HØGSKOLEN I ØSTFOLD

Avdeling for Informasjonsteknologi
Remmen
1757 Halden
Telefon: 69 21 50 00
URL: www.hiof.no

Høgskolen i Østfold

PROSJEKTRAPPORT

Prosjektkategori: Sluttraport for Bacheloroppgave	[X] Fritt tilgjengelig
Omfang i studiepoeng: 20	[ ] Fritt tilgjengelig etter:
Fagområde: Informasjonsteknologi	[ ] Tilgjengelig etter avtale med samarbeidspartner

Rapporttittel: Prosjekt Hessdalen	Dato: 24.05.2012 Antall sider: 51 Antall vedlegg: 5
Forfattere: Alex Nilsen, Dag Ole Kristoffersen, Denys Gorbach, Sondre Vego	Veileder: Einar von Krogh
Avdeling / linje: Informatikk	Prosjektnummer: H12D23

Utført i samarbeid med: Erling Strand, Høgskolen i Østfold	Kontaktperson hos samarbeidspartner: Erling Strand

Ekstrakt:
Dette prosjektet har gått ut på å utvikle et program for detektering av lysfenomener, med fokus på Hessdalsfenomenet. Formålet med programmet er å automatisere lagringen av videoklipp med funn av fenomenet.

3 emneord:

Hessdalen
Bildeanalyse
Modulært

Sluttrapport

Prosjekt Hessdalen

Gruppe H12D23

Sondre Vego, Denys Gorbach, Alex Nilsen og Dag Ole Kristoffersen

Våren 2012

Innholdsfortegnelse

1.Sammendrag

Denne rapporten beskriver et analyseprogram for deteksjon av lysfenomener i Hessdalen. Systemet som er utviklet baserer seg på en modulær oppbygging, og har et fokus på videre utvikling. Programmet har som mål å automatisere lagring av videoklipp hvor lysfenomener opptrer. Rapporten inneholder en teknisk beskrivelse av systemets komponenter, samt en konkret løsning for bruk i Hessdalen.

2.Forord

Målet med dette prosjektet er å utvikle et verktøy som skal detektere ukjente lysfenomen i Hessdalen, en dal i Sør-Trøndelag.

Prosjektet er gjennomført i tidsrommet fra januar 2012 til mai 2012 av studenter fra Informatikk og Digital Medieproduksjon. Disse er Alex Nilsen (Informatikk), Dag Ole Kristoffersen (Informatikk), Denis Gorbach (Digital Medieproduksjon) og Sondre Vego (Informatikk). Alle studerer siste året av en bachelorgrad ved avdeling for informasjonsteknologi, Høgskolen i Østfold, og leverer dette prosjektet som sitt avsluttende bachelorprosjekt.

Oppdragsgiver for dette prosjektet er Erling Strand, som arbeider med forskning av ukjente lysfenomen i Hessdalen (Project Hessdalen). Veileder til gruppen er Einar Von Krogh, som er ansatt ved Høgskolen i Østfold.

Vi valgte dette prosjektet fordi det er spennende å kunne få være med å utvikle et verktøy, som forskere senere har muligheten til å benytte for å finne mer ut om “Hessdalsfenomenet”. Det at fenomenet inneholder mye energi som i framtiden kan være mulig å benytte av mennesker, gjør det ekstra inspirerende. Gruppa har utenom dette en stor interesse for programmering, og et større utviklingsprosjekt som dette vil være en god mulighet til å utvide erfaring og kunnskap innenfor området.

Prosjektet har omhandlet flere spennende arbeidsområder, blant annet videostrømmer, bildeanalyse, arbeid med flyradar, webgrensesnitt og modularitet.

3.Innledning

I denne rapporten vil et system for deteksjon av ukjente lysfenomener bli presentert. Formålet med rapporten er å gi en oversikt over systemet som er utviklet, samt trekke fram de tekniske bitene som er nødvendig i forhold til dette. I tillegg vil rapporten beskrive prosessen underveis i prosjektet, tilknyttet både teori- og metodebruk.

Deteksjonssystemet som prosjektet omfatter har som formål å automatisk oppdage ukjente lysfenomener, med utgangspunkt i det såkalte "Hessdalsfenomenet". Fenomenet har vært mulig å observere i flere år, og tidligere er det gjennomført flere studentprosjekter. En utvikling av teknisk utstyr og ulike erfaringer gjennom årene har resultert i at flere ulike systemer er prøvd ut. Til forskjell fra tidligere prosjekter har man her hatt fokus på at kun det mest nødvendige av teknisk utstyr er lokalisert i Hessdalen hvor fenomenet opptrer. I kapittelet "Bakgrunn" vil det bli presentert mer informasjon om både "Hessdalsfenomenet" og forskningen rundt dette.

En konkret problemstilling kommer etter "Bakgrunn", og presenterer problemområdet for prosjektet med flere detaljer. I denne delen beskrives litt rundt strategien som ble lagt i starten for prosjektet, og målene med resultatet av prosjektet.

Teoridelen i rapporten tar for seg ulik teori som er tilknyttet utviklingen av systemet. Denne delen inneholder blant annet informasjon rundt bildeanalyse og modularitet som er benyttet i prosjektet.

Hoveddelen av prosjektet kommer i kapittelet "Systembeskrivelse". Her er det beskrevet hva programmet gjør og hvordan det fungerer. De ulike delene i programmet blir beskrevet, samt konsepter som benyttes for blant annet kommunikasjon og dataflyt.

I neste del blir selve oppsettet for Hessdalen beskrevet konkret, med utgangspunkt i det som tidligere står i systembeskrivelsen. Her kan man lese om løsningen som prosjektet har kommet fram til for å detektere lysfenomener.

Drøftings- og diskusjonsdelen inneholder videre argumentasjon for ulike valg tatt i prosjektet, samt utfordringer som har oppstått underveis. Denne delen inneholder også forslag til videre forbedringer av programmet, og elementer man ikke rakk å komme i mål med.

Deretter kommer konklusjonen som besvarelse på problemstillingen.

Rapporten har videre en del om hvordan prosjektet har vært organisert og dokumentert underveis. Litt ulike verktøy og utviklingsmetoder som er benyttet kan man også lese om her.

Til sist er det satt opp en begrepsliste etterfulgt av lister med referanser og figurer, samt oversikt over vedleggene som tilhører rapporten.

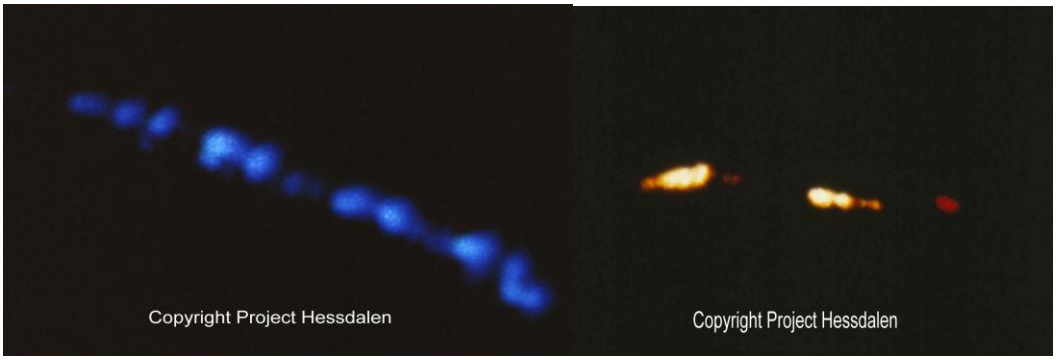
4.Bakgrunn

Hessdalen i Sør-Trøndelag er kjent for et hittil ukjent fenomen som har fått navnet "Hessdalsfenomenet". Dette fenomenet har gjennom flere år skapt stor interesse blant forskermiljøer i flere land. For å gi et bedre innblikk rundt "Hessdalsfenomenet" vil det i denne delen presenteres informasjon om hvordan fenomenet opptrer og litt om hvilke arbeider som er gjort tidligere i forhold til fenomenet.

Figur 1: Kart som viser Hessdalen (Kilde: Gulesider.no)

4.1.Hva er "Hessdalsfenomenet" ?

I starten på 80-tallet begynte et lysfenomen å opptre svært ofte. Mange av innbyggerne i Hessdalen fortalte om unaturlige lys som ble sett både oppe i atmosfæren og nede i dalen rundt der de bodde. Lysfenomenet kunne observeres i ulike hastigheter, alt fra å være et lys helt i ro til at lyset beveget seg med stor fart. Selve lyset opptrådde i ulike farger, blant annet hvit, gul og rød.



Figur 2: Hessdalsfenomenet (Kilde: www.hessdalen.org)

Disse observasjonene førte naturligvis til en del oppmerksomhet fra media, og det ble som regel omtalt som UFO fenomener. Interessen blant norske institusjoner var liten på starten av 80-tallet, dermed startet en gruppe på 5 personer selv et prosjekt (Project Hessdalen) i 1983 for å undersøke mer rundt dette fenomenet.

Siden fenomenet var ukjent visste man lite om hvordan man skulle gå frem for å finne ut mer om det. I 1984 ble den først feltaksjonen satt i gang, hvor rundt 40 personer deltok. Feltaksjonen brukte ulike instrumenter for å samle inn data om fenomenet. Ikke alle instrumentene ga utslag, men fenomenet ble registrert på blant annet radar, magnetometer, og spektrumanalysator. I tillegg var fenomenet synlig og ble dermed også fotografert. Denne feltaksjonen resulterte ikke i noe mer fakta om hva fenomenet var for noe, men en hel del data ble samlet inn, og man fikk kunnskaper om hvilke instrumenter som kunne være nyttig å bruke videre.
Flere detaljer om resultatene fra denne og andre feltaksjoner finnes i ulike rapporter på www.hessdalen.org.

Forekomsten av fenomenet ble mindre når man kom inn på 1990- og 2000-tallet i forhold til det som var på starten av 1980-tallet.

4.2.Hvorfor forskes det på "Hessdalsfenomenet" ?

Da "Project Hessdalen" ble opprettet i 1983 prøvde man å få forskningsmiljøer interessert i fenomenet. Dette viste seg å gi en del utfordringer fordi fenomenet så ofte var blitt omtalt som UFO i media. Etter feltaksjoner og flere foredrag har interessen blant forskere blitt større spesielt i andre europeiske land. I tillegg har "Project Hessdalen" prøvd å få UFO stempelet bort fra fenomenet, og heller bruke begrepet "Hessdalsfenomenet".

Etter hvert som fenomenet ble mer kjent begynte det å komme en del ulike teorier og bortforklaringer på hva det kunne være. Innbyggerne i dalen ble litt frustrerte over at veldig mange uttalte seg uten å engang ha vært i dalen for å observere fenomenet. Stort sett kunne man dele de ulike teoriene opp i to hovedgrupper. Hvor de i den ene gruppen trodde det var snakk om intelligente individer fra andre steder i verdensrommet, imens den andre gruppen hadde teorier som var basert på naturvitenskapelige fenomener. Siden det finnes lite som antyder at det er snakk om utenomjordiske individer var det naturvitenskapelige fenomener forskerne undersøkte videre.

Det er hittil ingen som har noen forklaring på fenomenet, altså er det hele et ukjent fenomen som fortsatt opptrer. Forskere flere steder i verden trigges til å finne ut mer om hva dette kan være, da det kan gi ny kunnskap om naturen, og muligens være med på å påvirke vitenskapen. En ny innsikt om vår verden vil derfor kunne være av stor verdi, og det vil være enklere for de som opplever slike fenomener, både i Hessdalen og andre plasser i verden, å kunne snakke med andre om det hvis man har mer kunnskap å bygge på.

Etter flere år med observasjoner og undersøking av fenomenet har man funnet ut at lyset består av store mengder med energi. Derfor stiller forskere seg spørsmålet om dette er mulig å benytte for oss mennesker på noen måte. I såfall vil det muligens gi en ny energikilde som er fornybar, noe som er av stor betydning i vår tid hvor energi er så mye i fokus.

4.3.Automatisering av datainnsamling

Fenomenet er utfordrende å forske på fordi det opptrer uten noen bestemt form, og forekommer til ulike tider. Da man på førsten av 80-tallet hadde svært mange forekomster er det i dag blitt mange færre, dette i seg selv gir utfordringer med at man ikke har like stor sjanse for å observere det til enhver tid. Derfor har "Project Hessdalen" satt fokus på å benytte moderne teknologi til for å automatisk samle inn data om fenomenet. Ulike prosjekter har jobbet med å utvikle systemer til datainnsamling, og videre vil noe av dette presenteres..

Helt siden rundt midten av 90-tallet har det vært utført mange ulike studentprosjekter knyttet til Hessdalen prosjektet. Mange av studentprosjektene har jobbet rundt en automatisk målestasjon som er plassert i Hessdalen. Målestasjonen brukes til å automatisk samle inn data om fenomenet på ulike måter, som senere kan analyseres. Fordelen med en slik automatisk målestasjon er for å minske tiden man må bruke på å luke ut data hvor fenomenet ikke befinner seg. I løpet av årene som studentprosjektene har foregått har det vært en stor utvikling av teknologisk utstyr, noe som også har preget prosjektene.

Figur 3: Målestasjon i Hessdalen (Kilde: www.hessdalen.org)

4.4.Formål for prosjektet

Hovedsakelig har utstyret tidligere vært plassert i Hessdalen. En slik lokalisering har gitt en del utfordringer på grunn av avstander til utstyret, blant annet i forbindelse med ulike værforhold som har ført til ustabilitet. Utstyret har i flere år vært koblet mot internett, men forbindelsen har blitt raskere den siste tiden i sammenheng med utvikling ellers i samfunnet.

Forrige studentprosjekt (Prosjekt Hessdalen AMS2010) anbefalte en videre utvikling med å oppgradere nettforbindelsen slik at man kunne flytte en del av utstyret til en annen plass. Noe

som forhåpentligvis vil kunne gjøre systemet mer stabilt, siden en del datautstyr dermed kan flyttes til et miljø med mer stabilt klima.

Formålet med dette prosjektet vil være å utvikle et system som via internett kommuniserer med kameraer og en flyradar lokalisert i Hessdalen. Dataene som hentes fra disse komponentene vil i systemet analyseres og lagres når systemet mener det er stor sannsynlighet for deteksjon av Hessdalsfenomenet. Det vil også være en fordel om man senere kan jobbe videre med og utvide systemet som utvikles.

5.Problemstilling og mål for prosjektet

I denne delen kommer den konkrete problemstillingen i prosjektet. Etter problemstillingen er det skrevet litt om strategien som ble lagt i samråd med oppdragsgiver for hvordan den skulle løses, og til sist noen deler om hvilke effekt- og resultatmål man har hatt for prosjektet.

5.1.Problemstilling

Prosjektet skal utvikle et analyseprogram som automatisk skal oppdage ukjente lysfenomener, med utgangspunkt i "Hessdalsfenomenet". Dette systemet skal ta imot videostrømmer fra kameraer i Hessdalen, og automatisk lagre film av fenomenet når det er mulig å observere med kamera. Programmet skal bufre begge videostrømmene, slik at det er mulig å lagre film også en liten periode før fenomenet opptrer, i tillegg skal lagring av film også omfatte en periode etter at fenomenet ikke lenger er å observere. Analyseringsdelen i programmet skal kunne skille ut de delene som oppfattes som fenomen, slik at feilkilder som fly, billys og hus i bygda lukes bort.

5.2.Strategi

I samråd med oppdragsgiver og veileder satte gruppen opp 4 hoveddeler som det var naturlig å dele prosjektet opp i. Disse var følgende:

1. Utvikle et program som detekterer lysfenomenet ut fra å sammenligne bilder med et "bakgrunnsbilde", som definerer en normaltilstand uten fenomener.
2. Utvide programmet med å luke ut fly ved hjelp av en flyradar.
3. Utvikle deteksjonsbiten til å bli mer avansert for å skille ut flere feilkilder.
4. Plotte inn i videofilmen hvor på bildet flyene blir detektert.

Delene over har en prioritering i stigende rekkefølge, men det ble ikke forventet at gruppa klarte å få implementert alle delene. Hovedmålet for gruppa skulle være å få på plass del 1 og 2, slik at man fikk et program som kunne benyttes til deteksjon. Del 3 og 4 var ting man kunne jobbe videre med om det var tilstrekkelig med tid.

5.3.Effektmål

Programmet som utvikles skal være et verktøy for videre forskning på Hessdalsfenomenet. Verktøyet vil være med på å minske manuelt arbeid ved å automatisere utplukking av filmklipp hvor fenomenet opptrer.

5.4.Resultatmål

Det ferdigutviklede programmet skal være kjørbart på et Linux operativsystem, og kunne kjøres på serveren som studentene får disponere under prosjektperioden. Programmet skal kunne detektere lysfenomener, og settes opp ut fra en konfigurasjonsfil.

6.Teori

Denne delen vil gi en teoretisk referanseramme for avsnittet som beskriver oppbygningen av systemet. Når det dukker opp tekniske termer i systembeskrivelsen kan man slå opp i denne delen for å få en kort innføring i de teknikkene som benyttes.

6.1.Bildeanalyse

Med bildeanalyse i denne sammenhengen menes det å prosessere og analysere et bilde digitalt i en datamaskin. En av de første bruksområdene for digitale bilder var i avis bransjen hvor bilder ble overført via sjøkabel mellom London og New York tidlig i 1920 årene. De første datamaskinene som var i stand til å utføre meningsfylte bildeprosesserings oppgaver kom tidlig i 1960 årene. I dag er det nesten ingen tekniske nyvinninger som er uberørt av digital bildebehandling(Gonzalez & Woods, 2002). Her vil det komme en kort forklaring av de grunnleggende teknikkene innen bildeanalyse som blir brukt i dette prosjektet. Denne delen kan brukes som en referanse når man leser beskrivelsen av vår deteksjonsrutine under systembeskrivelsen.

6.1.1.Støyfiltrering

Figur 4: Et stillbilde hentet fra en av videostrømmene som prosjektet skal analysere. Man kan klart se at det er en god del støy i dette bildet, noe som kompliserer deteksjonsprosessen betydelig.

Hovedkildene til støy i digitale bilder er i kameraet når bildet tas, eller under overføringen. Ytelsen til bildesensorer påvirkes av en mengde faktorer slik som lysnivå og temperatur(Gonzalez & Woods s. 222, 2002). Støy er et problem fordi det kan føre til at elementer i bildet ikke blir oppdaget, eller at støyen i seg selv blir tolket som et element i bildet. Det er mange teknikker for støyreduksjon av digitale bilder, men her vil vi begrense oss til å beskrive noen grunnleggende teknikker.

6.1.1.1. Gjennomsnittsfilter

Figur 5: Viser en versjon av bildet i figur 4 som er gjennomsnittsfiltrert med en maske på 9x9 piksler. Her kan man se at støyen har blitt mindre fremtredende i bildet.

Et gjennomsnittsfilter setter hver piksel til en ny verdi som er gjennomsnittet av verdien til omkringliggende piksler og eksisterende piksel. På denne måten reduserer man detaljene i bildet og derfor også de skarpe overgangene der hvor det er støy i bildet. Ulempen til denne filtreringsformen er at også ønskede detaljer i bildet reduseres.

6.1.1.2. Medianfilter

Figur 6: Figuren over viser en versjon av bildet i figur 4 som er medianfiltrert med en maske på 9x9 piksler. Det er verdt å legge merke til at denne filtreringen har fjernet betydelig mere støy enn gjennomsnittsfiltreringen med like stor maske.

Et medianfilter baserer seg også av at man bruker omkringliggende piksler til å beregne ny verdi på en piksel, men her benytter man en annen strategi enn å beregne gjennomsnittet. Som det fremgår av navnet benytter man her medianen til denne gruppen med pikselverdier som ny verdi. Det vil si at man sorterer pikselverdiene i stigende rekkefølge og plukker ut den verdien som havner i midten av denne listen. For enkelte typer støy har denne formen for filtrering klare fordeler. (Gonzalez & Woods s. 123-124, 2002)

6.1.2. Morfologi

Morfologi i denne sammenhengen betegner et sett med operatorer basert på matematiske prinsipper innen mengdelære. Disse operatorene brukes som verktøy for å trekke frem bildekomponenter som er interessante for å beskrive regioner i bildet. Det er to grunnleggende morfologiske operatorer; dilasjon og erosjon, de andre morfologiske operatorene er avledet fra disse. Figuren under(Steven W. Smith 1997) viser effekten til de forskjellige operatorene og en tekstlig forklaring til hver operator følger under figuren.

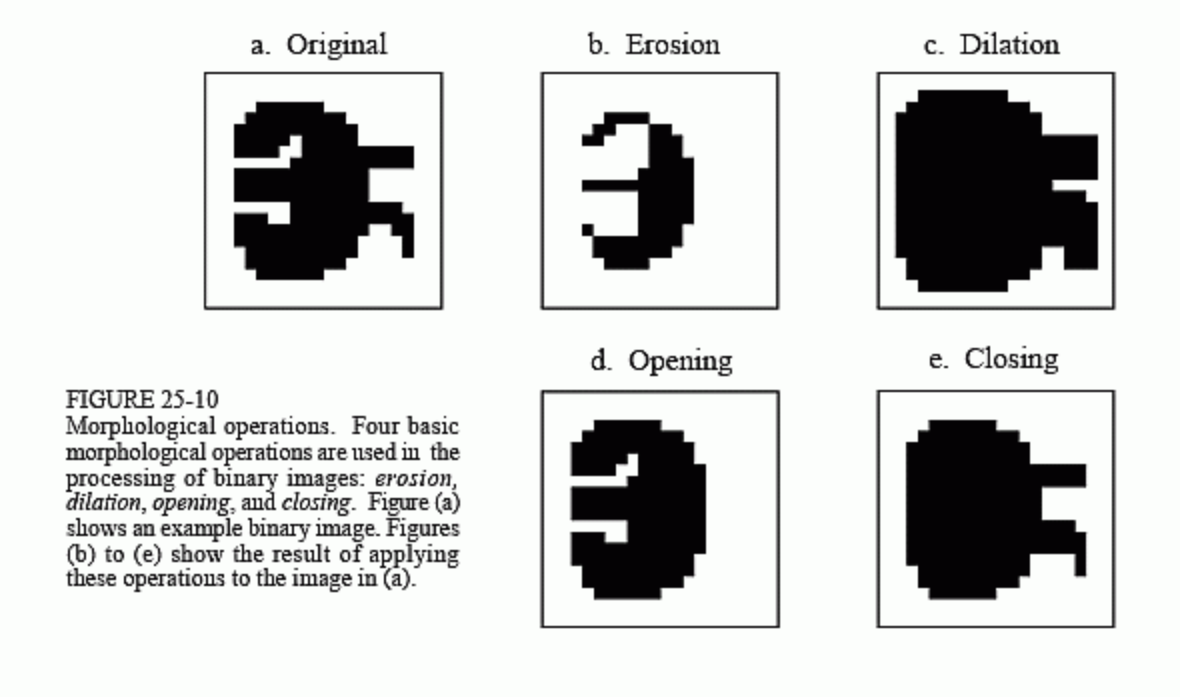


FIGURE 25-10
Morphological operations. Four basic morphological operations are used in the processing of binary images: *erosion*, *dilation*, *opening*, and *closing*. Figure (a) shows an example binary image. Figures (b) to (e) show the result of applying these operations to the image in (a).

