeslund. Computer vision-based human motion capture - a survey. Master's thesis, University of Aalborg, 1999.

[42] Emile A. Hendriks Ioannis Patras and Reginald L.Lagendijk. Semi-automatic object-based video segmentation with labeling of color segments. *Elsevier Science B.V.*, 2002.

[43] Berna Erol Dar-Shyang Lee and Jonathan J.Hull. Segmenting people in meeting videos using mixture background and object models. *Springer-Verlag Berlin Heidelberg*, pages 791–789, 2002.

[44] Hongqiang Bao and Zhaoyang Zhang. Motion objects segmentation using a new level set based method. *Springer-Verlag Berlin Heidelberg*, pages 312–318, 2004.

[45] Jie Yang Zhi Liu and Ningsong Peng. A new tracking mechanism for semi-automatic video object segmentation. *Springer-Verlag Berlin Heidelberg*, pages 824–832, 2004.

[46] M.H.Park J.B. Kim, H.S. Park and H.J. Kim. Unsupervised moving object segmentation and recognition using clustering and a neural network. *IEEE*, 2002.

[47] Marti Balcells Capellades. An appearance based approach for human and object tracking. Master's thesis, University of Maryland, 2002.

[48] K.M.Lee-T.S.Yun H.J.Kim J.B. Kim, C.W. Lee. Wavelet-based vehicle tracking for automatic surveillance systems. *in Proc. IEEE TENCON, 1*, pages 313–316, 2001.

[49] S. Peleg M. Irani. Motion analysis for image enhancement: Resolution, occlusion and transparency. *Int. Journal of Vis. Comm. Image Repr. , 4 (4)*, pages 324–335, 1993.

[50] H.J. Kim E.Y. Kim, E.Y. Park. A generic algorithm based segmentation of markov random field modeled images. *IEEE Signal Processing Letters, 7 (11)*, pages 301–303, 2000.

[51] D.R. Haynor S. Sun and Y.Kim. Semiautomatic video object segmentation using vsnakes. *IEEE*, 2003.

[52] L.J. Le Roux and J.J.D. van Schalkwyk. An overview of moving object segmentation in video images. *IEEE*, 1991.

[53] David Doermann Huiping Li and Omid Kia. Automatic text detection and tracking in digital video. Master's thesis, University of Maryland, 1998.

[54] Mai K. Zabih R., Miller J. A feature-based algorithm for detecting and classifying production effects. *ACM/Springer Multimedia Systems*, 7(2), 1999.

[55] P. Salembier and F. Marques. Region-based representations of image and video: segmentation tools for multimedia services, 1999.

[56] V. Berzins. Accuracy of laplacian edge detectors. *Comput. Vis. Graph. Image Process*, (27):195–210, 1984.

[57] D. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Machine Intell. PAMI-8*, pages 679–698, 1986.

[58] M. Perez-Luque M. Cebrian and G. Cisneros. Edge detection alternatives for multispectral remote sensing images. *in Proceedings of the 8th Scandinavian Conference on Image Analysis (NOBIM-Norwegian Soc. Image Process & Pattern Recognition, Tromso, Norway*, pages 1047–1054, 1993.

[59] Ashley Walker Erik Wolfart Robert Fisher, Simon Perkins. Image processing learning resources. *http://homepages.inf.ed.ac.uk/rbf/HIPR2/hipr_top.htm*, 2004.

[60] Woods R.E. Gonzalez R.C. Digital image processing. *Addison-Wesley Publishing Company, Inc.*, 1992.

[61] Øyvind Kolås. Aluminium. *pippin.gimp.org*, 2004.

[62] M.J. Sammon M. Seul, L. O'Gorman. *Practical Algorithms for Image Analysis*. Cambridge University Press, 2001.

[63] Otsuji K. Tonomura Y., Akutsu A. and Sadakata T. Videomap and videospaceicon: Tools for anatomizing video content. *Proceedings of ACM INTERCHI Conference on Human Factors in Computing Systems*, pages 131–141, 1993.

[64] Smoliar S.W. Zhang H., Low C.Y. Video parsing and browsing using compressed data, multimedia tools and applications. *Kluwer Academic Publishers*, 1:89–111, 1995.

[65] Ole Martin Bakke og Øystein Langerak. Datastøttet videoanalyse. Master's thesis, Norges Tekniske Naturvitenskapelige unvisersitet, 2000.

[66] C. Chrysafis D. Taubman and A. Drukarev. Embedded block coding with optimized truncation. *ISO/IEC JTC1/SC29/WG1, JPEG-2000 Document WG1N1129*, Nov. 1998.

[67] Andrei Ouglov. Panoramic video in video mediated education. *EI konferanse i San Jose*, 2005.

Appendix – Object segmentation and text detection in lecture video

Eirik Grythe
E-mail: eirik@grythe.dk
Project homepage: grythe.dk

Employer:
Gjøvik University College
http://www.hig.no
—
Supervisor:
Andrei Ouglov

2005/05/13

Innhold

Innhold . . . . . **iii**
1 Programbeskrivelse **1**
1.1 Oxide 1
1.2 RGB 1
1.2.1 Generering av masker(create mask) 1
1.2.2 Objektsegmentering (object remove) 1
1.3 YCbCr 1
1.4 Forbedring av masker 1
1.4.1 Fylling av hull i masker (Color fill) 1
1.4.2 Fjerning av objektkontur 2
1.5 Temporal medianbevegelse (mblur) 2
1.6 Behandling av skrift 2
1.6.1 Kantdeteksjon 2
1.6.2 Forbedring av leselighet (Text improvement) 2
1.6.3 Deteksjon av skriftredusksjon 2
1.7 Beskjæring av video i tid og rom 2
1.7.1 Bevegelsessegmentering (Motion segmentation) 2
1.7.2 Beskjæring av video (Area segmentation) 2
2 Kildekode **5**
2.1 Segmentering ved bruk av RGB-maske 5
2.1.1 objsegm_color.xml 5
2.1.2 ops/create_mask.c 5
2.1.3 ops/create_blackmask.c 8
2.1.4 ops/objsegm_color.c 11
2.1.5 SumRGB 12
2.2 Segmentering ved bruk av CbCr 16
2.2.1 objsegm_CbCr.xml 16
2.2.2 ops/Cb.c 17
2.2.3 ops/Cr.c 18
2.2.4 euklidsk_dist.c 18
2.2.5 ops/terskle_mask.c 20
2.2.6 ops/object_segm.c 22
2.3 Forbedring av masker 24
2.3.1 Fylling av hull 24
2.3.2 Fylling ved bruk av tekstmaske 29
2.3.3 Kontur 29
2.3.4 Fullstendig xml-fil 30
2.4 Temporal medianfiltrering 31
2.4.1 snitt.xml 31
2.4.2 ops/snitt.c 31

2.5 Tekst 34
2.5.1 Kant deteksjon 34
2.5.2 Forbedring av leselighet 39
2.5.3 Deteksjon av fjerning av skrift 49
2.6 Motion segmentation 56
2.6.1 motion_detect.c 56
2.6.2 ops/motion.c 61
2.7 Areasegmentation 64
2.7.1 area_segm.c 64
2.7.2 ops/area_restr.c 69
2.7.3 ops/crop.c 72
2.8 Diverse kode 74
2.8.1 visualize_motion.c 74
2.8.2 ops/bland_inn.c 75
2.8.3 ops/visual_motion.c 77
3 Installasjons 81
3.1 Linux 81
3.2 Oxide 83

1 Programbeskrivelse

1.1 Oxide

Ved eksekvering av Oxide angir man en xml-fil og eventuelt andre parametere. I xml-fila angir man hvilke bilde og videofiler som skal åpnes og lagres og hva kildekoden heter som skal gjøre selve prosesseringen.
Følgende paramtere kan sendes med:
–time-start value tid for første ramme som skal prosesseres (default 0.0)
–time-end value tid for siste ramme å prosessere (default 1.0)
–time-step value tid mellom hver ramme (default 1.0)
–gggl-ops-dir dir tilleggskatalog for gggl ops
–delay seconds antall sekunder forsinkelse mellom hver ramme (default 0)
-v prosess utskrift

1.2 RGB

1.2.1 Generering av masker(create mask)

For å generere masker må man ta målinger av fargerverdier i forskjellig belyste områder. Ved å ta utgangspunkt i en farge og regne ut hvor stor fargeavstanden er til de andre pikslene, kan man opprette en maske med gråtoner som indikerer avstand. I dette tilfellet indikerer lyse piksler at det er liten avstand til fargenen i referansefargen. En sort hvit maske viser seg å gi et bedre resultat enn en gråtonemaske. Ved å sette en terskelverdi hvor hva som regnes for hvitt, og sette resten av pikslene sorte får man en tydelig avgrensing. Det må genereres en maske for hver ramme i videoen. Dette gjøres ved å lage en xml fil som henter inn videofilen som skal behandles og kjører create mask for hver ramme.

1.2.2 Objektsegmentering (object remove)

Objekter i bevegelse skal ikke oppdateres. Ved kun å oppdatere piksler innenfor masken til hver ramme får man denne effekten. Siden områder som befinner seg utenfor masken skal forbli uberørt, blir teksten bak personen synlig selv om han står foran den.

1.3 YCbCr

1.4 Forbedring av masker

1.4.1 Fylling av hull i masker (Color fill)

Kun det hvite området i masken skal oppdateres. Color fill henter pikselverdiene i orginalevideon og legger verdienene inn i området som skal oppdateres. På den måten unngår man å oppdatere noe annet enn tavla og bakgrunnen fra første ramme i videoen kan brukes gjennom hele videoen.

1.4.2 Fjerning av objektkontur

1.5 Temporal medianbevegelse (mblur)

1.6 Behandling av skrift

1.6.1 Kantdeteksjon

1.6.2 Forbedring av leselighet (Text improvement)

For å forbedre leseligheten av tekst er man nødt til å legge sammen flere masker. Bruker man kun en maske vil bare deler av teksten bli representert og den vil derfor blir uleselig gjennom hele filmen. Totalverdien til alle maskene over et valgt tidsintervall legges i et bilde. Dette bildet inneholder representasjon av all teksten i bildet. Siden det også forekommer endel støy på ulike steder i bildet vil dette i løpet av noen sekunder utgjøre en betydelig representasjon. For å fjerne denne støyen kan man sette en terskelverdi for hvor hyppig representasjon man i et piksel for at det skal tolkes som tekst, og dermed ikke segmenteres vekk.

1.6.3 Deteksjon av skriftredusksjon

1.7 Beskjæring av video i tid og rom

1.7.1 Bevegelsessegmentering (Motion segmentation)

Motion segmentation regner ut differanseverdien mellom to rammer i video. Alle R,G og B-pikslenen sammenliknes i 2 rammer. På de stedene det har vært bevegelse, vil pikselverdien være forandret og dette maskeres som differanse. Dersom total differanse mellom to rammer overstiger valgt terskelverdi, blir den nye rammen markert som en differanseramme. Det opprettes så en ny video som kun består av differaserammer.
Ved å kun sammenligne to etterfølgende rammer vil man kun få markert brå bevegelser. I dette programmet settes det en peker til første ramme i videofilen. Denne rammen sammenlignes med nye rammer utover i videon helt til differansen overstiges. Første peker flyttes så til den nye differanserammen og peker nummer to leter seg videre utover i videoen. Programmet kjøres ved å skrive:
motion_segm ”videofil.avi”

Følgende paramtere kan sendes med:
–time-start value tid til første ramme som skal prosesseres (default 0.0)
–time-end value tid for siste ramme å prosessere (default 1.0)
–time-step value tid mellom hver ramme (default 1.0)
–gggl-ops-dir dir tilleggskatalog for gggl ops
–thumbnails for å generere ny videofil av differanserammene
–threshold value terskelverdi for bevegelse (default 0.03)
-v prosess utskrift

1.7.2 Beskjæring av video (Area segmentation)

Ved beskjæring kan man enten angi koordinater som vidoen skal beskjæres etter eller det kan brukes en maske. Ved bruk av maske blir det søkt etter tilstrekkelig lange maskerte horisontale og vertikale linjer. Første horisontale linjer som har en stor representasjon markerer toppen av tavla. Siste markerte linje innenfor dette området markerer nedre

del av tavla. På samme måte kan man detektere høyre og ventre punkt av maska ved å detektere tilstrekkelig lange vertikale linjer. Siden det forekommer noe støy i maska er det viktig å kjøre en sjekk på om det maskerte området utgjør en tilstrekkelig stor del av videobildet til å kunne være det området man ønsker å gjengi.

2 Kildekode

2.1 Segmentering ved bruk av RGB-maske

2.1.1 objsegm_color.xml

```
<oxide>
    <op type='ff_save'>
        <property name='resource'>objsegm_color.avi</property>
    </op>

    <op type='objsegm_color'>

<!--  <op name='median' type='median' horizontal='4' vertical='4'/> -->
<!--   <op type='create_mask' /> -->

        <op type='create_blackmask' />
        <op type='clone' idref='13'>
        </op>
    </op>

    <op type='avi'>
        <property name='resource'>hoyre.avi</property>
        <property name='frame' interpolation='proportional'></property>
        <output id='13'/>
    </op>
</oxide>
```

2.1.2 ops/create_mask.c

```
#include <gggl/op/gggl_op.h>
#include <math.h>   /* sqrt */
#include <stdio.h>  /* io debugging */

typedef struct {
    double red;
    double green;
    double blue;
    // Komplementærfarger for å kompanserer for opplyste områder
    double redk;
    double greenk;
    double bluek;
} InstanceData;


// Algoritme for å regne ut fargeavstand
static float rgb_distance ( float red0, float green0, float blue0,
                            float red1, float green1, float blue1)
{
    return sqrt ( (red1-red0)     * (red1-red0) +
                  (green1-green0) * (green1-green0) +
                  (blue1-blue0)   * (blue1-blue0));
}


static int
process (GgglOp *op)
{
    InstanceData *id;
    GgglImage    *src_img;
    GgglImage    *dst_img;
    int           width;
    int           height;
    int           x,y;

    float         r0,g0,b0;
    float         r2,g2,b2;

    src_img = op->input_pad[0];


    id = op->priv;
```

```
    /* leser inn fargekomponentene til første valgte farge */
    r0 = id->red;
    g0 = id->green;
    b0 = id->blue;

    /* leser inn fargekomponentene til komplementærfarge */
    r2 = id->redk;
    g2 = id->greenk;
    b2 = id->bluek;

    /* henter størrelsen til kilde bildet */
    width  = src_img->width;
    height = src_img->height;

    /* allokerer plass i minnet til bildet */
    dst_img = gggl_op_image_alloc (op, 0, op->out_pad_fmt[0], width, height);
    op->output_pad[0] = dst_img; // flyttes opp 1 linje?

    /* behandling av farge */
    for (y = 0; y < height; y++) {
        float *src;
float *dst;

/* Peker til bildene */
        src = (float *) gggl_image_offset (src_img, 0, y, 0);
        dst = (float *) gggl_image_offset (dst_img, 0, y, 0);

/* leser inn fargene til pikselet i bilde som skal analyseres */
        for (x = 0; x < width; x++) {
            float r1,g1,b1;
            float color_distance, color_distance1;

            r1 = src[0];
            g1 = src[1];
            b1 = src[2];

    // Mørk grønn
    color_distance = rgb_distance (r0,g0,b0,
                                           r1,g1,b1);

    // terskelverdi grønnfarge 0.18
    // Er ikke fargen tilstrekkelig lik den ønskede fargen,
    // settes pikselet sort
    if(color_distance < 0.19) {
        dst[0] = 1-color_distance;
     }
    else
        dst[0] = 0;


    // Lys grønn
            color_distance1 = rgb_distance (r2,g2,b2,
                                           r1,g1,b1);

    // terskelverdi andre grønnfarge 0.13
    if(color_distance1 < 0.13 && dst[0]==0 ) {
        dst[0] = dst[0] + (1-color_distance1);
     }
     // ellers skal dst[0] bil uforandret
// må ikke settes lik 0 for ikke å overskrive
// første maske

    // Fargebildet økes med 3 kanaler, gråtone med 1
            src += 3;
            dst += 1;
        }
    }

    // Kode for å generere maske for tavlekritt
    // Viser seg å være vanskelig å generere
    // Må ta utgangspunkt i avgrenset område og kontrast
    // forstere krittfargen
/*

    // Skriver direkte inn gjennomsnittsverdiene for krittfarge
    r0 = 0.42352;
    g0 = 0.52549;
    b0 = 0.49010;
```

```
    // Komplemenetærfarge der det er opplyst på tavla
    r2 = 0.43921;
    g2 = 0.61960;
    b2 = 0.55294;

     // behandling av krittfarge
    for (y = 0; y < height; y++) {
        float *src;
float *dst;

src = (float *) gggl_image_offset (src_img, 0, y, 0);
        dst = (float *) gggl_image_offset (dst_img, 0, y, 0);

        for (x = 0; x < width; x++) {
            float r1,g1,b1;
            float color_distance;
//     flaot color_distance1;
// leser inn fargene i pikselet i bilde som skal analyseres
            r1 = src[0];
            g1 = src[1];
            b1 = src[2];

// Mørk kritt
    color_distance = rgb_distance (r0,g0,b0,
                                           r1,g1,b1);

// terskelverdi hvit 0.15
    if(color_distance < 0.11) {
        dst[0] = 1-color_distance;
     }
    else
        dst[0] = 0;


// Restultatet blir ikke noe bedre når man kompanserer med
// lyst kritt. Det blir bare mye støy i maska
//
//     // Lys kritt
//            color_distance1 = rgb_distance (r2,g2,b2,
//                                            r1,g1,b1);
//
//      // terskelverdi andre grønnfarge 0.13
//     if(color_distance1 < 0.13 && dst[0]==0 ) {
//         dst[0] = dst[0] + (1-color_distance1);
//      }
//

            src += 3;
            dst += 1;
        }
    }

*/
    return 0;
}


// inputbilde skal være i rgbF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

// output i gF (gråtoner)
static int
query_out_pad_fmt (GgglOp *op,
                   int     pad_no,
                   int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}
```

```
static void
open (GgglOp *op)
{
    InstanceData *instance_data;

    instance_data = gggl_op_malloc (op, sizeof (InstanceData));

    /* legger inn standar grønnfarge */
    instance_data->red            = 0.35686; //0.22745;
    instance_data->green          = 0.50588; //0.37254;
    instance_data->blue           = 0.35686; //0.32156;

    /* lys grønn er en komplimentærfarge for at hele tavla skal bli maskert */
    instance_data->redk           = 0.55686; //0.33725;
    instance_data->greenk         = 0.66666; //0.56470;
    instance_data->bluek          = 0.46666; //0.47843;

    /* register our level property */
    gggl_op_property_new (op, "red",  GgglPropValue,
       &(instance_data->red),  "target red"   );
    gggl_op_property_new (op, "green",  GgglPropValue,
                          &(instance_data->green),  "target green"   );
    gggl_op_property_new (op, "blue",  GgglPropValue,
                          &(instance_data->blue),  "target blue"   );


    /* the GgglOp structure has a field that we are allowed
     * to use for our own purposes, by storing a pointer to
     * a structure, we can store arbitary large amounts of
     * data that are persistent for a Op instance */
    op->priv = instance_data;

    op->query_in_pad_fmt  = query_in_pad_fmt;
    op->query_out_pad_fmt = query_out_pad_fmt;
    op->process           = process;
}

static void *op_info[]=
{
    "create_mask" , open,
    "description", "create mask",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL
};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

2.1.3 ops/create_blackmask.c

```
#include <gggl/op/gggl_op.h>
#include <math.h>   /* sqrt */
#include <stdio.h>  /* io debugging */

typedef struct {
    double red;
    double green;
    double blue;
    // Komplementærfarger for å kompanserer for opplyste områder
    double redk;
    double greenk;
    double bluek;
} InstanceData;

// Algoritme for å regne ut fargeavstand
static float rgb_distance ( float red0, float green0, float blue0,
                            float red1, float green1, float blue1)
{
    return sqrt ( (red1-red0)     * (red1-red0) +
                  (green1-green0) * (green1-green0) +
                  (blue1-blue0)   * (blue1-blue0));
}


static int
process (GgglOp *op)
{
```

```
    InstanceData *id;
    GgglImage    *src_img;
    GgglImage    *dst_img;
    int           width;
    int           height;
    int           x,y;

    float         r1,g1,b1;
    float         r2,g2,b2;
//    float   r3,g3,b3;

    src_img = op->input_pad[0];


    /* leser inn fargekomponentene til første valgte farge */
    id = op->priv;

    r1 = id->red;
    g1 = id->green;
    b1 = id->blue;

    /* leser inn fargekomponentene til komplementærfarge */
    r2 = id->redk;
    g2 = id->greenk;
    b2 = id->bluek;

    // Skriver direkte inn gjennomsnittsverdiene for krittfarge
    // hoyre.avi // smove.avi
/*
    r3 = 0.43529; // 0.42352;
    g3 = 0.63529; // 0.52549;
    b3 = 0.53725; // 0.49010;

