Sensur av hovedoppgaver

Høgskolen i Sørøst-Norge
Fakultet for teknologi og maritime fag

Prosjektnummer: **2016-13**
For studieåret: **2015/2016**
Emnekode: **SFHO3201**

Prosjektnavn
Remote Tower Solution

Utført i samarbeid med: Kongsberg Defence and Aerospace, avd. Integrated Defence Systems

Ekstern veileder: Alexander Gosling

Sammendrag: Remote Tower Solution har utviklet et system som detekterer og identifiserer fly i en direkte 4K videostrøm og markerer disse. Dette er en tiltenkt tilleggsfunksjon til KONGSBERG sitt eget RTS prosjekt og skal hjelpe en flygeleder eller AFIS-operatør (Aerodrome Flight Information Service) å oppdage fly visuelt i en videostrøm.

Stikkord:
- Fly
- Detektere/Identifisere
- Markere

Tilgjengelig: DELVIS

Prosjektdeltagere og karakter:

Navn	Karakter
Heidi Troppen	
Silje Ågren Aas	
Dennis Aurbakken	
Aleksander Lillevold	

Dato: 9. Juni 2016

_____	_____	_____
Sigmund Gudvangen	Karoline Moholth	Alexander Gosling
Intern Veileder	Intern Sensor	Ekstern Sensor

KONGSBERG

Høgskolen i Sørøst-Norge

Remote Tower Solution

Bacheloroppgave 2016

Rev	Endring	Dato	Utarbeidet av	Godkjent av	
-	Opprettet	21.04.2016	Silje Ågren Aas		
1	Revisjon 1	13.05.2016	Gruppe 13		
2	Siste gjennomgang	20.05.2016	Silje Ågren Aas		

Abstrakt

Dette dokumentet vil presentere prosjektprosessen og dokumentasjonen rundt Remote Tower Solution (RTS) prosjektet. Prosjektets hovedoppgave er å gjøre det enklere for en flygeleder eller AFIS-operatør (Aerodrome Flight Information Service) å oppdage fly visuelt i en videostrøm. Ved å oppdage fly og markere disse vil det bli raskere og enklere for brukeren å oppdage flyene, samt hvis andre hjelpesystemer er nede vil fortsatt flyene kunne oppdages. Avinor Flysikring AS og Kongsberg Defence Systems (KONGSBERG) har inngått en kontrakt om en fjernstyrt tårntjeneste. Det er i forbindelse med dette vi fikk i oppgave å lage en ekstraktorapplikasjon for å oppdage fly, en såkalt Moving Target Indicator (MTI).

Innholdsfortegnelse

Liste over tabeller

Liste over figurer

1. Introduksjon

Avinor Flysikring AS har inngått en kontrakt med Kongsberg Defence Systems (KONGSBERG) om leveranse av en komplett løsning for fjernstyring av tårntjenesten ved flere lufthavner fra ett kontrollsenter. I den forbindelse har KONGSBERG gitt oss i oppgave å markere fly på et skjermbilde. Prosjektet har derfor gått ut på å lage en Moving Target Indicator (MTI). Dette er en ekstraktorapplikasjon som skal hjelpe flygelederen eller AFIS-operatøren (Aerodrome Flight Information Service) med å detektere innkommende fly. Systemet markerer fly visuelt på et skjermbilde over flyplassen, og vil gjøre det enklere for operatørene å oppdage fly visuelt. I tillegg til at dette vil gjøre det mer effektivt å oppdage flyene på skjermbilde vil det også være en sikkerhet hvis andre systemer er nede.

Oppgaven gikk da ut på å finne en løsning som gjorde dette mulig å utføre. I denne rapporten er det beskrevet hvilke valg vi gjorde av biblioteker og hvordan vi flettet disse sammen for å få det endelige produktet. Noen av de teknologiene som er brukt i prosjektet er blant annet “Blackmagic Design Software Development Kit” og biblioteket “OpenCV”. For å kunne oppdage og markere fly er det utviklet et system som benytter bevegelsesdeteksjon for å oppdage flyene. Dette er utført ved hjelp av “OpenCV”. Deretter markerer systemet flyene med en grønn firkant. I tillegg har systemet en funksjonalitet for å slå av og på markeringen av objektene.

1.1 Omfang

I denne rapporten vil vi beskrive prosjektplanen, kravspesifikasjonen og testspesifikasjonen i tillegg til en grundig teknisk oversikt over de forskjellige systemmodulene og hvordan de er satt sammen til et system for å fungere optimalt. Deretter vil testplanen og resultatene bli beskrevet. Avslutningsvis vil vi ha en etteranalyse og konklusjon av prosjektet.

Rapporten gir en grundig beskrivelse av prosjektprosessen og funksjonaliteten til systemet. Den gir også en overordnet beskrivelse av systemet og de tekniske spesifikasjonene. Dokumentet referer i stor grad til andre dokumenter, for en mer detaljert beskrivelse av innholdet.

2. Prosjektplan

2.1 Mål, avgrensinger og forutsetninger

Hovedmålet for dette prosjektet vil være å designe og utvikle en Moving Target Indicator(MTI). Dette er en ekstraktorapplikasjon som skal detektere innkommende fly i en videostrøm. Systemet skal markere fly på et skjermbilde over flyplassen. Det er derfor svært viktig at applikasjonen detekterer hva slags type objekt det er. Med hensyn på type objekt skal det bestemme om objektet er et fly, og om objektet skal markeres eller ikke.

2.2 Milepæler

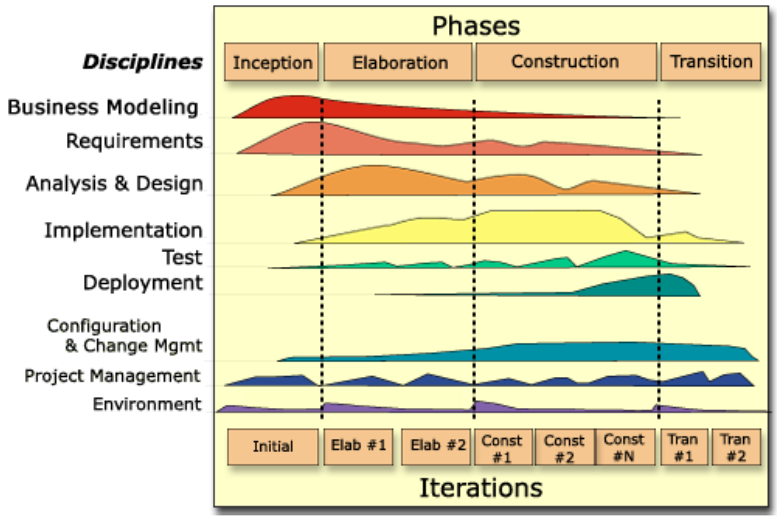
De viktigste milepælene i prosjektet vil være presentasjonene og den siste innleveringen. Den første presentasjonen vil inneholde hva prosjektet går ut på og planen for hvordan vi skal gjennomføre prosjektet. Den andre presentasjonen vil gå på mer tekniske detaljer og om hvordan prosjektet ligger an i forhold til planen. Før tredje og siste presentasjon skal alt av dokumentasjon leveres inn. Den siste presentasjonen er satt sammen av en del hvor vi presenterer det tekniske, og en del hvor vi skal selge produktet vårt.

Tabell 1: Milepæler

Milepæl	Tid
1. Presentasjon	Uke 6
2. Presentasjon	Uke 13
Innlevering	Uke 21
3. Presentasjon	Uke 21

2.3 Prosjektmodell

Som prosjektmodell har vi valgt å bruke Unified Process som er en iterativ og inkrementell utviklingsprosess. Hver tidsbestemte gjentagelse er delt inn i fire faser, innledning, utforming, bygging og overgang. Hver iterasjon resulterer i et delsystem, som er en utgivelse av systemet som enten er tilført ny eller forbedret funksjonalitet sammenlignet med den forrige utgivelsen. De fleste fasene vil omfatte gjentagelse under arbeidet, men fokuset vil endre seg i løpet av prosjektet. En viktig del av prosessen er muligheten for testing under utformingsfasen. Delvis implementering av systemet gjør at vi kan validere arkitekturen og sette grunnlaget for videre utvikling av systemet.



Figur 1: Fasene

2.3.1 Innledning

I innledningsfasen begynner vi med prosjektplanlegging og har fokus på å få et overblikk over prosjektet. Først har vi fokus på bakgrunnen og målet med selve prosjektoppgaven. Vi må skaffe oss et overblikk over frister og begynne å lage en prosjektplan slik at vi kan overholde de gitte fristene.