Figur 7: Morfologi

6.1.2.1. Dilasjon

Ved dilasjon kan man grovt si at alle regioner i bildet blir utvidet, på denne måten kan man lukke små hull og forbinde regioner som ligger tett inntil hverandre. Ulempen er at etter dilasjon vil arealet til regionen ha økt slik at eventuelle beregninger som går på areal kan feile. Figur Xc viser effekten av dilasjonsoperatoren på bildet i Figur Xa

6.1.2.2. Erosjon

Erosjon er det motsatte av dilasjon, erosjon vil minke alle regioner i bildet. Erosjon kan derfor brukes til å eliminere regioner i bildet under en viss størrelse, slik som støy. Et annet bruksområde for erosjon er å separere regioner som er kunstig knyttet sammen på grunn av støy eller andre forvrengninger av bildet. I likhet med dilasjon vil også denne operatoren forandre arealet til regioner slik at den påvirker slike beregninger. Figur Xb viser effekten av erosjonsoperatoren på bildet i Figur Xa

6.1.2.3. Åpning

Morfologisk åpning er en kombinasjon av erosjon og dilasjon ved at man først kjører en erosjon på bildet, for deretter å kjøre dilasjon. Det man oppnår med dette er mange av de samme fordelene som man får ved en ren erosjon, uten at arealet påvirkes i like stor grad. Figur Xd viser effekten av åpningsoperatoren på bildet i Figur Xa

6.1.2.4. Lukking

Morfologisk lukking er også en kombinasjon av dilasjon og erosjon, men utført i motsatt rekkefølge av det som gjelder for åpning. Det vil si at det først utføres en dilasjon, så en erosjon. Funksjonen til lukking er også stort sett det samme som ved å utføre den første operatoren for

seg selv, bortsett ifra at arealet til regionene i bildet ikke påvirkes i like stor grad. Figur Xe viser effekten av lukningsoperatoren på bildet i Figur Xa

6.1.3.Bakgrunnssubtraksjon

Bakgrunnssubtraksjon går ut på å forsøke å skille det som er statisk bakgrunn i en bildeserie ifra det som er dynamisk. En klassisk måte å gjøre dette på er å kjøre gjennomsnittet på bildene over tid for å lage en bakgrunnsmodell. Dette fungerer bra hvis bakgrunnen er synlig mesteparten av tiden, og objektene beveger seg kontinuerlig. Metoden fungerer ikke like bra hvis det er mange bevegelige objekter hele tiden, da vil disse påvirke bakgrunnsmodellen i altfor stor grad. (Grimson et al, 1999). Forbedringer av denne metoden har gått på at man har brukt modeller for hver piksel istedenfor bildet som helhet. Grimson(1999) har tatt dette ett skritt videre ved å bruke flere modeller for hver piksel for deretter å se på variansen til hver av disse modellene for å anslå hvilken modell som mest sannsynlig er bakgrunn. Bowden og KaewTraKulPong(2001) har foreslått videre forbedringer av metoden til Grimson(1999). Forbedringene deres ligger i oppdateringsalgoritmen, initialiseringsmetode og de har introdusert en metode for skygge deteksjon. De hevder at deres modifikasjoner medfører raskere læring og økt nøyaktighet i modellene. Deres metode for skyggedeteksjon skal ha en klar fordel når det gjelder prosesseringsbehov i forhold til andre lignende strategier.

6.2.Multithreading

Dagens prosessorer inneholder typisk flere kjerner, som regel 2 eller 4 og det er ventet at dette antallet i fremtiden vil øke. For at et program skal kunne utnytte denne kraften bruker man multithreading. Dette vil si at et program deles opp i mindre deler som kan kjøres parallelt. Disse delene kalles tråder. Multithreading gjør det mulig å lage programmer som eksekverer hurtigere, men innebærer samtidig at programmene blir mer komplekse. Dette fordi trådene bruker samme minneområde og kan påvirke hverandre slik at utfallet ikke blir slik det var tiltenkt. Det er noen velkjente fallgruver når man bruker multithreading. Disse fallgruvene er blant annet “Race Condition” og “Deadlock”. Videre kommer litt om slike problemer.

6.2.1.Race Condition

Trådene i et program bruker iblant samme minneområde når de skal oppdatere eller bruke samme variabel. Problemet oppstår når begge trådene ønsker å gjøre dette samtidig. Dette kan føre til at variabelen ikke blir som forventet. For å vise en slik situasjon kommer det videre et eksempel.

Hvis to tråder T1 og T2 ønsker å øke en variabel med 1 kan dette bli resultatet:

1. Integer i = 0; (minnet)
2. T1 leser i til sitt cpuregisterA: 0
3. T2 leser i til sitt cpuregisterB: 0
4. T1 inkrementerer i sitt cpuregisterA: i = 1
5. T2 inkrementerer i sitt cpuregisterB: i = 1
6. T1 lagrer verdien av registerA i minnet : i = 1
7. T2 lagrer verdien av registerB i minnet: i = 1
8. i = 1; (minnet)

Etter at begge trådene har lagt til 1 blir resultatet 1 istedenfor 2. Dette fordi tråd 2 ikke ventet på at tråd 1 fullførte sin addisjon (Wikipedia, Race condition, 2012).

For å unngå disse problemene må trådene synkroniseres. Dette fører til at de ikke forsøker å endre felles variabler samtidig, men venter til andre tråder har gjort seg ferdig. En ulempe med dette er at problemet "Deadlock" kan oppstå.

6.2.2. Deadlock

En deadlock oppstår dersom to eller flere tråder står og venter på hverandre uten at noen slipper delte ressurser. Effekten av et slikt problem vil kunne føre til at programmet står å "henger".

Et konkret eksempel vil være at tråd T1 holder ressursen R1 låst og venter på ressursen R2. Samtidig holder tråden T2 på ressursen R2 og venter på ressursen R1. Da kan begge trådene bli stående og vente i evighet, og man får det som kalles en "Deadlock" (Wikipedia, Deadlock, 2012).

6.2.3. Opencv og multithreading

Intel som startet utviklingen av OpenCv har også utviklet et bibliotek som inneholder optimaliserte funksjoner. Dette biblioteket kalles "Intel® Integrated Performance Primitives (Intel® IPP)". Her ligger det tusenvis av funksjoner som er optimalisert for flerkjernede Intel-cpu'er. Opencv kan kompileres til å bruke dette biblioteket og dette kan føre til at programmer som bruker opencv får en betydelig hastighetsøkning (Using Intel® IPP with OpenCV). Intel IPP har en lisenskostnad og dette prosjektet har derfor ikke tatt i bruk dette biblioteket.

6.3. Modularitet og objektorientert programmering

Programmet er i høy grad bygget på to viktige programmeringsprinsipper. I denne delen er det valgt å ta med teori for å kort beskrive disse to prinsippene.

6.3.1. Modularitet

Modularitet er i programmeringssammenheng et prinsipp om at programmet (for eksempel desktop-applikasjon, bibliotek eller web-applikasjon) er delt inn i separate enheter, som kalles moduler. Modularitet er ofte en måte til å forenkle oppgaven med å lage et program og fordele utviklingsprosessen mellom flere utviklere. Når et program er delt inn i moduler, angis det funksjonalitet gjennomført av hver modul i tillegg til koblinger til andre moduler (*Wikipedia, Modular Programming, 2012)*

Koden blir ofte delt opp i flere filer, hvor hver fil kan kompileres separat fra resten. På den måten kan modularitet i seg selv være med på å redusere kompileringstiden når det kun er få filer som er endret. Ved å modularisere programmet får man et større skille mellom ansvar for ulike oppgaver, og det gjør at programmet ofte blir enklere å feilsøke siden hver enhet har ansvaret for sine oppgaver. Komplekse problemer kan lettere løses av modularitet, da modulen ikke forholder seg så mye til de andre modulen.
Det finnes også mulighet til å erstatte individuelle komponenter (f.eks. dll-biblioteker eller jar filer) av det endelige programmet uten å rekompilere hele programmet på nytt. Det kan være

nyttig hvis man f.eks. utvikler plugins til et allerede ferdig program *(*Haas, Juergen. Definition: Modular programming, About.com).

Objektorientert programmering (OOP) brukes ofte i programmer som bygger på modulær tankegang. Dette fordi OOP består av mange prinsipper som spiller godt sammen med modularitet.

6.3.2. Objektorientert programmering

Objektorientert programmering (OOP) er en teknikk som brukes i programmering. De viktigste begrepene i OOP er klasser og objekter. Et objekt er en instans av en klasse som inneholder attributter og metoder.
(*Wikipedia, Object-oriented Programming, 2012)*

Grunnleggende begreper i OOP er:

- Arv – mulighet til å arve attributter og metoder fra en eller flere andre klasser.
- Innkapsling – skjuling av objektets virkemåte fra andre.
- Klasse – klasser er grunnlaget for objekter, og fungerer som en mal.
- Objekt – en enhet som operettes med nye instanser av en klasse.
- Polymorfi – objekter med samme grensesnitt kan ha forskjellig implementering.

OOP gir muligheten for å gjenbruke kode på en enkel måte. Det bidrar til å øke sikkerheten rundt endring av data, ved bruk av innkapslede attributter, som kun kan endres vi klassens egne metoder. Vedlikehold og feilsøking kan bli enklere ved bruk OOP-teknikkene.
(Lewallen, Raymond, 2005)

7.Systembeskrivelse

Denne delen tar for seg en beskrivelse av systemet som er utviklet. Først vil det bli en kort forklaring av hele systemet, før det videre vil gå litt mer inn i dybden på de enkelte delene.

Systemet har under prosjektperioden blitt endret flere ganger for å få det beste resultatet, og hovedmodellen for systemet har derfor vært under forandring. Til tross for flere iterasjoner vil denne delen presentere det endelige resultatet.

7.1.Hva gjør programmet?

Programmet benytter en videostrøm for å finne lysfenomener som lagres til filmklipp. For å finne lysfenomenene blir videostrømmen analysert av ulike verktøy. Disse verktøyene anslår sannsynligheten for at et lysfenomen er oppdaget, som senere brukes som utgangspunkt for at dette skal lagres til film eller ikke. Programmet baserer seg på en konfigurasjonsfil hvor ulike innstillinger kan modifiseres, og dermed gir programmet høy fleksibilitet.

7.2.Løsningen som er valgt

Slik som i mange andre prosjekter vil det være flere alternative måter å løse problemstillingen på. Videre presenteres noen av alternativene man kunne se for seg, og litt hvilket fokus som preger de ulike måtene. Etter disse er presentert vil det til slutt komme en del om det alternativet som er valgt som løsning i prosjektet.

7.2.1.Statisk program

Et alternativ er å lage et program som leser en videostrøm, analyserer denne og lagrer film med deteksjon til harddisk. En slik løsning vil være tett knyttet til både videostrømmer og ulikt utstyr som benyttes til analysering (for eksempel flyradar). Ved bytte av utstyr eller endring av funksjonalitet vil det ofte føre til at mange ulike deler i programmet må forandres. Derimot vil grunnstrukturen til et slikt program kunne utvikles på kort tid, for så senere å bruke mer tid på funksjonaliteten som skal være med.

7.2.2.Modulært(dynamisk) program

Et annet alternativ er å legge mer fokus på grunnstrukturen i programmet og gjøre det modulært. Da vil man kunne få en dynamisk løsning hvor det vil være store muligheter for konfigurasjoner mellom hver kjøring. En modulær løsning splitter opp de ulike oppgavene i moduler som enkelt kan byttes ut, tas bort eller legges til. Alt etter hvilket oppsett som ønskes. En slik løsning krever mer jobb med grunnstrukturen i programmet, men vil forhåpentligvis gjøre utviklingsarbeidet videre mer enkelt.

7.2.3.Alternativet som er valgt

Programmet som er utviklet bygger på det modulære alternativet. Mer om dette valget kan leses om under drøftingsdelen. Videre vil vi gå litt nærmere inn på programmet som er utviklet.

7.3.Krav til miljøet hvor programmet skal kjøre

Prosjektet er kjørt og testet ut med følgende versjoner av verktøy og rammeverk:

- FFmpeg 0.10
- OpenCV 2.3.1a
- Qt Creator 2.4.1
- Linux (Ubuntu 11.10 og Mint 12)
- En web browser med full støtte av CSS og JavaScript for bruk av webgrensesnitt

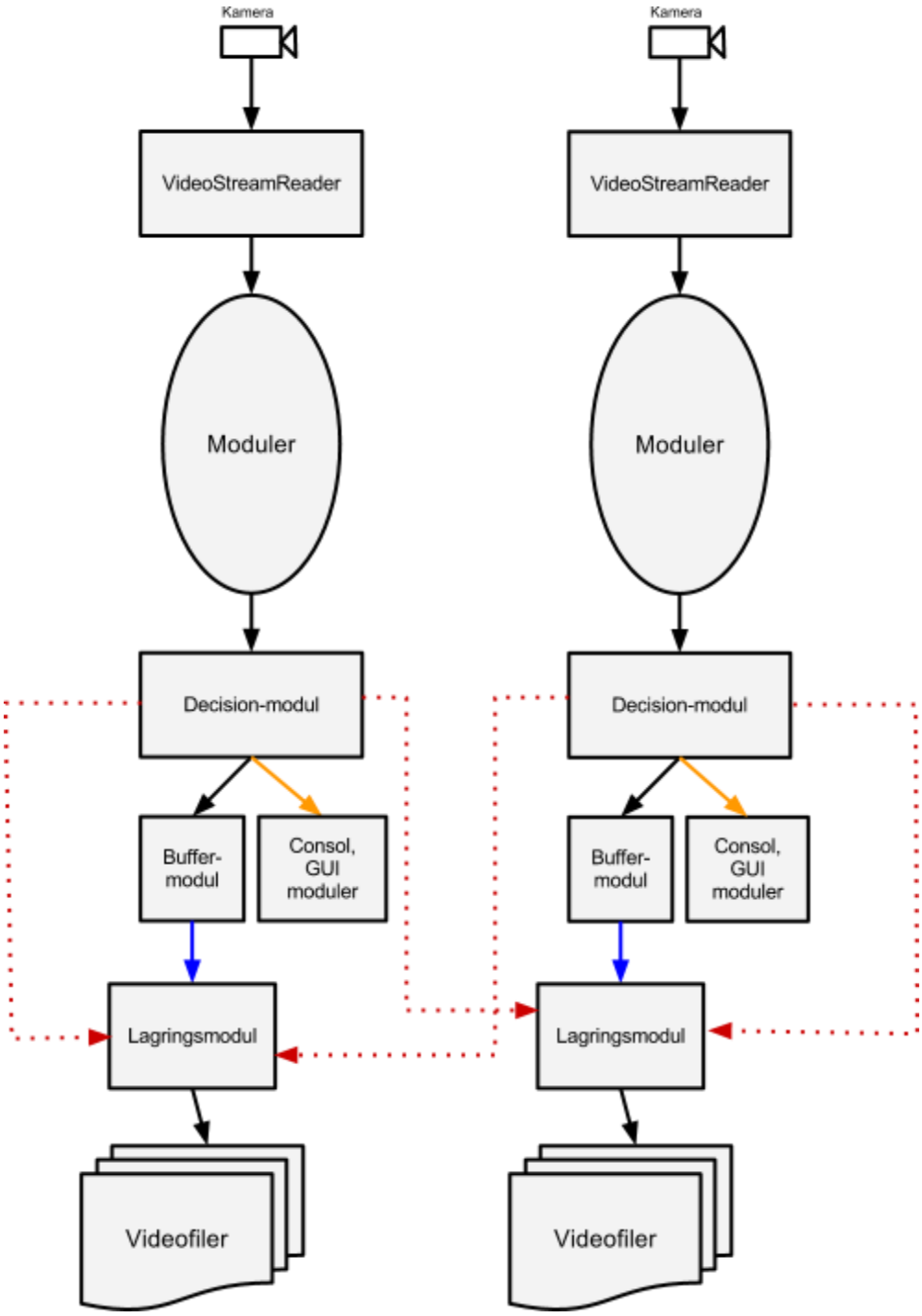
Dette miljøet som programmet er avhengig av er ferdig satt opp på serveren hvor programmet skal kjøres. Det er også satt sammen et script som kan kjøres for å installere alle rammeverkene ved hjelp av terminalen i Linux.

7.4.Konfigurasjonsmuligheter

Grunnlaget for hvordan modulene i systemet settes opp hentes fra en konfigurasjonsfil. I denne filen er det definert hvilke moduler som skal brukes, kommunikasjonen mellom dem og ulike innstillinger til hver enkelt modul. Denne måten å bygge opp systemet på gir en dynamisk effekt hvor oppsettet kan enkelt kan endres for hver kjøring av programmet. Fila kan enten endres direkte, eller ved hjelp av et webgrensesnitt som er utviklet.

7.5.Hovedmodell

For å illustrere hvordan systemet blir bygget opp ved hjelp av modulene er det laget en figur med et eksempel på oppsett. Siden systemet har en høy fleksibilitet er det mulig å benytte svært ulike oppsett etter ulike behov. Til tross for denne muligheten vil det her være fokus på et oppsett tilpasset bruk i forbindelse med “Hessdalsfenomenet”. Under kommer figuren som presenterer dette oppsettet, etterfulgt av en forklaring til de ulike komponentene.



Figur 8: Illustrasjon av et oppsett for programmet

Figuren over viser et oppsett med to modulkjeder i systemet. I toppen av figuren finner man et kamera som indikerer at data fra en videostrøm er input som resten av kjeden vil jobbe med. Neste ledd i kjeden er 'VideoStreamReader', dette er moduler som har til oppgave å ta imot data fra strømmen og bearbeide disse dataene for videre prosessering av påfølgende moduler. Ellipsen som har betegnelsen 'Moduler' illustrerer at det her kan variere hvor mange, og hvilke modultyper som finnes i et spesifikt oppsett ut ifra de konkrete bildebehandlingsoppgavene som

skal utføres. Når bildebehandlingen er gjennomført vil det være en decision modul som tolker informasjonen ifra bildebehandlingsmodulene for å bestemme om det er informasjon i bildet som tilsier at man skal starte lagring av strømmen til disk. Når decision modulen mottar informasjon som tilsier at strømmen bør lagres sender den et signal til lagringsmodulen om å starte lagring. I mellom decision og lagringsmodulen er det en buffermodul som mellomlagrer videostrømmen. Det er her lagringsmodulen vil hente ut videostrømmen ved lagring.

Under "Decisionmodul" finnes en boks med navnet "Console, GUI moduler" som en egen gren ut fra hovedstammen i strømmen. Dette er tatt med for å illustrere muligheten til å koble på flere moduler under en annen modul for å presentere informasjon på forskjellige måter.

7.6.Hovedkomponenter i systemet

Denne delen vil gi en nærmere beskrivelse av de enkeltkomponentene som utgjør systemet. Det vil være fokus på en funksjonell beskrivelse uten å gå altfor dypt inn på tekniske detaljer, for nærmere tekniske beskrivelser henvises det til vedlagt kodedokumentasjon.

7.6.1.FrameInformation

Programmet benytter en type objekter for å sende bildene fra videostrømmen gjennom modulkjede, disse er av typen FrameInformation. Istedenfor å sende bildet alene gjennom kjeden har man valg å legge det inn i et objekt, sammen med endel informasjon som modulene benytter.
Noe av innholdet i FrameInformation er følgende:

- OrginalFrame - bildet som kommer fra videostrømmen
- OverlayFrames - liste med bilder som kan brukes av moduler til å legge på originalbildet
- NaturalPoints - liste med sannsynligheten for naturlige fenomener (billys, fly o.l.)
- NotNaturalPoints - liste med sannsynligheten for uforklarte fenomener
- Id - unik id for dette objektet i kjøringen av programmet

En kort gjennomgang av innholdet over kommer i påfølgende del.

7.6.1.1.OrginalFrame

Bildet som skal sendes gjennom kjeden lagres i denne variabelen. Moduler for bildeanalyse vil ta utgangspunkt i dette bildet.

7.6.1.2.OverlayFrames

Dette er en liste med såkalte overlaybilder. Formålet med denne er at man har muligheten til å benytte moduler som lagrer bilder som skal brukes i sammenheng med OrginalFrame uten å redigere OrginalFrame. Da vil man kunne få muligheten til å f.eks legge på et bilde med tidsstempel eller påtegnede ting. I prosjektet har vi ikke fått tid til å ta denne funksjonaliteten så mye i bruk. Derimot er den likevel med for å gi en slik mulighet til senere bruk.

7.6.1.3.NaturalPoints

Liste med verdier som antyder sannsynligheten for at moduler har funnet naturlige lys i bildet. Som for eksempel lys fra fly eller biler. Settes av analysemoduler, ved å angi navn på modulen og en verdi fra 0 til 100. Benyttes senere av DecisionModule for å avgjøre om ukjente lysfenomen er funnet eller ikke.

7.6.1.4. NotNaturalPoints

Liste med verdier som antyder sannsynligheten for at moduler har funnet unaturlige lys i bildet. Settes av analysemoduler, ved å angi navn på modulen og en verdi fra 0 til 100. Benyttes senere av DecisionModule for å avgjøre om ukjente lysfenomen er funnet eller ikke.