    // Komplemenetær krittfarge der det er opplyst på tavla. Brukes ikke
    // hoyre.avi // smove.avi

    r4 = 0.39608; // 0.43921;
    g4 = 0.55686; // 0.61960;
    b4 = 0.48235; // 0.55294;
*/

    /* henter størrelsen til kildebildet */
    width  = src_img->width;
    height = src_img->height;

    /* allokerer plass i minnet til bildet */
    dst_img = gggl_op_image_alloc (op, 0, op->out_pad_fmt[0], width, height);
    op->output_pad[0] = dst_img;

    // generering av masker
    for (y = 0; y < height; y++) {
        float *src;
float *dst;

// Peker til bildene
        src = (float *) gggl_image_offset (src_img, 0, y, 0);
        dst = (float *) gggl_image_offset (dst_img, 0, y, 0);

// leser inn fargene til pikselet i bilde som skal analyseres
        for (x = 0; x < width; x++) {
            float r0,g0,b0;
            float color_distance,
  color_distance1;
//   color_distance2;

            r0 = src[0];
            g0 = src[1];
            b0 = src[2];

    dst[0] = 0;

    // Mørk grønn
    color_distance = rgb_distance (r0,g0,b0,
                                           r1,g1,b1);

    // Lys grønn
            color_distance1 = rgb_distance (r0,g0,b0,
                                           r2,g2,b2);
/*
       // Mørk kritt
```

```
    color_distance2 = rgb_distance (r0,g0,b0,
                                        r3,g3,b3);
*/
    // Lyst kritt
    // Restultatet blir ikke noe bedre når man kompanserer med
    // lyst kritt. Det blir bare mye støy i maska...
/*            color_distance1 = rgb_distance (r0,g0,b0,
                                        r4,g4,b4);
*/

    // terskelverdier på mørk grønnfarge for å få overlappende masker:
    // hoyre.avi: 0.19   video1.avi:0.16  smove.avi:0.18
    // Er ikke fargen tilstrekkelig lik den ønskede fargen,
    // settes pikselet sort
    if(color_distance < 0.19)
        // dst[0] = 1-color_distance;
        dst[0] = 1;

    // terskelverdi lys grønnfarge
    // hoyre.avi: 0.13    video.avi:0.12   smove.avi:0.13
    if(color_distance1 < 0.13 && dst[0]==0)
        // dst[0] = dst[0] + (1-color_distance1);
        dst[0] = 1;
/*
    // terskelverdi mørk krittfarge
    // Verdier for smove.avi, begge masker:0.11, lys maska: 0.15
    if(color_distance2 < 0.11 && dst[0]==0)
            dst[0] = 1;
*/
    /* Bruker ikke lys krittfarge siden denne skaper mye støy
    if(color_distance3 < 0.08 && dst[0]==0)
            dst[0] = 1;
    */

    // ellers skal dst[0] bil uforandret.
      // Dst[0] må ikke settes lik 0 for ikke å overskrive
    // en av de andre maskene

    // Fargebildet økes med 3 kanaler, gråtone med 1
            src += 3;
            dst += 1;
        }
    }
    return 0;
}


// inputbilde skal være i rgbF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

// output i gF (gråtoner)
static int
query_out_pad_fmt (GgglOp *op,
                   int     pad_no,
                   int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}


static void
open (GgglOp *op)
{
    InstanceData *instance_data;

    instance_data = gggl_op_malloc (op, sizeof (InstanceData));

    /* legger inn standar grønnfarge */
        // hoyre.avi // video.avi //smove.avi
    instance_data->red   = 0.15686; // 0.35686;  //0.22745;
```

```
    instance_data->green = 0.37255; // 0.50588;  //0.37254;
    instance_data->blue  = 0.30196; // 0.35686;  //0.32156;

    /* lys grønn er en komplimentærfarge for at hele tavla skal bli maskert */
        // hoyre.avi //video.avi //smove.avi
    instance_data->redk  = 0.32549; //0.55686;  //0.33725;
    instance_data->greenk= 0.55294; //0.66666;  //0.56470;
    instance_data->bluek = 0.45882; //0.46666;  //0.47843;

    /* register our level property */
    gggl_op_property_new (op, "red",  GgglPropValue,
             &(instance_data->red),  "target red"   );
    gggl_op_property_new (op, "green",  GgglPropValue,
              &(instance_data->green),  "target green"   );
    gggl_op_property_new (op, "blue",  GgglPropValue,
     &(instance_data->blue),  "target blue"   );


    /* the GgglOp structure has a field that we are allowed
     * to use for our own purposes, by storing a pointer to
     * a structure, we can store arbitary large amounts of
     * data that are persistent for a Op instance */
    op->priv = instance_data;

    op->query_in_pad_fmt  = query_in_pad_fmt;
    op->query_out_pad_fmt = query_out_pad_fmt;
    op->process           = process;
}



static void *op_info[]=
{
    "create_blackmask" , open,
    "description", "create blacmask",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL
};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

2.1.4 ops/objsegm_color.c

```
#include <gggl/op/gggl_op.h>
#include <stdio.h>

typedef struct {
    GgglImage *held_image;
} Priv;


static int
process (GgglOp *op)
{
    Priv *p = op->priv;
    int y;
    int ystart, yend;
    int width;

    // sjekker om om det er noe i input bildene
    // før det i det hele tatt skrives til output
    if (!op->input_pad[0] || !op->input_pad[1]) {
        op->output_pad[0]=NULL;
        return -1;
    }
    // allokerer plass
    if (!p->held_image) {
     p->held_image = gggl_image_duplicate (op->input_pad[0]);
gggl_op_reg_alloc (op, p->held_image);
    }

    op->output_pad[0]=p->held_image;

    ystart = 0;
    yend = op->input_pad[0]->height-1;
    width = op->input_pad[0]->width-1;
```

```
    for (y = ystart; y <= yend; y++) {
        int x;
float *held; // resultat
        float *new; // nytt bilde fra video
        float *mask; // maskert område

        held = (float *) gggl_image_offset (p->held_image, 0, y, 0);
        new  = (float *) gggl_image_offset (op->input_pad[0], 0, y, 0);
        mask = (float *) gggl_image_offset (op->input_pad[1], 0, y, 0);

        for (x = width; x ; --x) {
    int c;
    int channels = op->input_pad[0]->channels;
    // En frame med rigtig fargeinnhold skal legges
    // oppå den eksisterende framen. Begge bildene skal være i rgb
    for (c = 0; c < channels; c++)
                // Det skal legges til så stor andel av det nye bildet som
// gråtonemask[0] tilsier. Der foreleseren er, oppdateres ikke
// masken. Men der skal innholdet som allerede ligger i held
// fortsette å være (tavla bak læreren)
        held[c] = held[c] * (1.0 - mask[0]) + new[c] * mask[0];

    new   += channels; // antall kanaler op->input_pad[0];
    held  += channels;
    mask  += channels;
        }
    }
    return 0;
}

// Bildene som lesses inn skal være i rgbF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

// Det skal leses inn to inputbilder som skal
// blandes til ett output bilde
static void
open (GgglOp *op)
{
    Priv *p = gggl_op_calloc (op, 1, sizeof (Priv));

    p->held_image        = NULL;
    op->priv             = p;
    op->input_pads       = 2;
    op->query_in_pad_fmt = query_in_pad_fmt;
    op->process          = process;
}


static void *op_info[]=
{
    "objsegm_color"        , open,
    "description", "algoritme for å fjerne objekter i"
                   "ved bruk av fargemasker",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL};


void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

2.1.5 SumRGB

objsegm_sumRGB.xml

```
<oxide>
    <op type='ff_save'>
        <property name='resource'>objsegm_sumRGB.avi</property>
    </op>
```

```
    <op type='mblur'>
        <property name='opacity'>0.04</property>
    </op>

    <op type='objsegm_color'>
        <op type='create_black_sumRGB' />
        <op type='sumRGB' />
<op type='clone' idref='13'>
        </op>
    </op>

    <op type='avi'>
        <property name='resource'>hoyre.avi</property>
<property name='frame' interpolation='proportional'></property>
        <output id='13'/>
    </op>
</oxide>
```

ops/sumRGB.c

```
#include <gggl/op/gggl_op.h>
#include <stdio.h>

typedef struct {
    GgglImage *held_image;
} Priv;


static int
process (GgglOp *op)
{
    Priv *p = op->priv;
    int y;
    int ystart, yend;
    int width;

    // sjekker om om det er noe i input bildet
    // før det i det hele tatt skrives til output
    if (!op->input_pad[0]) {
        op->output_pad[0]=NULL;
        return -1;
    }
    // allokerer plass
    if (!p->held_image) {
     p->held_image = gggl_image_duplicate (op->input_pad[0]);
gggl_op_reg_alloc (op, p->held_image);
    }

    op->output_pad[0]=p->held_image;

    ystart = 0;
    yend = op->input_pad[0]->height-1;
    width = op->input_pad[0]->width-1;
    for (y = ystart; y <= yend; y++) {
        int x;
float *dst_img;
        float *src_img;

        dst_img = (float *) gggl_image_offset (p->held_image, 0, y, 0);
        src_img = (float *) gggl_image_offset (op->input_pad[0], 0, y, 0);

        for (x = width; x ; --x) {
    int c;
    float sumRGB = 0; // sum av fargekomponenter fra alle kanaler
    for (c = 0; c < op->input_pad[0]->channels; c++)  {
        sumRGB += src_img[c];
//         fprintf(stderr, "src_img: %f\n", src_img[c]);
    }

//     fprintf(stderr, "sumRGB:  %f\n", sumRGB);

    for (c = 0; c < op->input_pad[0]->channels; c++) {
        dst_img[c] = src_img[c] / sumRGB;
// fprintf(stderr, "\tDst: %f\n", dst_img[c]);
    }
//     fprintf(stderr, "\n\n");


    dst_img += 3; // antall kanaler op->input_pad[0];
    src_img += 3;
        }
```

```
    }
    return 0;
}

// Bildene som lesses inn skal være i rgbF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

// Det skal leses inn to inputbilder som skal
// blandes til ett output bilde
static void
open (GgglOp *op)
{
    Priv *p = gggl_op_calloc (op, 1, sizeof (Priv));

    p->held_image        = NULL;
    op->priv             = p;
    op->query_in_pad_fmt = query_in_pad_fmt;
    op->process          = process;
}


static void *op_info[]=
{
    "sumRGB"     , open,
    "description", "SumRGB"
                   "redusere fargespranget i bildet",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL};


void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

ops/create_black_sumRGB.c

```
/* Genererer en maske av sumRGB og terskler masken
 * I dette programmet trengs det kun 1 refereansefarge og 1 terskling
 * siden intensitetsforskjeller har blitt utjevnet */

#include <gggl/op/gggl_op.h>
#include <math.h>   /* sqrt */
#include <stdio.h>  /* io debugging */

typedef struct {
    double red;
    double green;
    double blue;
} InstanceData;

// Algoritme for å regne ut fargeavstand
static float rgb_distance ( float red0, float green0, float blue0,
                            float red1, float green1, float blue1)
{
    return sqrt ( (red1-red0)     * (red1-red0) +
                  (green1-green0) * (green1-green0) +
                  (blue1-blue0)   * (blue1-blue0));
}


static int
process (GgglOp *op)
{
    InstanceData *id;
    GgglImage    *src_img;
    GgglImage    *dst_img;
    int           width;
    int           height;
    int           x,y;

    float         r1,g1,b1;
```

```
    src_img = op->input_pad[0];


    /* leser inn fargekomponentene til første valgte farge */
    id = op->priv;

    r1 = id->red;
    g1 = id->green;
    b1 = id->blue;

    /* henter størrelsen til kildebildet */
    width  = src_img->width;
    height = src_img->height;

    /* allokerer plass i minnet til bildet */
    dst_img = gggl_op_image_alloc (op, 0, op->out_pad_fmt[0], width, height);
    op->output_pad[0] = dst_img;

    // generering av masker
    for (y = 0; y < height; y++) {
        float *src;
float *dst;

// Peker til bildene
        src = (float *) gggl_image_offset (src_img, 0, y, 0);
        dst = (float *) gggl_image_offset (dst_img, 0, y, 0);

// leser inn fargene til pikselet i bilde som skal analyseres
        for (x = 0; x < width; x++) {
            float r0,g0,b0;
            float color_distance;

            r0 = src[0];
            g0 = src[1];
            b0 = src[2];

    dst[0] = 0;

    color_distance = rgb_distance (r0,g0,b0,
                                           r1,g1,b1);

    if(color_distance < 0.04)
        // dst[0] = 1-color_distance;
        dst[0] = 1;

    // Fargebildet økes med 3 kanaler, gråtone med 1
            src += 3;
            dst += 1;
        }
    }
    return 0;
}


// inputbilde skal være i rgbF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

// output i gF (gråtoner)
static int
query_out_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}


static void
open (GgglOp *op)
```

```
{
    InstanceData *instance_data;

    instance_data = gggl_op_malloc (op, sizeof (InstanceData));

    /* legger inn standard grønnfarge */
    instance_data->red   = 0.26667;
    instance_data->green = 0.39608;
    instance_data->blue  = 0.32941;

    /* register our level property */
    gggl_op_property_new (op, "red",  GgglPropValue,
            &(instance_data->red),  "target red"   );
    gggl_op_property_new (op, "green",  GgglPropValue,
             &(instance_data->green),  "target green"   );
    gggl_op_property_new (op, "blue",  GgglPropValue,
     &(instance_data->blue),  "target blue"   );


    /* the GgglOp structure has a field that we are allowed
     * to use for our own purposes, by storing a pointer to
     * a structure, we can store arbitary large amounts of
     * data that are persistent for a Op instance */
    op->priv = instance_data;

    op->query_in_pad_fmt  = query_in_pad_fmt;
    op->query_out_pad_fmt = query_out_pad_fmt;
    op->process           = process;
}



static void *op_info[]=
{
    "create_black_sumRGB" , open,
    "description", "create blacmask",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL
};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

2.2 Segmentering ved bruk av CbCr

2.2.1 objsegm_CbCr.xml

```
<oxide>
    <op type='ff_save'>
        <property name='resource'>objsegm_CbCr.avi</property>
    </op>

<!--
    <op type='mblur'>
        <property name='opacity'>0.04</property>
    </op>
-->

   <op type='object_segm'>

    <!--hoyreterskel = 0.935 tavlepussterskel=0.92 venstre=0.96
        eirik.avi = 0.91  tavlepuss1 = 935-->
    <op type='terskle_mask' terskel='0.935' />

<!--    <op type='add_mask'> -->

        <op type='euklidsk_dist'>
         <op type='CB' />
         <op type='clone' idref='4'/>
        </op>

        <op type='CR' />
        <op type='clone' idref='4'/>
    </op>

    <op type='avi'>
        <property name='resource'>hoyre.avi</property>
```

```
        <property name='frame' interpolation='proportional'></property>
        <output id='4'/>
    </op>
</oxide>
```

2.2.2 ops/Cb.c

```
#include <gggl/op/gggl_op.h>
#include <stdio.h>

static int
process_lines (GgglOp *op, int ystart, int yend) {
    int       y;
    int       width;
    GgglImage *src_img;
    GgglImage *dst_img;

    src_img = op->input_pad[0];
    dst_img = op->output_pad[0];
    width = src_img->width;

    for (y = ystart; y <= yend; y++) {
        int       x;
        float    *src = (float *) gggl_image_offset (src_img, 0, y, 0);
        float    *dst = (float *) gggl_image_offset (dst_img, 0, y, 0);

        for (x = 0; x < width; x++) {
            float     R, G, B, Cb;
            R = src[0];
            G = src[1];
            B = src[2];
            Cb = (-0.1687) * R - 0.3313 * G + 0.5 * B + 0.5;
            *dst = Cb;

            src += 3;
            dst++;
        }
    }
    return 0;
}

static int
process (GgglOp *op) {
    if (!op->input_pad[0]) {
        op->output_pad[0] = NULL;
        return 0;
    }
    op->output_pad[0] = gggl_op_image_alloc (op, 0,
                                             op->out_pad_fmt[0],
                                             op->input_pad[0]->width,
                                             op->input_pad[0]->height);
    return process_lines (op, 0, op->output_pad[0]->height - 1);
}

static int
query_in_pad_fmt (GgglOp *op, int pad_no, int fmt) {
    if (fmt == gggl_pixfmt ("rgbF"))
      return 1;
    return 0;
}

static void
open (GgglOp *op) {
    op->out_pad_fmt[0] = gggl_pixfmt ("gF");
    op->query_in_pad_fmt = query_in_pad_fmt;
    op->set_in_pad_fmt = gggl_op_set_in_pad_fmt_dont_touch_out_pad;
    op->process = process;
}

static void *op_info[] = {
    "CB", open,
    "description", "henter cb komponenten i et bilde",
    NULL
};

void
gggl_op_init (void) {
    gggl_op_db_register (op_info);
}
```

2.2.3 ops/Cr.c

```
#include <gggl/op/gggl_op.h>
#include <stdio.h>

static int
process_lines (GgglOp *op, int ystart, int yend) {
    int       y;
    GgglImage *src_img = op->input_pad[0];
    GgglImage *dst_img = op->output_pad[0];
    int       width = src_img->width;

    for (y = ystart; y <= yend; y++) {
        int       x;
        float    *src = (float *) gggl_image_offset (src_img, 0, y, 0);
        float    *dst = (float *) gggl_image_offset (dst_img, 0, y, 0);

        for (x = 0; x < width; x++) {
            float     R, G, B, Cr;
            R = src[0];
            G = src[1];
            B = src[2];
            Cr = 0.5 * R - 0.4187 * G - 0.0813 * B + 0.5 ;
            *dst = Cr;
            src += 3;
            dst++;
        }
    }
    return 0;
}

static int
process (GgglOp *op) {
    if (!op->input_pad[0]) {
        op->output_pad[0] = NULL;
        return 0;
    }
    op->output_pad[0] = gggl_op_image_alloc (op, 0,
                                             op->out_pad_fmt[0],
                                             op->input_pad[0]->width,
                                             op->input_pad[0]->height);
    return process_lines (op, 0, op->output_pad[0]->height - 1);
}

static int
query_in_pad_fmt (GgglOp *op, int pad_no, int fmt) {
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

static void
open (GgglOp *op) {
    op->out_pad_fmt[0] = gggl_pixfmt ("gF");
    op->query_in_pad_fmt = query_in_pad_fmt;
    op->set_in_pad_fmt = gggl_op_set_in_pad_fmt_dont_touch_out_pad;
    op->process = process;
}

static void *op_info[] = {
    "CR", open,
    "description", "hente cr komponenten til et bilde",
    NULL
};

void
gggl_op_init (void) {
    gggl_op_db_register (op_info);
}
```

2.2.4 euklidsk_dist.c

```
#include <gggl/op/gggl_op.h>
#include <math.h>
#include <stdio.h>

static void
open (GgglOp *op);

static void *op_info[]=
{
```

```
    "euklidsk_dist", open,
    "description", "euklidsk dist",
    "copyright" , "Eirik Grythe <grythe.dk> 2005",
    "category"  , "mask",
    NULL
};


void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}

// Algoritme for å regne ut fargeavstand
static inline float euklidsk ( float a, float b)
{
    int dist = sqrt ( (b-a) * (b-a) );
    if (dist < 0.39)
     return 1;
    return dist;
}


static int
process (GgglOp *op)
{
    GgglImage *img = op->input_pad[0];
    GgglImage *aux = op->input_pad[1];
    int width;
    int height;

    op->output_pad[0] = img;

    if (!img){
            fprintf (stderr, "eek didn't get all image input_pads in %s\n", (const char*) op_info[0]);
        return 0;
    }

    if (!aux)
        return 0;

    width = img->width;
    height = img->height;

    if (width  != aux->width ||
        height != aux->height) {
            fprintf (stderr, "eek image sizes not matching for %s\n", (const char*) op_info[0]);
            return 0;
    }

    unsigned int map_pixstride = 1;

    if ( aux->fmt == gggl_pixfmt ("gAF") )
        map_pixstride = 2;
    else
        map_pixstride = 1;

    if ( img->fmt == gggl_pixfmt ("gF")) {
        int y;
        for (y = 0; y < height; y++) {
            int x;
            float *pix        =  (float *)gggl_image_offset (img,  0, y, 0);
            float *map        =  (float *)gggl_image_offset (aux,  0, y, 0);
            for (x=0;x<width;x++) {
                 float maskval=map[0];
                 pix[0]= euklidsk (pix[0], maskval);
                 pix += 1;
                 map += map_pixstride;
            }
        }
    } else if (img->fmt == gggl_pixfmt ("gaF")) {
        int y;
        for (y = 0; y < height; y++) {
            int x;
            float *pix        =  (float *)gggl_image_offset (img,  0, y, 0);
            float *map        =  (float *)gggl_image_offset (aux,  0, y, 0);
            for (x=0;x<width;x++) {
                 float maskval=map[0];
                 pix[0]= euklidsk (pix[0], maskval);
```