Prosjektplanene må følge en prosjektmodell, samt formidle omfang, begrensinger og forutsetninger for prosjektet. Det skal lages en foreløpig tidsplan og en oversikt over aktivitetstyper. Det er også viktig å oppdage risikoer tidlig, slik at vi kan ta høyde for dette under planleggingen.

Det er også viktig å identifisere interessenter og avdekke krav. Ut ifra dette lager vi en kravspesifikasjon, samt en testspesifikasjon med akseptkriterier.

2.3.2 Utforming

Under denne fasen vil vi også jobbe videre med prosjektplanen, og gjøre oppdateringer underveis. Her blir det laget en mer detaljert risikoplan med tanker om hva som skjer hvis noe går galt. Det vil også bli en del undersøkelse etter MTI algoritmer og måter vi kan modifisere disse på.

For å validere og verifisere oppgaven må vi ha en testplan for å føre resultatene riktig. Det blir laget en testspesifikasjon som også inneholder krav som forklarer hva som er gode nok resultater for at produktet skal oppfylle et bestemt krav.

Systemarkitektur og systemdesign har hovedfokuset i denne fasen. Her kan det bli laget ulike prototyper som vi kan validere og verifisere gjennom testing. Dette blir grunnsteinen til bygningsfasen som blir planlagt i denne fasen.

2.3.3 Bygging

Det er i denne fasen mesteparten av programmeringen finner sted. Systemet blir utviklet og undersøkelser fra tidligere faser blir viktige. Som i de andre fasene skal vi oppdatere tidsplanen og prosjektplanen underveis. I tillegg skal vi også lage iterasjonsplaner for denne fasen samt lage en plan for neste fase.

2.3.4 Overgang

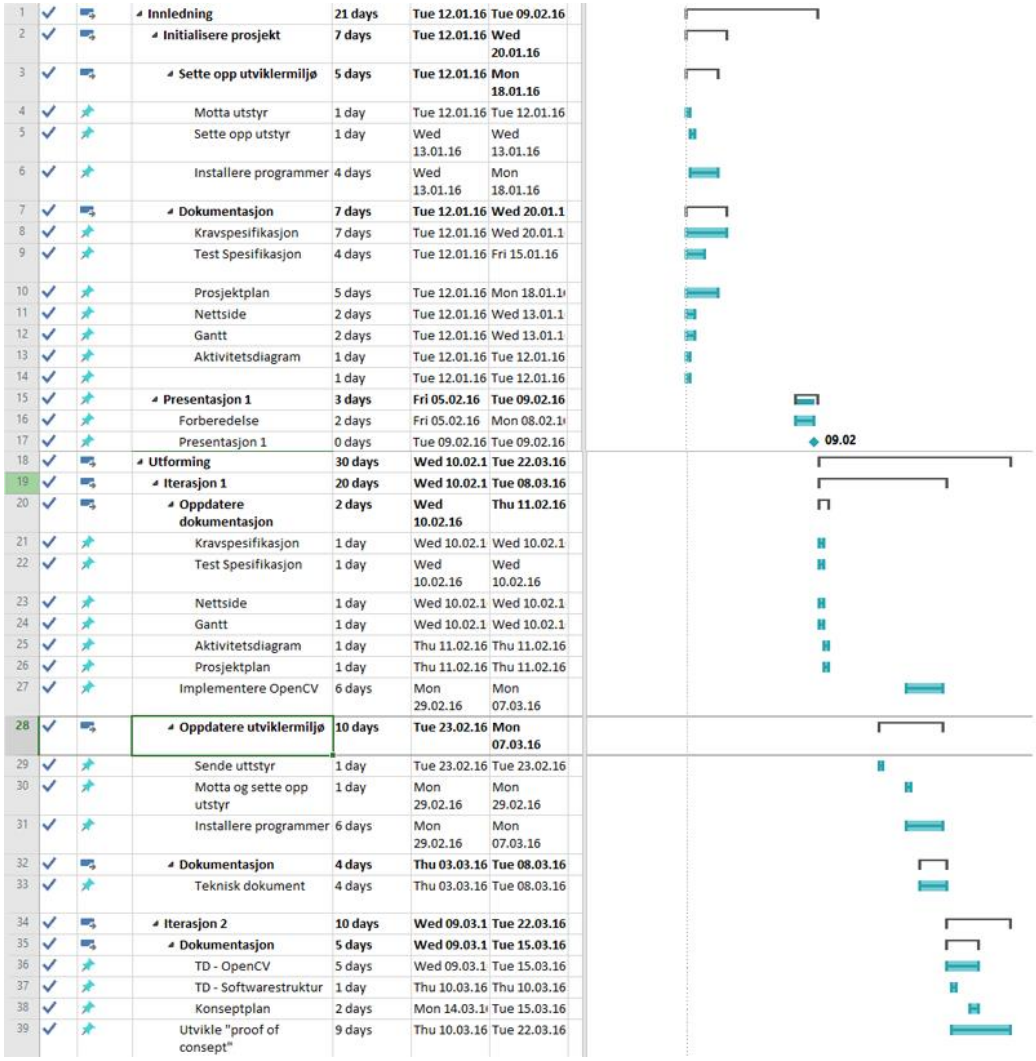
I denne fasen blir det mye testing, systemet skal både valideres og verifiseres. Prosjektplanen og tidsplanen blir ferdigstilt. Det blir også feilsøking og eventuelle feil blir da rettet opp eller fjernet på den mest optimale metoden. Ut ifra det blir det laget en feilliste som lister alle feil og om de er rettet eller ikke. En endelig rapport skal også bli skrevet i denne fasen. Det er viktig at oppdragsgiver er fornøyd med produktet, og dokumentasjonen gjør det mulig at oppdragsgiver får godt overblikk over alt som har skjedd, og hvorfor det har skjedd.

2.4 Prosjektplanen

Figur 2 viser prosjektets tidslinje med oversikt over tidsfrister og prosjektfasene. Her kan vi se hvor lang tid vi har planlagt å bruke på de forskjellige prosjektfasene, samt når presentasjonene kommer i forhold til disse.