7.6.1.5. Id

Angir en unik id til objektet for gjeldende kjøring. Dette brukes blant annet til debugging av programmet for å sjekke at objektene kommer i riktig rekkefølge.

FrameInformationobjektene opprettes i et stort antall, noe som ga utfordringer med minnehåndtering. Dette ble løst ved hjelp av ObjectManager, og kan leses mer om under delen som tar for seg ObjectManager.

7.6.2. Module Factory

Module factory sørger for å instansiere objekter av klasser som arver Module. I dette programmet gir det den fordelen at man kan legge til nye moduler ved å bare registrere typenavnet på den nye modulen. Dette gjøres i metoden getModule. Da samler man denne funksjonaliteten på en plass, noe som er en fordel hvis man senere ønsker å fjerne eller legge til nye moduler.

```
Module* ModuleFactory::getModule(settings::SettingsGroup *settingsGroup){
    QString moduleType = settingsGroup->getType();

    Module* newModule;
    if(moduleType.compare("VideoStreamProcessorModule", Qt::CaseInsensitive) == 0){
       // newModule = new VideoStreamProcessorModule();

    }
    else if(moduleType.compare("FileOutputModule", Qt::CaseInsensitive) == 0){
        newModule = new FileOutputModule(settingsGroup);
    }
    else if(moduleType.compare("CircularBufferModule", Qt::CaseInsensitive) == 0){
        newModule = new CircularBufferModule(settingsGroup);
    }
    else if(moduleType.compare("FaceDetectionModule", Qt::CaseInsensitive) == 0){
        newModule = new FaceDetectionModule(settingsGroup);
    }

    else {
        qDebug() << "Invalid Module name in module factory";
    }
     //newModule->setSettingsGroup(*settingsGroup);

    return newModule;
}
```

Figur 9: Registrering av modul

For å registrere en ny modul legger man den til ved å lage en ny else if-setning i koden over.

7.6.2.1. EventLogger

Eventloggeren er en klasse som sørger for å logge hendelser fra resten av systemet til loggfiler. Den tilbyr en statisk metode "logEvent" som kan kalles fra andre deler av systemet.

Eventloggerens virkemåte

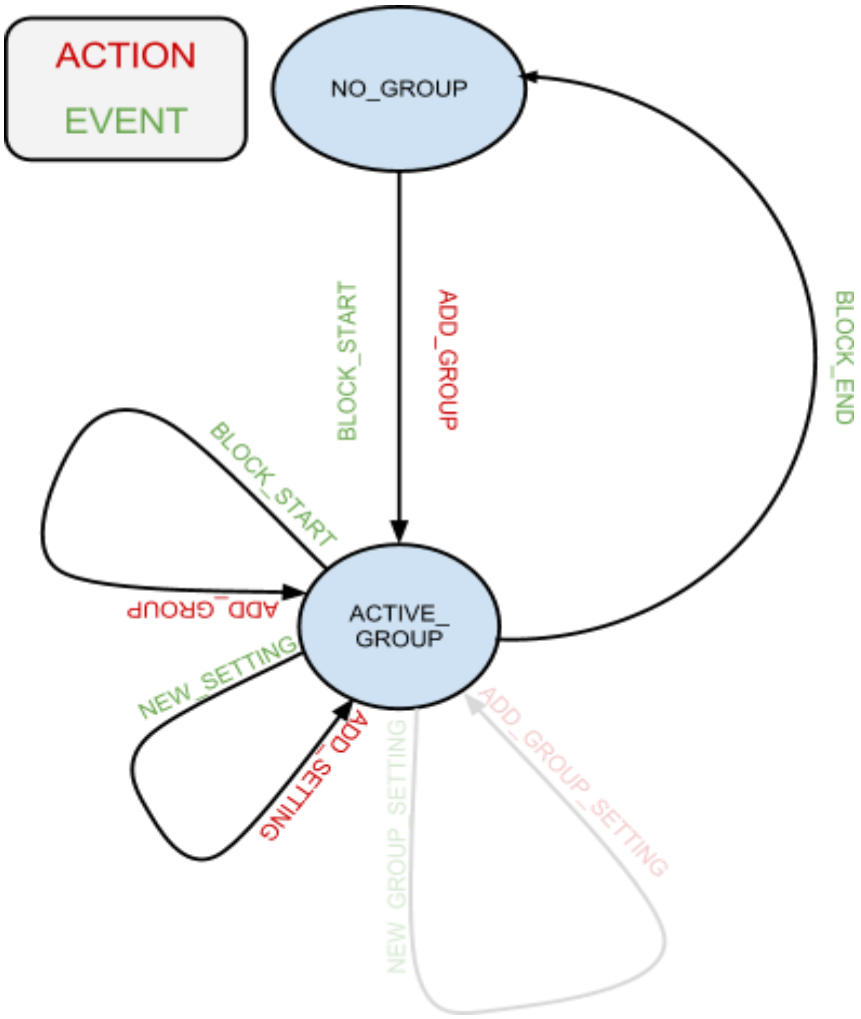
Når metoden logEvent kalles blir loggoppføringen lagt i en FIFO-kø. Denne køen blir ved jevne mellomrom skrevet til loggfilen av eventloggeren. Skrivingen til fil skjer på en egen tråd i eventloggeren, på den måten unngår man at resten av programmet må vente på filskrivingen.

Filskriving

Hendelsesloggene blir skrevet til filer med datostempel. Siden en kjøring av programmet fort kan produsere mange megabyte med logger, lagres hendelser av type "Debug" i egne filer. Disse ligger i en egen mappe under loggmappen. En fil som når en viss størrelse vil begynne å skrive til en ny loggfil. På denne måten unngår man veldig store filer som er upraktisk å åpne i en teksteditor.

7.6.2.2. SettingsParser

For parsing av konfig fila har vi brukt en finite state machine basert løsning hvor det settes forskjellige states ut ifra hvor man er i konfig fila. Figuren under viser de tilstandene som er i bruk og overgangen imellom disse. Tilstandene, eventene og actionene representeres med enum's i koden, i tillegg er det for hver state en egen klasse som arver en abstrakt 'State' klasse. Klassene for hver state definerer hvordan overgangene fra et state til et annet state foregår, og hvilken action som blir utført ved en gitt event.



Figur 10: Tilstander i SettingsParser

Ut ifra figuren kan man for eksempel se at hvis tilstanden er ACTIVE_GROUP og en linje i konfig fila tolkes som en NEW_SETTING event vil en ADD_SETTING action bli utført og man vil fortsatt være i ACTIVE_GROUP tilstanden etter at den er utført. Hvis man derimot i samme tilstand tolker en linje til å inneholde en BLOCK_END event, vil det ikke bli utført noen action, men tilstanden vil forandre seg til NO_GROUP.

7.6.3. ObjectManager

Klassen som heter ObjectManager er viktig for stabiliteten til programmet. Minneforbruket ble litt ut i prosjektet veldig økende, noe som gjorde at programmet ikke kunne kjøre lenge før det ble problemer. Et program som dette er tenkt til å holdes kjørende lenge, og da vil problemer av denne typen være alvorlige. Løsningen ble å benytte en klasse (ObjectManager) som skulle ha ansvaret for alle de objektene som skapte mest problemer med minneforbruket. Dette var FrameInformationobjektene som man opprettet og kastet for hvert bilde som ble sendt gjennom systemet.

ObjectManager består av to lister av objekter, som sees på som en objectpool. Hvor den ene lista inneholder objekter som ikke benyttes, og den andre lista inneholder de objektene som er i bruk. Et objekt som er i bruk har da en eller flere brukere, som i denne sammenheng vil være

moduler. ObjectManager sørger for å flytte objektene mellom de to listene avhengig av om de har brukere eller ikke. Fordelen med dette er å kunne gjenbruke objektene.
Dette skjer på følgende måte:

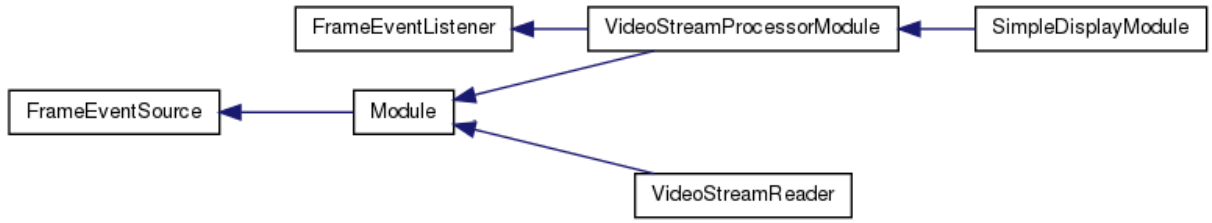
1. En modul trenger et nytt tomt FrameInformationobjekt, og spør ObjectManager om dette
2. ObjectManager finner et objekt som ikke er i bruk og sender det til modulen, samt melder på modulen som bruker til det objektet
3. Modulen sender objektet videre til en ny modul som skal ha tak i FrameInformationobjektet
4. Mottagermodul melder seg på som bruker av objektet, deretter melder sender seg av
5. Siste modul i kjeden vil melde seg av objektet uten å sende det videre. Da vil ObjectManager finne ut at det ikke lenger er noen brukere av objektet, og legge det inn i lista for ubrukte objekter. Slik at objektet er klart til bruk senere.

Objektene som brukes i denne sammenheng trenger man derfor ikke å opprette, samt slette for hvert bilde som sendes gjennom systemet. Eneste gangen man oppretter eller sletter objekter, i objectpoolen, er i de tilfellene det er for få eller for mange objekter der. I slike tilfeller vil det opprettes/slettes et sett med objekter istedenfor ett og ett.

Denne gjenbruksløsningen har fungert meget godt, og løst det økende minneforbruket man hadde tidligere.

7.7.Informasjonsflyt mellom modulene

For å gjøre flyten av informasjon og data mellom modulene så fleksibel som mulig har vi valgt å gjøre denne flyten eventbasert. Måten vi har implementert det på er at modulene abonnerer på events fra andre moduler ved å registrere seg som "Listener" for de event typene den er interessert i. Vi har i første omgang bruk for to typer events, FrameEvent som brukes til å signalisere at nye data er tilgjengelige, og DetectionEvent som sier om et lysfenomen er detektert. For å gjøre det lettere å forstå sammenhengen tar vi utgangspunkt i et minimalt oppsett av systemet hvor det kun er FrameEvents å forholde seg til, et arvediagram for eksempelet sees på figuren under.



Figur 11: Arvediagram til FrameEvent

I dette oppsettet har vi to konkrete moduler, en VideoStreamReader som leser en videostrøm, og en SimpleDisplayModule som viser en videostrøm i et vindu. Hvis vi begynner med VideoStreamReader som har det enkleste arvehierarkiet ser vi at denne arver Module, og at Module arver FrameEventSource igjen. Dette betyr at en VideoStreamReader er en kilde(source) for FrameEvents. Hvis vi nå ser på arvehierarkiet for SimpleDisplayModule ser vi at denne arver VideoStreamProcessorModule, som i tillegg til å arve det samme som VideoStreamReader også arver FrameEventListener. Som resultat av dette kan vi si at SimpleDisplayModule er både en FrameEventSource og en FrameEventListener og derav både tar imot og sender FrameEvents. For å vise litt mer konkret hva det innebærer å være en

FrameEventSource og en FrameEventListener kan vi se på følgende utdrag fra dokumentasjonen for disse klassene:

FrameEventSource

void	**addVideoStreamReciever** (**FrameEventListener** *listener) Adds a listener to the list of FrameEventListeners.
void	**notifyListeners** (**FrameEvent** *event) Loops through the list of FrameEventListeners and notifyes them that a new frame is available.
virtual **FrameInformation** *	**getFrameInformation** ()=0 A pure virtual method for the extraction of a **FrameInformation** object from a **FrameEventSource**.

Figur 12: Dokumentasjon FrameEventSource

7.7.1. FrameEventListener

virtual void	**newFrameEvent** (**FrameEvent** *event)=0 A pure virtual function for recieving FrameEvents.

Figur 13: Dokumentasjon FrameEventListener

Med denne informasjonen for hånden kan de konkrete metode kallene som vil foregå i dette oppsettet beskrives.

7.7.2. Opprette koblingen

Det første som vil skje er at SimpleDisplayModule vil si ifra til VideoStreamReader at den er interessert i de FrameEventene som den genererer ved å kalle addFrameEventListener metoden til VideoStreamReaderen med en referanse til seg selv som parameter. Resultatet av dette er at VideoStreamReaderen vil legge referansen til SimpleDisplayModulen inn i sin liste over "abonnenter".

7.7.3. Sende FrameEvent

La oss si at VideoStreamReaderen har lest en ny frame fra videokilden og ønsker å si ifra til sine abonnenter om dette, dette gjøres ved å kalle sin egen notifyListeners metode. Dokumentasjonen til denne metoden er i figuren under:

void FrameEventSource::notifyListeners (FrameEvent * event)

Loops through the list of FrameEventListeners and notifyes them that a new frame is available.

Parameters:

*event a pointer to a FrameEvent object containing a pointer to the source of the event.

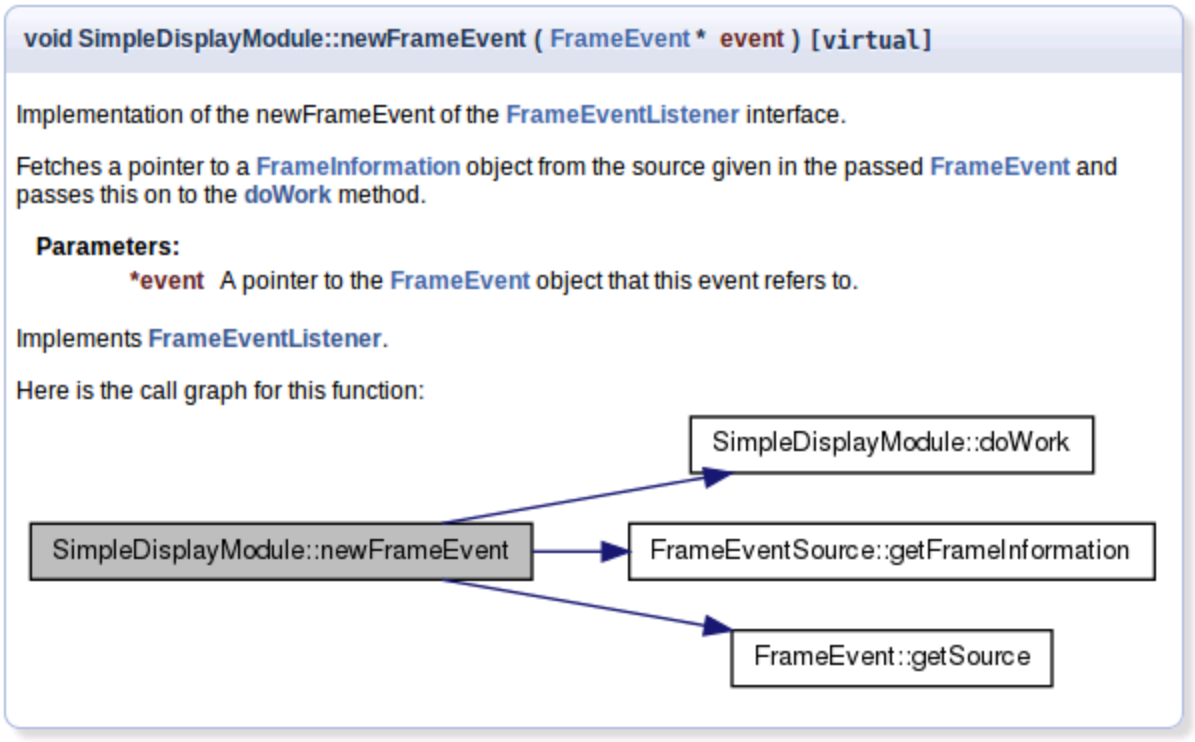
Here is the call graph for this function:

Figur 14: Dokumentasjon notifyListener

Som man ser vil denne metoden gå igjennom listen over abonnenter og kalle deres newFrameEvent metode, og da er det opp til abonnenten hva den ønsker å gjøre videre.

7.7.4. Ta imot en FrameEvent og hente en ny frame

Som vi så i avsnittet over blir SimpleDisplayModule sin newFrameEvent kalt når VideoStreamReader har en ny frame tilgjengelig.



Figur 15: Dokumentasjon newFrameEvent

Som dokumentasjonen viser vil SimpleDisplayModule hente ut kilden til eventen ved å bruke en metode på selve FrameEvent objektet som heter getSource. Når den vet hvor eventen kommer ifra vil den kalle getFrameInformation på kilden (som i dette tilfellet er VideoStreamReaderen) for å hente en peker til selve framen med bildedatene, disse sendes så videre til sin egen doWork metode som gjør selve jobben med å vise videostrømmen.

7.7.5. Andre event typer

I dette eksemplet har vi valgt å gjøre det enkelt ved å bare se på FrameEvents, men fremgangsmåten vil være akkurat den samme for andre event typer(såsom DetectionEvent). Denne måten å knytte modulene sammen på gjør at man står veldig fritt til å utvikle moduler som er avhengig av andre moduler på måter som vi ikke har tenkt på når vi laget vår implementasjon av systemet.

7.8. Moduler som er utviklet

I denne delen presenteres de konkrete modulene som er utviklet, for å gi et innblikk i hva som er oppgaven til modulen. Noen av modulene inneholder også informasjon rundt bakgrunnen for at de er utviklet.

7.8.1. Moduler for lesing av videostrømmer

Vi har utviklet to forskjellige moduler som har oppgave å lese en videostrøm. Grensesnittet og funksjonaliteten til disse modulene er like, hovedforskjellen er hvilket rammeverk som blir benyttet for å lese strømmen og dermed måten de løser den samme oppgaven på.

7.8.1.1. VideoStreamReader

Dette var den første modulen vi utviklet for lesing av videostrømmene. Den benytter seg av funksjonaliteten i opencv rammeverket for å lese videostrømmene. Modulen viste seg å ikke takle videostrømmene ifra Hessdalen spesielt godt, da den mistet forbindelsen og var treg til å koble opp igjen. Andre videostrømmer og videofiler takler den derimot tilfredsstillende.

7.8.1.2. FFMPEGVideoStreamReader

Denne modulen ble utviklet i et forsøk på å omgå de problemene som var med den opprinnelige VideoStreamReader modulen. Rammeverket som benyttes i denne er ffmpeg/libav. Dette rammeverket er det samme som benyttes av opencv, så å bruke det direkte er en mere lavnivå måte å lese videostrømmer på som gir en økt kompleksitet på koden. Fordelen med å bruke ffmpeg/libav direkte er at man får mere kontroll ved feilsituasjoner, som potensielt kan medføre bedre stabilitet. Vår erfaring med denne modulen tilsier at denne potensielle økningen i stabilitet også er realisert i praksis i vår implementasjon.

7.8.2. DetectionModule

Dette er selve kjernen i deteksjonssystemet vårt da det er denne som analyserer bildet for å finne lysfenomener.

Deteksjonsrutinen vår baserer seg på en opencv funksjon kalt “BackgroundSubtractorMOG”, som eliminerer statiske komponenter i bildet. En nærmere beskrivelse av denne, som baserer seg på arbeidet til Bowden (2001), og andre tekniske bildebehandlingstermer finnes i teoridelen av rapporten.

Det første som gjøres i DetectionModule er å lese inn et bilde som brukes til å maskere områder som ikke ønskes analysert. Dette gjøres for å begrense muligheter for feildeteksjoner fra busker som beveger seg i vinden, billys og lignende.

Før det maskerte bildet blir sendt til BackgroundSubtractorMOG funksjonen benytter vi en medianfiltrering av bildet, dette gjøres fordi det spesielt på nattestid er en betydelig andel av støy i bildet. En mer intens median filtrering hjelper på så det blir mindre feildeteksjoner, men øker også prosesseringstiden.

Etter at bildet har vært igjennom medianfiltrering og backgroundsubtractor, brukes morphologisk lukking for å kombinere klynger av småelementer i bildet. Dette gjøres fordi komponenter som egentlig hører sammen har en tendens til å splittes opp i backgroundsubtractor til separate komponenter.

Videre sitter vi igjen med et binært bilde hvor alt som tolkes som bakgrunn blir sorte piksler og alt som tolkes som forgrunn er hvite piksler. Det neste som da gjøres er å finne

sammenhengende komponenter i bildet, hver sammenhengende komponent vil da være et dynamisk objekt. Hvis det finnes dynamiske objekter på n antall påfølgende frames, der n gis av konfig fila, vil det legges inn verdien '100' i listen med NotNaturalPoints i FrameInformation for å angi at denne framen inneholder noe som sees på som et lysfenomen.

7.8.3.FlightRadarModule

For å unngå feildeteksjoner når fly er i nærheten av videokamera ble det kjøpt inn en flyradar (Kinetic SBS-3). Denne flyradaren henter GPS posisjoner til fly innenfor rekkevidde. Med hardwaren følger det også med software for å visualisere posisjoner til flyene. Dette programmet kalles Basestation og kan også kommunisere med flyradaren over Ethernet.

Vi benytter dette programmet i prosjektet da flyradaren er lokalisert i Hessdalen, mens programvaren kjøres fra en server lokalisert en annen plass. Basestation har i tillegg en mulighet for å logge informasjon til telnet, sånn at 3.parts programmer kan hente informasjon.

Figur 16: Flyradar SBS-3 (Kilde: www.thiecom.de/kinetic/sbs3/index.htm)

Flyradarmodulen som er utviklet i dette prosjektet, kobler seg til telnetserveren som Basestation lager for å hente flyinformasjon. Denne informasjonen trengs for å avgjøre om fly er i nærheten av kameraet. Input til denne modulen er et FrameInformation-objekt, og dersom det er fly innenfor et definert sett med koordinater vil modulen sette NaturalPoints = 100 i objektet, for å indikere en høy sannsynlighet for at et fly påvirker deteksjonen.

7.8.4.Buffermodul

I forbindelse med ønske om å kunne lagre film fra et tidspunkt før et fenomen oppdages, vil det være behov for en buffermodul. Buffermodulen er en sirkulær buffer. Det vil si at når den blir full vil den overskrive den tidligste framen som finnes fra før når en ny frame settes inn.