```
                pix += 2;
                map += map_pixstride;
            }
        }
    } else if (img->fmt == gggl_pixfmt ("rgbF")) {
        int y;
        for (y = 0; y < height; y++) {
            int x;
            float *pix        =  (float *)gggl_image_offset (img,  0, y, 0);
            float *map        =  (float *)gggl_image_offset (aux,  0, y, 0);
            for (x=0;x<width;x++) {
                float maskval=map[0];
                pix[0]= euklidsk (pix[0], maskval);
                pix[1]= euklidsk (pix[1], maskval);
                pix[2]= euklidsk (pix[2], maskval);
                pix += 3;
                map += map_pixstride;
            }
        }
    } else if (img->fmt == gggl_pixfmt ("rgbaF")) {
        int y;
        for (y = 0; y < height; y++) {
            int x;
            float *pix        =  (float *)gggl_image_offset (img,  0, y, 0);
            float *map        =  (float *)gggl_image_offset (aux,  0, y, 0);
            for (x=0;x<width;x++) {
                float maskval=map[0];
                pix[0]= euklidsk (pix[0], maskval);
                pix[1]= euklidsk (pix[1], maskval);
                pix[2]= euklidsk (pix[2], maskval);
                pix += 4;
                map += map_pixstride;
            }
        }
    }
   return 0;
}

static int
query_in_pad_fmt (struct GgglOp  *op,
                  int      pad_no,
                  int fmt)
{
            if (fmt == gggl_pixfmt ("rgbaF") ||
                fmt == gggl_pixfmt ("rgbF")  ||
                fmt == gggl_pixfmt ("gaF")   ||
                fmt == gggl_pixfmt ("gF")    ||
                fmt == gggl_pixfmt ("gAF")   ||
                fmt == gggl_pixfmt ("gF"))
                return 1;
    return 0;
}


static int
set_in_pad_fmt (GgglOp *op,
            int               no,
            int   fmt)
{
    int ret = gggl_op_default_set_in_pad_fmt (op, no, fmt);

    if (ret == PIXFMT_SUPPORTED)
        if (no == 0)
            op->out_pad_fmt[0] = fmt;
    return ret;
}

static void
open (GgglOp *op)
{
    op->input_pads          = 2;
    op->query_in_pad_fmt    = query_in_pad_fmt;
    op->set_in_pad_fmt      = set_in_pad_fmt;
    op->process             = process;
}
```

2.2.5 ops/terskle_mask.c

```
#include <gggl/op/gggl_op.h>
#include <stdio.h>
```

```
typedef struct {
    double terskel;
}Priv;

static int
process (GgglOp *op)
{
    Priv *p = op->priv;
    GgglImage    *src_img;
    GgglImage    *dst_img;
    int           width;
    int           height;
    int           x,y;

    src_img = op->input_pad[0];

    if(!src_img)
        return -1;

    /* henter størrelsen til kilde bildet */
    width  = src_img->width;
    height = src_img->height;

    /* allokerer plass i minnet til bildet */
    dst_img = gggl_op_image_alloc (op, 0, op->out_pad_fmt[0],
           width, height);
    op->output_pad[0] = dst_img;
    double terskel = p->terskel;

    /* behandling av farge */
    for (y = 0; y < height; y++) {
        float *src;
float *dst;

/* Peker til bildene */
        src = (float *) gggl_image_offset (src_img, 0, y, 0);
        dst = (float *) gggl_image_offset (dst_img, 0, y, 0);

/* leser inn fargene til pikselet i bilde som skal
 * analyseres */
        for (x = 0; x < width; x++) {
            float gray;

            gray = src[0];
    // Verdiene ved sammenslåing av farger blir svært lyse
    // Ved å sette alle piksler med en lysere verdi enn
    // en terkselverd helt hvite og resten sorte får
    // man en tydlig maske
    // video1: 0.925 hoyre:0.935 (minst konturstøy)
    // Etter areasegmentering må koden forandres til 0.96
    if(gray > terskel)
        dst[0] = 1;
    else
        dst[0] = 0;

    // Kun gråkanaler i hvert bilde
    src += 1;
            dst += 1;
        }
    }

    return 0;
}

// inputbilde skal være i gF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}

// output i gF (gråtoner)
static int
query_out_pad_fmt (GgglOp *op,
                   int     pad_no,
```

```
                int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}

static void
open (GgglOp *op)
{
    Priv     *p = gggl_op_calloc (op, 1, sizeof (Priv));

    p->terskel = 0.5;

    gggl_op_property_new (op, "terskel", GgglPropValue, &(p->terskel),                              "terskel value" );

    op->priv = p;
    op->query_in_pad_fmt  = query_in_pad_fmt;
    op->query_out_pad_fmt = query_out_pad_fmt;
    op->process           = process;
}

static void *op_info[]=
{
    "terskle_mask" , open,
    "description", "terskle mask",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL
};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

2.2.6 ops/object_segm.c

```
/* Foretar segmentering ved bruk av to rammer.
 * Denne algoritmen ble utviklet for å forbedre sgmenteringer
 * Selv om ikke noe av personen inngår i maskeringen blir det
 * spor etter konturen. Det ble derfor brukt to masker. En for å finne
 * hvor persoenen har beveget seg og en for å oppdatere området
 * han har beveget seg vekk fra
 * */
#include <gggl/op/gggl_op.h>
#include <stdio.h>

typedef struct {
    GgglImage *held_image;
    GgglImage *held_mask;
} Priv;


static int
process (GgglOp *op)
{
    Priv *p = op->priv;
    int y;
    int ystart, yend;
    int width;
    int channels;

    // sjekker om om det er noe i input bildene
    // før det i det hele tatt skrives til output
    if (!op->input_pad[0] || !op->input_pad[1]) {
        op->output_pad[0]=NULL;
        return -1;
    }
    // allokerer plass
    if (!p->held_image) {
     p->held_image = gggl_image_duplicate (op->input_pad[0]);
     p->held_mask  = gggl_image_duplicate (op->input_pad[1]);
gggl_op_reg_alloc (op, p->held_mask);
gggl_op_reg_alloc (op, p->held_image);
    }

    op->output_pad[0]=p->held_image;

    ystart = 0;
```

```
    yend = op->input_pad[0]->height-1;
    width = op->input_pad[0]->width-1;

    for (y = ystart; y <= yend; y++) {
        int x;
float *held_img; // resultat
float *held_msk; // previous mask
        float *new; // nytt bilde fra video
        float *mask; // maskert område

        held_img = (float *) gggl_image_offset (p->held_image, 0, y, 0);
        held_msk = (float *) gggl_image_offset (p->held_mask , 0, y, 0);
        new  = (float *) gggl_image_offset (op->input_pad[0], 0, y, 0);
        mask = (float *) gggl_image_offset (op->input_pad[1], 0, y, 0);

        for (x = width; x ; --x) {
    int c;
    // En frame med rigtig fargeinnhold skal legges
    // oppå den eksisterende framen. Begge bildene skal være i rgb
    channels = op->input_pad[0]->channels;
    for (c = 0; c < channels; c++) {
                // Det skal legges til så stor andel av det nye bildet som
// gråtonemask[0] tilsier. Der foreleseren er, oppdateres ikke
// masken. Men der skal innholdet som allerede ligger i held
// fortsette å være (tavla bak læreren)
//         held[c] = held[c] * mask[0] + new[c] * (1.0 - mask[0]);
if(held_msk[0] != 0)
    held_img[c] = held_img[c] * held_msk[0];
if(mask[0] == 0)
    held_img[c] = 0;
// held_img[c] = new[c] * (1.0 - mask[0]);


        held_msk = mask;
    }
    new      += channels;
    held_img += channels;
    held_msk += channels;
    mask     += channels;
        }
    }
    return 0;
}

// Bildene som lesses inn skal være i rgbF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

// Det skal leses inn to inputbilder som skal
// blandes til ett output bilde
static void
open (GgglOp *op)
{
    Priv *p = gggl_op_calloc (op, 1, sizeof (Priv));

    p->held_image        = NULL;
    p->held_mask         = NULL;
    op->priv             = p;
    op->input_pads       = 2;
    op->query_in_pad_fmt = query_in_pad_fmt;
    op->process          = process;
}


static void *op_info[]=
{
    "object_segm", open,
    "description", "age, "
                   "Algoritme som segmentere objeter i bevegelse",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL};
```

```
void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

2.3 Forbedring av masker

2.3.1 Fylling av hull

fillrotate.xml

```
<oxide>
    <op type='png_save'>
        <property name='resource'>fill.png</property>
    </op>

    <!-- Very timeconsuming to use fill_verticale
         Rotating instead
    <op type='fill_verticale' hole_limit='12'/>
    -->

    <!-- Rotate back -->
<!--    <op type='rot90'/>-->
    <!-- Rotate and fill verticale holes -->
<!--    <op type='fill_horisontale' hole_limit='16'/>
    <op type='rot90'/>
 -->   <!-- Fill horisontale holes -->
    <op type='fill_horisontale' hole_limit='16'/>

    <op type='png'>
        <property name='resource'>CbCrmask.png</property>
    </op>
</oxide>
```

ops/fill_horizontal.c

```
#include <gggl/op/gggl_op.h>
#include <stdio.h>

typedef struct {
    int hole_limit;
} InstanceData;


static int
process (GgglOp *op)
{
    InstanceData *id;
    id          = op->priv;

    GgglImage *src_img;
    GgglImage *dst_img;
    int        width;
    int        height;
    int        x,y;
    int        sum_x; // sum av piksler i hors retn.
    int        hole_size;

    /* hullet kan mask være 12 piksler i horisontal og vertikal
     * lengde for at det skal tettes */
    int        hole_limit;

    hole_limit = 16; // id->hole_limit;
    hole_size = 0;

    src_img = op->input_pad[0];

    /* henter størrelsen til kilde bildet */
    width  = src_img->width;
    height = src_img->height;

    /* allokerer plass i minnet til bildet */
    dst_img = gggl_op_image_alloc (op, 0,
   op->out_pad_fmt[0], width, height);
    op->output_pad[0] = dst_img;

    // Horisontal fylling
    for (y = 0; y <= height; y++) {
```

```
        float *src;
float *dst;

/* Peker til bildene */
        src = (float *) gggl_image_offset (src_img, 0, y, 0);
        dst = (float *) gggl_image_offset (dst_img, 0, y, 0);

sum_x = 0;

        for (x = 0; x <= width; x++) {
            /* 0 = svart. Dvs pikselet er allerede markert i
     * masken og skal ikke forandres */
    if(src[0] == 0) {
        dst[0] = 0;
if (sum_x < hole_limit)
    sum_x++;
/* Ved nok etterfølgende maskerte piksler
 * nullstilles hullet */
else
    hole_size = 0;

src += 1;
                dst += 1;
            }

    /* Pikselet er ikke maskert. Sjekker om det
     * er et lite hul som skal tettes */
    else {
        float *tmp;
/* Det må ha vært tilstrekkelig mye maskert område
 * før hullet hvis det skal tettes */
        if(hole_size == 0) {
    tmp = src;
    // Finner størrelsen på hullet
    while(tmp[0] != 0 ) {
        hole_size++;
tmp++;
    }
    int z;
            for (z = hole_size; z; z--) {
              // hullet er for stort til å tettes
        if(hole_size > hole_limit)
            dst[0] = src[0];

        // hullet er tilstrekkelig lite
else {
            /* Sjekker at det er tilstrekkelig stort maskert
     * område etter hullet. Hvis ikke dette gjøres
     * vil hull som etterfølges av en tynn linje
     * fylles igjen*/
    int mask_size;
            mask_size = 0;
    // Finner størrelsen på det maskerte området
    // etter hullet
    while(tmp[0] == 0 ) {
                mask_size++;
        tmp++;
    }
            if(hole_size >= hole_limit)
                // hullet fylles igjen
dst[0] = 1;
                    else
// hullet fylles ikke igjen
                dst[0] = 0;
                }
dst++;
src++;
    }
}
/* Det var ikke maskert stor nok område før hullet
 * Hullet skal ikke tettes */
else {
    dst[0] = src[0];
    /* Området som markerer maskert område må
     * krympe for hvert
     * nytt ikke maskert piksel som kommer */
    if(sum_x > 0)
                sum_x--;

    src++;
```

```
                    dst++;
}
    }
        }
    }
    return 0;
}

// inputbilde skal være i gF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}

// output i gF (gråtoner)
static int
query_out_pad_fmt (GgglOp *op,
                   int     pad_no,
                   int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}

static void
open (GgglOp *op)
{
    InstanceData *instance_data;
    instance_data = gggl_op_malloc (op, sizeof (InstanceData));

    instance_data->hole_limit   = 12;

    gggl_op_property_new (op,
            "hole_limit", GgglPropValue, &(instance_data->hole_limit),
            "Size of holes to fill");

    op->priv              = instance_data;

    op->query_in_pad_fmt  = query_in_pad_fmt;
    op->query_out_pad_fmt = query_out_pad_fmt;
    op->process           = process;
}

static void *op_info[]=
{
    "fill_horisontale" , open,
    "description", "fill hole in mask",
    "copyright"  , "Eirik Grythe, grythe.dk, 2005",
    NULL
};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

ops/fill_vertical.c

```
#include <gggl/op/gggl_op.h>
#include <stdio.h>

typedef struct {
    int hole_limit;
}Priv;


static float
get_value (GgglImage *img,
         int x,
         int y) {

    if (x<0 ||
        y<0 ||
        x>= img->width ||
```

```
        y>= img->height) {
        return 0.0;
    }

    return * ((float *) gggl_image_offset (img, x, y, 0));
}

static void
set_value (GgglImage *img,
         int x,
         int y,
         float value)
{
    if (x<0 ||
        y<0 ||
        x>= img->width ||
        y>= img->height) {
        return;
    }

    *((float *) gggl_image_offset (img, x, y, 0)) = value;
}


static int
process (GgglOp *op)
{
    Priv *p = op->priv;

    GgglImage *src_img;
    GgglImage *dst_img;
    int        width;
    int        height;
    int        x,y;
    int        sum_y; // sum av piksler i vertikal retn.
    int        hole_size;

    /* hullet kan mask være 12 piksler i horisontal og vertikal
     * lengde for at det skal tettes */
    int        hole_limit;

    hole_limit = 12; //p->hole_limit;
    fprintf(stderr, "Hole_limit %i",hole_limit);

    hole_size = 0;

    src_img = op->input_pad[0];

    /* henter størrelsen til kilde bildet */
    width  = src_img->width;
    height = src_img->height;

    /* allokerer plass i minnet til bildet */
    dst_img = gggl_op_image_alloc (op, 0,
   op->out_pad_fmt[0], width, height);
    op->output_pad[0] = dst_img;

    /* Vertikal fylling */
    for (x = 0; x <= width; x++) {
    sum_y = 0;
        for (y = 0; y <= height; y++) {
            if(get_value(src_img, x, y) == 0) {
        set_value(dst_img, x, y, 0);
if (sum_y < hole_limit)
    sum_y++;
// Ved nok etterfølgende maskerte piksler nullstilles hullet
else
    hole_size = 0;
            }

    /* Pikselet er ikke maskert. Sjekker om det
     * er et lite hul som skal tettes*/
    else {
int tmp;
/* Det må ha vært tilstrekkelig mye maskert område
 * for at hullet tettes */
        if(sum_y == hole_limit) {
    tmp = y;
    // Finner størrelsen på hullet
    while(get_value(src_img, x, tmp) != 0 ) {
```

```
        hole_size++;
tmp ++;
    }
    int z;
            for (z = hole_size; z; z--) {
        // hullet er tilstrekkelig lite
        if(hole_size <= hole_limit) {
            /* Sjekker at det er tilstrekkelig stort
     * maskert område etter hullet. Hvis ikke
     * dette gjøres vil hull som etterfølges
     * av en tynn linje fylles igjen*/
    int mask_size;
            mask_size = 0;
    // Finner størrelsen på det maskerte området
    // etter hullet
    while(get_value(src_img, x, tmp) == 0 ) {
                mask_size++;
        tmp++;
    }
            if(mask_size >= hole_limit)
        // hullet fylles igjen
                set_value(dst_img, x, y, 1);
                    else
                set_value(dst_img, x, y, 0);
                }
              // hullet er for stort til å tettes
else {
            set_value(dst_img, x, y, get_value(src_img, x, y));
}
    }
}
/* Det var ikke maskert stor nok område før hullet
 * Hullet skal ikke tettes */
else {
    set_value(dst_img, x, y, 1);
    /* Området som markerer maskert område
     * må krympe for hvert
     * nytt ikke maskert piksel som kommer */
    if(sum_y > 0)
                sum_y--;

}
    }
        }
    }
    return 0;
}

// inputbilde skal være i gF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}

// output i gF (gråtoner)
static int
query_out_pad_fmt (GgglOp *op,
                   int     pad_no,
                   int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}

static void
open (GgglOp *op)
{
    Priv     *p = gggl_op_calloc (op, 1, sizeof (Priv));


    p->hole_limit   = 12;

    gggl_op_property_new (op,
            "hole_limit", GgglPropValue, &(p->hole_limit),
```

```
            "Size of holes to fill");

    op->priv              = p;

    op->query_in_pad_fmt  = query_in_pad_fmt;
    op->query_out_pad_fmt = query_out_pad_fmt;
    op->process           = process;
}

static void *op_info[]=
{
    "fill_verticale" , open,
    "description", "fill verticale hole in mask",
    "copyright"  , "Eirik Grythe, grythe.dk, 2005",
    NULL
};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

2.3.2 Fylling ved bruk av tekstmaske

filltext.xml

```
<oxide>
    <op name='ff_save' type='ff_save'>
        <property name='resource'>textsegm.avi</property>
    </op>

    <op type='subtract_mask'>
        <op type='median'/>
        <op type='create_textmask'/>
       <op name='clone' type='clone' idref='4'/>
    </op>

    <op name='input' type='avi'>
        <property name='resource'>hoyre.avi</property>
        <property name='frame' interpolation='proportional'></property>
        <output id='4'/>
    </op>
</oxide>
```

2.3.3 Kontur

kontur.xml

```
<oxide>
    <op name='png_save' type='png_save'>
        <property name='resource'>kontur.png</property>
    </op>

    <op type='add_mask'>
        <op type='median'/>

<!-- Buttom, right edges -->
        <op type='edge'
    matrixB_0_0 = '-1'
     matrixB_0_1 = '-1'
         matrixB_0_2 = '-1'
         matrixB_1_0 = '0'
         matrixB_1_1 = '0'
         matrixB_1_2 = '0'
         matrixB_2_0 = '1'
         matrixB_2_1 = '1'
         matrixB_2_2 = '1'

      matrixA_0_0 = '-1'
         matrixA_0_1 = '0'
         matrixA_0_2 = '1'
         matrixA_1_0 = '-1'
         matrixA_1_1 = '0'
         matrixA_1_2 = '1'
         matrixA_2_0 = '-1'
         matrixA_2_1 = '0'
         matrixA_2_2 = '1'

          divisor = '1'
     offset  = '0.0'/>
```

```
        <!-- Upper, left edges -->
        <op type='edge'
     matrixB_0_0 = '1'
         matrixB_0_1 = '1'
         matrixB_0_2 = '1'
         matrixB_1_0 = '0'
         matrixB_1_1 = '0'
         matrixB_1_2 = '0'
         matrixB_2_0 = '-1'
         matrixB_2_1 = '-1'
         matrixB_2_2 = '-1'

     matrixA_0_0 = '1'
         matrixA_0_1 = '0'
         matrixA_0_2 = '-1'
         matrixA_1_0 = '1'
         matrixA_1_1 = '0'
         matrixA_1_2 = '-1'
         matrixA_2_0 = '1'
         matrixA_2_1 = '0'
         matrixA_2_2 = '-1'

          divisor = '1'
         offset  = '0.0'/>

        <op type='clone' idref='10'/>
    </op>

    <op type='png'>
        <property name='resource'>output.png</property>
        <output id='10'/>
    </op>
</oxide>
```

2.3.4 Fullstendig xml-fil

Denne xml-fila generere CbCr-maske, forbedre maske og segmentere objekter.