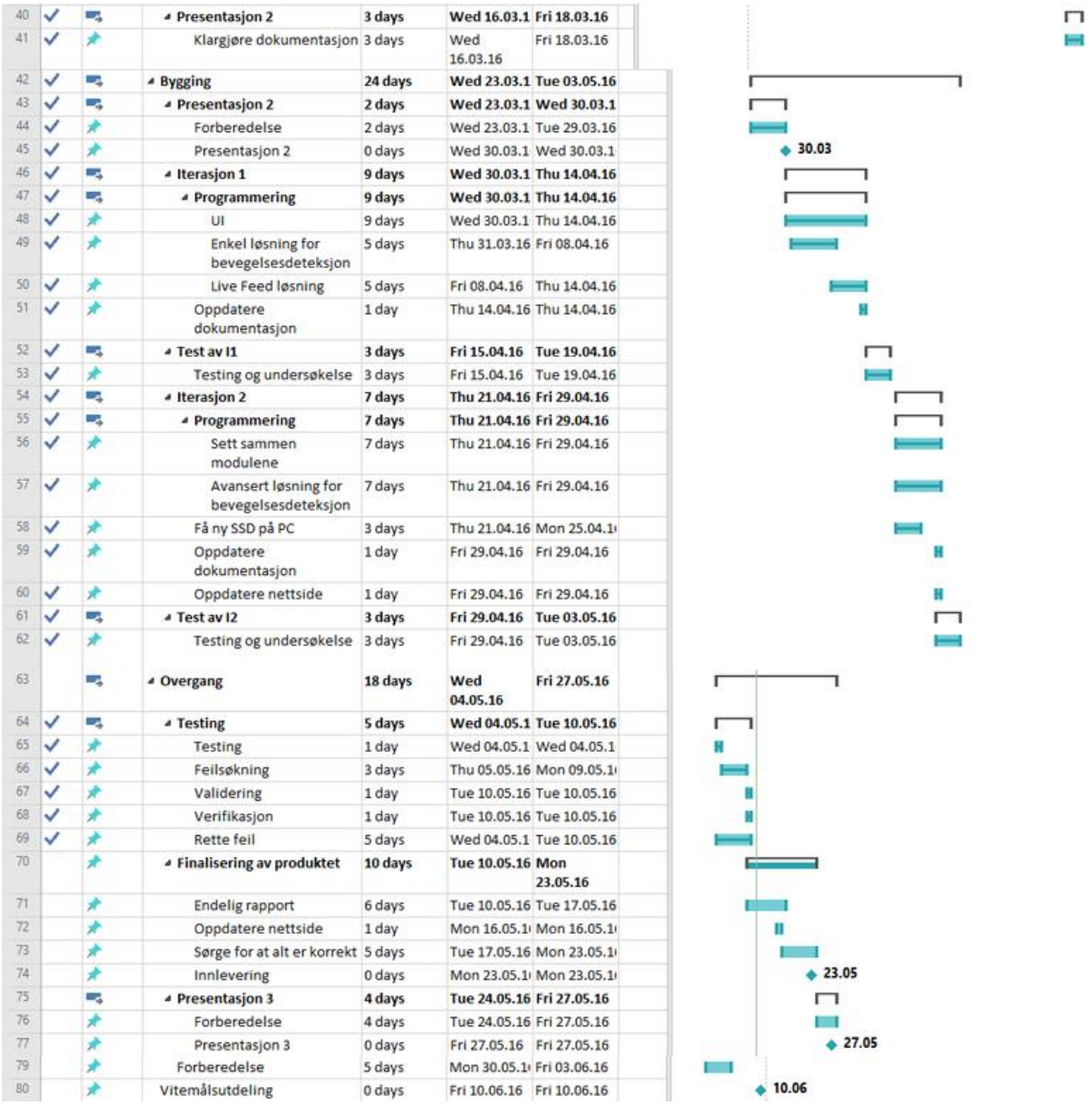
Figur 2: Tidslinje

Figur 3 og 4 viser Gantt-diagrammet som viser oppdatert tidsplan og hvordan aktiviteter avhenger av hverandre. I tillegg viser det hvor lenge hver aktivitet varer, og i hvilken rekkefølge de er gjort i forhold til prosjektmodellen.



1		Innledning	21 days	Tue 12.01.16	Tue 09.02.16
2		Initialisere prosjekt	7 days	Tue 12.01.16	Wed 20.01.16
3		Sette opp utviklermiljø	5 days	Tue 12.01.16	Mon 18.01.16
4		Motta utstyr	1 day	Tue 12.01.16	Tue 12.01.16
5		Sette opp utstyr	1 day	Wed 13.01.16	Wed 13.01.16
6		Installere programmer	4 days	Wed 13.01.16	Mon 18.01.16
7		Dokumentasjon	7 days	Tue 12.01.16	Wed 20.01.1
8		Kravspesifikasjon	7 days	Tue 12.01.16	Wed 20.01.1
9		Test Spesifikasjon	4 days	Tue 12.01.16	Fri 15.01.16
10		Prosjektplan	5 days	Tue 12.01.16	Mon 18.01.1
11		Nettside	2 days	Tue 12.01.16	Wed 13.01.1
12		Gantt	2 days	Tue 12.01.16	Wed 13.01.1
13		Aktivitetsdiagram	1 day	Tue 12.01.16	Tue 12.01.16
14			1 day	Tue 12.01.16	Tue 12.01.16
15		Presentasjon 1	3 days	Fri 05.02.16	Tue 09.02.16
16		Forberedelse	2 days	Fri 05.02.16	Mon 08.02.1
17		Presentasjon 1	0 days	Tue 09.02.16	Tue 09.02.16
18		Utforming	30 days	Wed 10.02.1	Tue 22.03.16
19		Iterasjon 1	20 days	Wed 10.02.1	Tue 08.03.16
20		Oppdatere dokumentasjon	2 days	Wed 10.02.16	Thu 11.02.16
21		Kravspesifikasjon	1 day	Wed 10.02.1	Wed 10.02.1
22		Test Spesifikasjon	1 day	Wed 10.02.16	Wed 10.02.16
23		Nettside	1 day	Wed 10.02.1	Wed 10.02.1
24		Gantt	1 day	Wed 10.02.1	Wed 10.02.1
25		Aktivitetsdiagram	1 day	Thu 11.02.16	Thu 11.02.16
26		Prosjektplan	1 day	Thu 11.02.16	Thu 11.02.16
27		Implementere OpenCV	6 days	Mon 29.02.16	Mon 07.03.16
28		Oppdatere utviklermiljø	10 days	Tue 23.02.16	Mon 07.03.16
29		Sende uttstyr	1 day	Tue 23.02.16	Tue 23.02.16
30		Motta og sette opp utstyr	1 day	Mon 29.02.16	Mon 29.02.16
31		Installere programmer	6 days	Mon 29.02.16	Mon 07.03.16
32		Dokumentasjon	4 days	Thu 03.03.16	Tue 08.03.16
33		Teknisk dokument	4 days	Thu 03.03.16	Tue 08.03.16
34		Iterasjon 2	10 days	Wed 09.03.1	Tue 22.03.16
35		Dokumentasjon	5 days	Wed 09.03.1	Tue 15.03.16
36		TD - OpenCV	5 days	Wed 09.03.1	Tue 15.03.16
37		TD - Softwarestruktur	1 day	Thu 10.03.16	Thu 10.03.16
38		Konseptplan	2 days	Mon 14.03.1	Tue 15.03.16
39		Utvikle "proof of consept"	9 days	Thu 10.03.16	Tue 22.03.16

Figur 3: Gantt



40	✓		Presentasjon 2	3 days	Wed 16.03.1 Fri 18.03.16
41	✓		Klargjøre dokumentasjon	3 days	Wed 16.03.16 Fri 18.03.16
42	✓		Bygging	24 days	Wed 23.03.1 Tue 03.05.16
43	✓		Presentasjon 2	2 days	Wed 23.03.1 Wed 30.03.1
44	✓		Forberedelse	2 days	Wed 23.03.1 Tue 29.03.16
45	✓		Presentasjon 2	0 days	Wed 30.03.1 Wed 30.03.1
46	✓		Iterasjon 1	9 days	Wed 30.03.1 Thu 14.04.16
47	✓		Programmering	9 days	Wed 30.03.1 Thu 14.04.16
48	✓		UI	9 days	Wed 30.03.1 Thu 14.04.16
49	✓		Enkel løsning for bevegelsesdeteksjon	5 days	Thu 31.03.16 Fri 08.04.16
50	✓		Live Feed løsning	5 days	Fri 08.04.16 Thu 14.04.16
51	✓		Oppdatere dokumentasjon	1 day	Thu 14.04.16 Thu 14.04.16
52	✓		Test av I1	3 days	Fri 15.04.16 Tue 19.04.16
53	✓		Testing og undersøkelse	3 days	Fri 15.04.16 Tue 19.04.16
54	✓		Iterasjon 2	7 days	Thu 21.04.16 Fri 29.04.16
55	✓		Programmering	7 days	Thu 21.04.16 Fri 29.04.16
56	✓		Sett sammen modulene	7 days	Thu 21.04.16 Fri 29.04.16
57	✓		Avansert løsning for bevegelsesdeteksjon	7 days	Thu 21.04.16 Fri 29.04.16
58	✓		Få ny SSD på PC	3 days	Thu 21.04.16 Mon 25.04.1
59	✓		Oppdatere dokumentasjon	1 day	Fri 29.04.16 Fri 29.04.16
60	✓		Oppdatere nettside	1 day	Fri 29.04.16 Fri 29.04.16
61	✓		Test av I2	3 days	Fri 29.04.16 Tue 03.05.16
62	✓		Testing og undersøkelse	3 days	Fri 29.04.16 Tue 03.05.16
63			Overgang	18 days	Wed 04.05.16 Fri 27.05.16
64	✓		Testing	5 days	Wed 04.05.1 Tue 10.05.16
65	✓		Testing	1 day	Wed 04.05.1 Wed 04.05.1
66	✓		Feilsøkning	3 days	Thu 05.05.16 Mon 09.05.1
67	✓		Validering	1 day	Tue 10.05.16 Tue 10.05.16
68	✓		Verifikasjon	1 day	Tue 10.05.16 Tue 10.05.16
69	✓		Rette feil	5 days	Wed 04.05.1 Tue 10.05.16
70			Finalisering av produktet	10 days	Tue 10.05.16 Mon 23.05.16
71			Endelig rapport	6 days	Tue 10.05.16 Tue 17.05.16
72			Oppdatere nettside	1 day	Mon 16.05.1 Mon 16.05.1
73			Sørge for at alt er korrekt	5 days	Tue 17.05.16 Mon 23.05.1
74			Innlevering	0 days	Mon 23.05.1 Mon 23.05.1
75			Presentasjon 3	4 days	Tue 24.05.16 Fri 27.05.16
76			Forberedelse	4 days	Tue 24.05.16 Fri 27.05.16
77			Presentasjon 3	0 days	Fri 27.05.16 Fri 27.05.16
79			Forberedelse	5 days	Mon 30.05.1 Fri 03.06.16
80			Vitemålsutdeling	0 days	Fri 10.06.16 Fri 10.06.16