Når en deteksjon skjer og lagringsmodulen begynner å hente ut frames fra bufferen, vil de første frames som hentes ut være fra et tidspunkt før deteksjonen. Hvor langt tilbake i tid vil bestemmes av størrelsen på bufferen. På den måten sørger man for å kunne lagre frames som også omhandler tiden før et fenomen oppdages.

7.8.5.Lagringsmodul

Denne modulen tar seg av lagring av frames til videoklipp, etter signaler fra "Decision" modulen. Oppgaven til modulen vil dermed være å ta imot signaler fra "Decision" modulen og lagre videoklipp til disk. Lagring til disk vil ofte gå saktere enn hastigheten på frames gjennom modulkjeden.

For å forebygge tap av frames, når resten av kjeden må vente på lagringsmodulen, er den satt til å kjøre på en egen tråd, og har i tillegg et eget mellomlager av frames. Når det ikke skjer noen deteksjon vil modulen sitte passiv og vente på signaler.

I det den får beskjed om en deteksjon vil den kalle på buffermodulen, og be om frames som den putter i sitt mellomlager. Den vil fortsette å be om frames til mellomlageret er fullt eller bufferen gir beskjed om den ikke har mere data å gi. Først da vil den sjekke om den trenger å eventuelt opprette en ny fil, for så å skrive hele mellomlageret til disk. Dette vil foregå i en løkke til modulen mottar et signal som sier at deteksjonen har stoppet opp.

Et slikt signal vil sette et flagg internt som sier at deteksjonen er over, men lagringen vil fortsette i ett gitt antall frames etter dette punktet, siden det var et ønske om å lagre noen sekunder også etter en deteksjon. Hvor lenge den vil fortsette er avhengig av innstillinger valgt i konfig filen. Etter dette vil modulen gå tilbake til en passivt lyttende tilstand, med mindre den mottar et nytt deteksjonssignal før denne tiden går ut, da vil lagring fortsette uten å starte på en ny fil.

7.8.6. Decisionmodul

Denne modulen har som formål å avgjøre når det skal sendes et signal (event) videre på om det er funnet en fenomen eller ikke. Input til denne modulen er et FrameInformationobjekt. Modulen er svært enkel med tanke på at den ikke har så mange oppgaver, men er en veldig viktig modul i systemet.

For å gå litt mer inn på det tekniske i modulen, er det viktig å ha en forståelse av FrameInformationobjektene som tas imot. Hvert av disse objektene inneholder to lister med verdier. Disse verdiene er satt av modulene som tidligere har tatt imot objektene i kjeden. Modulen er derfor avhengig av at det er noen moduler foran som vil påvirke listene i objektet for at den skal ha noen praktisk funksjon. Hvis den mottar FrameInformationobjekter hvor listene alltid er tomme, vil den aldri sende noe annet enn signalet for ingen deteksjon av fenomen og kun være en modul hvor FrameInformationobjekter går gjennom.

De to listene har hver sin oppgave. Den ene lista holder på verdier som angir sannsynligheten for at et uforklarlige fenomen er detektert, og heter “NotNaturalPoints”. Den andre inneholder verdier som angir sannsynligheten for at det er detektert naturlige fenomener, og heter “NaturalPoints”. Alle verdiene i disse listene består av et tall fra og med 0 til og med 100, og tolkes da som en prosentverdi.

En utregning mellom disse to listen avgjør, i sammenheng med en innstilling fra konfigurasjonsfila, om signalet som sendes skal indikere deteksjon eller ikke deteksjon av lysfenomen. Først vil modulen gå gjennom listene og finne gjennomsnittet av alle verdiene i hver av dem, og sitter igjen med to verdier. Gjennomsnittet for “NaturalPoints” trekkes fra gjennomsnittet for “NotNaturalPoints” (Gj.snittNotNaturalPoints - Gj.snittNaturalPoints). Resultatet her sier noe om sannsynligheten for at det som er detektert er naturlige lys eller unaturlige.

For å gi brukeren av programmet mulighet til å påvirke avgjørelsen som denne modulen tar, benyttes en innstilling i konfigurasjonsfila. Innstillingen heter “Limit”, og brukes til å sette en grense for utsendelse av signaler. Når resultatet (beskrevet over) går over denne grensa vil modulen sende signaler som indikerer at lysfenomen er oppdaget. I motsatt tilfelle vil det sendes signaler om at lysfenomen ikke er oppdaget. Signalene som sendes vil

mottakermodulen bruke etter eget ønske. Det vil for eksempel være naturlig at en modul som lagrer filmklipp benytter disse signalene til å avgjøre om bildene skal lagres til film eller ikke.

"Limit"-innstillingen som settes i konfigurasjonsfila kan være en utfordring å sette riktig. Man må derfor bruke tid på å teste det ut for å innstille det rett. Det vil være naturlig å ha en verdi mellom 0 og 100, siden det er snakk om prosentverdier. Negative verdier vil også kunne gi utslag, men et negativt resultat av utregningen over gir en tydelig indikasjon på at det vil være en minimal sannsynlighet for lysfenomener.

7.8.7.SimpleDisplay modul

Dette er en enkel modul som kun har som oppgave å vise bildet som den mottar, for så å sende det videre. Visningen av bildene skjer ved hjelp av en spiller i OpenCV, og gir en filmvisningseffekt når mange bilder går gjennom systemet. Modulen kan plasseres ulike steder i kjeden, utenom helt først hvor en videolesermodul bør være.

8.Hessdalenoppsettet

Her vil vi beskrive hvordan vi har brukt systemet som er utviklet, for å løse den konkrete oppgaven med å detektere lysfenomen i Hessdalen. Vi vil begrunne de valgene vi har gjort som gjør at løsningen ikke fullt ut oppfyller de målene vi hadde satt oss ved prosjektets oppstart, og beskrive innstillinger som kan justeres for å justere deteksjonsalgoritmen.

8.1.Konfig fil for Hessdalen

For å sette opp denne løsningen bruker vi følgende konfig fil:

```
#module
name: stream
type: FFMPEGVideoStreamReader
transport: tcp
printInfo: true
source: rtsp://stream.hiof.no:1935/rtplive/_definst_/hessdalen02.stream

#module
name: dshow
type: DetectAndShowModule
history: 30
nmixtures: 8
backgroundratio: 0.99
learningrate: 0.02
blursize: 9
noisesigma: 5
morphsize: 100
minacceptedcontoursize: 30
maxacceptedcontoursize: 50000
FrameEventSource: stream

#module
type: DecisionModule
name: decision
FrameEventSource: dshow
Limit: 40

#module
name: buffer
type: SimpleBufferModule
FrameEventSource: decision
fps: 15
seconds: 15

#module
type: FileOutputModule
name: fileoutput
BufferSource: buffer
```

writeBufferSize: 50
timeBefore: 10
DetectionEventSource: decision
filePath: /home/h12d23/Videos

For en nærmere forklaring på de innstillinger som vi ikke går nærmere inn på her henvises det til vedlagt bruksanvisning.

De viktigste innstillingene for dette oppsettet som det kan være verdt å gå nærmere inn på er de for DetectAndShowModule som er deteksjonsmodulen i oppsettet. Følgende innstilninger setter direkte parametere for BackgroundSubtractor algoritmen i opencv:

history: 30
Denne innstilningen angir hvor stor historikk som brukes for BackgroundSubtractor algoritmen, her har vi satt den til å bruke en historikk på 30 frames, som kombinert med en framerate på 15 frames per sekund gir en to sekunds historikk.

nmixtures: 8
Dette parameteret har vi rett og slett mangelfull forståelse for og henviser til beskrivelsen i opencv dokumentasjonen: "Number of Gaussian mixtures".

backgroundratio: 0.99
Dette parametere er en faktor som angir hvor "følsom" algoritmen er før den tolker noe som forgrunn i bildet, lavere tall angir høyere følsomhet, som gir flere feildeteksjoner.

learningrate: 0.02
Learningrate har med hvor fort algoritmen tilpasser seg til endringer i bildet. Høyere verdier kan medføre at et objekt som beveger seg langsomt tolkes som bakgrunn. Grunnen til dette er at algoritmen rekker å tolke det som statisk før det har beveget seg nok til å slå ut som forgrunn.

noisesigma: 5
Dette parametere er heller ikke forstått fullt ut, men det har med hvordan algoritmen håndterer støy å gjøre. Etter testing satte vi denne verdien til 5 som så ut til å være et godt utgangspunkt.

Følgende innstilninger setter parametere som blir benyttet av modulen direkte:
blursize: 9
Angir størrelsen på masken som brukes når bildet som skal analyseres blir medianfiltrert. Høyere verdier vil gjøre at støy påvirker deteksjonen i mindre grad, men krever også mye av cpu.

morphsize: 100
Størrelsen på strukturelementet som benyttes når det kjøres morphologisk lukning. Høyere verdier vil gjøre at detekterte objekter kan være lenger ifra hverandre før de blir kombinert til ett objekt.

minacceptedcontoursize: 30
Hvor stort et detektert objekt må være målt i antall piksler før det blir beregnet som et objekt.

maxacceptedcontoursize: 50000
Maks størrelse for et detektert objekt målt i piksler før det blir ignorert.

For alle disse parameterne så gjelder det at det ikke har vært tid til å eksperimentere noe særlig med oppsettet for å finne optimale verdier. Håpet er at selv om ting ikke er finjustert har vi gitt nok informasjon om parameterne som kan settes til at bruker selv kan justere deteksjonen.

Som det fremgår av konfig filen har vi i dette oppsettet valgt å lese bare en videostrøm. Dette ble valgt fordi kravet til prosessorhastighet ved to strømmer ble mer enn det vi hadde til rådighet. Mere om denne ytelsesproblematikken kommer under avsnittene om ytelse i drøfting og diskusjon delen av rapporten.

9.Drøfting og diskusjon

9.1.Forbedringspotensiale

Under utviklingen av programmet har det hele tiden vært et fokus på å utvikle en løsning som er fleksibel og konfigurerbar. Målet om å få til en slik fleksibel løsning har delvis lykkes, men det er fremdeles en del forbedringspunkter. Den begrensende faktoren for graden av suksess, har vært kombinasjonen av en stram tidsramme og behov for tidvis stor grad av kompetanseutvikling i gruppen. Som det vil fremgå av den videre drøftingen, så er det for en del av forbedringspunktene gjort konseptuelle planer for hvordan ting kunne vært utbedret. Når slike planer ikke er realisert er det hovedsakelig fordi tiden ikke har strukket til da andre mer kritiske oppgaver har blitt prioritert.

9.1.1.Modularitet

For å oppnå den fleksibiliteten som har vært ønskelig har vi valgt å la programmet bestå av selvstendige moduler. Vi har gjort dette ved å definere et grensesnitt for modulene slik at vi kan forholde oss til modulene som en “black box”. På denne måten vil det som skjer inne i modulen ikke være noe de andre modulene trenger å forholde seg til. For å konfigurere hvilke moduler som skal brukes, og relasjonene dem imellom vil programmet lese en konfigurasjonsfil ved oppstart. Denne løsningen virker robust og fleksibel for de ferdiglagde modulene som er utviklet i prosjektperioden.

En målsetning og motivasjon for denne løsningen har vært at den skal være lett å utvide og videreutvikle for andre enn de som har inngående kjennskap til prosjektet, denne målsetningen er nok bare oppnådd til en viss grad. Det har ikke blitt testet så langt i prosjektperioden å lage en modul i et selvstendig prosjekt, separat fra hovedprosjektet som programmet er utviklet i. Det er usikkerhet rundt hvordan denne prosessen vil fungere i praksis. I tillegg til selve problemstillingen med å kompilere og linke en selvstendig modul kommer problemet med å få programmet til å kjenne igjen denne modulen. Slik det er nå har vi en egen “factory” som instansierer alle modulene, og for hver modul som legges til må denne modifiseres og rekompileres. Det finnes muligheter i Qt rammeverket, som vi allerede bruker, for å loade plugins dynamisk. Dette er sannsynligvis en mulig løsning på problemet(How to Create Qt Plugins). Det er også mulighet for å benytte seg av reflection via Qt rammeverket som kan egne seg for å gjøre denne prosessen enklere(Using the Meta-Object Compiler (moc)).

En annen problemstilling har vært hvordan vi skulle definere interfacet til en modul. Modulen som leser strømmen ifra Hessdalen vil ikke benytte det samme interfacet for å ta imot data som en generisk modul og vil da skille seg ut, men for å passe inn i systemet må også denne kunne defineres som en modul. Vi tok utgangspunkt i denne problemstillingen da vi definerte vår modul klasse og valgte å standardisere interfacet for data ut fra en modul, men ikke for data inn. Senere, og da spesielt når det skulle skrives en veiledning for utvikling av moduler, har det blitt satt spørsmålstegn ved denne avgjørelsen. Det ville kanskje vært mer naturlig at man standardiserte input til en modul, da enhver modul er avhengig av data inn for å kunne gjøre sin jobb. En annen og kanskje enda bedre mulighet hadde vært å standardisere både input og output og heller sagt at moduler med andre krav er avvik fra standardmodellen.

Enda et tema som har vært oppe til diskusjon når det gjelder definisjonen av en modul har vært hvorvidt man skulle gjøre det slik at hver modul kjørte på sin egen tråd. For enkelte modultyper

som kanskje utfører tyngre beregninger kunne dette vært en fordel. Fordelen om vi hadde definert en modul til å ha sin egen tråd er jo hovedsakelig skalerbarhet med tanke på antall moduler. Men hvorvidt dette vil ha noen positiv effekt vil være veldig avhengig av bruksområdet. Hvis relasjonene mellom modulene bare er en lang rekke av moduler, hvor hver modul er avhengig av modulen før seg for å få gjort jobben sin, vil et slikt design mest sannsynlig bare bety ekstra overhead. Hvis relasjonene er slik at det er mange parallelle grener med moduler vil et slikt design utnyttet moderne flerkjerneprosessorer på en god måte. Valget falt på å kjøre enkelte moduler på en egen tråd, uten å gjøre dette til en del av modul interfacet. De modulene vi kjører på en egen tråd er den som leser videostrømmen ifra Hessdalen, og den som skriver til fil. Ved å la den modulen som leser strømmen ifra Hessdalen gå på en egen tråd vil også hele modulkjeden som lytter til denne modulen kjøre på denne tråden, slik vil hver videostrøm få sin egen prosesseringstråd. Valget om å kjøre modulen som skriver til fil på en egen tråd ble gjort fordi skriving til disk er en tidkrevende oppgave som ville kunne "blokkere" tråden.

9.1.2. Logging

For logging brukes Eventlogger-klassen. Denne lagrer logger til fil, men det kunne godt vært mulighet å stille inn hvor disse loggene skal lagres. Dette rakk vi ikke å få implementert.

Det kunne også vært hensiktsmessig å lage egne moduler som leser disse loggfilene og presenterer informasjonen på forskjellig vis med muligheter for filtrering på ulike kriterier.

9.1.3. Audio strøm

Lagring av audio strøm ifra Hessdalen ble ikke nevnt under planleggingsfasen av prosjektet, så hele systemet ble designet og utviklet uten at det var tatt hensyn til audio i det hele tatt. Underveis i prosjektet kom det frem at audio kanalene i strømmen ble benyttet til å overføre data fra blant annet en geigerteller som står i Hessdalen. Takket være den modulære oppbygningen vil det sannsynligvis ikke være mye eksisterende kode som må forandres for å få lagret audio under deteksjon, men oppgaven ble nedprioritert til fordel for å kvalitetssikre hovedoppgaven som er å lagre videostrømmene. Det ble nødvendig å skrive om den delen av koden som leser videostrømmen ifra Hessdalen, da det api'et som først ble brukt ga en veldig ustabil oppkobling. En positiv effekt av dette var at i motsetning til den gamle koden benytter denne nye koden et api som også har støtte for å håndtere audio pakker i strømmen. Det som må gjøres for å få ned audiostrømmen er å skrive om den modulen som leser strømmen slik at den inneholder et audio buffer og et interface for moduler som ønsker audio pakker. Det ville nok vært mest naturlig å legge til en ny event klasse for audio events som kunne sendes mellom moduler. I tillegg må enten modulen som lagrer videostrømmen i dag skrives om til å også håndtere audio, eller en ny modul som lagrer audiofiler må skrives.

9.1.4. Flyradar modul

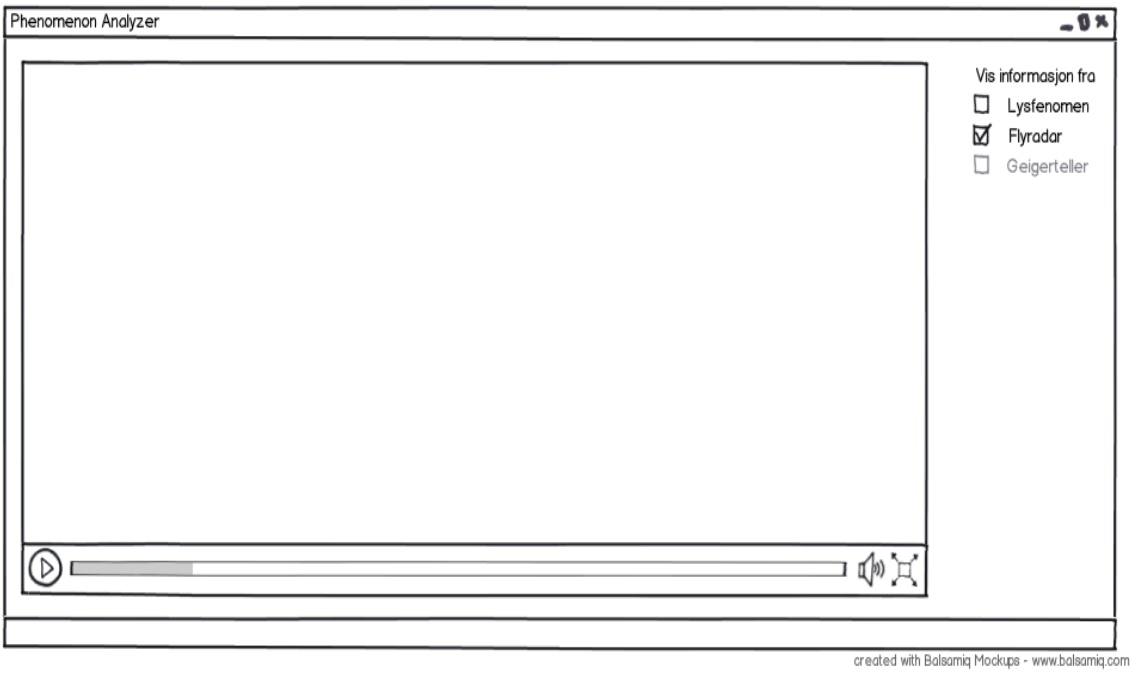
Flyradar modulen viste seg å bli en liten utfordring da dokumentasjonen på hva som skulle til for å få ut og tolke data fra denne boksen viste seg å være svært begrenset. Løsningen som ble valgt baserer seg på å bruke den medfølgende applikasjonen og sette opp denne til å skrive loggdata til Telnet. En stor ulempe med denne løsningen er at avhengighet av et eksternt program for å få tilgang på data. Programmet man blir avhengig av er i tillegg beregnet på windows plattformen og må kjøres via kompabilitetslaget wine for å kunne brukes i Linux. Arbeidet med å utvikle modulen begynte med et forsøk på å skrive kode som jobbet direkte mot boksen for å unngå denne problemstillingen. Grunnen til at denne løsningen ikke ble ferdigstilt er rett og slett at arbeidet ble for tidkrevende. Det bør også defineres koordinater som denne

modulen skal jobbe ut fra. Sånn som modulen fungerer nå ser den bare på om den får kontakt med fly uten å se nærmere på posisjonene deres.

Videre kunne det vært ønskelig å utvide denne modulen slik at den kunne regne om ifra geografiske koordinater til omtrentlige pikselkoordinater. En slik omregning forutsetter at man har en nøyaktig angivelse av posisjon og orientering av de to kameraene, og vil mest sannsynlig måtte kalibreres ved å utføre noen eksperimenter i Hessdalen.

9.1.5. Utvidelser av informasjon som lagres

Slik programmet er i dag er det kun videostrømmen som lagres, andre data blir bare brukt for å avgjøre om man på et gitt tidspunkt skal lagre eller ei. Det kunne vært ønskelig å lagre de dataene som har ligget til grunn for denne beslutningen sammen med videoklippet som lagres. Det kunne eksempelvis vært interessant hvor i bildet et fenomen er detektert, og koordinatene til fly som er i bildet. Et spørsmål i denne sammenhengen er hva slags format slik informasjon skulle lagres som. Opencv biblioteket har funksjoner for å serialisere objekter, noe som kunne vært en løsning for noe av informasjonen. En annen og muligens mere fleksibel løsning kan være å lagre dataene i en ren tekstfil på et egendefinert format. Det kunne også vært et alternativ med XML, men det vil sannsynligvis føre til at disse filene blir mye større enn de trenger å være. En egen avspiller for lagret
informasjon



Figur 17: Videoavspiller

Hvis meta data lagres sammen med videoklippene kunne det vært ønskelig å lage en egen avspiller for disse klippene hvor man kan kombinere videostrømmen med de andre dataene. Her vil det være mulig å gi brukeren mulighet til å skru av og på "overlay's" med informasjon fra forskjellige moduler. Hvis man for eksempel skrur på å vise informasjon om hvor i bildet programmet mener det er et lysfenomen vil man få opp en firkant som markerer dette området i bildet. Hvis man da i tillegg ønsker å se samme informasjon fra flyradar modulen kan man se visuelt om det er sannsynlig at det programmet mener er et lysfenomen er et fly eller ikke. Nå vet ikke gruppen nok om informasjonen som sendes i audio kanalene, men det er også mulig at

informasjon fra disse kan visualiseres. Med en slik spesiallaget avspiller kunne det også være mulig å spille av begge strømmene samtidig slik at man får med hele synsfeltet til det to kameraene.