```
<oxide>
    <op name='output' type='ff_save'>
        <property name='resource'>object.avi</property>
<!--    <property name='bitrate'>100000000</property>-->
    </op>

    <!-- Updating video in masked area-->
    <op name='segment' type='object_segm'>
        <!-- Add medianedges and mask -->
        <op name='terskle' type='terskle_mask' terskel='0.5'/>
        <op name='improve mask' type='add_mask'>
            <op name='median' type='median'/>

    <!-- Buttom, right edges -->
            <op type='edge'
    matrixB_0_0 = '-1'
    matrixB_0_1 = '-1'
         matrixB_0_2 = '-1'
         matrixB_1_0 = '0'
         matrixB_1_1 = '0'
         matrixB_1_2 = '0'
         matrixB_2_0 = '1'
         matrixB_2_1 = '1'
         matrixB_2_2 = '1'

    matrixA_0_0 = '-1'
         matrixA_0_1 = '0'
         matrixA_0_2 = '1'
         matrixA_1_0 = '-1'
         matrixA_1_1 = '0'
         matrixA_1_2 = '1'
         matrixA_2_0 = '-1'
         matrixA_2_1 = '0'
         matrixA_2_2 = '1'

            divisor = '1'
    offset  = '0.0'/>


            <!-- Upper, left edges -->
```

```
          <op type='edge'
    matrixB_0_0 = '1'
         matrixB_0_1 = '1'
         matrixB_0_2 = '1'
         matrixB_1_0 = '0'
         matrixB_1_1 = '0'
         matrixB_1_2 = '0'
         matrixB_2_0 = '-1'
         matrixB_2_1 = '-1'
         matrixB_2_2 = '-1'

    matrixA_0_0 = '1'
         matrixA_0_1 = '0'
         matrixA_0_2 = '-1'
         matrixA_1_0 = '1'
         matrixA_1_1 = '0'
         matrixA_1_2 = '-1'
         matrixA_2_0 = '1'
         matrixA_2_1 = '0'
         matrixA_2_2 = '-1'

            divisor = '1'
         offset  = '0.0'/>

           <op name='clone' type='clone' idref='10'/>
        </op>


        <!-- Rotate back -->
        <op type='rot90'>
           <output id='10'/>
        </op>
        <!-- Rotate and fill verticale holes -->
        <op type='fill_horisontale' hole_limit='12'/>
        <op type='rot90'/>
        <!-- Fill horisontale holes -->
        <op type='fill_horisontale' hole_limit='12'/>



        <!-- maske-->
        <op name='terskle' type='terskle_mask' terskel='0.935'/>

        <op name='cbcr' type='add_mask'>
         <op name='cb' type='CB' />
         <op name='clone' type='clone' idref='4'/>
        </op>
<op type='CR'/>
<op type='clone' idref='4'/>
    </op>

    <op name='input' type='avi'>
        <property name='resource'>hoyre.avi</property>
        <property name='frame' interpolation='proportional'></property>
        <output id='4'/>
    </op>
</oxide>
```

2.4 Temporal medianfiltrering

2.4.1 snitt.xml

```
<oxide>
    <op type='ff_save'>
        <property name='resource'>snitt.avi</property>
    </op>
    <op type='snitt'>
        <property name='blurtime'>0.04</property>
   </op>
   <op type='avi'>
        <property name='resource'>small.avi</property>
        <property name='frame' interpolation='proportional'></property>
    </op>
</oxide>
```

2.4.2 ops/snitt.c

```
#include <gggl/op/gggl_op.h>

/* Private instance variables */
```

```
typedef struct {
    double blurtime;      /*< also exposed as a property */
} Priv;


/* Prosess for å sammenlikne etterfølgende rammer og legge fortløpende gjennomsnittsbilde i videofil */
static int
process (GgglOp *op)
{
    Priv   *p=op->priv;

    GgglImage *src_img;
    GgglImage *acc_img;

    src_img = op->input_pad[0];

    if (!src_img)
        return -1;

    acc_img = gggl_image_duplicate (src_img);

    /* register the allocation with the instance */
    gggl_op_reg_alloc (op, acc_img);

    /* use local variables to avoid dereferencing in innerloops */
    float  blurtime  = p->blurtime;
    int    channels = acc_img->channels;
    int    width    = acc_img->width;
    int    height   = acc_img->height;
    int    y;

    for (y = 0; y < height; y++) {
        int x;
        float *acc = (float *) gggl_image_offset (acc_img, 0, y, 0);
        float *src = (float *) gggl_image_offset (src_img, 0, y, 0);

        for (x = 0; x < width; x++) {
            int c;

for (c = 0; c < channels; c++)
            acc[c] = (acc[c] * (1.0-blurtime) + src[c]) * blurtime;

        acc += channels;
        src += channels;
        }
    }

    op->output_pad[0] = acc_img;

    return 0;
}


/* instantiate a processing op */
static void
open (GgglOp *op)
{
    /* allocate memory for private instance variables */
    Priv *p = gggl_op_calloc (op, 1, sizeof (Priv));

    p->blurtime = 0.5;

    gggl_op_property_new (op,
            "blurtime",
            GgglPropValue,
            &p->blurtime,
            "interval to calculate median movement");

    op->priv             = p;
    op->input_pads       = 1;
    op->output_pads      = 1;
    op->query_in_pad_fmt = gggl_fmt_accept_float;
    op->process          = process;
}

static void *op_info[]=
{
    "snitt",       open,
    "description", "median movemen",
    "category"   , "realtime",
```

```
    NULL
};


void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

2.5 Tekst

2.5.1 Kant deteksjon

sobel.xml

```
<oxide>
    <op type='png_save'>
        <property name='resource'>sobel.png</property>
    </op>

    <op type='terskle_mask' terskel='1.50'/>

    <!-- Lower, right edges -->
    <op type='edge'
matrixB_0_0 = '-1'
     matrixB_0_1 = '0'
     matrixB_0_2 = '1'
     matrixB_1_0 = '-2'
     matrixB_1_1 = '0'
     matrixB_1_2 = '2'
     matrixB_2_0 = '-1'
     matrixB_2_1 = '0'
     matrixB_2_2 = '1'

matrixA_0_0 = '-1'
     matrixA_0_1 = '-2'
     matrixA_0_2 = '-1'
     matrixA_1_0 = '0'
     matrixA_1_1 = '0'
     matrixA_1_2 = '0'
     matrixA_2_0 = '1'
     matrixA_2_1 = '2'
     matrixA_2_2 = '1'

      divisor = '1'
offset  = '0.0'/>

    <!-- Upper, left edges -->
    <op type='edge'
matrixB_0_0 = '1'
     matrixB_0_1 = '0'
     matrixB_0_2 = '-1'
     matrixB_1_0 = '2'
     matrixB_1_1 = '0'
     matrixB_1_2 = '-2'
     matrixB_2_0 = '1'
     matrixB_2_1 = '0'
     matrixB_2_2 = '-1'

matrixA_0_0 = '1'
     matrixA_0_1 = '2'
     matrixA_0_2 = '1'
     matrixA_1_0 = '0'
     matrixA_1_1 = '0'
     matrixA_1_2 = '0'
     matrixA_2_0 = '-1'
     matrixA_2_1 = '-2'
     matrixA_2_2 = '-1'

      divisor = '1'
offset  = '0.0'/>

    <op type='png'>
        <property name='resource'>input2.png</property>
    </op>
</oxide>
```

prewitt.xml

```
<oxide>
    <op type='png_save'>
        <property name='resource'>prewitt.png</property>
    </op>

    <!-- Ved å terksle bildet vil man få fjernet mye støy -->
    <op type='terskle_mask' terskel='0.80'/>

    <!-- Buttom, right edges -->
    <op type='edge'
matrixB_0_0 = '-1'
```

```
     matrixB_0_1 = '-1'
     matrixB_0_2 = '-1'
     matrixB_1_0 = '0'
     matrixB_1_1 = '0'
     matrixB_1_2 = '0'
     matrixB_2_0 = '1'
     matrixB_2_1 = '1'
     matrixB_2_2 = '1'

matrixA_0_0 = '-1'
     matrixA_0_1 = '0'
     matrixA_0_2 = '1'
     matrixA_1_0 = '-1'
     matrixA_1_1 = '0'
     matrixA_1_2 = '1'
     matrixA_2_0 = '-1'
     matrixA_2_1 = '0'
     matrixA_2_2 = '1'

      divisor = '1'
offset  = '0.0'/>


    <!-- Upper, left edges -->
    <op type='edge'
matrixB_0_0 = '1'
     matrixB_0_1 = '1'
     matrixB_0_2 = '1'
     matrixB_1_0 = '0'
     matrixB_1_1 = '0'
     matrixB_1_2 = '0'
     matrixB_2_0 = '-1'
     matrixB_2_1 = '-1'
     matrixB_2_2 = '-1'

matrixA_0_0 = '1'
     matrixA_0_1 = '0'
     matrixA_0_2 = '-1'
     matrixA_1_0 = '1'
     matrixA_1_1 = '0'
     matrixA_1_2 = '-1'
     matrixA_2_0 = '1'
     matrixA_2_1 = '0'
     matrixA_2_2 = '-1'

      divisor = '1'
offset  = '0.0'/>

    <op type='png'>
        <property name='resource'>input2.png</property>
    </op>
</oxide>
```

freichen.xml

```
<oxide>
    <op type='png_save'>
        <property name='resource'>freiChen.png</property>
    </op>

    <op type='terskle_mask' terskel='0.90'/>

    <!-- Buttom, right edges -->
    <op type='edge'
matrixB_0_0 = '-1'
     matrixB_0_1 = '-1.414213'
     matrixB_0_2 = '-1'
     matrixB_1_0 = '0'
     matrixB_1_1 = '0'
     matrixB_1_2 = '0'
     matrixB_2_0 = '1'
     matrixB_2_1 = '1.414213'
     matrixB_2_2 = '1'

matrixA_0_0 = '-1'
     matrixA_0_1 = '0'
     matrixA_0_2 = '1'
     matrixA_1_0 = '-1.414213'
     matrixA_1_1 = '0'
     matrixA_1_2 = '1.414213'
     matrixA_2_0 = '-1'
```

```
     matrixA_2_1 = '0'
     matrixA_2_2 = '1'

      divisor = '1'
offset  = '0.0'/>

    <!-- Top, right edges -->
    <op type='edge'
matrixB_0_0 = '1'
     matrixB_0_1 = '1.414213'
     matrixB_0_2 = '1'
     matrixB_1_0 = '0'
     matrixB_1_1 = '0'
     matrixB_1_2 = '0'
     matrixB_2_0 = '-1'
     matrixB_2_1 = '-1.414213'
     matrixB_2_2 = '-1'

matrixA_0_0 = '1'
     matrixA_0_1 = '0'
     matrixA_0_2 = '-1'
     matrixA_1_0 = '1.414213'
     matrixA_1_1 = '0'
     matrixA_1_2 = '-1.414213'
     matrixA_2_0 = '1'
     matrixA_2_1 = '0'
     matrixA_2_2 = '-1'

      divisor = '1'
offset  = '0.0'/>

    <op type='png'>
        <property name='resource'>input2.png</property>
    </op>
</oxide>
```

roberts.xml

```
<oxide>
    <op type='png_save'>
        <property name='resource'>robert.png</property>
    </op>

    <op type='terskle_mask' terskel='0.08'/>
    <!-- Upper, right edges -->
    <op type='edge'
matrixB_0_0 = '0'
     matrixB_0_1 = '0'
     matrixB_0_2 = '0'
     matrixB_1_0 = '0'
     matrixB_1_1 = '-1'
     matrixB_1_2 = '0'
     matrixB_2_0 = '0'
     matrixB_2_1 = '0'
     matrixB_2_2 = '1'

matrixA_0_0 = '0'
     matrixA_0_1 = '0'
     matrixA_0_2 = '0'
     matrixA_1_0 = '0'
     matrixA_1_1 = '0'
     matrixA_1_2 = '-1'
     matrixA_2_0 = '0'
     matrixA_2_1 = '1'
     matrixA_2_2 = '0'

      divisor = '1'
offset  = '0.0'/>

    <!-- Buttom,right edges -->
    <op type='edge'
matrixB_0_0 = '0'
     matrixB_0_1 = '0'
     matrixB_0_2 = '0'
     matrixB_1_0 = '0'
     matrixB_1_1 = '1'
     matrixB_1_2 = '0'
     matrixB_2_0 = '0'
     matrixB_2_1 = '0'
     matrixB_2_2 = '-1'
```

```
matrixA_0_0 = '0'
     matrixA_0_1 = '0'
     matrixA_0_2 = '0'
     matrixA_1_0 = '0'
     matrixA_1_1 = '0'
     matrixA_1_2 = '1'
     matrixA_2_0 = '0'
     matrixA_2_1 = '-1'
     matrixA_2_2 = '0'

      divisor = '1'
offset  = '0.0'/>

    <op type='png'>
        <property name='resource'>input2.png</property>
    </op>
</oxide>
```

edgedetect.c

```
#include <gggl/op/gggl_op.h>
#include <stdio.h>

typedef struct
{
    double    kernelA[3][3];
    double    kernelB[3][3];
    double    divisor;      // sum av vekter i matrise
    double    offset;     // "standard" verdi
} Priv;


static int
process (GgglOp *op)
{
    Priv      *p = op->priv;
    int        x, y;
    int        width;
    int        height;
    int        channels;
    GgglImage *src_img;
    GgglImage *dst_img;


    src_img = op->input_pad[0];
    width = src_img->width;
    height = src_img->height;
    channels = src_img->channels;

    if (!src_img)
    {
        op->output_pad[0] = NULL;
        return 0;
    }

    dst_img = gggl_op_image_alloc (op, 0, op->out_pad_fmt[0], width, height);
    op->output_pad[0] = dst_img;

    for (y = 1; y < height - 1; y++) {
      for (x = 1; x < width - 1; x++) {
        float *src00 = (float *) gggl_image_offset (src_img, x-1, y-1, 0);
        float *src10 = (float *) gggl_image_offset (src_img, x+0, y-1, 0);
        float *src20 = (float *) gggl_image_offset (src_img, x+1, y-1, 0);

        float *src01 = (float *) gggl_image_offset (src_img, x-1, y+0, 0);
        float *src11 = (float *) gggl_image_offset (src_img, x+0, y+0, 0);
        float *src21 = (float *) gggl_image_offset (src_img, x+1, y+0, 0);

        float *src02 = (float *) gggl_image_offset (src_img, x-1, y+1, 0);
        float *src12 = (float *) gggl_image_offset (src_img, x+0, y+1, 0);
        float *src22 = (float *) gggl_image_offset (src_img, x+1, y+1, 0);

        float *dst   = (float *) gggl_image_offset (dst_img, x, y, 0);

        double result, result2;

/* Horisontale kanter */
        result = p->kernelA[0][0] * *src00 +
                 p->kernelA[0][1] * *src01 +
                 p->kernelA[0][2] * *src02 +
```

```
                p->kernelA[1][0] * *src10 +
                p->kernelA[1][1] * *src11 +
                p->kernelA[1][2] * *src12 +
                p->kernelA[2][0] * *src20 +
                p->kernelA[2][1] * *src21 +
                p->kernelA[2][2] * *src22;

        result2 = p->kernelB[0][0] * *src00 +
                  p->kernelB[0][1] * *src01 +
                  p->kernelB[0][2] * *src02 +
                  p->kernelB[1][0] * *src10 +
                  p->kernelB[1][1] * *src11 +
                  p->kernelB[1][2] * *src12 +
                  p->kernelB[2][0] * *src20 +
                  p->kernelB[2][1] * *src21 +
                  p->kernelB[2][2] * *src22;

result += result2;

  result /= p->divisor;
        result += p->offset;

        *dst = result;

        src00 += channels;
        src01 += channels;
        src02 += channels;
        src10 += channels;
        src11 += channels;
        src12 += channels;
        src20 += channels;
        src21 += channels;
        src22 += channels;
        dst++;
        }
    }

  return 0;
}

// Input bilder kan være i rgb eller gråtoner
static int
query_in_pad_fmt (GgglOp *op, int pad_no, int fmt)
{
  if (fmt == gggl_pixfmt ("rgbF") ||
      fmt == gggl_pixfmt ("rgbAF") ||
      fmt == gggl_pixfmt ("gF") || fmt == gggl_pixfmt ("gAF"))
    return 1;
  return 0;
}

// output i gråtoner
static int
query_out_pad_fmt (GgglOp *op, int pad_no, int fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}


static void
open (GgglOp *op)
{
  Priv     *p = gggl_op_calloc (op, 1, sizeof (Priv));

  /* kernel A */
  gggl_op_property_new (op,
                        "matrixA_0_0", GgglPropValue, &p->kernelA[0][0],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixA_0_1", GgglPropValue, &p->kernelA[1][0],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixA_0_2", GgglPropValue, &p->kernelA[2][0],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixA_1_0", GgglPropValue, &p->kernelA[0][1],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
```

```
                        "matrixA_1_1", GgglPropValue, &p->kernelA[1][1],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixA_1_2", GgglPropValue, &p->kernelA[2][1],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixA_2_0", GgglPropValue, &p->kernelA[0][2],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixA_2_1", GgglPropValue, &p->kernelA[1][2],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixA_2_2", GgglPropValue, &p->kernelA[2][2],
                        "a component in the convolve matrix");

  /* kernel B*/
  gggl_op_property_new (op,
                        "matrixB_0_0", GgglPropValue, &p->kernelB[0][0],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixB_0_1", GgglPropValue, &p->kernelB[1][0],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixB_0_2", GgglPropValue, &p->kernelB[2][0],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixB_1_0", GgglPropValue, &p->kernelB[0][1],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixB_1_1", GgglPropValue, &p->kernelB[1][1],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixB_1_2", GgglPropValue, &p->kernelB[2][1],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixB_2_0", GgglPropValue, &p->kernelB[0][2],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixB_2_1", GgglPropValue, &p->kernelB[1][2],
                        "a component in the convolve matrix");
  gggl_op_property_new (op,
                        "matrixB_2_2", GgglPropValue, &p->kernelB[2][2],
                        "a component in the convolve matrix");

  /* Divisor */
  gggl_op_property_new (op,
                        "divisor", GgglPropValue, &p->divisor,
                        "divisor");
  /* Offset */
  gggl_op_property_new (op,
                        "offset", GgglPropValue, &p->offset,
                        "offset");

  op->priv = p;
  op->query_in_pad_fmt  = query_in_pad_fmt;
  op->query_out_pad_fmt = query_out_pad_fmt;
  op->process = process;
}


static void *op_info[] = {
  "edge" , open,
  "description", "edge detect algorithm",
  "category"   , "edge",
  NULL
};


void
gggl_op_init (void)
{
  gggl_op_db_register (op_info);
}
```

2.5.2 Forbedring av leselighet

create_RGBtext.c

```
#include <gggl/op/gggl_op.h>
#include <math.h>   /* sqrt */
#include <stdio.h>  /* io debugging */
```

```
typedef struct {
    double red;
    double green;
    double blue;
} InstanceData;


static int  process           (GgglOp *op);
static int  query_in_pad_fmt  (GgglOp *op, int pad_no, int fmt);
static int  query_out_pad_fmt (GgglOp *op, int pad_no, int fmt);
static void open              (GgglOp *op);

static void *op_info[]=
{
    "create_textmask" , open,
    "description", "create mask",
    "category"   , "tutorial",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL
};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}

static void
open (GgglOp *op)
{
    InstanceData *instance_data;

    instance_data = gggl_op_malloc (op, sizeof (InstanceData));

    /* legger inn krittfarge */
    instance_data->red   = 0.48627; /* 0.46275; */
    instance_data->green = 0.58431; /* 0.59216; */
    instance_data->blue  = 0.52941; /* 0.52549; */


    /* register our level property */
    gggl_op_property_new (op, "red",  GgglPropValue,
            &(instance_data->red),  "target red"   );
    gggl_op_property_new (op, "green",  GgglPropValue,
             &(instance_data->green),  "target green"   );
    gggl_op_property_new (op, "blue",  GgglPropValue,
     &(instance_data->blue),  "target blue"   );

    op->priv = instance_data;

    op->query_in_pad_fmt  = query_in_pad_fmt;
    op->query_out_pad_fmt = query_out_pad_fmt;
    op->process           = process;
}

// inputbilde skal være i rgbF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

// output i gF (gråtoner)
static int
query_out_pad_fmt (GgglOp *op,
                   int     pad_no,
                   int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}

// Algoritme for å regne ut fargeavstand
static float rgb_distance ( float red0, float green0, float blue0,
```

```
                           float red1, float green1, float blue1)
{
    return sqrt ( (red1-red0)     * (red1-red0) +
                  (green1-green0) * (green1-green0) +
                  (blue1-blue0)   * (blue1-blue0));
}

static int
process (GgglOp *op)
{
    InstanceData *id;
    GgglImage    *src_img;
    GgglImage    *dst_img;
    int           width;
    int           height;
    int           x,y;

    float         r1,g1,b1;

    src_img = op->input_pad[0];

    /* leser inn fargekomponentene til første valgte farge */
    id = op->priv;

    r1 = id->red;
    g1 = id->green;
    b1 = id->blue;

    /* henter størrelsen til kildebildet */
    width  = src_img->width;
    height = src_img->height;

    /* allokerer plass i minnet til bildet */
    dst_img = gggl_op_image_alloc (
   op, 0, op->out_pad_fmt[0], width, height);
    op->output_pad[0] = dst_img;

    // generering av masker
    for (y = 0; y < height; y++) {
        float *src;
float *dst;

// Peker til bildene
        src = (float *) gggl_image_offset (src_img, 0, y, 0);
        dst = (float *) gggl_image_offset (dst_img, 0, y, 0);

// leser inn fargene til pikselet i bilde som skal analyseres
        for (x = 0; x < width; x++) {
            float r0,g0,b0;
            float color_distance;

            r0 = src[0];
            g0 = src[1];
            b0 = src[2];

    // Kritt
    color_distance = rgb_distance (r0,g0,b0,
                                           r1,g1,b1);

    if(color_distance < 0.11)
        dst[0] = 0;

    else
dst[0] = 1;

    // Fargebildet økes med 3 kanaler, gråtone med 1
            src += 3;
            dst += 1;
        }
    }
    return 0;
}
```

text_impr.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <oxide/oxide.h>
#include <gggl/gggl.h>   /* for loading additional op dir */
```