Figur 4: Gantt

2.4.4 Oppgavebeskrivelser og aktiviteter

Oppgavene som har blitt gjort i løpet av prosjektet er hovedsakelig produksjon av produktet og dokumentasjon. Dokumentasjonen og beskrivelsen av prosjektet og produktet har stor betydning for resultatet av prosjektet. Aktivitetslisten, i Vedlegg 1, beskriver de forskjellige aktivitetene og er med på å ha en god oversikt over prosjektplanen. Hver aktivitet får ett identifikasjonsnummer og en ansvarsperson. Hver aktivitet er direkte linket til et krav slik at vi lett kan spore dette og lettere kan se om er krav har riktig oppfølgning eller ikke. Det er i tillegg laget en oversikt over hvor mye tid som er brukt på hver aktivitet, og hvordan tiden er fordelt på de forskjellige aktivitetene. [1]

2.4.5 Justeringer underveis

Etter en del utfordringer rundt installasjon av programvare og det valgte biblioteket for bevegelsesdeteksjon, ble det midt i utformingsfasen bestemt at prosjektet skulle bytte plattform fra Linux til Windows. [2] Dette var en avgjørelse som ble gjort sammen med oppdragsgiver, og avgjørelsen ble tatt på grunnlag av oppdragsgivers fokus på selve applikasjonen og

bildedeteksjonen. Dette førte til noe ekstra arbeid og vi måtte derfor gå inn å endre på den opprinnelige planen slik at vi på ny fikk satt opp utviklingsmiljøet.

Dette tok noe tid, men etter bytte av plattform gjorde det at ting gikk lettere og det var lite utfordringer med selve biblioteket slik at vi klarte å hente oss greit inn igjen og kunne fortsette å følge den opprinnelige planen som var satt.

Mot slutten av prosjektet bestemte vi også at vi skulle begynne noe tidligere enn planlagt på dokumentasjonen som skulle leveres inn. Dette gjorde vi for å forsikre oss om at vi hadde fått med oss alt som skulle være med og at vi hadde tid til å gå igjennom å rette opp eventuelle feil og mangler.

2.5 Interessenter

Det kan være enkeltpersoner eller grupper med interesser i prosjektet eller sluttproduktet av prosjektet.

Tabell 2: Interessenter

Navn	Beskrivelse	Området
Remote Tower Solution	Utviklerne representert i gruppe 13	Design, analyse og utvikling av bevegelsesindikator
Integrated Defence Systems	Ekstern veileder Alexander Gosling	Laget prosjektbeskrivelse, samt bidrar med hjelp til spesifikasjon og krav til prosjektet
Sigmund Gudvangen	Intern veileder fra HSN	Hjelpe til med gjennomføringen av prosjektplan og dokumentasjon
Regjeringen	Den norske regjering	Må følge Norges lover og regler
Leverandører	Elektronisk eller fysisk butikk	Tilbyr produkter og tjenester

2.6 Brukere

Hovedbrukerne av systemet og deres krav til sluttproduktet.

Tabell 3: Brukere

Brukere	Krav
Flygeleder/AFIS-operatør	System for å oppdage fly visuelt
Andre	System for å oppdage spesielle objekter

2.7 Risikoanalyse

Det er viktig å avdekke risikoer i en tidlig fase av prosjektet slik at man kan ta stilling til dette når man legger en prosjekt plan. Små problemer kan by på store tidsavvik. Risikoanalysen går ut på å

finne sannsynligheten og konsekvensen av at noe kan gå galt i løpet av prosjektløpet og hva man eventuelt kan gjøre hvis noe uforutsett skulle oppstå.

Tabell 4: Sannsynligheter

Sannsynlighet	**Beskrivelse**	**Skala**
Usannsynlig	Mindre enn en hendelse hver tusende time	1
Mindre sannsynlig	Gjennomsnittlig en hendelse hver tusende time	2
Sannsynlig	Gjennomsnittlig en hendelse hver hundrede time	3
Mer Sannsynlig	Gjennomsnittlig en hendelse hver tiende time	4
Veldig Sannsynlig	Mer enn en hendelse hver tiende time	5

Tabell 5: Konsekvenser

Konsekvens	**Beskrivelse**	**Skala**
Insignifikant	Prosjekter blir nesten ikke påvirket	1
Mindre Konse-kvenser	Prosjektet får problemer, men stopper ikke opp	2
Middels Konse-kvenser	Prosjekter blir ganske påvirket og en evaluering for hva som skal gjøres bør bli tatt	3
Store Konsekven-ser	Prosjektet stopper opp og en evaluering av hva som gjøres må tas	4
Katastrofe	Prosjektet blir avbrutt	5

Tabell 6: RISK-analyse

		Sannsynlighet				
		Usannsyn-lig	Mindre sann-synlig	Sannsyn-lig	Mer Sann-synlig	Veldig Sann-synlig
Konsekvens	Insignifikant	5	10	15	20	25
	Mindre Konsekvenser	4	8	12	16	20
	Middels Konse-kvenser	3	6	9	12	15
	Store Konsekvenser	2	4	6	8	10
	Katastrofe	1	2	3	4	5

Tabell 7: Forklaring til RISK

Lav	Akseptabel risiko, ingen handling trengs
Middels	Akseptabel risiko, handling må bli tenkt på
Høy	Uakseptabel risiko, handling må bli tatt

Tabell 8: Hendelsesrisiko

Hendelse	**Sjanse (1-5)**	**Konse-kvens (1-5)**	**Risiko (1-5)**	**Risikoreduserende handling**
Ureparerbar ut-styrsfeil	1	2	2	Ha liste over all hardware så det er mulighet for å bestille nytt
Et gruppemedlem går bort	1	4	4	Ha oversikter over hva alle gjør slik at det vil være mulig og plukke opp igjen
Filer blir ødelagte/korrupte	1	5	5	Ha flere backuper tilgjengelig som jevnlig blir oppdatert
Mer tid enn det som er estimert brukes	3	2	6	Stadig følge opp og oppdatere fremgangen i prosjektet med jevnlig revaluering av tidslin-jen

2.8 Prosjektgruppe

Vi er fire studenter fra dataingeniørlinja; to fra virtuelle systemer og to fra embedded systems.

Heidi Troppen
21 år, Virtuelle Systemer
Leder

Silje Ågren Aas
21 år, Embedded Systems
Dokumentasjon

Dennis Aurbakken
22 år, Virtuelle Systemer
Design

Aleksander Lillevold
21 år, Embedded Systems
Test

2.9 Utgiftsrapport

Prosjektet har ikke hatt noe budsjett og vi har fått godkjenning av oppdragsgiver for større kjøp som er gjort underveis. De største utgiftene har vært kjøp av kamera og capture card, ellers er det bare mindre kjøp som er gjort. En fullstendig oversikt over hvilke kostnader Remote Tower Solution prosjektet har hatt er representert i Tabell 9.

Tabell 9: Utgifter

Produkt	Leverandør	Antall	Pris
GoPro Hero4 Black	KONGSBERG	1	kr 4 990,00
Kamerastativ	KONGSBERG	1	kr 139,00
DeckLink Studio 4K	KONGSBERG	1	kr 6 945,00
HDMI kabel	Clas Ohlson	1	kr 179,00
Minnekort 16GB	Elkjøp	2	kr 398,00
Kontorrekvisita	Clas Ohlson	-	kr 339,00
Diverse	-	-	kr 333,00
Forfriskninger	Kiwi/Haspa cafe	-	kr 200,00
Plakater	Kopisenteret Kongsberg	4	kr 940,00
		Sum	kr 15 123,00

2.10 Møter

Møter er viktig for at alle de involverte partene skal få oppdateringer underveis i prosjektet, samt sørge for at prosjektet drives i riktig retning. Ettersom vi er en liten gruppe med kun 4 medlemmer har vi samarbeidet tett sammen slik at behovet for møter innad i gruppen ikke har vært så stort. Vi har allikevel gjort en oppsummering av hver uke samtidig mens vi har skrevet oppfølgingsdokumentet. Dette har bidratt til at vi har hatt god oversikt over hva som har blitt gjort og av hvem, i tillegg har vi sett på hva som trenger og bli gjort i uken som kommer.

Vi har hatt ukentlige møter med intern veileder Sigmund Gudvangen, hvor vi har gått igjennom hva som er gjort og hvordan vi ligger an i forhold til planen. De ukentlige oppfølgningsdokumentene er skrevet og sendt en dag i forveien av møte slik at alle som deltok på møtene hadde en hvis oversikt over hva som måtte tas opp. Dette gjorde at møtene ble mer effektive og gruppa ble mer bevisst på hvordan vi lå an underveis. Møtene med intern veileder har forsikret oss om at dokumentasjonen og generell prosjektutvikling har holdt seg til skolens standard for gjennomføring av bachelorprosjektet.