9.1.6. Ytelse

Programmets mål var å lese to videostrømmer, tolke dem og eventuelt lagre dem. Videostrømmene som skal leses og tolkes er relativt høyoppløselige (1920x720) og overføres med en kodek som krever mye av maskinen som skal dekode dem. Slik programmet er i dag, med den maskinvaren som er tilgjengelig vil programmet kjøre altfor tregt til at en slik konfigurasjon lar seg gjennomføre i praksis. Det er endel ting som kan gjøres for å utbedre dette, en liten gjennomgang av noen mulige utbedringer følger.

9.1.6.1. Optimalisert kompilering av støttebibliotek

For å unngå å bruke altfor mye tid på oppsettet av utviklingsmiljø, har vi ikke satt oss inn i eller benyttet oss av compiler options for å optimalisere for prosessorarkitektur. Det kan være at ytelsen kan økes nevneverdig om man tar seg tid til å sørge for at alle støttebiblioteker, og programmet i seg selv, blir kompilert med et oppsett som utnytter en moderne prosessorarkitektur. Blant annet har opencv støtte for Intel Performance Primitives og Threading Building Blocks. Dette er støttebiblioteker laget for å utnytte de avanserte mulighetene i moderne intel cpu'er, Intel performance primitives er kort beskrevet i teoridelen. Selv om opencv har denne støtten, må man kompilere opencv mot disse bibliotekene for å få utnyttet de fordelene dette gir. Det er verdt å merke seg at i informasjonen om den nye opencv versjonen, som først har kommet mot slutten av prosjektperioden, er backgroundsubtractor funksjonen, som er kjernen i vår deteksjonsrutine, spesielt nevnt med at den har blitt skrevet om for å utnytte Intels Threading Building Blocks. Disse bibliotekene er ikke gratis, og må eventuelt lisensieres om man ønsker å benytte dem.

9.1.6.2. Hardware akselerert dekoding av videostrømmer

I dagens oppsett vil all dekoding av videostrømmer skje i software, noe som i seg selv er en tung oppgave. En måte å minske ressursbruken på kan være å sørge for at denne dekodingen blir hardware akselerert. For å få til dette må man mest sannsynlig lage en ny VideoStreamReader som benytter et annet api for lesing av strømmene.

9.1.6.3. Optimalisering av kode

Koden som har blitt skrevet har i første rekke blitt skrevet med tanke på funksjonalitet og ikke så stor vekt på ytelse. Hvis tiden hadde strekt til ville det vært naturlig å gå over koden og forsøke å optimalisere bort åpenbare ytelsesproblemer. Vi ser for oss at det er spesielt når det gjelder lesing av strøm og analysen i forbindelse med deteksjon at det kan være endel å hente.

Lesing av strøm

Modulen som er skrevet for å lese videostrømmene er preget av at vi har vært svært usikre på rammeverket som benyttes grunnet vanskelig tilgang på dokumentasjon av dette. Om man fikk gjort seg bedre kjent med rammeverket er det mulig at man kunne løst enkelte ting her mer effektivt. Blant annet konverteringen av bildedata fra ffmpeg sitt interne format til formatet som resten av koden forholder seg til kan sannsynligvis skrives mer effektivt.

Deteksjon

Tiden har ikke strekt til når det gjelder å gå igjennom deteksjonsrutinen som vi benytter for å optimalisere denne med tanke på ytelse. I en slik ytelseskritisk applikasjon kan det være et alternativ å avvike noe ifra god programmeringsskikk med oppdeling i generelle metoder og klasser, for å optimalisere med tanke på ytelse. Slik deteksjonen fungerer i dag vil også områder av bildet som er maskert bli prosessert av analysemetodene, selv om disse aldri vil påvirke utfallet. Når masken dekker ganske store områder av bildet er det mye å hente på å få ordnet opp i denne svakheten.

9.1.6.4. Hardware

Et alternativ som kan vurderes, hvis programmet skal brukes til sitt tiltenkte formål, er å sette opp en dedikert server som er satt opp hardwaremessig med tanke på kravene til en slik applikasjon. Dagens prosessor, på serveren som brukes, er en Intel E7500 bestående av en dual core som skal kjøre på 2,93Ghz. Nærmere undersøkelser viser at denne prosessoren er klokket ned i bios, og ikke kjører på full kapasitet. Programmet benytter seg av omfattende multithreading, så det skalerer godt over flere prosessorkjerner. Derav kan det være et godt alternativ med en cpu med flere kjerner som opererer på en tilsvarende eller høyere frekvens. Kombinert med optimalisert kompilering, som beskrevet over, bør gi en merkbar bedre ytelse enn dagens oppsett.

9.1.7. Oppstart av programmet via terminal/ssh

Siden programmet fungerer som en tjeneste på serveren er det tilpasset å kunne kjøres fra terminalvinduet i Linux eller via ssh fra en annen pc. På denne delen gjenstår det noe tilpasninger for at funksjonaliteten ved å styre programmet via terminal skal fungere greit. Denne delen har vært avhengig av at programmet ellers hovedsakelig har vært ferdig, og når andre ting har tatt lengre tid enn beregnet har også denne delen blitt utsatt. Programmet kan startes (beskrevet i bruksanvisningsdokumentet) i terminalvinduet, men har ikke noen naturlig måte å avslutte. Gruppa kommer til å jobbe litt videre med funksjonaliteten rundt start og stopp av programmet mot presentasjonen, slik at det vil være mer forståelig.

9.2. Prosjektomfang

Omfanget av prosjektet har vist seg å være mer omfattende enn det vi så for oss før prosjektet startet. I oppstarten av prosjektperioden var vi litt bekymret for at omfanget på dette prosjektet ville bli i minste laget, da selve oppgaven med å lese, analysere og lagre en videostrøm kan gjøres relativt fort. Dette avsnittet vil gi et innblikk i hvordan omfanget på prosjektet ble større enn først beregnet, og hvilke faktorer som påvirket denne veksten.

9.2.1. Designvalg

Det ble det klart tidlig i prosessen at det var ønskelig å utvikle løsningen på en så generell måte som mulig, slik at den skulle kunne gjenbrukes til andre lignende formål. Fordelen med denne måten å løse oppgaven på er at det blir veldig enkelt å utvide eller skifte ut funksjonaliteten til programmet. Hvis man for eksempel ønsker å skifte ut den algoritmen som analyserer bildet for å finne lysfenomener, er det bare å skrive en ny modul som implementerer denne algoritmen, og sette opp programmet til å bruke denne nye modulen. Det som har vært en ulempe med denne løsningen er den økte kompleksiteten med å utvikle selve systemet, som har tatt lengre tid enn ønskelig.

9.2.2. Tekniske utfordringer

Underveis i prosjektforløpet har det vært endel tekniske utfordringer som vi har måttet forholde oss til og finne en løsning på, her følger en beskrivelse av de viktigste av disse.

9.2.2.1. Stabilitet på videostrøm

Etterhvert som prosjektet gikk fremover og vi begynte å forholde oss til videostrømmen ifra Hessdalen ble det klart at vi her hadde en utfordring når det gjaldt stabiliteten til denne strømmen. Det api'et som først ble brukt for å lese strømmen mistet forbindelsen til strømmen med jevne mellomrom og brukte lang tid på å koble til igjen. Dette problemet medførte at det var vanskelig å teste programmet vårt, da det ikke leste strømmen kontinuerlig lenge nok til at det ga noen mening. Det ble etterhvert tatt en avgjørelse på at vi skulle forsøke å bruke et annet api som hadde en mere "low level" kontroll over videostrømmen, selv om vi ikke var sikre på om dette ville medføre noen bedring . Valget medførte at vi måtte bruke ekstra tid på å sette oss inn i hvordan vi brukte et nytt api på et tidspunkt når denne funksjonaliteten burde vært ferdig utviklet.
Dette viste seg å bli en ganske betydelig utfordring da dette api'et var svært dårlig dokumentert og store deler av informasjonen som var å finne var utdatert. Senere ble det satt opp en streamingserver på høyskolen som virket som en proxy for strømmene, denne streamingserveren ble resatt en gang i døgnet som ble en ny utfordring for programmet som tok tid å finne noen god løsning på.

9.2.2.2. Hardware problemer

Da vi skulle sette opp serveren som skal kjøre programmet vårt hadde vi problemer med å få kompilert de bibliotekene som koden vår er avhengig av. Etter å ha tatt ned hele oppsettet og satt det opp på nytt uten noen bedring kom vi på å kjøre en test av minnet. Denne testen viste seg å feile og vi kjørte en del nye tester for å bekrefte at det var snakk om en hardware feil. Konklusjonen ble at vi hadde en ødelagt rambrikke som måtte byttes, etter å ha gjort dette fikk vi satt opp miljøet og kjørt programmet på serveren. Senere fant vi ut at selve prosessoren i serveren også ser ut til å ha problemer. Når vi skulle sjekke ut om prosessoren hadde to eller fire kjerner fant vi ut at prosessoren som egentlig skulle kjøre på 2,93Ghz var klokket ned til 1,6Ghz. Vi forsøkte oss på å skru opp hastigheten i bios til opprinnelig hastighet med det resultat at serveren frøs. Status i skrivende stund er at vi skrudde av en option i bios som skal justere hastigheten opp automatisk for så å forsøke 2,93Ghz på nytt. Hvorvidt dette er stabilt over tid vet vi ikke. Men den nedjusterte hastigheten er for lite til å kjøre programmet vårt på en tilfredsstillende måte, så vi ser oss nødt til å gjøre et forsøk.

9.2.2.3. Minnehåndtering

Siden programmet skal håndtere og bufre videostrømmer med høy oppløsning vil det hele tiden allokeres ganske store minneblokker, hvis ikke disse blir frigitt igjen vil programmet fort allokere alt tilgjengelig minne og krasje. I moderne språk som Java og C# gjøres denne minnehåndteringen automatisk, men i C++ som vi bruker må dette gjøres manuelt. Da funksjonaliteten til programmet begynte å komme på et slikt nivå at det var ønskelig å teste det over tid ble dette en problemstilling vi måtte forholde oss til. Minneforbruket økte kontinuerlig så lenge programmet kjørte, en klar indikator på manglende deallokkering av minne. Vi brukte ganske mye tid på å prøve å finne manglende deallokkering av minne uten å få kontroll på minneforbruket. Etterhvert ble vi enige om å lage en såkalt "object pool" som er en egen del av systemet som administrerer bruken av objekter. Den store fordelen med en slik pool er at den

gir mulighet til å registrere informasjon om hvor mange objekter som er i bruk til enhver tid, og hvilke moduler som bruker dem. Dette ga oss mer informasjon og gjorde at vi fant ut hvor problemene var slik at vi fikk rettet opp i dem.

10.Konklusjon

Programmet som er utviklet er nærmest som et modulært rammeverk for prosessering av videostrømmer. Bruksområdet for dette rammeverket er ikke begrenset til deteksjon av lysfenomen i Hessdalen, men det vil også kunne brukes i andre videoovervåkningssammenhenger. Et eksempel på et annet bruksområde er for eksempel videoovervåkning av en minibank hvor man kan lagre videoklipp når det skjer noe foran minibanken. Et slikt bruksområde innebærer også et mer veldefinert problem som er lettere å konfigurere og teste. Det som gjør det vanskelig å teste tiltenkt bruksområde er at det er veldig lite rammer for hva som skal defineres som et fenomen eller ikke. I tillegg har de tidligere nevnte utfordringene rundt stabiliteten på videostrømmen gjort det vanskelig å kjøre programmet kontinuerlig lenge nok til å få testet. Det gjenstår noe testing både på deteksjonsbiten og lagringsmodul før vi anser programmet for å være helt i mål.

11.Organisering og fremdrift

Prosjektperioden har hatt en fremdrift drevet av metoder og organisering i gruppa. I denne delen blir det først presentert hvordan gruppa har organisert seg i forhold til arbeid, møter, dokumentasjon og bruk av ulike verktøy. Deretter beskrives metodebruken i prosjektet.

11.1.Organisering

11.1.1.Møter

Prosjektet startet med en fase for planlegging hvor det ble satt opp en fremdriftsplan, og i møte med oppdragsgiver bestemt hva som skulle utvikles. Gruppa har hatt møter med veileder regelmessig hver 14.dag, hvor det har blitt en statusoppdatering for prosjektet og planlegging av arbeidet videre. Til disse møtene er det sendt ut møteinnkalling i forveien, samt at selve møtet er dokumentert med et referat.

I tillegg til disse møtene med veileder, har gruppa flere ganger i uka hatt korte statusmøter med hverandre. Disse møtene har enten foregått på skolen eller over internett ved hjelp av programvare for videokonferanse. Møtene har blitt brukt til å komme med korte oppdateringer om fremdriften, samt diskusjon og løsning på en del utfordringer som har kommet underveis. Møtene har ikke hatt en like fast struktur som møtene med veileder. Det er ikke skrevet referat fra disse møtene, da de ofte har vært svært korte og mer vært et verktøy for statusoppdatering.

Gruppa har vært i kontakt med oppdragsgiver flere ganger under prosjektperioden. Kontakten har bestått av samtaler med status på prosjektet, og for å ta opp utfordringer som har ført til større endringer i prosjektet. I tillegg har det vært sendt e-poster mellom oppdragsgiver og gruppa for å informere om mindre saker i prosjektet. Gruppa har selv ønsket å holde oppdragsgiver oppdatert, blant annet for å forebygge at resultatet som leveres ikke står til oppdragsgiver sine forventninger.

11.1.2.Arbeidsprosessen og dokumentasjon

Arbeidet med planlegging av prosjektet har gruppa for det meste gjort i arbeidsmøter på skolen. Utviklingsarbeidet har foregått både på skolen og hjemmefra, med hyppig bruk av statusmøtene nevnt ovenfor.

Under prosjektperioden valgte gruppa å ha fire ulike roller som rullerte, slik at alle fikk prøvd ut hver rolle. Prosjektperioden ble delt opp i fire deler, og rollene ble da rullert fort hver del som ble påbegynt. Rollene som ble brukt hadde hver sine oppgaver, og er følgende:

- **Prosjektleder**: styrer møtene og oppfølger oppgavene gruppen jobber på.
- **Sekretær**: skriver møtereferater og sender dem til resten av gruppen, innkaller gruppen til møter og oppdaterer prosjektbloggen ukentlig.
- **Koordinator:** sørger for at prosjektstyringsverktøy er oppdatert og legger inn tickets.
- **Konsulent:** reserverer grupperom når det er behov.

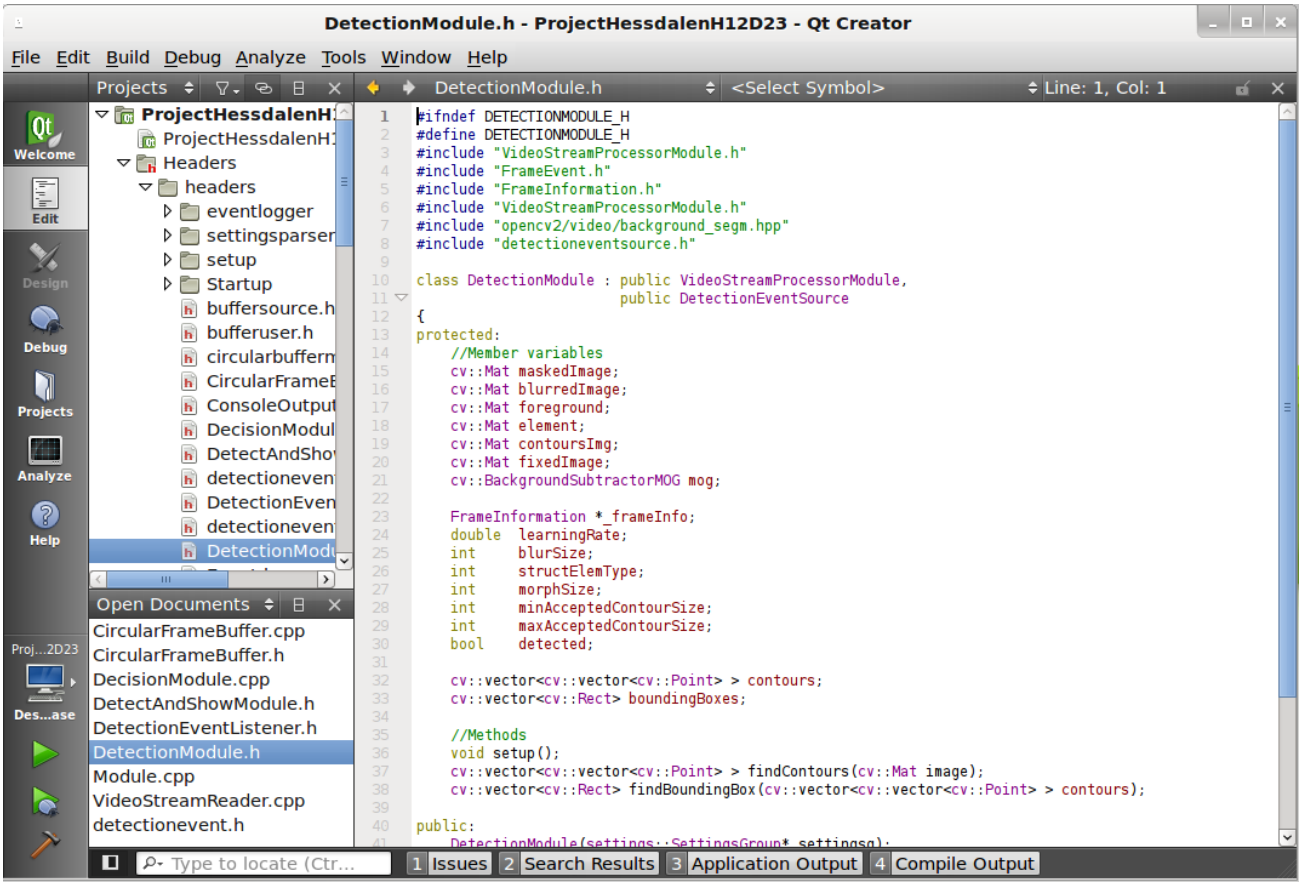
Alle i gruppa har hatt ansvar for å dokumentere sitt arbeid ved hjelp av timeliste og dagbok. I starten av prosjektet begynte vi å bruke et ticketsystem for at hver enkelt skulle dokumentere sitt arbeid. Etterhvert som utviklingsperioden var i gang, og Git ble brukt som versjonskontroll, gikk vi over til å se på kommenteringa av arbeidet i Git som god nok dokumentasjon i tillegg til

timelista. Ulempen med å måtte føre tickets i tillegg var at det ble så mye duplisering av det samme i ulike systemer. Timelista ligger med som vedlegg til rapporten.

Gruppa har tatt utgangspunkt i de milepælene som ble satt opp i forprosjektrapporten, men underveis justert noe i forhold til at enkelte ting som tok lengre tid enn planlagt.

11.1.3. Verktøy

I prosjektet er det benyttet ulike verktøy for utvikling og dokumentasjon. C++ er språket som programmet er skrevet i, med bruk av rammeverket Qt. For å gjøre utviklingen av C++ mer behagelig ble det fra starten bestemt at man ønsket å bruke IDE. Ulike IDE'er ble testet ut og valget falt på Qt Creator, som også integrerer rammeverket Qt.



Figur 18: Screenshot av Qt Creator

GIT har blitt benyttet som versjonskontroll i prosjektperioden, hvor utviklingen av koden underveis har blitt dokumentert med kommentarer. OpenCV og FFmpeg er rammeverk brukt til bildebehandling og videostrømlesing i programmet. Foruten disse verktøyene har gruppa også valgt å benytte Doxygen som program til å generere dokumentasjon av kildekoden direkte fra filene i programmet.

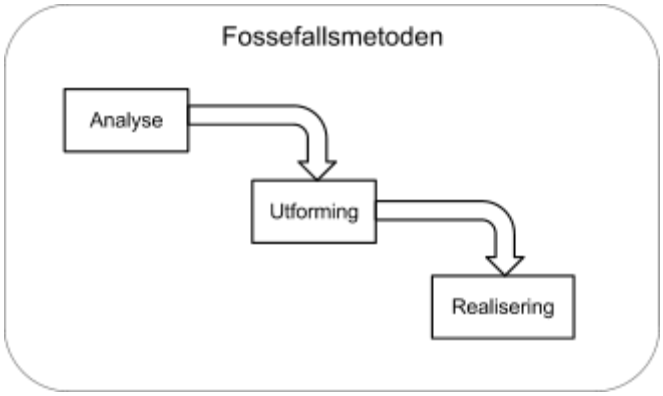
Google Docs sine verktøy er benyttet til dokumentasjon, illustrering og rapportskriving.

11.2.Utviklingsmetoder

For å beskrive metodebruken i prosjektperioden er det nødvendig å trekke fram flere ulike metoder. Studentene i prosjektgruppa hadde ikke noen stor erfaring med utviklingsmetoder før prosjektet, og dette har dermed før til at prosjektet preges av en blanding fra flere metoder. Videre er det skrevet en del for hver at metodene som har påvirket prosjektperioden, og litt om hvordan ulike elementer er brukt fra disse.

11.2.1.Fossefall

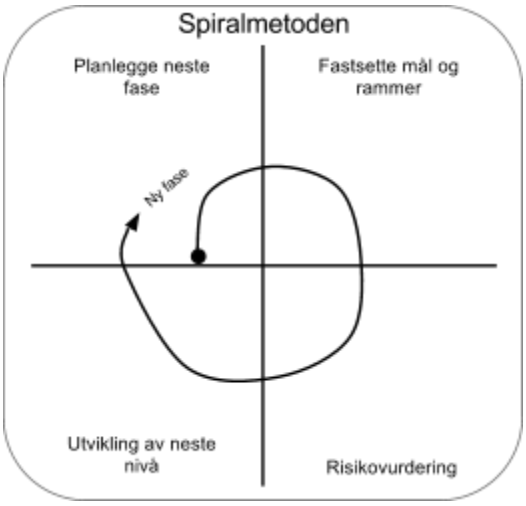
I starten av utviklingsperioden, etter at forprosjektet var over, begynte man direkte med en overordnet planlegging av systemet. En slik måte å gjøre det på er hentet fra fossefallsmetoden hvor resultatet planlegges ferdig i første fase. Grunnen til at gruppa valgte å legge en overordnet plan for utviklingen var for å danne seg et mer konkret bilde for hva som skulle utvikles, siden rammene gitt av oppdragsgiver ikke var så detaljerte. Dermed ble planleggingsfasen svært naturlig å bruke en del tid på i starten.