```
char    *movie_name = "improved.avi";
double   start      = 130.0;      // under testing
double   end        = 200.0;      // under testing
double   step       = 1.0;
double   threshold  = 0.0;
char    *movie_file = NULL;
char    *op_dir     = "ops";
int      verbose    = 0;
int      write_thumbnails = 0;
 /* default krittfarge */
double   red        = 0.47843;
double   green      = 0.59216;
double   blue       = 0.53333;

/* utility function to parse arguments */
static void
parse_args (int    argc,
            char **argv);

/* utility function to get information about a movie */
static void
movie_get_info (const char *file,
                int        *duration,
                int        *width,
                int        *height);

static Oxide *
find_op_named (Oxide *oxide,
               const char *op_name);


/* utility operation to extract a single frame from a movie file */
static void
write_frame (const char *movie,
             int frame,
             const char *png_file);

/* Opretter xml-tree for lagring av ny film */
static Oxide *saver_open  (const char *movie,
                   const char *movie_out);

/* Funksjon som lagerer hver enkelt ramme */
static void saver_store (Oxide *root,
         int    frame);

/* Lukker xml-treet */
static void saver_close (Oxide *root);


int
main (int    argc,
      char **argv)
{
    Oxide  *ringbutt_img;
    Oxide  *read = NULL;
    Oxide  *text_det;
    Oxide  *saver;
    double  time;
    double  text_mask;
    double  width, height;
    float   dst_img;

//    int buffersize = 50;
//    int ringbuffer[width][height][buffersize];

    parse_args (argc, argv);

    oxide_init();

    if (op_dir) {
        if (verbose)
            fprintf (stderr, "loading gggl ops from '%s'\n", op_dir);
        gggl_op_db_load_plugins (op_dir);
    }

    read = oxide_parse_xml_chunk (NULL,
        "<oxide>"
"  <op name='saver' type='ff_save'/>"
"  "
        "  <op name='readability' type='readability'/>"
```

```
"  <op type='autostretch'/>"
        "  <op name='movie' type='avi'>"
        "    <property name='resource'>input.avi</property>"
        "    <property name='frame' interpolation='linear'>"
        "      <value time='0'     >0</value>"
        "      <value time='100000'>100000</value>"
        "    </property>"
        "  </op>"
        "</oxide>");


    oxide_set_property (find_op_named (read, "movie"),
                        "resource", movie_file);

    oxide_set_property (find_op_named (read, "saver"),
                        "resource", movie_name);

    text_det = find_op_named (read, "readability");

    if (verbose)
        fprintf (stderr, "processing\n");

    if (end==0) {
        int duration;
        movie_get_info (movie_file, &duration, NULL, NULL);
        end = duration;
    }

    time = start;

    oxide_set_property_double (find_op_named (read, "movie"),
                        "_time", time);

    oxide_process (read);

    width   = oxide_get_property_double (text_det, "width");
    height  = oxide_get_property_double (text_det, "height");
//      dst_img = oxide_get_property_double (text_det, "acc_img");
//    fprintf (stdout, "width %4.0f\n", height);

    oxide_set_property_double (read, "time" , time);
    oxide_set_property_double (read, "red"  , red);
    oxide_set_property_double (read, "green", green);
    oxide_set_property_double (read, "blue" , blue);

    /* Oppretter videofila det skal skrives til */
    saver = saver_open(movie_file, movie_name);

    do{
        oxide_set_property_double (find_op_named (read, "movie"),
                        "_time", time);

        oxide_process (read);

 //       oxide_set_property_double (text_det, "time", time);
/* Detektert tekst */
// text_mask = oxide_get_property_double (text_det, "acc_img");

if (verbose)
            fprintf (stdout, "Movie: %4.0f. \t",
oxide_get_property_double (
    find_op_named (read, "movie"), "_time"));

//RINGBUFFER

        // Dersom thumbnail er satt skal videoen genereres
/* if (write_thumbnails) {
            char     buf[50];
            sprintf (buf, "thumb-%05d.png", (int)time);
            fprintf (stdout, "Ramme %4.0f skrives til png bilde\n", time);
            write_frame (movie_file, time, buf);
}
*/
// saver_store (saver, time);

        time += step;
    } while (time <= end);

    saver_close(saver);
    oxide_free (read);
```

```
    return 0;
}

/* Skrives til skjerm dersom programmet kjøres uten parametere */
void
usage (char *application_name)
{
    fprintf (stderr,
            "usage: %s [options] <movie file>\n"
            "\n"
            "  Options:\n"
            "     --name               name of the new movie\n"
    "     --time-start value   time of first frame to process (default 0.0)\n"
            "     --time-end value     time of last frame to process (default 1.0)\n"
            "     --time-step value    amount of time between each frame (default 1.0)\n"
            "     --gggl-ops-dir dir   additional directory to look for gggl ops\n"
            "     --thumbnails         write png thumbnails of keyframes\n"
            "     --threshold value    threshold considered scene change\n"
            "     --red value          red color value (0-1)\n"
            "     --green value        green color value (0-1)\n"
            "     --blue value         blue color value (0-1)\n"
            "     -v                   print diagnostic output\n"
            "\n"
            "all parameters following -- are considered ops to be chained together\n"
            "into a small compoistion instead of using an xml file, this allows for\n"
            "easy testing of filters, be aware that the default value for all\n"
            "properties will be used\n"
            , application_name);
    exit (-1);
}


/* helper macros to make the commandline easier to parse */

#define match(string) (!strcmp (*curr_arg, (string)))
#define argument() \
do{\
  if (!curr_arg[1] || curr_arg[1][0]=='-'){\
    fprintf (stderr, "error parsing commandline %s needs argument\n", *curr_arg);\
    exit (-1);\
  }\
  curr_arg++;\
}while(0)

void
parse_args (int    argc,
            char **argv)
{
    char **curr_arg;

    if (argc==1) {
        usage (argv[0]);
    }

    curr_arg = argv + 1;

    while (*curr_arg) {
        if (match ("-h") ||
            match ("--help")) {
            usage (argv[0]);
        } else if (match ("--end") ||
                   match ("--time-end") ||
                   match ("-out") ||
                   match ("-e")) {
            argument ();
            end = atof (*curr_arg);
        } else if (match ("--threshold")) {
            argument ();
            threshold = atof (*curr_arg);
        } else if (match ("--op_dir" )||
                   match ("--gggl-ops-dir")) {
            argument ();
            op_dir = strdup (*curr_arg);
        } else if (match ("--name" )) {
            argument ();
            movie_name = strdup (*curr_arg);
        } else if (match ("--start") ||
                   match ("--time-start") ||
                   match ("-in") ||
```

```
                match ("-s")) {
            argument ();
            start = atof (*curr_arg);
        } else if (match ("-v")) {
            verbose = 1;
        } else if (match ("--thumbnails")) {
            write_thumbnails = 1;
        } else if (match ("--red" )) {
            argument ();
            movie_name = strdup (*curr_arg);
        } else if (match ("--green" )) {
            argument ();
            movie_name = strdup (*curr_arg);
        } else if (match ("--blue" )) {
            argument ();
            movie_name = strdup (*curr_arg);
        } else {
            if (movie_file) {
                fprintf (stderr, "Error parsing commandline: don't know how to handle multiple xml files\n");
                exit (-1);
            }
            movie_file = strdup (*curr_arg);
        }

        curr_arg++;
    }

    if (verbose)
        fprintf (stderr,
            "Commandline values:\n"
            "\tend        = %f\n"
            "\top_dir     = '%s'\n"
            "\tstart      = %f\n"
            "\tverbose    = %i\n"
            "\tthreshold  = %f\n"
            "\tthumnails  = %s\n"
            "\tred        = %f\n"
            "\tgreen      = %f\n"
            "\tblue       = %f\n"
            "\tmovie_file = '%s'\n"
            , end, op_dir, start
            , verbose, threshold, write_thumbnails?"yes":"no", red, green, blue, movie_file);

    if (!movie_file) {
        fprintf (stderr, "Error: no movie file specified\n");
        exit (-1);
    }
}


static void
movie_get_info (const char *file,
                int        *duration,
                int        *width,
                int        *height)
{
    Oxide *root, *null, *load;
    double ret;

    root = oxide_new (NULL, OxideRoot);
    null = oxide_new (root, OxideOp);
    load = oxide_new (root, OxideOp);

    oxide_set_property (null, "type", "null");
    oxide_set_property (load, "type", "ff");
    oxide_append_child (root, null);
    oxide_append_child (root, load);
    oxide_set_property (load, "resource", file);
    oxide_process (root);

    if (duration)
        *duration = oxide_get_property_double (load, "frames");
    if (width)
        *width    = oxide_get_property_double (load, "width");
    if (height)
        *height   = oxide_get_property_double (load, "height");

    oxide_free (root);
}
```

```
/* Når forskjellen mellom to frames overgår en satt terskelverdi,
 * skal det genereres bilder dersom thumbnails er satt
 * Denne funksjonen brukes til å dokumentere differanse
 * mellom hvert bilde som skrives til video bildet sendes
 * til ny videofil
 */
static void
write_frame (const char *movie,
             int frame,
             const char *png_file)
{
    Oxide *root;
    Oxide *loader;
    Oxide *saver;

    root = oxide_parse_xml_chunk (NULL,
          "<oxide>"
          "<op name='saver'  type='png_save' />"
          "<op name='loader' type='ff' />"
          "</oxide>");

    loader = find_op_named (root, "loader");
    saver  = find_op_named (root, "saver");

    oxide_set_property (loader,  "resource", movie);
    oxide_set_property (saver,   "resource", png_file);
    oxide_set_property_double (loader, "frame", frame);

    oxide_process (root);
    oxide_free (root);
}

/* Genererer xml for å lagre rammer i video */
static Oxide *saver_open  (const char *movie,
                   const char *movie_out)
{
    Oxide *root;
    Oxide *loader;
    Oxide *saver;

    root = oxide_parse_xml_chunk (NULL,
  "<oxide>"
  "<op name='saver'  type='ff_save'/>"
  "<op name='loader' type='avi' />"
  "</oxide>");

    loader = find_op_named (root, "loader");
    saver  = find_op_named (root, "saver");
    oxide_set_property (saver,  "resource", movie_out);
    oxide_set_property (loader, "resource", movie);

    return root;
}

/* Legger til nye rammer i videoen */
static void saver_store (Oxide *root, int frame)
{
    Oxide *loader;

    loader = find_op_named (root, "loader");
    oxide_set_property_double (loader, "frame", frame);
    oxide_process (root);
}

/* Frigjør treet fra minnet */
static void saver_close (Oxide *root)
{
    oxide_free (root);
}

/* helper funciton to locate an op based on it's name */
static Oxide *
find_op_named (Oxide *oxide,
               const char *op_name)
{
    Oxide *iter = oxide;

    while (iter) {
        char *name = oxide_get_property (iter, "name");
        if (name) {
```

```
            if (!strcmp (name, op_name)) {
                free (name);
                return iter;
            }
            free (name);
        }
        iter = oxide_recurse (iter);
    }
    return NULL;
}
```

ops/readability.c

```
// 150 til 200 rammer trengs for å oppdatere skrift

// Skriften som skrives på tavlen skiller seg lite fra grønnfargen
// Ved å benytte autostretch forbedres dette en tankte.
// Hvis man over et tidsinterval legger sammen alle maskene som finner
// krittfargen blir skriften straks mye mer leslig
#include <gggl/op/gggl_op.h>
#include <math.h>   /* sqrt */
#include <stdio.h>  /* io debugging */

typedef struct
{
    double red, green, blue;
    double width, height;
    double time;
    GgglImage *acc_img;
} InstanceData;

// Algoritme for å regne ut fargeavstand
static float rgb_distance ( float red0, float green0, float blue0,
                            float red1, float green1, float blue1)
{
    return sqrt ( (red1-red0)     * (red1-red0) +
                  (green1-green0) * (green1-green0) +
                  (blue1-blue0)   * (blue1-blue0));
}

static int
process (GgglOp *op)
{
    InstanceData *id;
    GgglImage *src_img;
    GgglImage *dst_img;
//    GgglImage *acc_img;

    int       channels;
    double    width;
    double    height;
    int       x,y;
    float     r0,g0,b0;
    double    time;
    float    *tot;
    src_img  = op->input_pad[0];
    /* henter data om kildebildet */
    channels = src_img->channels;
    width    = src_img->width;
    height   = src_img->height;

    if (!src_img)
        return -1;

    id   = op->priv;
    time = id->time;
    /* Legger verdiene i en struct */
    id->width  = width;
    id->height = height;
//    fprintf(stderr, "Tid %4.0f\n",time);

    /* leser inn fargekomponentene til første valgte farge */
/*    r0 = id->red;
    g0 = id->green;
    b0 = id->blue;
*/
    r0 = 0.47843;
    g0 = 0.59216;
    b0 = 0.53333;

    /* allokerer plass i minnet til bildet */
```

```
    dst_img = gggl_op_image_alloc (op, 0, op->out_pad_fmt[0], width, height);
    op->output_pad[0] = dst_img;

    // totalmasken skal kun lagres i minnet, og den
    // skal være av samme størrelse om src_img og dst_img
//    id->acc_img = gggl_image_duplicate (src_img);

    /* generering av masker */
    for (y = 0; y < height; y++) {
float *src;
float *dst;

// Peker til bildene
        src   = (float *) gggl_image_offset (src_img, 0, y, 0);
        dst   = (float *) gggl_image_offset (dst_img, 0, y, 0);
        tot   = (float *) gggl_image_offset (src_img, 0, y, 0);

        for (x = 0; x < width; x++) {
//     fprintf(stderr, "Time: %4.0f\n",time);
    float r1,g1,b1;
            float color_distance;

            r1 = src[0];
            g1 = src[1];
            b1 = src[2];

    color_distance = rgb_distance (r0,g0,b0,
                                           r1,g1,b1);

    /* krittfarge detektert */
    if(color_distance < 0.11) {
               // øker verdi i totalmaske
       tot[0]  += 1;
          fprintf(stderr, "Tot: %4.0f\n",tot[0]);

       /* For at å fjerne totalstøyen fra summeringen
* av maskene, tas det ikke hensyn til
* pikselrepresentasjoner som forekommer
* sjeldent*/
//               if(tot[0] > time*0.7 ) {
          dst[0] = 0;
//            fprintf(stderr, "Rammenr %4.0f\n",time);
//            fprintf(stderr, "Total %4.0f\n",tot[0]);
              }

       else
   dst[0] = 1;
//     }
    // Fargebildet økes med 3 kanaler, gråtone med 1
            src += channels;
            dst ++;
    tot ++;
        }
    }

//    op->output_pad[0] = dst_img;
//    id->dst =(float*) tmp;
//    fprintf(stderr,"loading %f",(float*) id->dst);
    return 0;
}

// inputbilde skal være i rgbF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

// output i gF (gråtoner)
static int
query_out_pad_fmt (GgglOp *op,
                   int     pad_no,
                   int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
```

```
    return 0;
}

static void
open (GgglOp *op)
{
    InstanceData *instance_data;

    instance_data = gggl_op_malloc (op, sizeof (InstanceData));

  // Etter,  før,  kontraststekking
//    instance_data-> red   = 0.48627;  /* 0.43529; */
//    instance_data-> green = 0.58431;  /* 0.63529; */
//    instance_data-> blue  = 0.52941;  /* 0.53725; */

    instance_data-> width  = 0;
    instance_data-> height = 0;


    gggl_op_property_new (op, "red",  GgglPropValue,
             &(instance_data->red), "Red value");
    gggl_op_property_new (op, "green",  GgglPropValue,
             &(instance_data->green), "Green value");
    gggl_op_property_new (op, "blue",  GgglPropValue,
             &(instance_data->blue), "Blue value");
    gggl_op_property_new (op, "width",  GgglPropValue,
     &(instance_data->width), "width");
    gggl_op_property_new (op, "height",  GgglPropValue,
             &(instance_data->height), "height");
    gggl_op_property_new (op, "time",  GgglPropValue,
             &(instance_data->time), "time"   );

    op->priv              = instance_data;
    op->query_in_pad_fmt  = query_in_pad_fmt;
    op->query_out_pad_fmt = query_out_pad_fmt;
    op->process           = process;
}


static void *op_info[]= {
    "readability"  , open,
    "description", "Optimize textconture",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL
};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

2.5.3 Deteksjon av fjerning av skrift

text_save.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <oxide/oxide.h>
#include <gggl/gggl.h>   /* for loading additional op dir */

char    *movie_name ="motion_segm.avi";
double   start      = 0.0;
double   end        = 250.0; // under testing
double   step       = 1.0;
/* Når det sammenlignes på to etterfølgende rammer
 * bør diff være rundt 0.02
 * Når det ramme 1 sammenlignes med rammer fram
 * til overgått diff, må verdi være ca 3 %
 * Ca hver 5. ramme blir skrevet til ny film */
double   threshold  = 0.03;
char    *movie_file = NULL;
char    *op_dir     = "ops";
int      write_thumbnails = 0;
int      verbose    = 1; // under testing

/* utility function to parse arguments */
static void
parse_args (int    argc,
            char **argv);
```

```
/* utility function to get information about a movie */
static void
movie_get_info (const char *file,
                int        *duration,
                int        *width,
                int        *height);

static Oxide *
find_op_named (Oxide *oxide,
               const char *op_name);


/* utility operation to extract a single frame from a movie file */
static void
write_frame (const char *movie,
             int frame,
             const char *png_file);

/* Opretter xml-tree for lagring av ny film */
static Oxide *saver_open  (const char *movie,
                   const char *movie_out);

/* Funksjon som lagerer hver enkelt ramme */
static void saver_store (Oxide *root,
         int    frame);

/* Lukker xml-treet */
static void saver_close (Oxide *root);


int
main (int    argc,
      char **argv)
{
    Oxide   *oxide = NULL;
    Oxide   *reduction;
    Oxide   *saver;
    double   timeA, timeB;
    double   difference;
    double   new_text_frame = 0;
    double   interval = 75; // 75 rammer tilsvarer 3 sekuder fram i tid

    parse_args (argc, argv);

    oxide_init();

    if (op_dir) {
        if (verbose)
            fprintf (stderr, "loading gggl ops from '%s'\n", op_dir);
        gggl_op_db_load_plugins (op_dir);
    }

    oxide = oxide_parse_xml_chunk (NULL,
        "<oxide>"
        "  <op type='display'/>"
        "  <op type='clone' idref='1'/>"
        "  <op name='textremove' type='textremove'>"
        "    <op name='movieA' type='avi'>"
        "      <property name='resource'>input.avi</property>"
        "      <property name='frame' interpolation='linear'>"
        "        <value time='0'     >0</value>"
        "        <value time='100000'>100000</value>"
        "      </property>"
        "    </op>"
        "  </op>"
        "  <op name='movieB' type='avi'>"
        "    <output id='1'/>"
        "    <property name='resource'>input.avi</property>"
        "    <property name='frame' interpolation='linear'>"
        "      <value time='0'     >1</value>"
        "      <value time='100000'>100001</value>"
        "    </property>"
        "  </op>"
        "</oxide>");


    oxide_set_property (find_op_named (oxide, "movieA"),
                        "resource", movie_file);
```

```
    oxide_set_property (find_op_named (oxide, "movieB"),
                        "resource", movie_file);

    reduction = find_op_named (oxide, "textremove");

    if (verbose)
        fprintf (stderr, "processing\n");

    if (end==0) {
        int duration;
        movie_get_info (movie_file, &duration, NULL, NULL);
        end = duration;
    }

    timeA = start;

    do{
        /* Peker A skal settes til første ramme */
        oxide_set_property_double (find_op_named (oxide, "movieA"),
                        "_time", timeA);

timeB = timeA + interval;
        oxide_set_property_double (find_op_named (oxide, "movieB"),
                        "_time", timeB);

if (verbose) {
           fprintf (stdout, "MovieA: %4.0f. \t",
oxide_get_property_double (
      find_op_named(oxide, "movieA"), "_time"));
           fprintf (stdout, "MovieB: %4.0f. \t",
oxide_get_property_double (
             find_op_named(oxide, "movieB"), "_time"));
        }

oxide_process (oxide);

difference = oxide_get_property_double (reduction, "reduction");


        if (difference > threshold) {
fprintf (stdout, "Differanse: %f.", difference);
    new_text_frame = timeA;

           /* Dersom thumbnail er satt skal det genereres
     * png bilder av diff_frames */
           if (write_thumbnails) {
               char buf[50];
               sprintf (buf, "text-%05d.png", (int)new_text_frame);
               fprintf (stdout, "Ramme %4.0f skrives til png bilde\n",
 new_text_frame);
write_frame (movie_file, new_text_frame, buf);
    }