Det har også vært ukentlige møter med ekstern veileder Alexander Gosling, i tillegg til jevn kontakt via mail. Ekstern veileder er også oppdragsgiver for prosjektoppgaven, og har derfor vært gruppas fremste ressursperson i forhold til utvikling av systemet og kravene som ble stilt til systemet. Han har også vært den som har kommet med nødvending utstyr og ressurser for gjennomføringen av prosjektet. Møtene har gått ut på å finne ut hva som er forventet av systemet, samt status om hvordan ting har gått.

Alle møteinnkallinger er gjort av gruppeleder, møtereferater er skrevet på rundgang blant alle gruppemedlemmene og oppfølgingsdokumenter er skrevet i samarbeid. Disse finnes vedlagt utenom rapporten.

2.11 Timelister

En detaljert oversikt over timeforbruket både totalt og for det enkelte gruppemedlemmet er beskrevet i et eget Excel dokument. [1] For å gjøre jobben med timeføring mest mulig effektiv valgte vi å bruke Google Skjemaer. Det vil si at vi sendte inn et “svar” for hver gang vi førte timer. Svarene ble samlet ryddig og automatisk inn i Skjemaer, og svarinformasjon og diagrammer kom dermed ut i sanntid. Det gjorde også at vi enkelt kunne sette opp en oversikt over hvor mye tid som er brukt på hver aktivitet, og hvordan tiden er fordelt på de forskjellige aktivitetene.

3. Kravspesifikasjon

Denne delen spesifiserer alle krav som systemet skal tilfredsstille i henhold til de ønskene som er kommet frem fra oppdragsgiver. Hensikten er at alle involverte parter har samme forståelse av hvilke betingelser, funksjonalitet og ytelse som stilles til systemet. Kravene er utarbeidet fra hva Kongsberg Defence Systems så for seg at systemet kunne gjøre, og hvilke funksjoner de ville ha i systemet. [3] Dette er brukt som grunnlag for vår definisjon av systemkravene.

3.1 Prioritet

Alle kravene har blitt vektet, da noen krav er viktigere for systemets funksjonalitet enn andre. Kravene vil bli bevilget prioritet A, B eller C, hvor A er det viktigste og er nødvendig for at system skal fungere. B er viktige egenskaper, og C er krav som har mindre innvirkning på systemets funksjonalitet. KDS står her for Kongsberg Defence Systems.

Hvert krav har sin egen unike ID, slik at man i testspesifikasjonen kan spore hvilke tester som tilhører hvert av kravene. Kravnummeret forteller om det er et hovedkrav eller underkrav, dette markeres med punktum og etterfølgende nummer.

3.1.1 Funksjonelle krav

Tabell 10: Funksjonelle krav

ID	Krav	Prioritet	Interessent
1	Systemet skal detektere bevegelse fra en videostrøm	A	KDS
1.1	Systemet skal ikke detektere værelementer	B	Gruppe 13
1.2	Systemet skal ikke detektere skygger	B	Gruppe 13
2	Systemet skal identifisere type objekt	A	KDS
2.1	Systemet skal markere fly	A	KDS
2.1.1	Systemet skal markere fly med en grønn firkant rundt flyet	B	KDS
2.2			
2.3	Systemet skal ikke markere fly som står stille etter landing	B	KDS
2.4	Brukeren skal kunne skru av/på individuelle markeringer	B	KDS
3	Videostrømmen må være på minimum 4096 x 2160 (4K)	B	KDS
3.1	Systemet skal prosessere 5 bilder per sekund (5 fps)	B	KDS
3.2	Systemet skal prosesseres på en direkte videostrøm	C	KDS

3.1.2 Maskinvare krav

Tabell 11: Maskinvare krav

ID	Krav	Prioritet	Interessent
4			
4.1	Systemet skal kunne kjøre på operativsystemet Windows	A	KDS
5	Systemet skal være skrevet i programmeringsspråket C++	B	KDS
5.1			
5.2	Systemets kildekoden skal dokumenteres med Doxygen	C	KDS
6	Systemet skal bruke versjonskontrollsystemet Mercurial(HG)	C	KDS

3.1.3 Andre krav

Tabell 12: Andre krav

ID	Krav	Prioritet	Interessent
7			
8			
9			
10	Systemet må ha en playbackfunksjonalitet for testing	C	Gruppe 13

3.2 Forklaring

Forklaring til hvert av kravene beskrevet i Tabell 10-12.

ID 1: Systemet skal detektere bevegelse fra en videostrøm.
Absolutt all bevegelse skal detekteres, uansett hvor stor eller liten den er. Hvorvidt den er relevant, det vil si at bevegelsen er et fly, er ikke et krav. At en bevegelse blir detektert er ikke synonymt med at den blir markert.

ID 1.1: Systemet skal ikke detektere værelementer.
Med værelementer menes for eksempel skyer, regn og sol. I tilfellene hvor disse beveger seg, skal disse ikke detekteres. Fordi disse elementene er bakgrunnsobjekter, skal de regnes som om de er en del av bakgrunnen.

ID 1.2: Systemet skal ikke detektere skygger.
Alle objekter lager skygger, noe som kan føre til falske positiver. I tilfellene hvor fly lager disse skyggene, kan det føre til falske positiver svært ofte.

ID 2: Systemet skal identifisere type objekt.
Hvis objektet er en bil, skal objektet bli identifisert som en bil, samme gjelder for eksempel fly, fugl og menneske.

ID 2.1: Systemet skal markere fly.
Det vil si en visuell markering på skjermen hvor fly-typen (passasjerfly, privatjet, lite fly) ikke er av betydning. Alle fly skal markeres uansett hvilken type de er.

ID 2.1.1: Systemet skal markere fly med en grønn firkant rundt flyet.

Flyene skal i første omgang markers med en grønn firkant, dette fordi grønn indikerer at det er en "venn" altså ufarlig.

ID 2.3: Systemet skal ikke markere fly som står stille etter landing
Definisjonen vi bruker på "stille etter landing" er 10 sekunder. Med andre ord, hvis et fly har stått stille i mer enn 10 sekunder, skal det ikke lenger markeres.

ID 2.4: Brukeren skal kunne skru av/på individuelle markeringer.
Det vil si at brukeres skal kunne skru av eller på markeringen på de individuelle flyene som er detektert.

ID 3: Videostrømmen må være på minimum 4096 x 2160 (4K).
Videostrømmen som kommer inn til applikasjonen skal være på 4096 x 2160 (4K) fordi hovedprosjektet til KDS bruker 4K som oppløsning. Applikasjonen skal da kunne ha god prosessering på 4K for framtidig bruk av prosesseringen.

ID 3.1: Systemet skal prosessere 5 bilder per sekund (5 fps).
Systemet trenger ikke prosessere mer enn 5 bilder per sekund.

ID 3.2: Systemet skal prosesseres på en direkte videostrøm.
Systemet skal direkte kunne detektere og markere fly visuelt på videostrømmen.

ID 4.1: Systemet skal kunne kjøre på operativsystemet Windows.
Ble gjort om fra Linux etter problemer med programinstallasjoner. Windows i vårt tilfelle gjelder Windows 7 og eldre versjoner.

ID 5: Systemet skal være skrevet i programmeringsspråket C++.
Dette er et krav KDA stiller på grunn av muligheten for at de skal kunne bruke det vi har laget.

ID 5.2: Systemets kildekoden skal dokumenteres med Doxygen.
Etter ønske fra KDA om å dokumentere kildekoden med Doxygen har vi valgt å gjøre dette til et krav.

ID 6: Systemet skal bruke versjonskontrollsystemet Mercurial(HG).
Dette har ingen innvirkning på systemet direkte, men hjelper oss å hele tiden ha styring/kontroll på versjonene av prosjektet vårt.

ID 10: Systemet må ha en playbackfunksjonalitet for testing.
Dette er ekstrafunksjonalitet som kan for å kunne teste enkelte deler av for eksempel en video, for å kunne se at programmet fungerer slik det skal.

4. Testspesifikasjon

Dette kapittelet inneholder en testspesifikasjon for hvert av kravene, altså en beskrivelse som beskriver hva som skal testes, og hvordan det skal gjøres. Det viktigste med en test er å verifisere om forventningene til kravet er møtt. Hver test har sporbarhet tilbake til kravet som blir testet, samt akseptkriterier og en beskrivelse av hvordan testen skal gjennomføres.