Figur 19: Illustrasjon av fossefallsmetoden

11.2.2.Spiral

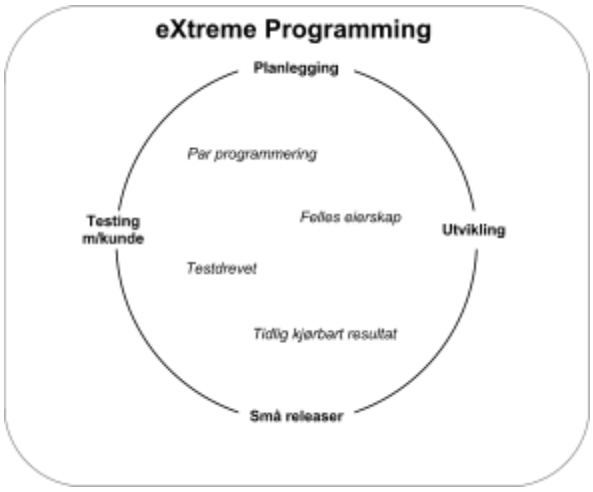
Etter at planleggingen var over gikk gruppa over i en fase preget av mer konkret utvikling. Denne perioden var preget mer av spiralmetoden enn fossefallsmetoden. Her tok gruppa tak i mindre deler av prosjektet for å planlegge, utvikle og vurdere dette. Endringer ble gjort mer direkte enn hva man har mulighet for i fossefallsmetoden, og det var under hele perioden en stor fokus på å bygge et system som enkelt kunne endres uten at for mye ble påvirket.



Figur 20: Figur av spiralmetoden

11.2.3. Agile metoder

Regelmessig under hele utviklingsperioden har studentene hatt statusmøter. Hyppigheten har vært mellom 1 til 3 dager mellom hvert møte, alt etter hva som har passet best i forhold til oppgavene det jobbes med. Disse møtene har vært preget av at hver enkelt i gruppa har informert de andre om hva som er gjort siden sist, hva som skal gjøres til neste gang og problemer som har kommet underveis. Disse statusmøtene har ofte vært i muntlig form via kommunikasjonsprogrammet Skype, og ligner mye på møtene som brukes i Scrum.



Figur 21: Konsepter ved eXtreme Programming

Oppdragsgiver som kan defineres som "kunden" i prosjektet, og har underveis blitt oppdatert gjennom e-post og korte uformelle møter gjennom prosjektperioden. På den måten har studenten gitt informasjon til oppdragsgiver ved store utfordringer i prosjektet som har ført til ulike endringer. Det var tidlig et ønske og mål blant gruppa om å få opp en kjørbar versjon av systemet for både testing og for at oppdragsgiver og veileder kunne følge med underveis. Dette er et viktig fokus innen Agile metoder. På grunn av store utfordringer underveis klarte ikke studenten å få til en kjørbar versjon av programmet så tidlig som man hadde håper på.

11.3.Erfaring til senere prosjekter

Ved å ha tatt i bruk flere av komponentene fra ulike kjente metoder har utviklingsperioden vært preget av en fast struktur som på mange måter har hjulpet til med framgangen i prosjektet. Derimot ser og erfarer man i ettertid at det kunne ført til enda bedre struktur og framgang om gruppa holdt seg mer til en bestemt metode, hvor en av de Agile metodene antageligvis hadde vært valgt. Etter å ha fått mer erfaring fra større utviklingsprosjekter stiller man sterkere til å velge bruk av metoder til å styre utviklingsprosessen.

12.Begrepsliste

I denne delen beskrives en del av begrepene i rapporten nærmere, for at leseren skal kunne få hjelp underveis til å forstå ukjente ord og uttrykk.

- **Frame:** Et enkeltbilde tatt ut fra en videostrøm.
- **Event:** Brukes i forbindelse med signaler som sendes mellom modulene, da i forbindelse med DetectionEvent eller FrameInformationEvent, les mer under informasjonsflyt i systemet. Finnes også et objekt med navnet "EventLogger" uten at det har noen tilknytning til kommunikasjonen mellom modulene. Dette er klassen som brukes for å logge hendelser i programmet til fil.
- **FrameInformation:** Objekt som inneholder et bilde med tilhørende informasjon. Står nærmere forklart i systembeskrivelsesdelen.
- **Lysfenomen:** Synlig ukjent lys som opptrer uten at det kan forklares.
- **FFmpeg:** Rammeverk for video- og audiobehandling.
- **OpenCV:** Rammeverk for bildeanalyse.
- **Hessdalsfenomenet:** Navnet på det ukjente lysfenomenet som opptrer i Hessdalen, og er fenomenet programmet har til formål å oppdage. Les mer i bakgrunn.
- **Konfigurasjonsfil(konfigfil):** Fila som sendes inn til programmet som parameter. Programmet bygger opp sitt oppsett for gjellende kjøring basert på innholdet i denne fila. Uten denne fila, eller hvis fila er tom, vil ikke programmet være kjørbart.
- **Modulært:** Basert på moduler.
- **Modul:** En utskiftbar del i systemet. Finnes i ulike typer, hvor hver type har sin oppgave. Benyttes i konfigurasjonsfila for å bestemme oppsettet til programmet.
- **Kjede:** Brukes for å illustrere hvordan modulene settes opp i systemet ut fra konfigurasjonsfila. En kjede er to eller flere moduler som henger sammen, altså kommuniserer med hverandre.
- **Videostrøm:** en video som blir sendt gjennom et nettverk. En videostrøm består av flere bilder (frames).

13.Referanse- og litteraturliste

Bjarne Stroustrup's FAQ (2012) URL: http://www2.research.att.com/~bs/bs_faq.html#invention *(Lesedato: 16.04.2012)*

Bow den, R., P. Kaew TraKulPong (2001). An Improved Adaptive Background Mixture Model for Real-time Tracking w ith Shadow Detection

Bro, Jonas, Geir Eilertsen, Magnus Skalsveen (2010). Prosjekt Hessdalen AMS2010
URL: http://frigg.hiof.no/h10d09/filer/prosjektinnlevering/[h10d09]Prosjektrapport.pdf *(Lesedato 17.04.2012)*

Skagestein, Gerhard (2005) Systemutvikling - fra kjernen og ut, fra skallet og inn 2.utgave

Smith, W. Steven(1997) The Scientist and Engineer's Guide to Digital Signal Processing

Gonzales, Rafael C, Richard E. Woods (2002). Digital Image Processing

Grimson W. E. L, Chris Stauffer (1999). Adaptive background mixture models for real-time tracking. in Proceedings.

Haas, Juergen. Definition: Modular programming, About.com. URL: http://linux.about.com/cs/linux101/g/modularprogramm.htm *(Lesedato: 17.04.2012)*

How to create qt plugins URL: http://qt-project.org/doc/qt-5.0/plugins-howto.html (Lesedato 18.04.2012)

Lew allen, Raymond (2005). Advantages of an Object-Oriented Approach (for new programmers).
URL: http://codebetter.com/raymondlew allen/2005/02/08/advantages-of-an-object-oriented-approach-for-new-programmers/ *(Lesedato: 17.04.2012)*

Sommerville, Ian (2007) Softw are Engineering 8. edition

Strand, Erling (1984). Project Hessdalen 1984 - Teknisk hovedrapport
URL: *http://www.hessdalen.org./rapporter/hprapport84.shtml (Lesedato 16.04.2012)*

Strand, Erling (2002). Hessdalfenomenet
URL: *http://www.hessdalen.org./rapporter/PH-artikkel032002.pdf (*Lesedato 16.04.2012)

Using Intel® IPP w ith OpenCV URL: http://software.intel.com/file/33161

Using the Meta-Object Compiler (moc) URL:http://qt-project.org/doc/qt-4.8/moc.html (Lesedato: 18.04.2012)

Wikipedia, Deadlock (2012) URL: http://en.w ikipedia.org/wiki/Deadlock (Lesedato: 16.04.2012)

Wikipedia, Race condition (2012) URL: http://en.w ikipedia.org/wiki/Race_condition (Lesedato 16.04.2012)

Wikipedia, Object-oriented Programming (2012) URL: http://en.wikipedia.org/wiki/Object-oriented_programming *(Lesedato: 17.04.2012)*

Wikipedia, Modular Programming (2012) URL: http://en.wikipedia.org/wiki/Modular_programming (Lesedato: 17.04.2012)

Wikipedia, Scrum URL: http://no.w ikipedia.org/wiki/Scrum *(Lesedato: 18.04.12)*

14.Figurliste

15. Vedlegg

- Brukerveiledning
- Veiledning for utvikling av moduler
- Webgrensesnitt – teknisk beskrivelse
- Bruk av TeamViewer
- Timeliste

Bruksanvisning for TeamViewer

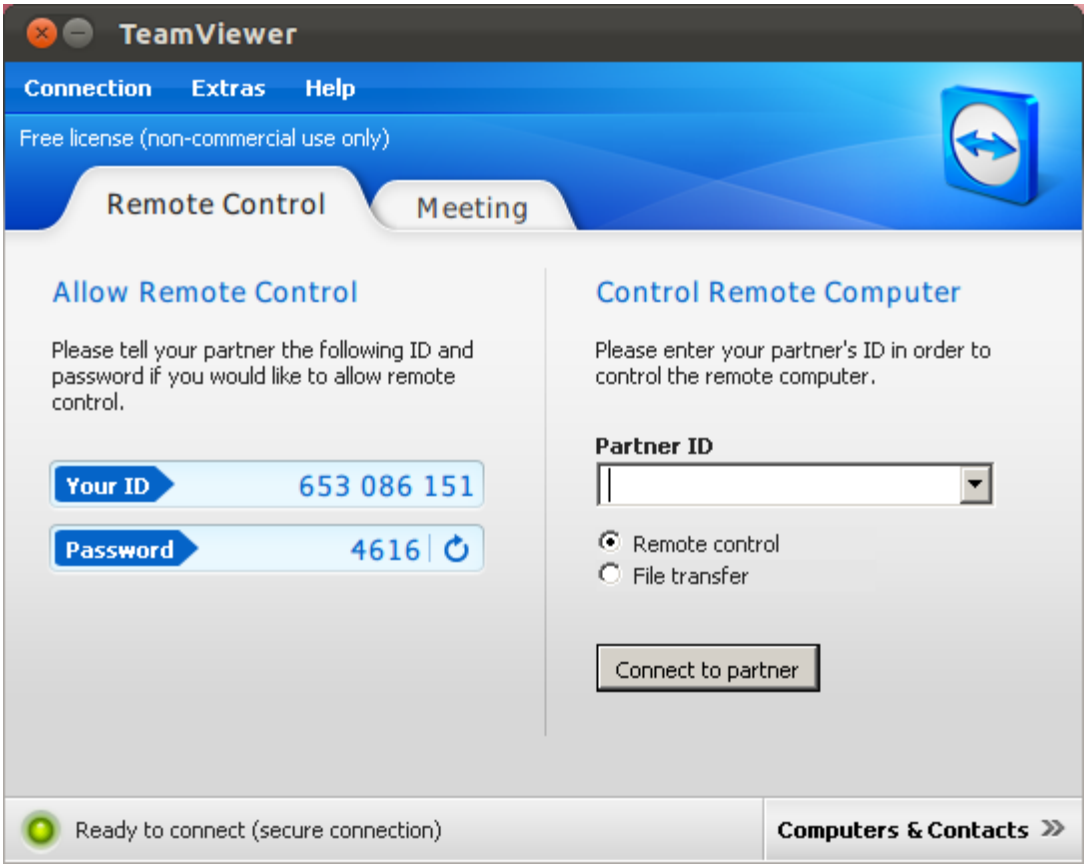
TeamViewer er et verktøy for fjernstyring, skrivebordsdeling, nettmøter, webkonferanser og filoverføring mellom datamaskiner. I dette prosjektet har TeamViewer vært brukt for å styre serveren med programmet som befinner seg på skolen fra en annen pc. Programmet testes og kjøres ofte på serveren, så derfor er det viktig å alltid ha tilgang til den. TeamViewer forenkler denne oppgaven svært mye, fordi man da slipper å sitte direkte på pcen.

Hvordan bruke TeamViewer?

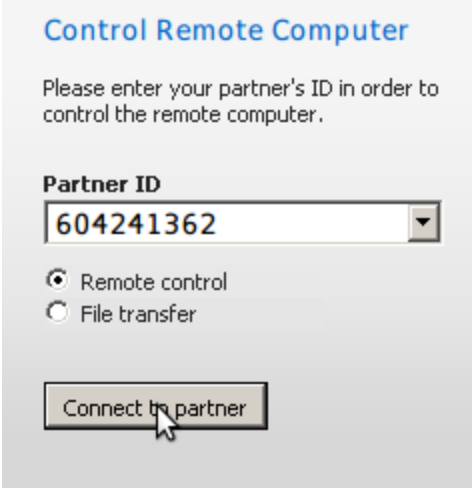
Først trenger man å laste ned TeamViewer fra linken nedenfor:
http://www.teamviewer.com/no/download/

Programmet er multiplattform, så det er mulig å få en versjon for ønskede operasjonssystem, f. eks Windows, Linux, MacOS, eller Android.

Etter å ha lastet ned, installert og startet programmet, får brukeren et slikt vindu:

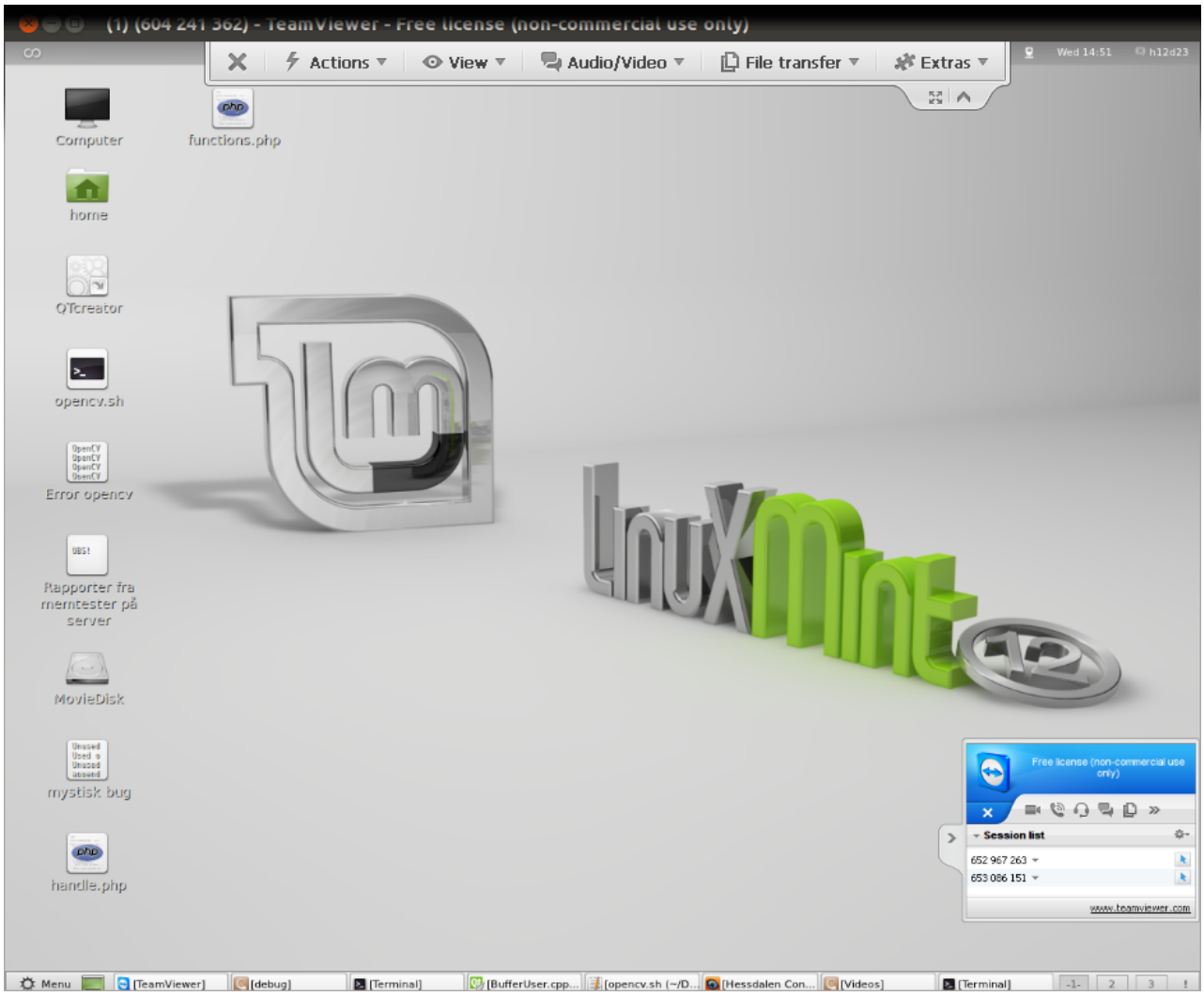


Til venstre er det ID og passord som kan benyttes hvis brukeren ønsker å dele skrivebordet sitt med andre. Hvis brukeren ønsker til å koble til et skrivebord som er delt fra et annet sted, må brukeren angi partneres (hvem det skal kobles til) ID, passe på at “Remote control” er valgt og trykke “Connect to partner”. I tilfelle med serveren, er ID **604241362**:



Så får brukeren en dialog hvor passordet skal skrives inn, passordet til serveren er **ufo007Hessdalen**.

Etter å ha logget inn, kommer det et nytt vindu med skrivebordet til den eksterne datamaskinen. Det er viktig å vite at det er ikke en privat sesjon, og alt som gjøres foregår også på datamaskinen i sanntid. Så dersom maskinen allerede brukes, er det best å vente til den andre personen er ferdig.



Bildet øverst viser det eksterne skrivebordet. På toppen befinner seg et panel med forskjellige innstillinger og verktøy til TeamViewer. Der kan brukeren f. eks. endre oppløsning eller kvalitet hvis koblingen er sakte. Nederst til høyre er det et panel som viser alle brukere (og deres ID) som er koblet til skrivebordet akkurat nå. Der er det mulig å ha en samtale eller sende over filer.

NB: På serveren vil maskinen være låst om ingen andre brukeren, passordet for å låse opp dette er samme som ved pålogging. Når brukeren er ferdig med arbeid på det eksterne skrivebordet, er det viktig å låse skjermen slik at datamaskinen er trygg under passordet. Dette er spesielt viktig om man benytter TeamViewer på serveren fordi den står i et rom hvor mange andre har tilgang, og om pcen ikke låses vil andre i samme rom kunne få adgang til den. Låsing av serveren gjøres i menyen oppe til høyre som heter “h12d23”, og man velker “Lock screen”.

Web-grensesnitt - teknisk beskrivelse

Dette vedlegget beskriver funksjonaliteten i webgrensesnittet ved å se på det tekniske. Her vil ulike metoder bli beskrevet. Brukerveiledning for webgrensesnittet er lagt i samme dokument som brukerveiledningen til programmet.

Funksjoner

Ved hjelp av web-grensesnittet er det mulig å:

- lage nye konfigfiler (navn til filen kan oppgis av brukeren)
- lage nye moduler i en konfigfil (brukeren velger type av modulen fra en liste)
- redigere konfigfiler (altså innstillinger til moduler inn i konfigfilene)
- lagre eksisterende konfigfiler med andre navn
- starte programmet

Teknisk beskrivelse

I denne kapittelen skal web-grensesnittets virkemåten beskrives.

Teknologi brukt

- PHP
- JavaScript
- Ajax
- jQuery
- CSS
- HTML

Filer og kataloger

index.php - hovedsiden
functions.php - inneholder hovedfunksjoner av grensesnittet
handle.php - filen som håndterer forespørlser

config.js - hoved JavaScript funksjoner
jQuery.js - jQuery JavaScript-bibliotek

style.css - sidens stil

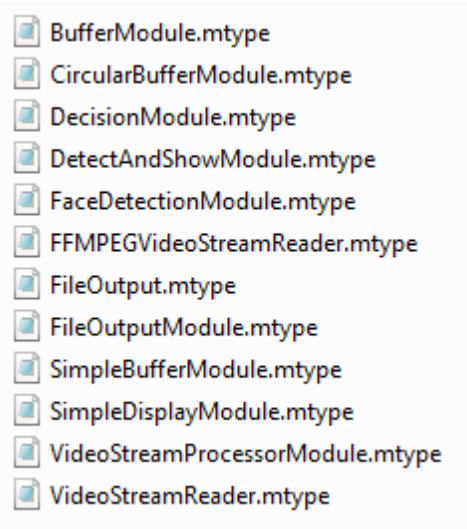
configs - katalogen som inneholder konfigfiler
modules - katalogen som inneholder modulenes hjelpefiler

Virkemåten

Når brukeren åpner hovedsiden, lager et PHP-script i *index.php* en liste med konfigfilene som finnes i *configs* katalogen ved å bruke **getFileList()** funksjonen fra *functions.php*. Denne funksjonen tar navn til en mappe som argument og returnerer en liste med filer i den angitte mappen. Når brukeren velger en konfigfil fra listen, kjører **selectConfig()** funksjonen i *configs.js* som bruker Ajax for å sende en forespørsel med navnet til konfigfilen til *handle.php.* Så sjekker scriptet hva slags data ble mottatt, og i dette tilfellet er det kun navnet til konfigfilen. Da kjøres det 3 funksjoner:

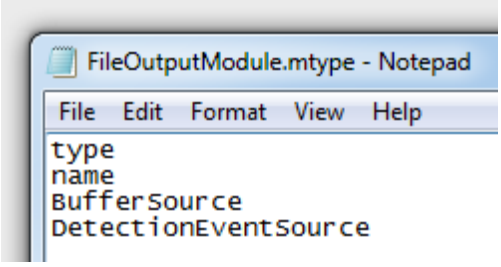
getModuleTypes()

Denne funksjonen parser hjelpefiler til modulene for å definere hvilke innstillinger hver modul må ha. Den tar navn til mappe med hjelpefilene som argument som brukes i **getFileList()** funksjonen for å få liste med alle hjelpefilene i mappen.