//          if (verbose)

    timeA = timeB;
}
fprintf (stdout, "\n");
        timeA += step;
    } while (timeA <= end);

    oxide_free (oxide);

    return 0;
}

/* Skrives til skjerm dersom programmet kjøres uten parametere */
void
usage (char *application_name)
{
    fprintf (stderr,
           "usage: %s [options] <movie file>\n"
           "\n"
           "  Options:\n"
           "     --name               name of output movie\n"
    "     --time-start value   time of first frame to process (default 0.0)\n"
           "     --time-end value     time of last frame to process (default 1.0)\n"
           "     --time-step value    amount of time between each frame (default 1.0)\n"
           "     --gggl-ops-dir dir   additional directory to look for gggl ops\n"
           "     --thumbnails         write png thumbnails of keyframes\n"
```

```
            "     --threshold value    threshold considered scene change\n"
            "     -v                   print diagnostic output\n"
            "\n"
            "all parameters following -- are considered ops to be chained together\n"
            "into a small compoistion instead of using an xml file, this allows for\n"
            "easy testing of filters, be aware that the default value for all\n"
            "properties will be used\n"
            , application_name);
    exit (-1);
}


/* helper macros to make the commandline easier to parse */

#define match(string) (!strcmp (*curr_arg, (string)))
#define argument() \
do{\
  if (!curr_arg[1] || curr_arg[1][0]=='-'){\
    fprintf (stderr, "error parsing commandline %s needs argument\n", *curr_arg);\
    exit (-1);\
  }\
  curr_arg++;\
}while(0)

void
parse_args (int    argc,
            char **argv)
{
    char **curr_arg;

    if (argc==1) {
        usage (argv[0]);
    }

    curr_arg = argv + 1;

    while (*curr_arg) {
        if (match ("-h") ||
            match ("--help")) {
            usage (argv[0]);
        } else if (match ("--end") ||
                   match ("--time-end") ||
                   match ("-out") ||
                   match ("-e")) {
            argument ();
            end = atof (*curr_arg);
        } else if (match ("--threshold")) {
            argument ();
            threshold = atof (*curr_arg);
        } else if (match ("--op_dir" )||
                   match ("--gggl-ops-dir")) {
            argument ();
            op_dir = strdup (*curr_arg);
        } else if (match ("--name" )) {
            argument ();
            movie_name = strdup (*curr_arg);
        } else if (match ("--start") ||
                   match ("--time-start") ||
                   match ("-in") ||
                   match ("-s")) {
            argument ();
            start = atof (*curr_arg);
        } else if (match ("-v")) {
            verbose = 1;
        } else if (match ("--thumbnails")) {
            write_thumbnails = 1;
        } else {
            if (movie_file) {
                fprintf (stderr, "Error parsing commandline: don't know how to handle multiple xml files\n");
                exit (-1);
            }
            movie_file = strdup (*curr_arg);
        }

        curr_arg++;
    }

    if (verbose)
        fprintf (stderr,
            "Commandline values:\n"
```

```
            "\tend        = %f\n"
            "\top_dir     = '%s'\n"
            "\tstart      = %f\n"
            "\tverbose    = %i\n"
            "\tthreshold  = %f\n"
            "\tthumnails  = %s\n"
            "\tmovie_file = '%s'\n"
            , end, op_dir, start
            , verbose, threshold, write_thumbnails?"yes":"no", movie_file);

    if (!movie_file) {
        fprintf (stderr, "Error: no movie file specified\n");
        exit (-1);
    }
}


static void
movie_get_info (const char *file,
                int        *duration,
                int        *width,
                int        *height)
{
    Oxide *root, *null, *load;
    double ret;

    root = oxide_new (NULL, OxideRoot);
    null = oxide_new (root, OxideOp);
    load = oxide_new (root, OxideOp);

    oxide_set_property (null, "type", "null");
    oxide_set_property (load, "type", "ff");
    oxide_append_child (root, null);
    oxide_append_child (root, load);
    oxide_set_property (load, "resource", file);
    oxide_process (root);

    if (duration)
        *duration = oxide_get_property_double (load, "frames");
    if (width)
        *width    = oxide_get_property_double (load, "width");
    if (height)
        *height   = oxide_get_property_double (load, "height");

    oxide_free (root);
}

/* Når forskjellen mellom to frames overgår en satt terskelverdi,
 * skal det genereres bilder dersom thumbnails er satt
 * Denne funksjonen brukes til å dokumentere differanse
 * mellom hvert bilde som skrives til video bildet sendes
 * til ny videofil
 */
static void
write_frame (const char *movie,
             int frame,
             const char *png_file)
{
    Oxide *root;
    Oxide *loader;
    Oxide *saver;

    root = oxide_parse_xml_chunk (NULL,
          "<oxide>"
          "<op name='saver'  type='png_save' />"
          "<op name='loader' type='ff' />"
          "</oxide>");

    loader = find_op_named (root, "loader");
    saver  = find_op_named (root, "saver");

    oxide_set_property (loader,  "resource", movie);
    oxide_set_property (saver,   "resource", png_file);
    oxide_set_property_double (loader, "frame", frame);

    oxide_process (root);
    oxide_free (root);
}

/* helper funciton to locate an op based on it's name */
```

```
static Oxide *
find_op_named (Oxide *oxide,
               const char *op_name)
{
    Oxide *iter = oxide;

    while (iter) {
        char *name = oxide_get_property (iter, "name");
        if (name) {
            if (!strcmp (name, op_name)) {
                free (name);
                return iter;
            }
            free (name);
        }
        iter = oxide_recurse (iter);
    }
    return NULL;
}
\begin{verbatim}
```

ops/textremove.c

```
#include <gggl/op/gggl_op.h>
#include <math.h>   /* sqrt */
#include <stdio.h>  /* io debugging */

typedef struct {
    double red;
    double green;
    double blue;
} InstanceData;


// Algoritme for å regne ut fargeavstand
static float rgb_distance ( float red0, float green0, float blue0,
                            float red1, float green1, float blue1)
{
    return sqrt ( (red1-red0)     * (red1-red0) +
                  (green1-green0) * (green1-green0) +
                  (blue1-blue0)   * (blue1-blue0));
}

static int
process (GgglOp *op)
{
    InstanceData *id;
    GgglImage    *src_img;
    GgglImage    *dst_img;
    int           width;
    int           height;
    int           x,y;
    int           textpixels = 0;
    float         r1,g1,b1;

    src_img = op->input_pad[0];

    /* leser inn fargekomponentene til første valgte farge */
    id = op->priv;

    r1 = id->red;
    g1 = id->green;
    b1 = id->blue;

    /* henter størrelsen til kildebildet */
    width  = src_img->width;
    height = src_img->height;

    /* allokerer plass i minnet til bildet */
    dst_img = gggl_op_image_alloc (
   op, 0, op->out_pad_fmt[0], width, height);
    op->output_pad[0] = dst_img;

    // generering av masker
    for (y = 0; y < height; y++) {
        float *src;
float *dst;

// Peker til bildene
        src = (float *) gggl_image_offset (src_img, 0, y, 0);
```

```
        dst = (float *) gggl_image_offset (dst_img, 0, y, 0);

// leser inn fargene til pikselet i bilde som skal analyseres
        for (x = 0; x < width; x++) {
            float r0,g0,b0;
            float color_distance;

            r0 = src[0];
            g0 = src[1];
            b0 = src[2];

    // Kritt
    color_distance = rgb_distance (r0,g0,b0,
                                           r1,g1,b1);

    if(color_distance < 0.11) {
        dst[0] = 0;
        textpixels++;
    }
    else
dst[0] = 1;

    // Fargebildet økes med 3 kanaler, gråtone med 1
            src += 3;
            dst += 1;
        }
    }
    fprintf(stderr,"Antal textpiksler%i\n", textpixels);
    return 0;
}


// inputbilde skal være i rgbF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

// output i gF (gråtoner)
static int
query_out_pad_fmt (GgglOp *op,
                   int     pad_no,
                   int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}



static void
open (GgglOp *op)
{
    InstanceData *instance_data;

    instance_data = gggl_op_malloc (op, sizeof (InstanceData));

    /* legger inn krittfarge */
    instance_data->red   = 0.48627; /* 0.46275; */
    instance_data->green = 0.58431; /* 0.59216; */
    instance_data->blue  = 0.52941; /* 0.52549; */


    /* register our level property */
    gggl_op_property_new (op, "red",  GgglPropValue,
             &(instance_data->red),  "target red"   );
    gggl_op_property_new (op, "green",  GgglPropValue,
              &(instance_data->green),  "target green"   );
    gggl_op_property_new (op, "blue",  GgglPropValue,
     &(instance_data->blue),  "target blue"   );

    op->priv = instance_data;

    op->query_in_pad_fmt  = query_in_pad_fmt;
```

```
    op->query_out_pad_fmt = query_out_pad_fmt;
    op->process           = process;
}



static void *op_info[]=
{
    "textremove" , open,
    "description", "detect when text is removed",
    "category"   , "text",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL
};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

2.6 Motion segmentation

2.6.1 motion_detect.c

```
/* motion detection, writes the frames with a terskled
difference to a new movie file*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <oxide/oxide.h>
#include <gggl/gggl.h>   /* for loading additional op dir */

char    *movie_name ="motion_segm.avi";
double   start      = 0.0;
double   end        = 250.0; // under testing
double   step       = 1.0;
/* Når det sammenlignes på to etterfølgende rammer
 * bør diff være rundt 0.02
 * Når det ramme 1 sammenlignes med rammer fram
 * til overgått diff, må verdi være ca 3 %
 * Ca hver 5. ramme blir skrevet til ny film */
double   threshold  = 0.03;
char    *movie_file = NULL;
char    *op_dir     = "ops";
int      write_thumbnails = 0;
int      verbose    = 1; // under testing

/* utility function to parse arguments */
static void
parse_args (int    argc,
            char **argv);

/* utility function to get information about a movie */
static void
movie_get_info (const char *file,
                int        *duration,
                int        *width,
                int        *height);

static Oxide *
find_op_named (Oxide *oxide,
               const char *op_name);


/* utility operation to extract a single frame from a movie file */
static void
write_frame (const char *movie,
             int frame,
             const char *png_file);

/* Opretter xml-tree for lagring av ny film */
static Oxide *saver_open  (const char *movie,
                   const char *movie_out);

/* Funksjon som lagerer hver enkelt ramme */
static void saver_store (Oxide *root,
         int    frame);
```

```
/* Lukker xml-treet */
static void saver_close (Oxide *root);


int
main (int    argc,
      char **argv)
{
    Oxide   *oxide = NULL;
    Oxide   *motion;
    Oxide   *saver;
    double   timeA, timeB;
    double   difference;
    double   diff_frame = 0;

    parse_args (argc, argv);

    oxide_init();

    if (op_dir) {
        if (verbose)
            fprintf (stderr, "loading gggl ops from '%s'\n", op_dir);
        gggl_op_db_load_plugins (op_dir);
    }

    oxide = oxide_parse_xml_chunk (NULL,
        "<oxide>"
        "  <op type='display'/>"
        "  <op type='clone' idref='1'/>"
        "  <op name='motion' type='motion'>"
        "    <op name='movieA' type='avi'>"
        "      <property name='resource'>input.avi</property>"
        "      <property name='frame' interpolation='linear'>"
        "        <value time='0'     >0</value>"
        "        <value time='100000'>100000</value>"
        "      </property>"
        "    </op>"
        "  </op>"
        "  <op name='movieB' type='avi'>"
        "    <output id='1'/>"
        "    <property name='resource'>input.avi</property>"
        "    <property name='frame' interpolation='linear'>"
        "      <value time='0'     >1</value>"
        "      <value time='100000'>100001</value>"
        "    </property>"
        "  </op>"
        "</oxide>");


    oxide_set_property (find_op_named (oxide, "movieA"),
                        "resource", movie_file);

    oxide_set_property (find_op_named (oxide, "movieB"),
                        "resource", movie_file);

    motion = find_op_named (oxide, "motion");

    if (verbose)
        fprintf (stderr, "processing\n");

    if (end==0) {
        int duration;
        movie_get_info (movie_file, &duration, NULL, NULL);
        end = duration;
    }

    timeA = start;
    timeB = start + 1;

    saver = saver_open (movie_file, movie_name);
    /* Peker A skal settes til første ramme */
    oxide_set_property_double (find_op_named (oxide, "movieA"),
                        "_time", timeA);
    do{
        oxide_set_property_double (find_op_named (oxide, "movieB"),
                        "_time", timeB);

        oxide_process (oxide);

difference = oxide_get_property_double (motion, "difference");
```

```
if (verbose) {
           fprintf (stdout, "MovieA: %4.0f. \t",
oxide_get_property_double (
find_op_named(oxide, "movieA"), "_time"));
           fprintf (stdout, "MovieB: %4.0f. \n",
oxide_get_property_double (
find_op_named(oxide, "movieB"), "_time"));
        }

        if (difference > threshold) {
            /* timeA skal peke til ny diff_ramme og timeB
     * skal fortsette å telle oppover i rammer */
    timeA = timeB;
    oxide_set_property_double (find_op_named (oxide, "movieA"),
                        "_time", timeA);
    diff_frame = timeB;
            if (verbose) {
                fprintf (stdout, "Differanse: %f", difference);
        fprintf (stdout, "\t Ramme:%4.0f skrives til video \n",
 diff_frame);
    }

            /* Dersom thumbnail er satt skal det genereres
     * png bilder av diff_frames */
            if (write_thumbnails) {
                char buf[50];
                sprintf (buf, "thumb-%05d.png", (int)diff_frame);
                fprintf (stdout, "Ramme %4.0f skrives til png bilde\n",
 diff_frame);
write_frame (movie_file, diff_frame, buf);
    }
}
saver_store (saver, diff_frame);

        timeB += step;
    } while (timeB <= end);

    saver_close (saver);
    oxide_free (oxide);

    return 0;
}

/* Skrives til skjerm dersom programmet kjøres uten parametere */
void
usage (char *application_name)
{
    fprintf (stderr,
            "usage: %s [options] <movie file>\n"
            "\n"
            "  Options:\n"
            "     --name               name of output movie\n"
    "     --time-start value   time of first frame to process (default 0.0)\n"
            "     --time-end value     time of last frame to process (default 1.0)\n"
            "     --time-step value    amount of time between each frame (default 1.0)\n"
            "     --gggl-ops-dir dir   additional directory to look for gggl ops\n"
            "     --thumbnails         write png thumbnails of keyframes\n"
            "     --threshold value    threshold considered scene change\n"
            "     -v                   print diagnostic output\n"
            "\n"
            "all parameters following -- are considered ops to be chained together\n"
            "into a small compoistion instead of using an xml file, this allows for\n"
            "easy testing of filters, be aware that the default value for all\n"
            "properties will be used\n"
            , application_name);
    exit (-1);
}


/* helper macros to make the commandline easier to parse */

#define match(string) (!strcmp (*curr_arg, (string)))
#define argument() \
do{\
  if (!curr_arg[1] || curr_arg[1][0]=='-'){\
    fprintf (stderr, "error parsing commandline %s needs argument\n", *curr_arg);\
    exit (-1);\
  }\
  curr_arg++;\
```

```
}while(0)

void
parse_args (int    argc,
            char **argv)
{
    char **curr_arg;

    if (argc==1) {
        usage (argv[0]);
    }

    curr_arg = argv + 1;

    while (*curr_arg) {
        if (match ("-h") ||
            match ("--help")) {
            usage (argv[0]);
        } else if (match ("--end") ||
                   match ("--time-end") ||
                   match ("-out") ||
                   match ("-e")) {
            argument ();
            end = atof (*curr_arg);
        } else if (match ("--threshold")) {
            argument ();
            threshold = atof (*curr_arg);
        } else if (match ("--op_dir" )||
                   match ("--gggl-ops-dir")) {
            argument ();
            op_dir = strdup (*curr_arg);
        } else if (match ("--name" )) {
            argument ();
            movie_name = strdup (*curr_arg);
        } else if (match ("--start") ||
                   match ("--time-start") ||
                   match ("-in") ||
                   match ("-s")) {
            argument ();
            start = atof (*curr_arg);
        } else if (match ("-v")) {
            verbose = 1;
        } else if (match ("--thumbnails")) {
            write_thumbnails = 1;
        } else {
            if (movie_file) {
                fprintf (stderr, "Error parsing commandline: don't know how to handle multiple xml files\n");
                exit (-1);
            }
            movie_file = strdup (*curr_arg);
        }

        curr_arg++;
    }

    if (verbose)
        fprintf (stderr,
            "Commandline values:\n"
            "\tend        = %f\n"
            "\top_dir     = '%s'\n"
            "\tstart      = %f\n"
            "\tverbose    = %i\n"
            "\tthreshold  = %f\n"
            "\tthumnails  = %s\n"
            "\tmovie_file = '%s'\n"
            , end, op_dir, start
            , verbose, threshold, write_thumbnails?"yes":"no", movie_file);

    if (!movie_file) {
        fprintf (stderr, "Error: no movie file specified\n");
        exit (-1);
    }
}


static void
movie_get_info (const char *file,
                int        *duration,
                int        *width,
                int        *height)
```

```
{
    Oxide *root, *null, *load;
    double ret;

    root = oxide_new (NULL, OxideRoot);
    null = oxide_new (root, OxideOp);
    load = oxide_new (root, OxideOp);

    oxide_set_property (null, "type", "null");
    oxide_set_property (load, "type", "ff");
    oxide_append_child (root, null);
    oxide_append_child (root, load);
    oxide_set_property (load, "resource", file);
    oxide_process (root);

    if (duration)
        *duration = oxide_get_property_double (load, "frames");
    if (width)
        *width    = oxide_get_property_double (load, "width");
    if (height)
        *height   = oxide_get_property_double (load, "height");

    oxide_free (root);
}

/* Når forskjellen mellom to frames overgår en satt terskelverdi,
 * skal det genereres bilder dersom thumbnails er satt
 * Denne funksjonen brukes til å dokumentere differanse
 * mellom hvert bilde som skrives til video bildet sendes
 * til ny videofil
 */
static void
write_frame (const char *movie,
             int frame,
             const char *png_file)
{
    Oxide *root;
    Oxide *loader;
    Oxide *saver;

    root = oxide_parse_xml_chunk (NULL,
          "<oxide>"
          "<op name='saver'  type='png_save' />"
          "<op name='loader' type='ff' />"
          "</oxide>");

    loader = find_op_named (root, "loader");
    saver  = find_op_named (root, "saver");

    oxide_set_property (loader,  "resource", movie);
    oxide_set_property (saver,   "resource", png_file);
    oxide_set_property_double (loader, "frame", frame);

    oxide_process (root);
    oxide_free (root);
}

/* Genererer xml for å lagre rammer i video */
static Oxide *saver_open  (const char *movie,
                   const char *movie_out)
{
    Oxide *root;
    Oxide *loader;
    Oxide *saver;

    root = oxide_parse_xml_chunk (NULL,
  "<oxide>"
  "<op name='saver'  type='ff_save'/>"
  "<op name='loader' type='avi' />"
  "</oxide>");

    loader = find_op_named (root, "loader");
    saver  = find_op_named (root, "saver");
    oxide_set_property (saver,  "resource", movie_out);
    oxide_set_property (loader, "resource", movie);

    return root;
}

/* Legger til nye rammer i videoen */
```

```
static void saver_store (Oxide *root, int frame)
{
    Oxide *loader;

    loader = find_op_named (root, "loader");
    oxide_set_property_double (loader, "frame", frame);
    oxide_process (root);
}

/* Frigjør treet fra minnet */
static void saver_close (Oxide *root)
{
    oxide_free (root);
}

/* helper funciton to locate an op based on it's name */
static Oxide *
find_op_named (Oxide *oxide,
               const char *op_name)
{
    Oxide *iter = oxide;

    while (iter) {
        char *name = oxide_get_property (iter, "name");
        if (name) {
            if (!strcmp (name, op_name)) {
                free (name);
                return iter;
            }
            free (name);
        }
        iter = oxide_recurse (iter);
    }
    return NULL;
}
```