Vi har delt inn i to forskjellige testtyper, funksjonelle og ikke-funksjonelle tester. Funksjonell test betyr at vi ser på hva systemet skal gjøre. Ikke-funksjonelle tester derimot beskriver hvordan og på hvilken måte systemet skal utføre det. Vi har en mal for alle testene, med beskrivelse av de forskjellige kolonnene. De originale testene kan sees i vedlegg 2.

Tabell 13: Testmal

Test ID	**Unik ID for testen**
Krav ID	Krav ID(er) som testen omfatter
Beskrivelse	Beskriver hva slags fremgangsmåte som skal brukes for å verifisere akseptkriteriene
Akseptkriterier	Kriterier som må oppfylles for at testen skal være vellykket og at kravet som tilhører testen også er oppfylt. Hvis testen er vellykket dokumenteres dette, det samme gjelder også for eventuelle tester som feiler. Det som blir testet dokumenteres i testrapporten
Testtype	Skiller mellom funksjonelle og ikke-funksjonelle tester
Status	Testet/Ikke testet
Godkjent	Godkjent/Ikke godkjent/Delvis godkjent

4.1 Testplan

Testing av systemet er viktig for å evaluere om systemet oppfyller de kravene som ble spesifisert eller ikke. Hvis systemet ikke blir testet, kan vi heller ikke verifisere om systemet vil fungere som antatt. Ettersom dette er en ren softwareapplikasjon gjør det det enklere å teste ved kun å se om resultatet er slik vi ønsker eller ikke. Behovet for å teste individuelle moduler faller også litt bort ettersom det er en mindre applikasjon som består av kun noen få funksjoner. Underveis har vi laget “proof of concept” applikasjoner som har gjort at vi kunne verifisere at enkeltdeler av systemet fungerer som spesifisert. Testspesifikasjonen beskriver hvordan hvert krav skal testes. Den beskriver hvordan testen skal bli gjennomført og hva som skal til for at kravet er oppfylt eller ikke.

4.1.1 Krav som skal testes

ID 1: Systemet skal detektere bevegelse fra en videostrøm.
ID 1.1: Systemet skal ikke detektere værelementer.
ID 1.2: Systemet skal ikke detektere skygger.
ID 2: Systemet skal identifisere type objekt.
ID 2.1: Systemet skal markere fly.
ID 2.1.1: Systemet skal markere fly med en grønn firkant rundt flyet.
ID 2.4: Brukeren skal kunne skru av/på individuelle markeringer.
ID 3: Videostrømmen må være minimum 4096 x 2160 (4K).
ID 3.2: Systemet skal prosessere på en direkte videostrøm.
ID 3.1: Systemet skal prosessere 5 bilder per sekund (5 fps).
ID 4.1: Systemet skal kunne kjøre på operativsystemet Windows.
ID 10: Systemet må ha en playbackfunksjonalitet for testing.

4.1.2 Krav som ikke skal testes

ID 5: Systemet skal være skrevet i programmeringsspråket C++.
ID 5.2: Systemets kildekode skal dokumenteres med Doxygen.
ID 6: Systemet skal bruke versjonskontrollsystemet Mercurial(Hg).

4.2 Testtilfeller

4.2.1 Krav ID 1

Beskrivelse av testtilfellet for krav ID 1 er representert i Tabell 28.

Tabell 14: Gjennomføring Test 1

Test ID	1
Krav ID	1
Kravbeskrivelse	Systemet skal detektere bevegelse fra en videostrøm
Akseptkriterier	All bevegelse skal kunne oppdages, selv om bevegelsen ikke er relevant for sluttresultatet (det vil si et fly i bevegelse). Bevegelsen(e) skal være fra en videostrøm
Status	Testet
Godkjent	Ja

Funksjon som skal testes: I denne testen skal vi se om systemet klarer å detektere bevegelse i videostrømmen.

Akseptkriteriet: Systemet må kunne detektere all bevegelse i videostrømmen.

Prosedyre: For å kunne teste dette må vi se på forskjellige videoer med bevegelige objekter. For å kunne se om systemet detekterer bevegelsen markeres disse med firkanter.

Forventet resultat: Systemet markerer alle objekter som beveger seg i videostrømmen.

4.2.2 Krav ID 1.1

Beskrivelse av testtilfellet for krav ID 1.1 er representert i Tabell 29.

Tabell 15: Gjennomføring Test 12

Test ID	12
Krav ID	1.1
Kravbeskrivelse	Systemet skal ikke detektere værelementer
Akseptkriterier	Værelementer som skyer, regn og sol skal ikke bli markert
Status	Testet
Godkjent	Ja

Funksjon som skal testes: I denne testen skal vi se om systemet klarer å skille ut våtelementer slik at disse ikke blir markert.

Akseptkriteriet: Systemet må kunne skille ut værelementer for å kunne la være å markere dette.

Prosedyre: For å kunne teste dette har vi sett på forskjellige videoer. For å kunne se om systemet klarer å skille ut værelementer skal ikke disse markeres med firkanter.

Forventet resultat: Systemet markerer kun relevante objekter som beveger seg i videostrømmen, ikke vær, vind, skyer eller andre værelementer.

4.2.3 Krav ID 1.2

Beskrivelse av testtilfellet for krav ID 1.2 er representert i Tabell 30.

Tabell 16: Gjennomføring Test 13

Test ID	13
Krav ID	1.2
Kravbeskrivelse	Systemet skal ikke detektere skygger
Akseptkriterier	Skygger, uansett hvilket objekt de tilhører, skal ikke markeres
Status	Testet
Godkjent	Nei

Funksjon som skal testes: I denne testen skal vi se om systemet klarer å skille skyggen fra selve objektet som beveger seg slik at skyggen ikke blir markert.

Akseptkriteriet: Systemet må kunne skille skyggen fra selve objektet som beveger seg for å kunne la være å markere denne.

Prosedyre: For å kunne teste dette har vi sett på forskjellige videoer. For å kunne se om systemet klarer å skille ut skyggen fra selve objektet skal ikke skyggen til objektet være med i firkanten som markerer objektet.

Forventet resultat: Systemet markerer kun det relevante objektet som beveger seg og ikke skyggen som er med.

4.2.4 Krav ID 2

Beskrivelse av testtilfellet for krav ID 2 er representert i Tabell 31.

Tabell 17: Gjennomføring Test 2

Test ID	2
Krav ID	2
Kravbeskrivelse	Systemet skal identifisere type objekt
Akseptkriterier	Programmet skal kunne skille mellom forskjellig type objekter. Det vil si at en gruppe objekter er for eksempel fugler, biler og mennesker
Status	Ikke testet
Godkjent	Nei

Funksjon som skal testes: I denne testen skal vi se om systemet klarer å skille mellom forskjellige type objekter. Det vil si at systemet skal kunne skille mellom for eksempel fly, biler og mennesker.

Akseptkriteriet: Systemet må kunne skille mellom forskjellige type objekter for å kunne la være å markere alt som ikke er fly.

Prosedyre: For å kunne teste dette har vi sett på videoer med forskjellig type objekter. For å kunne se om systemet klarer å skille disse skal systemet kun markere den den oppfatter som fly.

Forventet resultat: Systemet markerer kun fly som beveger seg og ikke andre objekter.

4.2.5 Krav ID 2.1

Beskrivelse av testtilfellet for krav ID 2.1 er representert i Tabell 32.

Tabell 18: Gjennomføring Test 3

Test ID	**3**
Krav ID	2.1
Kravbeskrivelse	Systemet skal markere fly
Akseptkriterier	Fly skal markeres visuelt i videostrømmen
Status	Testet
Godkjent	Ja

Funksjon som skal testes: I denne testen skal vi se om systemet klarer å markere alle fly

Akseptkriteriet: Systemet må kunne markere fly i en videostrøm.

Prosedyre: For å kunne teste dette har vi sett på videoer med fly både på bakken og i lufta, og sett om disse ble markert.

Forventet resultat: Systemet markerer fly.

4.2.6 Krav ID 2.1.1

Beskrivelse av testtilfellet for krav ID 2.1.1 er representert i Tabell 33.

Tabell 19: Gjennomføring Test 14

Test ID	**14**
Krav ID	2.1.1
Kravbeskrivelse	Systemet skal markere fly med en grønn firkant rundt flyet
Akseptkriterier	Flyet skal bli markert med en grønn firkant rundt
Status	Testet
Godkjent	Ja

Funksjon som skal testes: I denne testen skal vi se om systemet klarer å markere fly med en grønn firkant.

Akseptkriteriet: Systemet må kunne markere fly med en grønn firkant i en videostrøm.

Prosedyre: For å kunne teste dette har vi sett på flyvideoer, og sett om disse ble markert med en grønn firkant.