Noen hjelpefiler i modules mappen.

Scriptet går gjennom hver linje i hver fil som kan se slik ut:

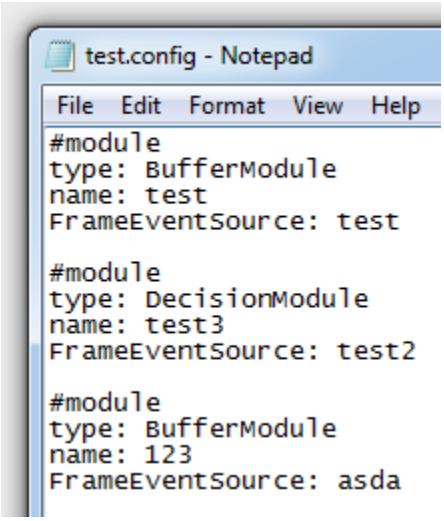


Her er hver linje en setting som må være i gjeldende modul. For hver modultype opprettes det et **ModuleType** objekt som har navnet til filen som eget navn, men uten katalogen eller filendelsen. Hvis et objekt med slikt navn allerede eksisterer, opprettes det ikke et nytt objekt. Så er hver setting (altså hver linje i filen) lagt til objektes sin array. Disse innstillingene kan

hentes ved å bruke **ModuleType** objektes funksjon **getOptions()**, som returnerer en liste med nødvendige innstillinger for gjeldende modultype. Når løkken er ferdig med å legge til innstillinger til et objekt, legges objektet til **moduleTypesArray** lista.

getSettings()

Denne funksjonen parser en angitt konfigfil og oppretter et objekt for hver modul som finnes i konfigfilen.



Eksempel på en konfigfil.

getSettings() tar gjeldende konfigs navn og en array med modultyper som argumenter. **moduleTypesArray** ble oprettet av **getModuleTypes()** funksjonen og navn til konfigfilen ble sendt til *handle.php* fra *index.php* via JavaScript og Ajax. Alt som finnes mellom to nøkkelord (#module) lagres til et **Module** objekt. Hvis et objekt med navnet til en modul ikke finnes, opprettes det et nytt objekt. Så skiller funksjonen linjens innhold på kolontegnet ": ", og legger de to nye strengene til objektes midlertig i en assosiativ array. For eksempel for "FrameEventSource: test", vil FrameEventSource bli key og test bli value i den assosiative arrayen. Den arrayen brukes til å sjekke om modulen har alle nødvendige innstillinger før alt er lagt til til en "endelig" array. Det gjøres i objektets **addSetting()** funksjonen. Den tar en array med modultyper som argument og sjekker om den midlertige arrayen har alle nødvendige innstillinger til gjeldende modultype. Hvis både innstilling og verdi er i arrayen, legges det til til objektes "endelig" array. Hvis det kun finnes innstilling uten at noen verdi er satt, legges instillingen til den endelige arrayen med "UNDEFINED" som verdi. Når løkken er ferdig med en modul, legges det modul objektet til en array som inneholder alle moduler for gjeldende konfigfil (**moduleArray**).

printHTML()

Nå inneholder **moduleTypeArray** og **moduleArray** nødvendige objekter, og objekter inneholder nødvendige innstillinger og verdier. **printHTML()** funksjonen skriver ut side med informasjon om alle modulene i gjeldende konfigfil (hvor man kan redigere innstillinger og lagre

moduler). Siden gir også brukeren mulighet til å opprette en ny modul, og en dialog som gir mulighet til å lagre konfig med et forskjellig navn. Alle disse sidene er først skjult ved hjelp av CSS og kan bli vist når brukeren trykker en gjeldende knapp, ved å kjøre **showDiv()** funksjonen i *config.js,* som viser og skjuler nødvendige sider (divs).

Opprettelse av moduler

Når brukeren benytter Create knappen i Create a module fanen, kjøres det **request()** funksjon i *config.js* som tar navnet til formen som argument. Så sendes dataen fra formen til *handle.php* hvor det kjøres **createModule()** funksjonen som tar navnet til konfigfilen, og type og navn til modulen som argumenter. Den skriver modulen rett til konfigfilen med #module nøkkelord uten å kjøre noen andre funksjoner. Så neste gang konfigfilen er parset (ved å trykke View the config f.eks), er modulen i liste med alle innstillingene som UNDEFINED, siden man bare kunne gi navnet til modulen og velge en type når modulen ble opprettet.

Lagring av separate moduler

Når brukeren trykker på Save Module knappen, kjøres det en **request()** funksjon i *config.js* med modulens navn som argument. Navnet til modulen er også navnet til formen for denne modulen. Det gir mulighet til å kjøre **serialize()**, som er en jQuery funksjon hvor data samles fra en form og returnerer en query string. Den strengen er videre sendt til *handle.php* hvor strengen er skilt til value og key og sendt til **editModuleSettings()** funksjonen fra *functions.php* som argumenter. Den bruker tidligere beskrevet funksjoner for å legge til ny data til Module objektet -- **addTempSettings()** og **addSettings()**. Videre kjøres det **writeModulesToFile()** funksjonen, som åpner ønskede konfigfil, sletter alt data som var der, og videre kjører en foreach for hver modul i **moduleArray** hvor det brukes Module objektes funksjon **writeToFile()** som skriver gjeldende modul til filen.

Lagring av alle moduler

For å lagre alle modulene samtidig, brukes det samme funksjoner, men de er kjørt i rekke etter hverandre ved hjelp av Ajax og JavaScript **saveForms()** - funksjonen i *config.js*.

Brukerveiledning for Programmet

Dette er en brukerveiledning for programmet som er utviklet av gruppe H12D23 ved Høyskolen i Østfold, våren 2012.

Programmet er utviklet til analyse av videostrømmer for å finne ukjente lysfenomener. Ved funn vil programmet lagre videoklippet. Det er mulig å endre ulike innstillinger i en konfigurasjonsfil som følger med programmet.

Systemet er bygget på modulær tankegang. Dette gir muligheter for å konfigurere programmet på mange ulike måter. Videre vil bruken av programmet bli forklart mer detaljert, sammen med noen praktiske eksempler. Først blir det beskrevet ulike konfigurasjonsmuligheter til programmet, deretter litt om hvordan man starter opp programmet. Til sist presenteres et webgrensesnitt som kan brukes som en alternativ måte å starte programmet på eller endre konfigurasjonsfila.

Konfigurasjon av programmet

Hele programmet baserer sitt oppsett på konfigurasjonsfila, dette gir muligheten til å kjøre et svært ulikt oppsett fra gang til gang. I denne delen går man inn på hvordan en slik fil settes opp.

Hver modul har sine egne innstillinger som brukeren endrer ved hjelp konfigurasjonsfila som benyttes i programmet. Filene kan skrives og endres enten ved hjelp av en texteditor (som f.eks. Notepad, gedit, eller Notepad++), eller ved hjelp av en webgrensesnitt (les mer om webgrensesnittet senere i veiledningen). For å spesifisere at det er en konfigurasjonsfil benyttes endelsen .config på fila, for eksempel “MinKonfig.config”.

En modul i konfigurasjonsfila blir angitt ved at man starter med en *#module* tag. Deretter kommer påfølgende innstillinger man ønsker å sette til modulen. Hver innstilling består av et par med key og value, hvor key angir typen innstilling og value angir verdien. Det kan være en god vane å avslutte en modulblokk med *#end* for leslighetens sin del, men er ikke noe krav fra programmets side.

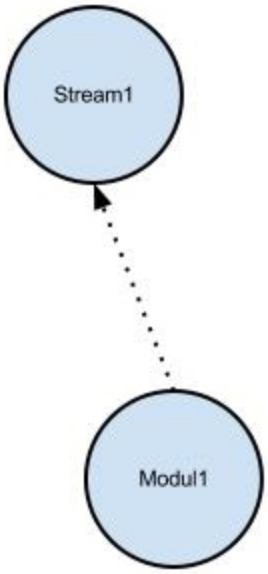
Viktig: Alle modulene krever at innsillingene *name* og *type* er satt.

Her kommer et eksempel på hvordan en modul kan settes opp:

```
#module
type: DecisionModule
name: decision1
FrameEventSource: stream1
#end
```

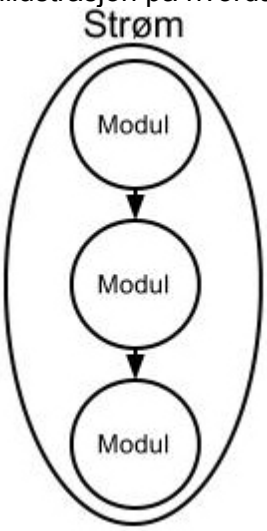
En modul kan være kilde(source) og/eller lytter(listener) i forhold til en eller flere andre moduler. Dette vil være måten modulene kommuniserer med hverandre på. Det vil være forskjellig fra modul til modul hva som takles, derfor anbefaler man å se på moduloversikten lengre ned i dokumentet for å finne ut hva modulene støtter. Kommunikasjonen skjer ved hjelp av signaler, og det finnes derfor ulike former for signaler. Denne kommunikasjonen settes opp i konfigurasjonsfila som en eller flere innstillinger til en modul.

I eksempelet over er “FrameEventSource: stream1” en innstilling som sier at modulen skal være en kilde som har modulen med navnet “stream1” som kilde til å motta FrameEvent signaler fra. Modulen som er angitt vil altså registrere seg som lytter hos “stream1” modulen. Dataflyten vil dermed gå fra “stream1” til “decision1”. Figuren under illustrerer konseptet.



Figur: “Modul1” henviser til “Stream1” som sin foreldre. Stiplet pil viser at “Modul1” i configfilen henviser til “Stream1”, og står dermed i motsatt retning i forhold til hvordan dataflyten vil gå.

Når modulene kobles sammen i forhold til kommunikasjon kan man se for seg at det vil danne en kjede. En slik kjede vil starte med en modul som kalles rot, og naturlig vil være en modul som leser videostrøm. Det vil også være naturlig å kalle hele kjeden for en strøm, siden det er utgangspunktet. Figuren under gir en illustrasjon på hvordan det kan bli.



F*igur: En strøm består av flere moduler, som kan sees på som en kjede.*

Signaltyper

I systemet finnes det to ulike typer av signaler(eventer) som er FrameInformationEvent og DetectionEvent. Videre blir disse beskrevet, samt noen eksempler på praktisk bruk.

FrameInformationEvent

Dette signalet benyttes for å si ifra om at det finnes et nytt bilde er tilgjengelig hos en modul. De fleste modulene i systemet kan både motta og sende slike signaler.

DetectionEvent

Enkelte moduler støtter detectionsignaler. Disse signalene brukes for å si ifra om det er funnet noe lysfenomen eller ikke. Modulene av typen “Decision” og “FileOutput” kommuniserer med denne type signaler for å avgjøre når film skal lagres til fil.

Praktiske eksempler

Bruken av signaler er lik uavhengig av hvilken type det er. Hvis to moduler skal kommunisere med et spesielt signal må modulen avsendermodul støtte signaltypen som output, og mottager støtte signaltypen som input. I beskrivelsen av hver modul, lenger ned i denne veiledningen, vil det være en oversikt over hva som støttes. Hvis to moduler settes etter hverandre og kildemodul sender DetectionEvents som lyttermodulen ikke støtter å ta imot, vil disse signalene bare ignoreres. Programmet vil i en slik situasjon kjøre som vanlig, men detectionsignalene vil stoppe opp etter kildemodulen.

For å sette opp to moduler til å kommunisere med hverandre i konfigurasjonsfila angir man hvilken modulen som skal lytte til en annen på følgende måte: <EventType>Source: <navn på modul> Da vil modulen hvor dette settes bli registrert som lytter hos den modulen som har angitt navn.

For eksempel en modul med navnet “LytterModul” ønsker å lytte til FrameInformationsignaler fra “KildeModul”. Da settes skriver man følgende innstilling i konfigurasjonsfila til “LytterModul”: *FrameEventSource: KildeModul.*

Liste over moduler

Her følger en kort beskrivelse av de modulene som er tilgjengelig i systemet, for en mer detaljert beskrivelse av hvordan man bruker hver enkelt modul henvises det til vedlagt kodedokumentasjon. Overskriftene angir hvilken type modul som blir angitt, og er da det som skal brukes som *type* når man setter opp en modul i konfigurasjonsfila.

DecisionModule

Denne modulen avgjør om et lysfenomen har intruffet. Det avgjøres basert på data genert fra andre moduler.

Input:
FrameEvent

Ouput:
FrameEvent / DetectionEvent

Innstillinger:

- Limit: En grenseverdi som brukes iforhold til hvor følsom detekteringen skal være. Angis

fra 0-100.
- FrameEventSource: Hvilken modul som leverer en FrameEvent til denne.

DetectAndShowModule

Denne modulen analyserer et bilde og finner

Input:
FrameEvent

Ouput:
FrameEvent

Innstillinger:
- history: Denne instillingen angir hvor stor historikk som brukes for BackgroundSubtractor algoritmen, her har vi satt den til å bruke en historikk på 30 frames, som kombinert med en framerate på 15 frames per sekund gir en to sekunds historikk.
- nmixtures: Dette parametret har vi rett og slett mangelfull forståelse for og henviser til beskrivelsen i opencv dokumentasjonen: “Number of Gaussian mixtures”.
- backgroundratio: Dette parameteret er en faktor som angir hvor “følsom” algoritmen er før den tolker noe som forgrunn i bildet, lavere tall angir høyere følsomhet, som gir flere feildeteksjoner.
- learningrate: Learningrate har med hvor fort algoritmen tilpasser seg til endringer i bildet. Høyere verdier kan medføre at et objekt som beveger seg langsomt tolkes som bakgrunn. Grunnen til dette er at algoritmen rekker å tolke det som statisk før det har beveget seg nok til å slå ut som forgrunn.
- blursize: Angir størrelsen på masken som brukes når bildet som skal analyseres blir medianfiltrert. Høyere verdier vil gjøre at støy påvirker deteksjonen i mindre grad, men krever også mye av cpu.
- noisesigma: Dette parameteret er heller ikke forstått fullt ut, men det har med hvordan algoritmen håndterer støy å gjøre. Etter testing satte vi denne verdien til 5 som så ut til å være et godt utgangspunkt.
- morphsize: Størrelsen på strukturelementet som benyttes når det kjøres morphologisk lukning. Høyere verdier vil gjøre at detekterte objekter kan være lenger ifra hverandre før de blir kombinert til ett objekt.
- minacceptedcontoursize: Hvor stort et detektert objekt må være målt i antall piksler før det blir beregnet som et objekt.
- maxacceptedcontoursize: Maks størrelse for et detektert objekt målt i piksler før det blir ignorert.
- FrameEventSource: Hvilken modul som leverer en FrameEvent til denne.
- maskImage: filnavnet som skal brukes som maske.

FileOutputModule

Denne modulen lagrer en videofil ut fra de bildedata den får inn.

Input:
DetectionEvent

Ouput:
FrameEvent

Innstillinger:

- BufferSource: Hvilken modul som er en buffersource for denne.
- writeBufferSize: Antall frames som skal lagres i mellombufferen for disk i/o
- timeBefore: Antall sekunder som skal lagres før deteksjonen starter.
- DetectionEventSource: Hvilken modul som leverer DetectionEvent til denne.
- filePath: stien til mappen der filmene skal lagres.
- height: høyden på videostrømmen (i pixler)
- width: bredden på videostrømmen (i pixler)

FlightRadarModule

Denne modulen brukes sammen med en kinetic sbs-3 flyradar. Dette krever at programmet «Basestation» som leveres med denne flyradaren kjører i bakgrunnen. Modulen henter data fra dette programmet gjennom telnet.

Input:
FrameEvent

Ouput:
FrameEvent

Innstillinger:

- server: ip som “basestation”-programmet sender telnetdata til.
- serverport: Porten telnetserveren bruker, denne er default 30003

SimpleBufferModule

Fungerer som et mellomlager for frameinformation-objekter i minnet.

Input:
FrameEvent

Ouput:
FrameEvent

Innstillinger:
FrameEventSource: Hvilken modul som leverer en FrameEvent til denne.
fps: Antall bilder per sekund på videostrømmen
seconds: hvor mange sekunder med video som skal bufres

SimpleDisplayModule

En enkel modul som viser videostrømmen

Input:
FrameEvent

Ouput:
FrameEvent

Innstillinger:

- FrameEventSource: Navnet på modulen som leverer FrameEvent

FFMPEGVideoStreamReader
Dette er modulen som har ansvar for å lese fra videostrømmer. Den henter et bilde, genererer et FrameInformation-Objekt og kaller så andre moduler som har registrert seg som en FrameEventListener. Dette er den eneste modulen i systemet som ikke lytter på events.

Input:

Ouput:
FrameEvent

Innstillinger:
transport: protokollen som streamingserveren bruker (tcp eller udp).
printInfo: (true/false) dersom denne er satt til true vil en frameid legge på bildet.
source: Adressen som videodata skal hentes fra. (rtsp://adresse eller file://filenavn)

Oppstart av programmet

Programmet som er utviklet ligger i filstien /home/h12d23/HessdalenVideoAnalyzer, og heter ProjectHessdalenH12D23. For at programmet skal kjøre kreves det i samme mappe ligger en mappe med navnet “logs”, som igjen inneholder mappen “debug”. Det er til disse mappene loggfiler og debugfiler blir lagret. Illustrasjonen under viser mappestrukturen sett fra mappa hvor programmet ligger. I illustrasjonen ser man også mappen “images” som inneholder filen “mask.png”, dette kreves kun om man bruker deteksjonsmoduler som detectandshowmodule.

```
h12d23@h12d23 ~/HessdalenVideoAnalyzer $ ls -R
.:
detectandshow2.config  images  logs  ProjectHessdalenH12D23

./images:
mask.png

./logs:
debug

./logs/debug:
```

Oppstart av programmet er tilpasset at man skal kunne gjøre via terminalvinduet, eller over ssh

om man ønsker å starte det uten å sitte foran serverpcen. Kommandoen som kjøres for å starte programmet vil fra rotmappa være følgende:
./ProjectHessdalenH12D23 <konfigurasjonsfil>
Hvor *<konfigurasjonsfil>* byttes ut med stien til den fila som programmet skal settes opp med. Har man konfigurasjonsfila "detectandshow2.config" i samme mappe, vil følgende kommando starte programmet: *./ProjectHessdalenH12D23 detectandshow2.config.*

Oppstart av programmet kan også gjøres ved hjelp av webgrensesnittet, som beskrives nedenfor. Et annet alternativ er å gå til serveren for å starte programmet direkte på pcen, eller via remote programmet TeamViewer som er beskrevet nærmere i et vedlegg.

Webgrensesnitt for konfigurasjon av programmet

Det ble utviklet et webgrensesnitt som hovedsaklig kan brukes for å endre konfigurasjonsfiler med, om man ønsker et litt mer grafisk grensesnitt. Dette verktøyet ble også utvidet til å kunne starte programmet direkte med en angitt konfigurasjonsfil. Videre vil bruken av dette webverktøyet presenteres

Funksjoner

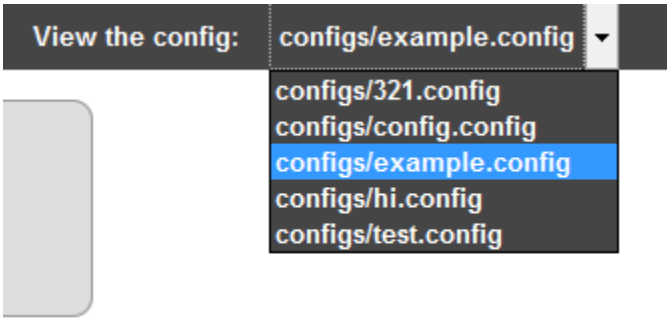
Ved hjelp av web-grensesnittet er det mulig å:

- lage nye konfigfiler (navn til filen kan oppgis av brukeren)
- lage nye moduler i en konfigfil (brukeren velger type av modulen fra en liste)
- redigere konfigfiler (altså innstillinger til moduler inne i konfigfilene)
- lagre eksisterende konfigfiler med andre navn
- starte programmet

Bruk av webgrensesnittet

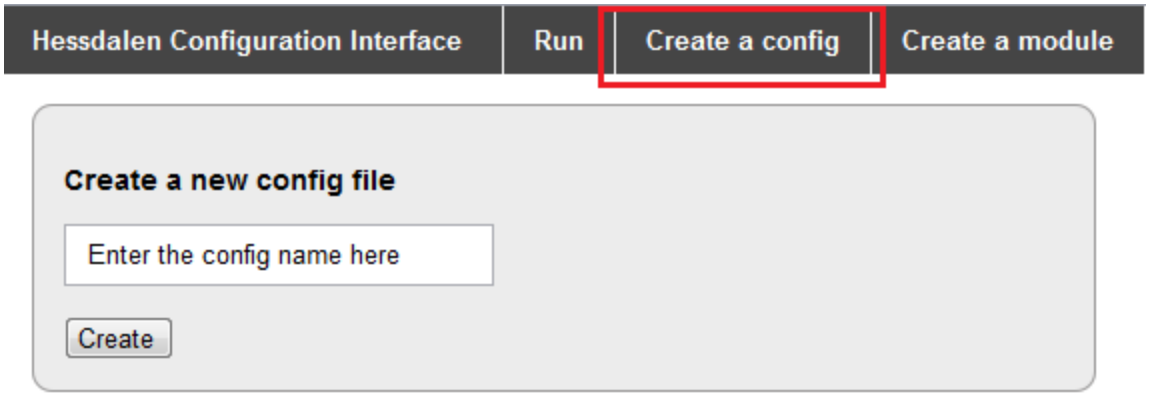
Valg av konfigfiler

For å velge en konfigfil som skal redigeres kan brukeren benytte menyen til høyre. Ved å trykke på navnet til konfigfil eller "View config" knappen, kommer brukeren på siden med moduler som finnes i konfigfilen.



Oprette konfigfiler

For å opprette en konfigfil trykker man på “Create a config” knappen, og kommer da til en side hvor man kan skrive inn ønsket navn på konfigfilen.