2.6.2 ops/motion.c

```
#include <gggl/op/gggl_op.h>
#include <math.h>
#include <stdio.h>

typedef struct {
    double difference;
} InstanceData;

static int  process           (GgglOp *op);
static int  query_in_pad_fmt  (GgglOp *op, int pad_no, int fmt);
static void open              (GgglOp *op);

static void *op_info[]=
{
    "motion"      , open,
    "description", "frame difference",
    NULL
};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}

static void
open (GgglOp *op)
{
    InstanceData *instance_data;

    instance_data = gggl_op_malloc (op, sizeof (InstanceData));

    instance_data->difference = 0;

    gggl_op_property_new (op,
            "difference", GgglPropValue, &(instance_data->difference),
            "difference between frames"
    );

    op->priv             = instance_data;
    op->input_pads       = 2;
    op->output_pads      = 0;
```

```
    op->query_in_pad_fmt = query_in_pad_fmt;
    op->process          = process;
}

static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}


/* create histograms for input images, set the difference property
 * of the op, to the sum of the absolute difference between all
 * bins in the two histograms
 */
#define BINS 10

static int
process (GgglOp *op)
{
    InstanceData *id;
    GgglImage    *imgA;
    GgglImage    *imgB;  /* the secondary image is an "auxiliary" image */
    int           width, height;
    int           x,y;
    int           bin;

    int red_binA   [BINS];
    int green_binA [BINS];
    int blue_binA  [BINS];

    int red_binB   [BINS];
    int green_binB [BINS];
    int blue_binB  [BINS];

    imgA     = op->input_pad[0];
    imgB     = op->input_pad[1];
    id       = op->priv;
    width    = imgA->width;
    height   = imgA->height;

    /* clear arrays */
    memset (red_binA,   0, sizeof (red_binA));
    memset (green_binA, 0, sizeof (green_binA));
    memset (blue_binA,  0, sizeof (blue_binA));

    /* fill bins for image A */
    for (y = 0; y < height; y++) {
        float *pix;;

        pix = (float *) gggl_image_offset (imgA, 0, y, 0);

        for (x = 0; x < width; x++) {
            int   r,g,b;  /* the bin to increment */

            r = pix[0] * BINS;
            g = pix[0] * BINS;
            b = pix[0] * BINS;

            if (r>=BINS)
                r=BINS-1;
            if (r<0)
                r=0;
            if (g>=BINS)
                g=BINS-1;
            if (g<0)
                g=0;
            if (b>=BINS)
                b=BINS-1;
            if (b<0)
                b=0;

            red_binA[r]++;
            green_binA[g]++;
            blue_binA[b]++;
```

```
            pix += 3;
        }
    }

    memset (red_binB,   0, sizeof (red_binB));
    memset (green_binB, 0, sizeof (green_binB));
    memset (blue_binB,  0, sizeof (blue_binB));

    /* fill bins for image A */
    for (y = 0; y < height; y++) {
        float *pix;;

        pix = (float *) gggl_image_offset (imgB, 0, y, 0);

        for (x = 0; x < width; x++) {
            int   r,g,b;  /* the bin to increment */
            r = pix[0] * BINS;
            g = pix[0] * BINS;
            b = pix[0] * BINS;

            if (r>=BINS)
                r=BINS-1;
            if (r<0)
                r=0;
            if (g>=BINS)
                g=BINS-1;
            if (g<0)
                g=0;
            if (b>=BINS)
                b=BINS-1;
            if (b<0)
                b=0;

            red_binB[r]++;
            green_binB[g]++;
            blue_binB[b]++;

            pix += 3;
        }
    }

    id->difference = 0;

    for (bin=0; bin< BINS; bin++) {
        id->difference += abs(red_binA[bin]-red_binB[bin]);
        id->difference += abs(green_binA[bin]-green_binB[bin]);
        id->difference += abs(blue_binA[bin]-blue_binB[bin]);
    }

    id->difference = id->difference / (width * height);
    return 0;
}
```

2.7 Areasegmentation

2.7.1 area_segm.c

```
/* motion detection program, writes the frames with a terskled difference to a new movie file*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <oxide/oxide.h>
#include <gggl/gggl.h>   /* for loading additional op dir */

char    *movie_name ="area_segm.avi";
double   hori       = 0.4;
double   vert       = 0.2;
double   start      = 0.0;
double   end        = 250.0; // under testing
double   step       = 1.0;
char    *movie_file = NULL;
char    *op_dir     = "ops";
int      verbose    = 1; // under testing
double   topp0       = 0.0;
double   buttom0     = 0.0;
double   right0      = 0.0;
double   left0       = 0.0;


/* utility function to parse arguments */
static void
parse_args (int    argc,
            char **argv);

/* utility function to get information about a movie */
static void
movie_get_info (const char *file,
                int        *duration,
                int        *width,
                int        *height);


static Oxide *
find_op_named (Oxide *oxide,
               const char *op_name);


/* utility operation to extract a single frame from a movie file */
static void
write_frame (const char *movie,
             int frame,
             const char *png_file);

/* Opretter xml-tree for lagring av ny film */
static Oxide *saver_open  (const char *movie,
                   const char *movie_out);

/* Funksjon som lagerer hver enkelt ramme */
static void saver_store (Oxide *root,
         int    frame);

/* Lukker xml-treet */
static void saver_close (Oxide *root);


int
main (int    argc,
      char **argv)
{
    Oxide   *dst_img;
    Oxide   *marg = NULL;
    Oxide   *crop = NULL;
    Oxide   *area_restr;
    Oxide   *cropping;
    Oxide   *saver;
    double   timeA;
    double   topp, buttom, left, right;
    double   width, height;
    double   new_frame = 0;

    parse_args (argc, argv);

    oxide_init ();
```

```
    if (op_dir) {
        if (verbose)
            fprintf (stderr, "loading gggl ops from '%s'\n", op_dir);
        gggl_op_db_load_plugins (op_dir);
    }

    marg = oxide_parse_xml_chunk (NULL,
        "<oxide>"
        "  <op type='png_save'>"
        "    <property name='resource'>terskleCbCr.png</property>"
        "  </op>"
        "  <op name='area_restr' type='area_restr'>"
        "    <op type='terskle_mask' terskel='0.935' />"
        "    <op type='add_mask'>"
        "      <op type='CB' />"
        "      <op type='clone' idref='4'/>"
        "    </op>"
        "    <op type='CR' />"
        "    <op type='clone' idref='4'/> "
        "  </op>"
        "  <op name='movieA' type='avi'>"
"    <output id='4'/>"
        "    <property name='resource'>input.avi</property>"
        "    <property name='frame' interpolation='linear'>"
        "      <value time='0'     >1</value>"
        "      <value time='100000'>100001</value>"
        "    </property>"
        "  </op>"
        "</oxide>");


    oxide_set_property (find_op_named (marg, "movieA"),
                        "resource", movie_file);

    area_restr = find_op_named (marg, "area_restr");

    if (verbose)
        fprintf (stderr, "Margine detection\n");

    if (end==0) {
        int duration;
        movie_get_info (movie_file, &duration, NULL, NULL);
        end = duration;
    }

    timeA = 0;

    /* Peker A skal settes til fxrste ramme */
    oxide_set_property_double (find_op_named (marg, "movieA"),
                        "_time", timeA);

    oxide_set_property_double (area_restr, "hori"  , hori);
    oxide_set_property_double (area_restr, "vert"  , vert);

    oxide_process (marg);

    // Dersom brukeren ikke allerede har angitt avgrensningspunkter blir
    // disse detektert
    if(topp0 == 0 && buttom0 ==0 && right0== 0 && left0 == 0) {
       topp   = oxide_get_property_double (area_restr, "topp");
       buttom = oxide_get_property_double (area_restr, "buttom");
       left   = oxide_get_property_double (area_restr, "left");
       right  = oxide_get_property_double (area_restr, "right");
    }

    width  = oxide_get_property_double (area_restr, "tot_width");
    height = oxide_get_property_double (area_restr, "tot_height");

    fprintf(stderr, "\nTopp:  %4.0f",  topp);
    fprintf(stderr, "\nBunn:  %4.0f",  buttom);
    fprintf(stderr, "\nLeft:  %4.0f",  left);
    fprintf(stdout, "\nRight: %4.0f\n",right);

    /* Dersom det ikke ble detekter et omrede som er stort nok
     * tilfredstilte e tilfredstille brukerens parametere */
    if(buttom == 0 || right == 0) {
        fprintf(stderr, "Could not find a large enough area!\n"
"Cropping stopped\n");
// TESTING
```

```
//        return 1;
    }

    fprintf(stderr, "\nInput width: %4.0f",  width);
    fprintf(stderr, "\nInput height:%4.0f\n",height);


    crop = oxide_parse_xml_chunk (NULL,
        "<oxide>"
        "  <op name='saver' type='ff_save'>"
        "    <property name='bitrate'>80000000</property>"
        "  </op>"
        "  <op name='crop' type='crop'/>"
"  <op name='movie' type='avi'>"
        "    <property name='resource'>input.avi</property>"
        "    <property name='frame' interpolation='linear'>"
        "      <value time='0'     >1</value>"
        "      <value time='100000'>100001</value>"
        "    </property>"
        "  </op>"
        "</oxide>");


    oxide_set_property (find_op_named (crop, "movie"),
                        "resource", movie_file);

    oxide_set_property (find_op_named (crop, "saver"),
                        "resource", movie_name);

    cropping = find_op_named (crop, "crop");

    if (verbose)
        fprintf (stderr, "Croping video\n");

    do{
        oxide_set_property_double (cropping, "topp"  , topp);
oxide_set_property_double (cropping, "buttom", buttom);
oxide_set_property_double (cropping, "left"  , left);
oxide_set_property_double (cropping, "right" , right);

oxide_set_property_double (find_op_named (crop, "movie"),
                        "_time", timeA);

        oxide_process (crop);

if (verbose)

fprintf(stderr, "Cropping frame:%4.0f\n", timeA);

timeA += step;
    } while (timeA <= end);

    return 0;
}

/* Skrives til skjerm dersom programmet kjxres uten parametere */
void
usage (char *application_name)
{
    fprintf (stderr,
            "usage: %s [options] <movie file>\n"
            "\n"
            "  Options:\n"
    "       Optional restriction points\n"
            "     --topp     topp\n"
            "     --buttom buttom\n"
            "     --right right\n"
            "     --left left\n\n"
            "     --hori-const         horisontale constaint of area\n"
            "     --vert-const         verticale constraint of area\n"
            "     --name               name of the new movie\n"
    "     --time-start value   time of first frame to process (default 0.0)\n"
            "     --time-end value     time of last frame to process (default 1.0)\n"
            "     --time-step value    amount of time between each frame (default 1.0)\n"
            "     --gggl-ops-dir dir   additional directory to look for gggl ops\n"
            "     -v                   print diagnostic output\n"
            "\n"
            "all parameters following -- are considered ops to be chained together\n"
            "into a small compoistion instead of using an xml file, this allows for\n"
            "easy testing of filters, be aware that the default value for all\n"
```

```
            "properties will be used\n"
            , application_name);
    exit (-1);
}


/* helper macros to make the commandline easier to parse */

#define match(string) (!strcmp (*curr_arg, (string)))
#define argument() \
do{\
  if (!curr_arg[1] || curr_arg[1][0]=='-'){\
    fprintf (stderr, "error parsing commandline %s needs argument\n", *curr_arg);\
    exit (-1);\
  }\
  curr_arg++;\
}while(0)

void
parse_args (int    argc,
            char **argv)
{
    char **curr_arg;

    if (argc==1) {
        usage (argv[0]);
    }

    curr_arg = argv + 1;

    while (*curr_arg) {
        if (match ("-h") ||
            match ("--help")) {
            usage (argv[0]);
        } else if (match ("--topp")) {
            argument ();
            topp0 = atof (*curr_arg);
        } else if (match ("--button")) {
            argument ();
            buttom0 = atof (*curr_arg);
        } else if (match ("--right")) {
            argument ();
            right0 = atof (*curr_arg);
        } else if (match ("--left")) {
            argument ();
            left0 = atof (*curr_arg);
        } else if (match ("--horizontale") ||
                   match ("--hori-const")) {
            argument ();
            hori = atof (*curr_arg);
        } else if (match ("--verticale") ||
                   match ("--vert-const")) {
            argument ();
            vert = atof (*curr_arg);
        } else if (match ("--end") ||
                   match ("--time-end") ||
                   match ("-out") ||
                   match ("-e")) {
            argument ();
            end = atof (*curr_arg);
        } else if (match ("--op_dir" )||
                   match ("--gggl-ops-dir")) {
            argument ();
            op_dir = strdup (*curr_arg);
        } else if (match ("--name" )) {
            argument ();
            movie_name = strdup (*curr_arg);
        } else if (match ("--start") ||
                   match ("--time-start") ||
                   match ("-in") ||
                   match ("-s")) {
            argument ();
            start = atof (*curr_arg);
        } else if (match ("-v")) {
            verbose = 1;
        } else {
            if (movie_file) {
                fprintf (stderr, "Error parsing commandline: don't know how to handle multiple xml files\n");
                exit (-1);
            }
```

```
            movie_file = strdup (*curr_arg);
        }

        curr_arg++;
    }

    if (verbose)
        fprintf (stderr,
            "Commandline values:\n"
            "\ttopp       = %f\n"
            "\tbuttom     = %f\n"
            "\tright      = %f\n"
            "\tleft      = %f\n"
            "\thoriz      = %f\n"
            "\tvert       = %f\n"
            "\tstart      = %f\n"
            "\tend        = %f\n"
            "\top_dir     = '%s'\n"
            "\tverbose    = %i\n"
            "\tmovie_file = '%s'\n"
            , topp0, buttom0, right0, left0, hori, vert, start, end, op_dir
            , verbose, movie_file);

    if (!movie_file) {
        fprintf (stderr, "Error: no movie file specified\n");
        exit (-1);
    }
}


static void
movie_get_info (const char *file,
                int        *duration,
                int        *width,
                int        *height)
{
    Oxide *root, *null, *load;
    double ret;

    root = oxide_new (NULL, OxideRoot);
    null = oxide_new (root, OxideOp);
    load = oxide_new (root, OxideOp);

    oxide_set_property (null, "type", "null");
    oxide_set_property (load, "type", "ff");
    oxide_append_child (root, null);
    oxide_append_child (root, load);
    oxide_set_property (load, "resource", file);
    oxide_process (root);

    if (duration)
        *duration = oxide_get_property_double (load, "frames");
    if (width)
        *width    = oxide_get_property_double (load, "width");
    if (height)
        *height    = oxide_get_property_double (load, "height");

    oxide_free (root);
}

/* Genererer xml for e lagre rammer i video */
static Oxide *saver_open  (const char *movie,
                  const char *movie_out)
{
    Oxide *root;
    Oxide *loader;
    Oxide *saver;

    root = oxide_parse_xml_chunk (NULL,
"<oxide>"
"<op name='saver'  type='ff_save'/>"
"<op name='loader' type='avi' />"
"</oxide>");

    loader = find_op_named (root, "loader");
    saver  = find_op_named (root, "saver");
    oxide_set_property (saver,  "resource", movie_out);
    oxide_set_property (loader, "resource", movie);

    return root;
```

```
}

/* Legger til nye rammer i videoen */
static void saver_store (Oxide *root, int frame)
{
    Oxide *loader;

    loader = find_op_named (root, "loader");
    oxide_set_property_double (loader, "frame", frame);
    oxide_process (root);
}

/* Frigjxr treet fra minnet */
static void saver_close (Oxide *root)
{
    oxide_free (root);
}

/* helper funciton to locate an op based on it's name */
static Oxide *
find_op_named (Oxide *oxide,
               const char *op_name)
{
    Oxide *iter = oxide;

    while (iter) {
        char *name = oxide_get_property (iter, "name");
        if (name) {
            if (!strcmp (name, op_name)) {
                free (name);
                return iter;
            }
            free (name);
        }
        iter = oxide_recurse (iter);
    }
    return NULL;
}
```

2.7.2 ops/area_restr.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <gggl/op/gggl_op.h>

typedef struct {
    double hori, vert;
    double topp, buttom, left, right;
    double tot_width, tot_height;
} InstanceData;


static int
process (GgglOp *op)
{
    InstanceData *id;
    int    x, y;
    int    ystart;
    double topp, buttom, left, right;
    double segm_width, segm_height;
    double tot_width, tot_height;
    int    sum_x, sum_y; // piksel i hors. og vert retn.
    double hori, vert;

    id     = op->priv;
    hori   = id->hori;
    vert   = id->vert;
    topp   = 0;
    buttom = 0;
    left   = 0;
    right  = 0;
    ystart = 0;
    segm_width = segm_height = 0;
    tot_height = op->input_pad[0]->height;
    tot_width  = op->input_pad[0]->width;

    /* Går gjennom horisontale linjer */
    for (y = ystart; y <= tot_height; y++) {
        float *mask; // maskert område
```

```
        mask = (float *) gggl_image_offset (op->input_pad[1], 0, y, 0);

        sum_x = 0;
        for (x = tot_width; x ; --x) {
        if(mask[0] == 0)
        sum_x++;
    mask++;
        }

/* tilfredstillller kravet til lengde */
if(sum_x > tot_width * hori) {
            if (!topp)
                topp = y;
}

/* slutt på areal er første linje som ikke tilfredstiller
         * kravet -1*/
        else
    if (topp && !buttom)
        buttom = y-1;

/* Segm_heigt kan ikke regnes ut før både topp
 * og bunn er detektert. Ellers vil man få et negativt
 * resultat */
  if (topp && buttom) {
       segm_height = buttom - topp;

    /* Sjekker om området som nå er markert er stort
     * nok til å være tavle (20 % av totalhøyden).
     * Hvis ikke må det letes videre nedover i bildet */
       if(segm_height < tot_height * vert) {
        topp   = 0;
        buttom = 0;
    }
}
    }

    /* Vertikale linjer
     * Starter fra venstre og markerer første vertikale linje som
     * tilfredsstille kravet til lengde*/
    for (x = 0; x <= tot_width;  x++) {
sum_y  = 0;
        ystart = 0;
        for (y = ystart; y <= tot_height; y++) {
            float *mask; // maskert område
            mask = (float *) gggl_image_offset (op->input_pad[1], x, y, 0);
    if(mask[0] == 0)
                sum_y++;
            mask++;
        }

        /* tilfredstillller kravet til lengde */
        if(sum_y > tot_height * (vert/2)) {
            if (!left)
                left = x;
}
/* første linje som ikke tilfredstiller kravet blir markert
 * som slutt */
else if (left && !right)
    right = x-1;


  if (left && right) {
    segm_width = right - left;
    /* Sjekker om området som nå er markert er stort
    * nok til å være tavle (40 % av totalbredden).
    * Hvis ikke må det letes videre nedover i bildet */
       if(segm_width < tot_width * hori) {
        left   = 0;
        right  = 0;
    }
}
    }

    id->topp   = topp;
    id->buttom = buttom;
    id->left   = left;
    id->right  = right;

    id->tot_width = tot_width;
```

```
    id->tot_height= tot_height;

    return 0;
}

// Bildene som lesses inn skal være i rgbF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}



// Det skal leses inn to inputbilder som skal
// blandes til ett output bilde
static void
open (GgglOp *op)
{
    InstanceData *instance_data;

    instance_data = gggl_op_malloc (op, sizeof (InstanceData));

    instance_data->hori       = 0;
    instance_data->vert       = 0;
    instance_data->topp       = 0;
    instance_data->buttom     = 0;
    instance_data->left       = 0;
    instance_data->right      = 0;
    instance_data->tot_width  = 0;
    instance_data->tot_height = 0;


    gggl_op_property_new (op,
            "hori", GgglPropValue, &(instance_data->hori),
            "horizontale constaints of area");
    gggl_op_property_new (op,
            "vert", GgglPropValue, &(instance_data->vert),
            "verticale constraints of area");
    gggl_op_property_new (op,
            "topp", GgglPropValue, &(instance_data->topp),
            "topp of croped video");
    gggl_op_property_new (op,
            "buttom", GgglPropValue, &(instance_data->buttom),
            "buttom of croped video");
    gggl_op_property_new (op,
            "left", GgglPropValue, &(instance_data->left),
            "left of croped video");
    gggl_op_property_new (op,
            "right", GgglPropValue, &(instance_data->right),
            "right of croped video");
    gggl_op_property_new (op,
            "tot_width", GgglPropValue, &(instance_data->tot_width),
            "total image width");
     gggl_op_property_new (op,
            "tot_height", GgglPropValue, &(instance_data->tot_height),
            "total image height");

    op->priv              = instance_data;
    op->input_pads        = 2;
    op->query_in_pad_fmt  = query_in_pad_fmt;
    op->process           = process;
}


static void *op_info[]=
{
    "area_restr"  , open,
    "description", "Avgrensing av areal",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL};


void
gggl_op_init (void)
{
```

```
    gggl_op_db_register (op_info);
}
```

2.7.3 ops/crop.c

```
#include <gggl/op/gggl_op.h>
#include <stdio.h>

typedef struct {
    double topp, buttom, left, right;
    GgglImage *held_image;
} InstanceData;


static int
process (GgglOp *op)
{
    InstanceData *id;
    id          = op->priv;

    GgglImage    *src_img;
    int           width;
    int           height;
    int           x,y;
    double        topp, buttom;
    double        left, right;

    topp   = id->topp;
    buttom = id->buttom;
    left   = id->left;
    right  = id->right;

//    topp   = 106;
//   buttom = 411;
//    left   = 1;
//    right  = 581;

    src_img = op->input_pad[0];

    /* regner ut størrelsen til bildet */
    width  = right  - left;
    height = buttom - topp;

    /* sjekker om om det er noe i input bildet
     * før det i det hele tatt skrives til output */
    if (!op->input_pad[0]) {
        op->output_pad[0]=NULL;
        return -1;
    }

    /* allokerer plass i minnet til bildet */
    id->held_image = gggl_op_image_alloc (op, 0,
     op->out_pad_fmt[0], width, height);

    /* Legger bildet i output (testing) */
    op->output_pad[0] = id->held_image;

    // Pekere for å flytte området øverst i bildet
    int y_segm;
    y_segm = 0;

    /* Avgrenset område skrives til nytt bilde */
    for (y = topp; y < buttom; y++) {
        float *src;
float *dst;

/* Peker til bildene */
        src = (float *) gggl_image_offset (src_img, 0, y, 0);
        dst = (float *) gggl_image_offset (id->held_image, 0, y_segm, 0);

/* scr må peke til begynnelsen av tavla */
for (x = 0; x < left; x++)
    src += 3;

/* Venstre avgrensning flyttes helt til venstre i skjermen
 * Området skrives til ny fil */
for (x = left; x < right; x++) {
    dst[0] = src[0];
    dst[1] = src[1];
    dst[2] = src[2];
```

```
    dst += 3;
    src += 3;
        }
y_segm++;
    }
    return 0;
}


// inputbilde skal være i gF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

// output i gF (gråtoner)
static int
query_out_pad_fmt (GgglOp *op,
                  int      pad_no,
                  int      fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

static void
open (GgglOp *op)
{
    InstanceData *instance_data;

    instance_data = gggl_op_malloc (op, sizeof (InstanceData));

    instance_data->held_image  = NULL;
    instance_data->topp   = 0;
    instance_data->buttom = 0;
    instance_data->left   = 0;
    instance_data->right  = 0;

    gggl_op_property_new (op,
            "help_image", GgglPropValue, &(instance_data->held_image),
            "frame to write");

    gggl_op_property_new (op,
            "topp", GgglPropValue, &(instance_data->topp),
            "topp of croped video");

    gggl_op_property_new (op,
            "buttom", GgglPropValue, &(instance_data->buttom),
            "buttom of croped video");

    gggl_op_property_new (op,
            "left", GgglPropValue, &(instance_data->left),
            "left of croped video");

    gggl_op_property_new (op,
            "right", GgglPropValue, &(instance_data->right),
            "right of croped video");

    op->priv              = instance_data;

    op->query_in_pad_fmt  = query_in_pad_fmt;
    op->query_out_pad_fmt = query_out_pad_fmt;
    op->process           = process;

    op->input_pads = 1;
    op->output_pads = 1;
}

static void *op_info[]=
{
    "crop" , open,
    "description", "skriver avgrenset område til nytt bilde",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
```

```
    NULL
};


void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

2.8 Diverse kode

For å generere bilder som viser hvilke piksler som har vært i bevegelse ble følgende kode utviklet.