Forventet resultat: Systemet markerer fly med en grønn firkant.

4.2.7 Krav ID 2.1.1

Beskrivelse av testtilfellet for krav ID 2.1.1 er representert i Tabell 34.

Tabell 20: Gjennomføring Test 14

Test ID	**14**
Krav ID	2.1.1
Kravbeskrivelse	Systemet skal markere fly med en grønn firkant rundt flyet
Akseptkriterier	Flyet skal bli markert med en grønn firkant rundt
Status	Testet
Godkjent	Ja

Funksjon som skal testes: I denne testen skal vi se om systemet klarer å markere fly med en grønn firkant.

Akseptkriteriet: Systemet må kunne markere fly med en grønn firkant i en videostrøm.

Prosedyre: For å kunne teste dette har vi sett på flyvideoer, og sett om disse ble markert med en grønn firkant.

Forventet resultat: Systemet markerer fly med en grønn firkant.

4.2.8 Krav ID 2.3

Beskrivelse av testtilfellet for krav ID 2.3 er representert i Tabell 35.

Tabell 21: Gjennomføring Test 15

Test ID	15
Krav ID	2.3
Kravbeskrivelse	Systemet skal ikke markere fly som står stille etter landing
Akseptkriterier	Fly som ikke har vært i bevegelse på 10 sekunder skal ikke markeres
Status	Testet
Godkjent	Ja

Funksjon som skal testes: I denne testen skal vi se om systemet slutter å markere fly etter at det har stått stille i mer enn 10 sekunder.

Akseptkriteriet: Fly som ikke har vært i bevegelse på 10 sekunder skal ikke markeres.

Prosedyre: For å kunne teste dette har vi sett på flyvideoer, hvor fly både kjører på rullebanene og stopper.

Forventet resultat: Systemet slutter å markere flyet etter at det har stått stille i mer enn 10 sekunder.

4.2.9 Krav ID 2.4

Beskrivelse av testtilfellet for krav ID 2.4 er representert i Tabell 36.

Tabell 22: Gjennomføring Test 16

Test ID	16
Krav ID	2.4
Kravbeskrivelse	Brukeren skal kunne skru av/på individuelle markeringer
Akseptkriterier	Alle markeringer som er synlige, skal kunne skrus av/på. Dette gjelder for fly som blir markert, men også eventuelle falske positiver
Status	Testet
Godkjent	Ja

Funksjon som skal testes: I denne testen skal vi se om systemet kan skru av individuelle markeringer. Markeringen av alle de individuelle objektene skal kunne skrus av og på etter brukerens ønske.

Akseptkriteriet: Alle markeringer som er synlige, skal kunne skrus av/på. Dette gjelder for fly som blir markert, men også eventuelle falske positiver.

Prosedyre: For å kunne teste dette har vi sett på videoer med flere objekter som blir markert samtidig, og deretter testet av og på funksjonen til systemet.

Forventet resultat: Systemet markerer automatisk objektene og vi kan deretter velge å skru av eller på markeringen.

4.2.10 Krav ID 3

Beskrivelse av testtilfellet for krav ID 3 er representert i Tabell 37.

Tabell 23: Gjennomføring Test 5

Test ID	5
Krav ID	3
Kravbeskrivelse	Videostrømmen må være minimum 4096 x 2160 (4K)
Akseptkriterier	Videostrømmen som kommer inn til applikasjonen skal være på 4096 x 2160 (4K)
Status	Testet
Godkjent	Ja

Funksjon som skal testes: I denne testen skal vi se om systemet kan ta inn en videostrøm på 4096 x 2160 (4K).

Akseptkriteriet: Videostrømmen som kommer inn til applikasjonen skal være på 4096 x 2160 (4K).

Prosedyre: For å kunne teste dette har vi lest av størrelsen på den innkommende videopakken.

Forventet resultat: Størrelsen på videopakken er 4096 x 2160 (4K).

4.2.11 Krav ID 3.1

Beskrivelse av testtilfellet for krav ID 3.1 er representert i Tabell 38.

Tabell 24: Gjennomføring Test 17

Test ID	17
Krav ID	3.1
Kravbeskrivelse	Systemet skal prosessere 5 bilder per sekund (5 fps)
Akseptkriterier	Alt fungerer korrekt samtidig som kun 5 bilder per sekund vises
Status	Testet
Godkjent	Ja

Funksjon som skal testes: I denne testen skal vi se om systemet fungerer på 5 bilder per sekund.

Akseptkriteriet: Alt fungerer korrekt samtidig som kun 5 bilder per sekund vises.

Prosedyre: For å kunne teste dette ser vi i programkonsollen hvor mange "frames" som blir tatt opp i forhold til hvor mange som blir prosessert og vist fram.

Forventet resultat: Siden videostrømmen inn i applikasjonen er 30 bilder per sekund vil den opprettholde 5 bilder per sekund hvis hvert sjette bilde blir prosessert og vist frem.

4.2.12 Krav ID 3.2

Beskrivelse av testtilfellet for krav ID 3.2 er representert i Tabell 39.

Tabell 25: Gjennomføring Test 18

Test ID	18
Krav ID	3.2
Kravbeskrivelse	Systemet skal prosessere på en direkte videostrøm
Akseptkriterier	Vi skal kunne ta inn en direkte videostrøm og systemet skal prosessere denne
Status	Testet
Godkjent	Ja

Funksjon som skal testes: I denne testen skal vi se om systemet fungerer på en direkte videostrøm.

Akseptkriteriet: Vi skal kunne ta inn en direkte videostrøm og systemet skal prosessere denne.

Prosedyre: For å kunne teste dette sender vi inn en direkte videostrøm og prosesserer denne.

Forventet resultat: Systemet prosesserer den direkte videostrømmen.

4.2.13 Krav ID 4.1

Beskrivelse av testtilfellet for krav ID 4.1 er representert i Tabell 40.

Tabell 26: Gjennomføring Test 6

Test ID	6
Krav ID	4.1
Kravbeskrivelse	Systemet skal kunne kjøre på operativsystemet Windows
Akseptkriterier	Alt som lages i forhold til systemet skal kunne kjøre på et Windows operativsystem
Status	Testet
Godkjent	Ja

Funksjon som skal testes: I denne testen skal vi se om systemet fungerer på operativsystemet Windows.

Akseptkriteriet: Alt som lages i forhold til systemet skal kunne kjøre på et Windows operativsystem.

Prosedyre: For å kunne teste dette kjører vi programmet på en maskin med Windows operativsystem.

Forventet resultat: Systemet fungerer med Windows operativsystem.

4.2.14 Krav ID 10

Beskrivelse av testtilfellet for krav ID 10 er representert i Tabell 41.

Tabell 27: Gjennomføring Test 9

Test ID	**9**
Krav ID	10
Kravbeskrivelse	Systemet må ha en playbackfunksjonalitet for testing
Akseptkriterier	Spille av, pause, og stoppet videoen, samt velge hvor i videoen avspillingen skal starte
Status	Testet
Godkjent	Ja

Funksjon som skal testes: I denne testen skal vi se om systemet spiller av, kan pause, og stoppe videoen.

Akseptkriteriet: Spille av, pause, og stoppet videoen, samt velge hvor i videoen avspillingen skal starte.

Prosedyre: prosessere en video som allerede finnes samt starte og stoppe denne mens den spilles av.

Forventet resultat: At man kan starte en video som allerede finnes også stoppe og starte denne mens applikasjonen kjører.

5. Systemarkitektur

Denne delen beskriver designet av systemet, altså definerer den strukturen, atferden og representerer systemet. Den består av systemkomponenter og relasjonene mellom dem.

5.1 Systemkomponenter

En oversikt over de forskjellige hovedområdene i systemet. [4]

5.1.1 RTSManager

"RTSManager" er den som styrer og starter alt som skjer i systemet. Dette er klassen som initialiserer de andre funksjonene og holder styr på det overordnede. Under initialiseringen tillater denne klassen de andre undergruppene å kommunisere ved å gi dem pekere til hverandre hvis de skulle trenge det. For eksempel gir den prosesseringsdelen tilgang til inputdelen slik at den kan hente bilder å prosessere.

5.1.2 DeckLinkDeviceDiscovery

“DeckLinkDeviceDiscovery” er en liten hjelpeklasse som hjelper “RTSManager” med å finne opptakskortet som vil brukes senere for å ta imot videostrømmen.

5.1.3 DeckLinkDevice

“DeckLinkDevice” er den klassen som håndterer alt som har med videostrøm input og konvertering av denne til et format som kan brukes av resten av programmet. Den inneholder også metoder arvet fra "IDeckLinkInputCallbak" slik at den kan mota selve "framen” fra videostrømmen og detektere at formatet på videostrømmen endres.