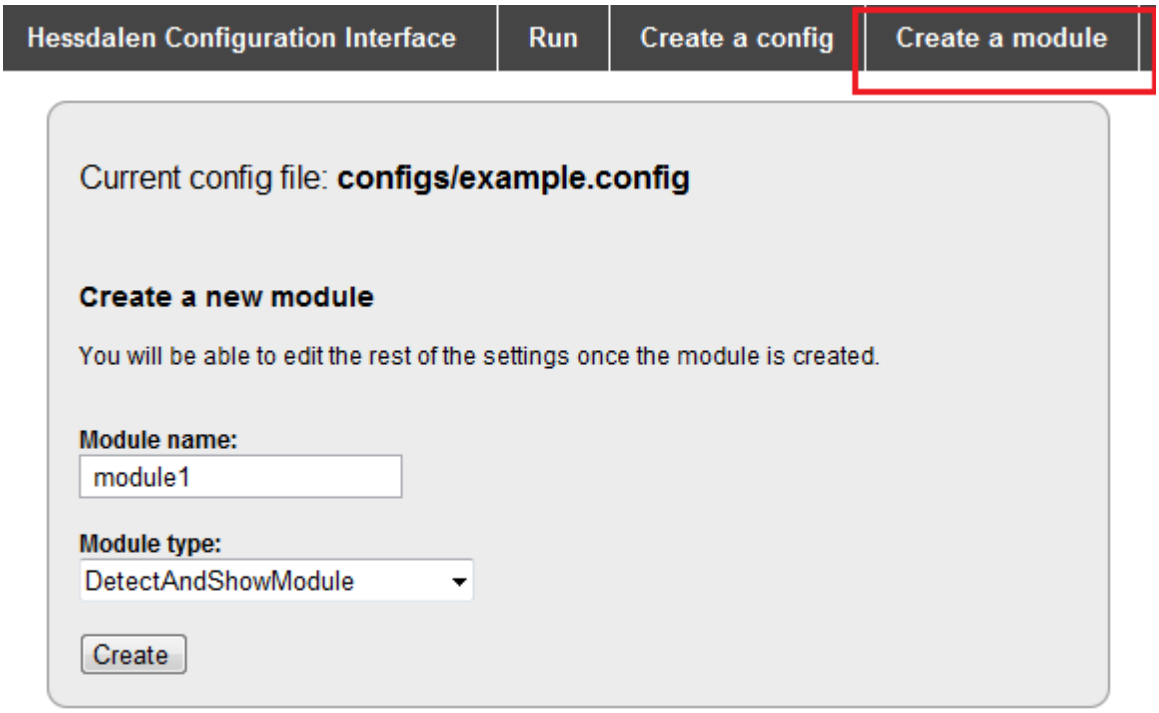


Navnet skal ikke inneholde mellomrom, spesialtegn, eller filendelse. For eksempel kan man skrive “*example”.* Etter å ha skrevet inn navnet trykker man “Create”, og da blir konfigfilen opprettet. For å se konfigfil i listen må siden oppdateres. Konfigfilene lagres i “configs” mappen.

NB: Hvis en fil med navnet brukeren valgte allerede eksisterer, blir filen overskrevet.

Opprette moduler

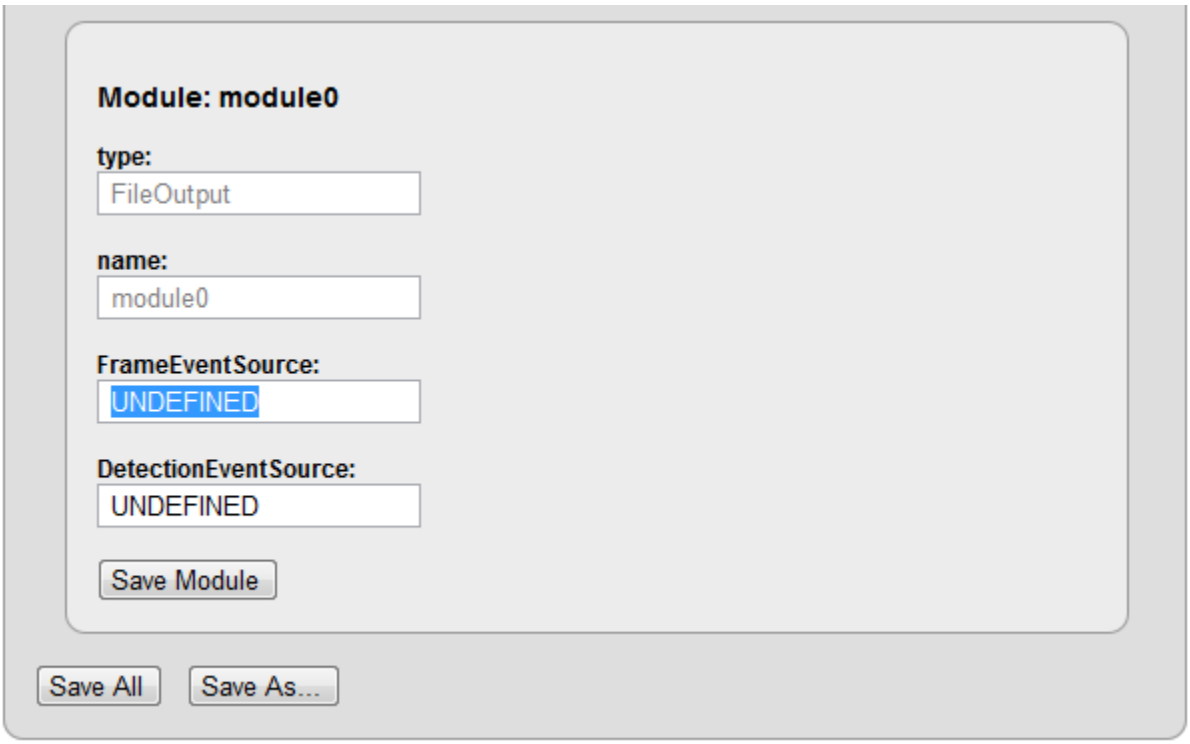
For å opprette en modul må konfigfilen hvor modulen skal opprettes først velges. Etter det kan brukeren trykke på “Create a module” knappen. Så skal typen og navnet til modulen oppgis (uten mellomrom eller spesialtegn).



Etter å ha oppgitt modultype og navn, kan brukeren trykke på “Create” knappen for å opprette modulen. Modulene kan redigeres ved å trykke “View config”.

Redigere moduler

Etter å ha kommet på siden med moduler (ved å tryke “View config”), kan brukeren redigere alle innstillinger til modulene i gjeldende konfigfil. Hvis en modul akkurat ble opprettet, eller hvis innstillingene ikke er oppgitt i konfigfilen, settes de automatisk til “UNDEFINED”. For å lagre konfigfilen med nye innstillinger kan brukeren enten trykke på “Save module” eller “Save All”.

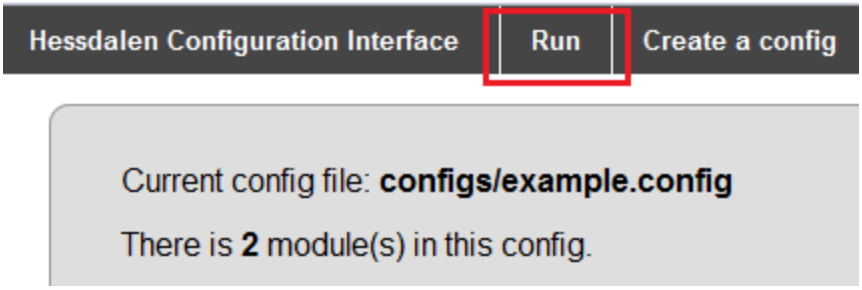


Hvis brukeren har lyst til å lagre konfigfilen med noen andre navn, kan man trykke på “Save As...” knappen. Da får man en dialog hvor navnet til den nye konfigfilen skal oppgis. Reglene for filnavnet her er samme som når man opretter en konfigfil.

NB: Hvis en fil med navnet brukeren valgte eksisterer allerede, blir filen overskrevet.

Starte programmet

Det er også mulig å starte programmet fra webrensesnittet. Det kan gjøres ved å tryke “Run” knappen. Da blir programmet startet med gjellende konfigurasjonsfil.



Veiledning for utvikling av moduler

Hva er en modul

Selv om systemet i utgangspunktet er laget for et spesifikt formål er det laget på en fleksibel måte som gjør at det kan konfigureres for å tilfredsstille andre krav og oppgaver. Endel oppgaver kan løses med de modulene som leveres med systemet, men i tillegg har man muligheten til å skrive sine egne moduler for å utvide funksjonaliteten.
En modul er en klasse som tilfredsstiller visse krav til interface for data inn og data ut. Ved å ha et standardisert interface for dette kan man enkelt skrive egne moduler som utfører de oppgaver man ønsker.

Hvilke krav må en modul tilfredsstille

For å tilfredsstille kravene som stilles til en modul må den være en subklasse av den abstrakte *Module* klassen. *Module* klassen implementerer grunnleggende funksjonalitet for en modul og er definert som en datakilde i systemet ved å være en subklasse av *FrameEventSource*. I tillegg til den arvede funksjonaliteten ifra *Module* klassen må man i modulen tilfredsstille kravet til *FrameEventSource* interfacet ved å implementere den abstrakte metoden *getFrameInformation*. Det er ikke et krav at en modul implementerer *FrameEventListener* interfacet som vil si at den er interessert i å motta data i tillegg til å være en datakilde, men dette vil nok ofte være ønskelig alikevel.

Håndtere input til modulen

For at en modul skal kunne ta imot input ifra andre moduler må den implementere *FrameEventListener* interfacet. Dette gjøres ved å subklasse *FrameEventListener* klassen og implementere *newFrameEvent* metoden i dette interfacet. Moduler man vil motta data fra vil sende en *FrameEvent* til denne metoden når de har nye data å tilby. Hvilke moduler man ønsker å motta data fra setter man i konfig filen, mer om det under konfigurasjon av modulen. Når man mottar en *FrameEvent* kan man velge å kalle kildens *getFrameInformation* metode for å hente oppdaterte data, eller ignorere denne eventen.

Konfigurasjon av modulen

Konfigurasjon av modulen gjøres ved å lage en egen seksjon i konfig filen som parses når programmet starter. En slik seksjon begynner med kommandoen #Module, og kan valgfritt avsluttes med #End. Innholdet i denne seksjonen er key/value par for de innstillingene man ønsker å konfigurere, et eksempel på hvordan en slik seksjon kan se ut er:

```
#Module
name:              display
type:              SimpleDisplayModule
FrameEventSource:  streamsource
#End
```

To av disse innstilningene er påkrevd og må finnes i enhver slik seksjon.

Name
Denne instilningen setter en unik id som gjelder en instans av en modul. Denne id'en

bruker man når man definerer relasjoner til andre moduler, i eksempelet over er streamsource et eksempel på id'en til en datakilde for display modulen.

Type

Denne instilningen sier hva slags modul type denne seksjonen gjelder for og det er god skikk å bruke klassenavnet til modulen til å navngi typen. Systemet bruker denne innstilningen til å vite hva slags klasse som skal brukes for å opprette en instans av modulen. For at systemet skal kunne opprette instanser av modulen må denne typen registreres i *ModuleFactory*.

Legge til modulen i ModuleFactory

For at systemet skal kjenne til modulen og være i stand til å opprette objekter av denne klassen må man "registrere" modulen i den klassen som står for opprettelsen av alle objektene. Dette er klassen ModuleFactory, og dette gjøres ved å legge til en elseif blokk slik som i kodesnutten under:

```
else if(moduleType.compare("SimpleDisplayModule", Qt::CaseInsensitive)
== 0){
      newModule = new SimpleDisplayModule(settingsGroup);
  }
```

Det man gjør her er at man sjekker om typen som er satt i config filen matcher med navnet på modulen, gjør det det så oppretter man et objekt av denne modulen og tilordner det SettingsGroup objektet som tilhører denne modulen.

Klasser i systemet som man må forholde seg til når man utvikler moduler

Her følger en kort beskrivelse av de viktigste klassene man må forholde seg til når man ønsker å skrive sin egen modul. For en mer detaljert beskrivelse av klassene henvises det til kode dokumentasjonsvedlegget.

Module

Dette er en abstrakt klasse som alle moduler må subklasse. Den gir grunnleggende funksjonalitet for moduler og definerer de interfacene som trengs for å kunne regnes som en modul.

VideoStreamProcessorModule

Dette er en abstrakt klasse som er subklasse av både Module klassen og FrameEventListener, det gjør denne klassen velegnet som parent hvis man skal lage en modul som trenger å både sende og motta data.

FrameEvent

Dette er en event klasse som sendes med når en modul ønsker å fortelle sine lyttere at den har nye data tilgjengelig.

FrameEventSource

Denne klassen definerer interfacet for en modul som er en datakilde for andre moduler. Alle moduler må tilfredsstille dette interfacet da Module klassen er en subklasse av denne klassen.

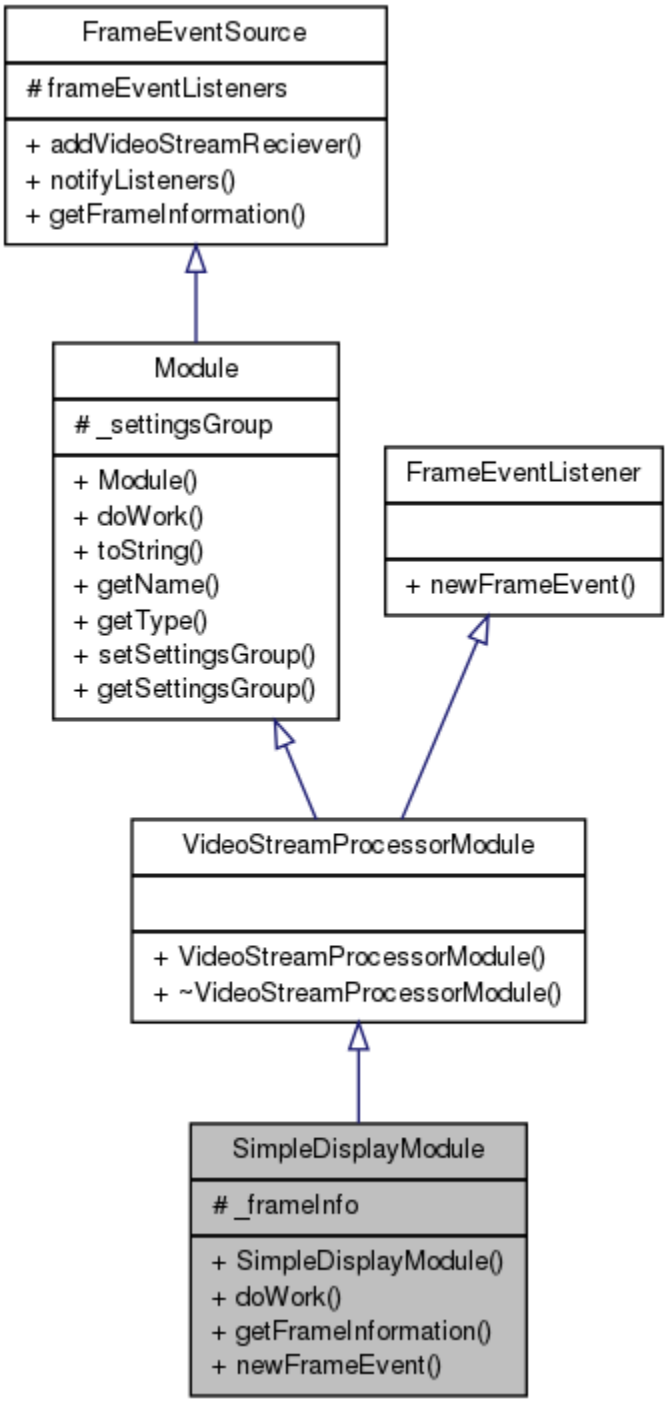
FrameEventListener
Denne klassen definerer interfacet for en modul som tar imot data ifra andre moduler.

SettingsGroup
Dette er en “beholder” klasse som inneholder alle innstilninger som er knyttet til en modul.

Setting
Dette er en klasse som inneholder en enkelt innstillning for en modul.

En enkel modul

Under sees et arvediagram for en veldig enkel modul som tar imot en videostrøm og viser denne til skjerm. Hvordan man kan skrive en slik modul følger etter bildet, beskrivelsen forutsetter kjennskap til programmeringsspråket c++ og bare det som er spesifikt i forbindelse med utvikling av en modul til dette systemet forklares



nærmere.

Header

Her er den komplette header fila for denne modulen:

```
#ifndef SIMPLEDISPLAYMODULE_H
#define SIMPLEDISPLAYMODULE_H
#include "VideoStreamProcessorModule.h"
#include "FrameEvent.h"
```

```
#include "FrameInformation.h"
#include "VideoStreamProcessorModule.h"

/**
* A minimal Module that displays a stream using opencv's imshow
method.
**/
class SimpleDisplayModule : public VideoStreamProcessorModule
{
protected:
  FrameInformation *_frameInfo;
public:
  SimpleDisplayModule(settings::SettingsGroup* settings);
  virtual void doWork(FrameInformation* frameInfo);
  virtual FrameInformation *getFrameInformation();
  void newFrameEvent(FrameEvent *event);
};

#endif // SIMPLEDISPLAYMODULE_H
```

Det som er verdt å merke seg her er at denne modulen subklasser ikke *Module* klassen direkte, men den subklasser *VideoStreamProcessorModule*, da den er interessert i å motta input fra andre moduler. Ellers tar klassen imot en peker til et *SettingsGroup* objekt i konstruktøren, dette objektet forteller modulen hvilken konfigurasjon som ønskes for denne instansen av modulen. Videre ser vi at FrameEventSource interfacet implementeres ved at vi implementerer den abstrakte getFrameInformation metoden fra dette interfacet. Også FrameEventListener interfacet implementeres ved newFrameEvent metoden.

Implementasjon

Her kommer selve implementasjonen av modulen, det er valgt å splitte opp denne i de enkelte metodene for at det skal være litt lettere å lese.

Konstruktøren

Da dette er en veldig enkel modul, implementert på en veldig grunnleggende måte så er det faktisk i realiteten ingen konstruktør. Det eneste som blir gjort her er at *SettingsGroup* pekeren som blir gitt som parameter blir sendt videre til *VideoStreamProcessorModule* sin konstruktør via initialiseringslisten.

```
SimpleDisplayModule::SimpleDisplayModule(settings::SettingsGroup*
settings) :VideoStreamProcessorModule(settings) {
}
```

FrameEventSource interfacet

Her er koden som kreves for å implementere dette interfacet:

```
/**
* The method that exposes the FrameInformation field
* to listeners subscribing to this module.
*
* @return FrameInformation the current frame stored by this module.
*/
FrameInformation *SimpleDisplayModule::getFrameInformation(){

  return _frameInfo;
}
```

Som det fremstår av koden er dette en ren “getter” metode som returnerer det siste bildet fra videostrømmen som ble mottatt.

FrameEventListener interfacet

Koden som kreves for å implementere dette interfacet er ikke stort mere komplisert enn det som krevdes for å implementere FrameEventSource interfacet:

```
/**
* Implementation of the newFrameEvent of the FrameEventListener
interface.
*
* Fetches a pointer to a FrameInformation object from the source given
in
* the passed FrameEvent and passes this on to the #doWork method.
*
* @param *event A pointer to the FrameEvent object that this event
refers to.
*
* \callgraph
*/
void SimpleDisplayModule::newFrameEvent(FrameEvent *event){

  doWork(event->getSource()->getFrameInformation());
}
```

Det som skjer her er at man henter ut en peker til den modulen som sender en event ifra FrameEvent objektet som sendes med. Deretter kaller man denne modulen sin getFrameInformation for å få det siste bildet ifra videostrømmen, dette bildet sender man så videre til modulens doWork metode som er den metoden som tar seg av selve “prosesseringen”.

Selve kjernen i modulen

Dette er koden som gjør selve jobben som modulen er ment å utføre, de andre kodebitene er bare der for å kommunisere med andre moduler for å kunne integreres i systemet. Denne biten av koden kan se svært forskjellig ut ifra modul til modul avhengig av hva slags oppgave modulen skal utføre. I dette eksempelet skal ikke modulen utføre noe annet enn å vise et bilde

til skjerm og varsle eventuelle lyttere om at den har et nytt bilde tilgjengelig.

```
/**
* Displays a frame.
*
*
* @param *frameInfo a pointer to the FrameInformation object
* containing the frame to be displayed.
*/
void SimpleDisplayModule::doWork(FrameInformation *frameInfo){
  _frameInfo = frameInfo;
  cv::imshow("SimpleDisplay", _frameInfo->getOriginalFrame());
  notifyListeners(new FrameEvent(this));
}
```

Linjen
`cv::imshow("SimpleDisplay", _frameInfo->getOriginalFrame());`
Tar seg av å vise bildet til skjerm ved å benytte en funksjon i bildebehandlingsbiblioteket opencv, mens:
`notifyListeners(new FrameEvent(this));`
varsler alle moduler som har registrert seg som lyttere til denne modulen om at det er nye data tilgjengelige. Den gjør dette ved å sende med et FrameEvent objekt med en peker til seg selv som kilde for denne eventen. Mer om kommunikasjonen mellom moduler kan leses i avsnittet “Informasjonsflyt mellom modulene”.

Registrering av modulen i ModuleFactory

For å registrere modulen i ModuleFactory kan man bruke den samme kodesnutten som i avsnittet “Legge til modulen i ModuleFactory”:

```
else if(moduleType.compare("SimpleDisplayModule", Qt::CaseInsensitive)
== 0){
      newModule = new SimpleDisplayModule(settingsGroup);
  }
```

Bruke modulen

Når all kode er skrevet er det eneste som gjenstår å skrive en konfigurasjonsfil som tar i bruk den nye modulen, et eksempel på en slik konfigurasjonsfil som bruker modulen vi nettop har skrevet følger:

```
#module
name: stream1
type: VideoStreamReader
source: rtsp://217.22.39.22/stream1

#module
type: SimpleDisplayModule
name: display
```

```
FrameEventSource: stream1
```

For en nærmere beskrivelse av hvorfor denne filen ser slik ut kan man se avsnittet “Konfigurasjon av modulen” over.