2.8.1 visualize_motion.c

```
#include <gggl/op/gggl_op.h>
#include <math.h>   /* sqrt */

// Algoritme for e regne ut fargeavstand
static float rgb_distance ( float red0, float green0, float blue0,
                            float red1, float green1, float blue1)
{
    return sqrt ( (red1-red0)     * (red1-red0) +
                  (green1-green0) * (green1-green0) +
                  (blue1-blue0)   * (blue1-blue0));
}


static int
process (GgglOp *op)
{
    GgglImage    *src_imgA, *src_imgB;
    GgglImage    *dst_img;
    int           width;
    int           height;
    int           x,y;

    src_imgA = op->input_pad[0];
    src_imgB = op->input_pad[1];


    /* henter stxrrelsen til kildebildet */
    width  = src_imgA->width;
    height = src_imgA->height;

    /* allokerer plass i minnet til bildet */
    dst_img = gggl_op_image_alloc (op, 0, op->out_pad_fmt[0],
     width, height);
    op->output_pad[0] = dst_img;

    // generering av masker
    for (y = 0; y < height; y++) {
        float *srcA, *srcB;
float *dst;

// Peker til bildene
        srcA = (float *) gggl_image_offset (src_imgA, 0, y, 0);
        srcB = (float *) gggl_image_offset (src_imgB, 0, y, 0);
dst =  (float *) gggl_image_offset (dst_img,  0, y, 0);

// leser inn fargene til pikselet i bilde som skal analyseres
        for (x = 0; x < width; x++) {
            float rA,gA,bA;
            float rB,gB,bB;

    float color_distance;

            rA = srcA[0];
            gA = srcA[1];
            bA = srcA[2];

    rB = srcB[0];
            gB = srcB[1];
            bB = srcB[2];

    dst[0] = 0;

    color_distance = rgb_distance (rA,gA,bA,
```

```
                                        rB,gB,bB);

            /* Det me aksepteres en liten differanse for at
       det ikke skal gjengis mye stxy i bildet */
    if(color_distance < 0.10 && dst[0]==0)
        dst[0] = 1;


    // Fargebilder xkes med 3 kanaler, gretone med 1
            srcA += 3;
            srcB += 3;
    dst  += 1;
        }
    }
    return 0;
}


// inputbilde skal vfre i rgbF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

// output i gF (gretoner)
static int
query_out_pad_fmt (GgglOp *op,
                   int     pad_no,
                   int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}


static void
open (GgglOp *op)
{

    op->input_pads = 2;
    op->query_in_pad_fmt  = query_in_pad_fmt;
    op->query_out_pad_fmt = query_out_pad_fmt;
    op->process           = process;
}



static void *op_info[]=
{
    "visual_motion" , open,
    "description", "visualize motion",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL
};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

For å forbedre prosesseringshastigheten er det flere metoder som ikke lenger blir brukt.

2.8.2 ops/bland_inn.c

Når objekter skulle fjernes, ble det før kjørt to operasjoner:

- en for å maskere bildet
- og en for å fylle masken

Nedenforstående kode for å fylle maske blir ikke lenger brukt. Begge opreasjonene kjøres nå i i obj_segm.c

```
#include <gggl/op/gggl_op.h>
#include <stdio.h>

typedef struct {
    double mixture;
} Priv;


// antall horisontale linjer i bildet
static int
process_lines (GgglOp *op,
               int ystart,
               int yend);


// sjekker om om det er noe i input bildene
// før det i det hele tatt skrives til output
static int
process (GgglOp *op)
{
    if (!op->input_pad[0] && !op->input_pad[1]) {
        op->output_pad[0]=NULL;
        return 0;
    } else if (!op->input_pad[0]) {
        op->output_pad[0]=op->input_pad[1];
        return 0;
    } else if (!op->input_pad[1]) {
        op->output_pad[0]=op->input_pad[0];
        return 0;
    }

    op->output_pad[0] = op->input_pad[0];
    return process_lines (op, 0, op->input_pad[0]->height - 1);
}

// Bestemer pikselformatet til inputbildene
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{  // Første inputbilde er frame x som leses inn og det skal være i farger
// Andre inputbilde er maska som skal være i sort/hvitt
    if ((pad_no == 0 && fmt == gggl_pixfmt ("rgbF"))    ||
        (pad_no == 1 && fmt == gggl_pixfmt ("gF")))
        return 1;
    return 0;
}

// Det skal leses inn to inputbilder som skal
// blandes til ett output bilde
static void
open (GgglOp *op)
{
    Priv *p = gggl_op_calloc (op, 1, sizeof (Priv));

    p->mixture = 0.5;
    gggl_op_property_new (op,
            "ratio", GgglPropValue, &p->mixture,
            "how much of the auxiliary image is mixed in");

    op->priv             = p;
    op->input_pads       = 2;
    op->query_in_pad_fmt = query_in_pad_fmt;
    op->process          = process;
}


static int
process_lines (GgglOp *op, int ystart, int yend) {
    Priv *p = op->priv;
    int y;
    int width = op->input_pad[0]->width;
    if (yend>=op->input_pad[1]->height)
        yend=op->input_pad[1]->height-1;
    if (width>=op->input_pad[1]->width)
        width=op->input_pad[1]->width-1;
```

```
    if (p->mixture <= 0.0) {     /* no need to do anything */
    } else {
        for (y = ystart; y <= yend; y++) {
            int x;
            float *new_frame;
    float *mask;

    new_frame = (float*) gggl_image_offset (op->input_pad[0], 0, y, 0);
            mask      = (float*) gggl_image_offset (op->input_pad[1], 0, y, 0);

            for (x = width; x ; --x) {
                int c;

// input_pad[0] er fargebildet med 3 kanaler
// terskelverdien for at tavle skap tolkes som grønn
// er i maskeringsbildet satt til noe over 0.8
                    if (mask[0]>= 0.8){
    // legger inn farge for alle kanalene i fargebilet
                for (c = 0; c < op->input_pad[0]->channels; c++)
       new_frame[c] = new_frame[c];
    // skal ikke bildet oppdateres settes det sort
    } else {
new_frame[0] = 0;
new_frame[1] = 0;
new_frame[2] = 0;
    }

//                      (new_frame[c] * (1.0 - p->mixture)
//                      mask[c] * (p->mixture));
//    float *test;
//    float *test1;
//    test =(float*) op->input_pad[0]->packstride;
// fprintf (stderr, "Input_pad:%f", test[0]);
// fprintf (stderr, "Input_pad2:%f", (float) op->input_pad[1]->packstride);
//
                new_frame += 3; // op->input_pad[0]->packstride;
                mask   += 1; // op->input_pad[1]->packstride;
            }
        }
    }
    return 0;
}

static void *op_info[]=
{
    "blandinn"        , open,
    "description", "age, "
                   "can be animated to become a crossfade, note that the "
                   "current version blatantly expect both inputs to be of the same "
                   "pixel format, something that is not always the case ",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

2.8.3 ops/visual_motion.c

For å visualisere bevegelse i bildet ble følgende algoritme utviklet.

```
#include <gggl/op/gggl_op.h>
#include <math.h>   /* sqrt */

// Algoritme for å regne ut fargeavstand
static float rgb_distance ( float red0, float green0, float blue0,
                            float red1, float green1, float blue1)
{
    return sqrt ( (red1-red0)     * (red1-red0) +
                  (green1-green0) * (green1-green0) +
                  (blue1-blue0)   * (blue1-blue0));
}


static int
process (GgglOp *op)
{
```

```
    GgglImage    *src_imgA, *src_imgB;
    GgglImage    *dst_img;
    int           width;
    int           height;
    int           x,y;

    src_imgA = op->input_pad[0];
    src_imgB = op->input_pad[1];


    /* henter størrelsen til kildebildet */
    width  = src_imgA->width;
    height = src_imgA->height;

    /* allokerer plass i minnet til bildet */
    dst_img = gggl_op_image_alloc (op, 0, op->out_pad_fmt[0],
     width, height);
    op->output_pad[0] = dst_img;

    // generering av masker
    for (y = 0; y < height; y++) {
        float *srcA, *srcB;
float *dst;

// Peker til bildene
        srcA = (float *) gggl_image_offset (src_imgA, 0, y, 0);
        srcB = (float *) gggl_image_offset (src_imgB, 0, y, 0);
dst =  (float *) gggl_image_offset (dst_img,  0, y, 0);

// leser inn fargene til pikselet i bilde som skal analyseres
        for (x = 0; x < width; x++) {
            float rA,gA,bA;
            float rB,gB,bB;

    float color_distance;

            rA = srcA[0];
            gA = srcA[1];
            bA = srcA[2];

    rB = srcB[0];
            gB = srcB[1];
            bB = srcB[2];

    dst[0] = 0;

    color_distance = rgb_distance (rA,gA,bA,
                                           rB,gB,bB);

            /* Det må aksepteres en liten differanse for at
       det ikke skal gjengis mye støy i bildet */
    if(color_distance < 0.10 && dst[0]==0)
        dst[0] = 1;


    // Fargebilder økes med 3 kanaler, gråtone med 1
            srcA += 3;
            srcB += 3;
    dst  += 1;
        }
    }
    return 0;
}


// inputbilde skal være i rgbF
static int
query_in_pad_fmt (GgglOp *op,
                  int     pad_no,
                  int     fmt)
{
    if (fmt == gggl_pixfmt ("rgbF"))
        return 1;
    return 0;
}

// output i gF (gråtoner)
static int
query_out_pad_fmt (GgglOp *op,
                   int     pad_no,
```

```
                int     fmt)
{
    if (fmt == gggl_pixfmt ("gF"))
        return 1;
    return 0;
}


static void
open (GgglOp *op)
{

    op->input_pads = 2;
    op->query_in_pad_fmt  = query_in_pad_fmt;
    op->query_out_pad_fmt = query_out_pad_fmt;
    op->process           = process;
}



static void *op_info[]=
{
    "visual_motion" , open,
    "description", "visualize motion",
    "copyright"  , "Eirik Grythe <grythe.dk> 2005",
    NULL
};

void
gggl_op_init (void)
{
    gggl_op_db_register (op_info);
}
```

3 Installasjons

3.1 Linux

Installer
- Windows XP først
- Debian testing versjonen // trenger å laste ned første CD
- systemkommander i hdaX // legges den i MBR krasjer Windows når man oppdaterer kjernen og systemkommanderen

- Partisjonstabellen min ble seende slik ut:

Disk /dev/hda: 60.0 GB, 60011642880 bytes
255 heads, 63 sectors/track, 7296 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot Start End Blocks Id System
/dev/hda1 1 5099 40957686 f W95 Ext'd (LBA)
/dev/hda2 5100 5226 1020127+ 17 Hidden HPFS/NTFS // Dok (win)
/dev/hda3 5227 6373 9213277+ 7 HPFS/NTFS // Windows XP
/dev/hda4 6374 7296 7413997+ 83 Linux // Linxu root
/dev/hda5 1 39 313204+ 83 Linux // boot (unødvendig for debian)
/dev/hda6 40 166 1020096 82 Linux swap // swap
/dev/hda7 167 5099 39624291 b W95 FAT32 // delt fat for win og linux

Bruk partisjonssystemet (fsnative). Ved bruk av ext2 og ext3 kan man få problemer hvis systemet henger seg og du må restarte. Jeg måtte installere på nytt fordi jeg mistet dpkg lista under en tvungen restart.

cfdisk /dev/hda // oppsett av partisjonstabeller. Ikke bruk partisjonstabell eller diskmanagement
// i windows etter at linux er installert. Da risikerer du å krasje
// partisjonstabellen og miste alt.

Insatallasjon av kjerne // 2.6.10 er nyeste ferdige versjon
tar xvzf kernel-2.6.10.tar.gz // pakker opp kjerne

cd /usr/src/linux2.6.10
make clean // sletter gamle filer
make menuconfig // laster meny for kjernevalg (krever mye konfigurasjon)
// prøv gjerne å finn en ferdig utviklet kjerne til maskinen på
// nettet så slipper du å bruke mye tid på å sette opp systemet riktig.
make bzImage // kompilerer kjernen
cp arch/i386/boot/bzImage /boot // kopierer kjernepekeren til boot katalogen
make modules
make modules_install // krever installasjon av modules_init_tools (lastes ned)
vim /etc/grub/menu.list // redigere grub oppstartmeny
vim /etc/lilo.conf // redigere lilo oppstartmeny
lilo // oppdaterer lilo bootmanageren

Ved nedlasting av Debian stable må man oppdatere til testingversjonen.
Grunnen er at stable versjonen er ment til bruk på servere og den er ikke
oppdatert siden 2000. For at nye drivere til skjermkort o.l skal vare
oppdateret, er man nødt til å gjøre følgende:

Forandre stable til testing i /etc/apt/sources.list
apt-get update // henter ny pakkeliste
apt-get dist-upgrade // laster ned 400 MB med nye drivere
/usr/sbin/apt-setup // velg kilde dkpg-reconfigure xserver-xfree86 // for å konfigurer X i ettertid:

/etc/xf86Config // ved å kjøre denne fila istedet, benyttes driverene
// som ble lagt til i kjerna. Da fikk jeg i mitt tilfelle
// mulighet til å velge Ati radeon skjermkortet
// og får lagt til at det har 128 MB RAM slik at videobehandling går
// raskt

Håndtering av pakker
apt-cache search ..... // søke etter pakker
apt-get install modules_init_tools // laster ned og installerer pakken modules_init_tools
apt-get remove .. // avinstallerer pakker
apt-get update // henter liste over nye pakker fra nettet
apt-get dist-upgrade // oppdaterer alle pakkene i dpkg

dpkg -l | grep xserver // viser installerte pakker som har server i navnet
dpkg -r –purge .... // fjerner pakker fra listen over hva som skal oppdateres
// brukes ved konflikt mellom pakker

tasksel // program for å legge inn grafiskbrukergrensesnitt, sql, mailserver o.l
synaptic // startes i X. Oversikt over alle pakker med innhold.
// Kan handtere enkeltpakker grafisk

Noen nyttige kommandoer:
tar xvzf filnavn // pakker ut gz og tar filer samtidig
fdisk -l // lister opp partisjoner på hd
mc // filcommander
ncftp // ftpserver
/etc/xf86config // konfigurere X
/etc/profile // forandre path (husk export)
locate // lokalisere filer
ps aux // aktive prosesser
kill -17 9202 // avslutter en prosess
chown eirik.eirik MPlayer-20050201 -R // setter rettigheter rekursivt
chown eirik.eirik * -Rv // setter rettigheter til alle filer i en katalog
chown bruker.gruppe filer -Rv // forandrer eier og bruker til alle filene i en katalog
gdm, kdm, xdm // valgfritt shell
mcedit // fargekode på tekst
syntax on // fargekode
cat /proc/cpuinfo // cpu info
| grep // filter
| // pipe
| grep rs | less // filtrert rs, sidevis
| grep rs | more // filtrert rs, sidevis more
/etc/init.d // oppstartscript
rc0 // første prosesser som startes
init 6 // reboot
mount /dev/hda7 /mnt/hda7 // monterer opp en partisjon

// Legg til DMA støtte i kjernen. Systemet jobber da raskere.
// (For at prosessoren skal slippe å bruke ressurser på å hente data fra hd)
apt-get install hdparm // DMA kontroll
hdparm -tT /dev/hda // hd installinger. Sett (sleep)
hdparm -d1 /dev/hda // config

3.2 Oxide

Installasjon av oxide
- last ned de nyeste versjon av oxide fra pippin.gimp.org/files
- pakk opp alle filene: tar xjf .....
- for å få riktige rettigheter til program må bruker kjøre:
./configure
make
og root kjøre:
make install

Dette kan forandres i ettertid ved å sette eier av filer i en katalog rekursivt:
chown bruker.gruppe filer -Rv

Legg inn Gtk+ utviklingsbiblioteket
apt-get install libgtk2.0-dev // laster ned og installerer

Etter å ha lagt inn gggl-ops-0.0.4 får man filmelding på pkg_config_path.
Følgende må gjøres:
vim /etc/profil // rediger
export PKG_CONFIG_PATH=¨/usr/lib/pkgconfig:/usr/local/lib/pkgconfig¨// legg til

For å få lagt inn gggl-ops-image på man laste den pngbilblioteket
apt-get install libpng3-dev

For å kunne håndtere videofiler må en mpeg kodek lastes ned og installeres
wget http://mplayerhq.hu/MPlayer/cvs/FFMpeg-20050201.tar.bz2 // FFmpeg, last også ned mplayer av samma dato

ffmpeg-dir: ./configure –enable-shared // må enable shared

Legg til:
/usr/local/lib
i /etc/ld.so.conf slik at runtime linkenen finner det lokalt installerte bibliotekefiler også

// når man prøver å kjøre oxide_run får man en feilmelding på av_codec (lyd)
// IKKE last ned libavcodec!! Dette overstyrer referenseintegriteten til ffmpeg filene
// og du vil ikke være i stand til å kjøre oxide under noen omstendighet. Prosessen kan
// ikke reverseres og du vil bli nødt til å legge inn debian på nytt for å gjenopprette
// den orginale linkingen mellom filer. Fjern heller lydbehandling ved å sette
// p->audio_st = NULL; i oxide/gggl-ops-video-0.0.5/ops/ffmpeg/ff_save.c og kjør make og make install etterpå

if (p->fmt->video_codec != CODEC_ID_NONE)

p->video_st = add_video_stream (op, p->oc, p->fmt->video_codec);

if (p->oxide_audio_query && p->fmt->audio_codec != CODEC_ID_NONE)

//p->audio_st = add_audio_stream (op, p->oc, p->fmt->audio_codec);
p->audio_st = NULL;

/usr/src/oxide/gggl-ops-video-0.0.5/ops/ffmpeg# make install // må kjøres etter at lydkompfila er forandret

ldconfig -v // oppdaterer bibliotekfiler

Oxide skal nå fungere. Gratulerer!

Bilblotekene som trengs for å installere det grafiske grensesnittet bauxite ligger ikke i debian pakkebiblioteket på nett lenger. Så foreløpig må kode kompileres og kjøres før resultatet så kan vises med qiv eller mplayer.
For å spille av filmer, last ned og installer mplayer
wget http://mplayerhq.hu/MPlayer/cvs/MPlayer-20050201.tar.bz2 // Mplayer

syntax on // viser fargekode i vim og shell