5.1.4 VideoProcessing

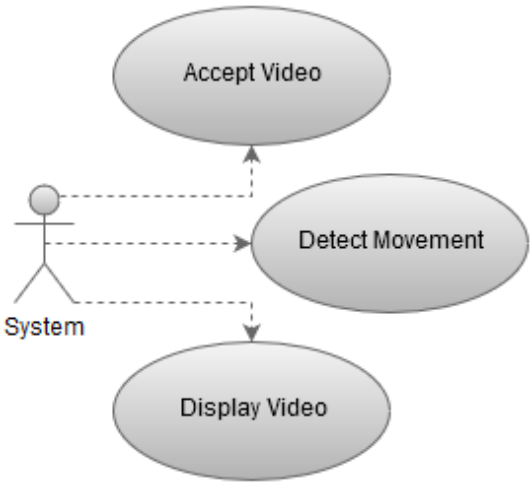
“VideoProcessing” tar seg av prosesseringen av videostrømmen. Denne klassen mottar et signal om at det er kommet en ny "frame" den kan prosessere og henter denne før den går over den og markerer det som er forskjellig og har beveget seg. Resultatet av denne prosesseringen kan da vises frem.

5.2 Systemfunksjoner

Softwaresystemet har tre hovedfunksjoner, som er illustrert i Figur 5. Disse funksjonene er:

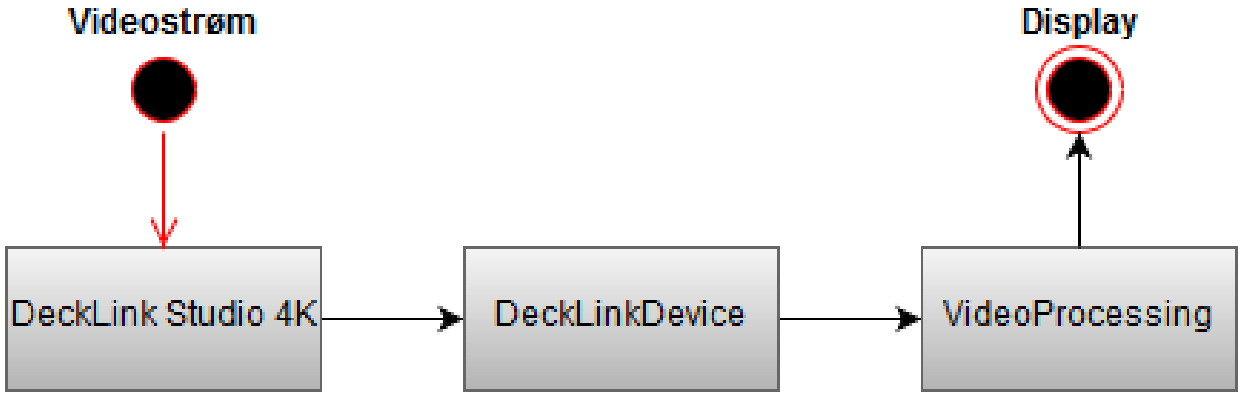
- Å bearbeide en videostrøm slik at den kan manipuleres.
- Detektere bevegelse i videostrømmen.
- Visualisere detektert bevegelse.

5.3 Systemets komponenter



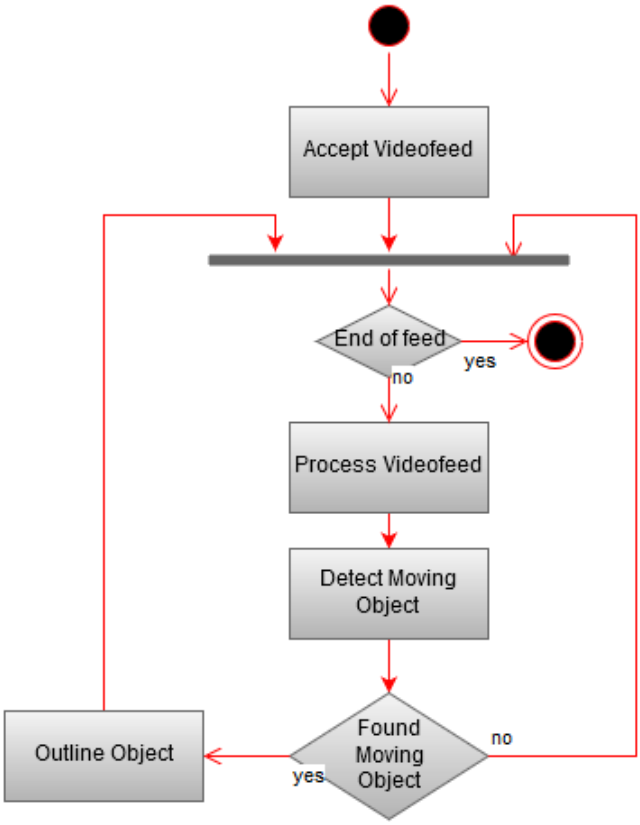
Figur 5: Use Case diagram

Systemet består av tre komponenter som er illustrert i Figur 6.



Figur 6: Komponentdiagram

5.4 Systemets kjernerepetisjon

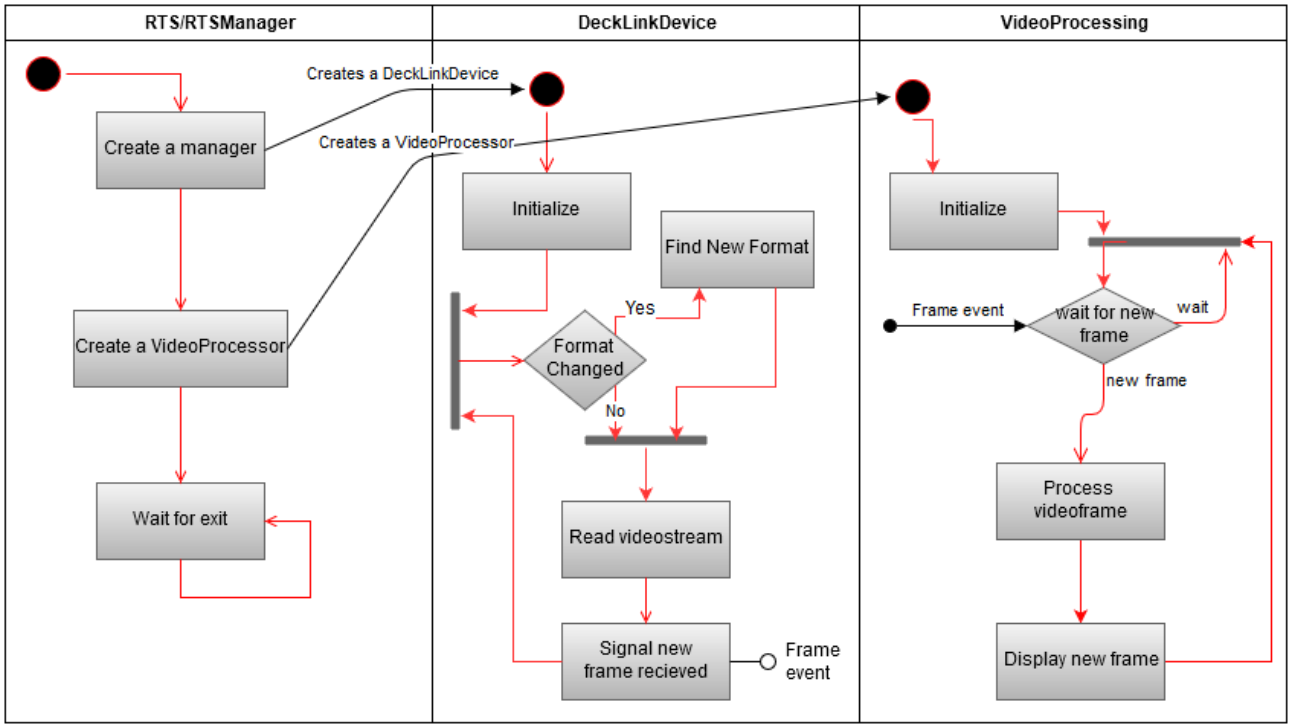


Figur 7: Aktivitetsdiagram

Figur 7 viser systemets “core loop”, det vil si hva systemet rutinemessig går igjennom. I dette systemet vil “core loopen” være prosessering av videostrømmen og detektering av bevegelse i denne strømmen. Aktivitetsdiagrammet i Figur 7 illustrerer også start og stopp av videofeeden.

5.4.1 Programflyt

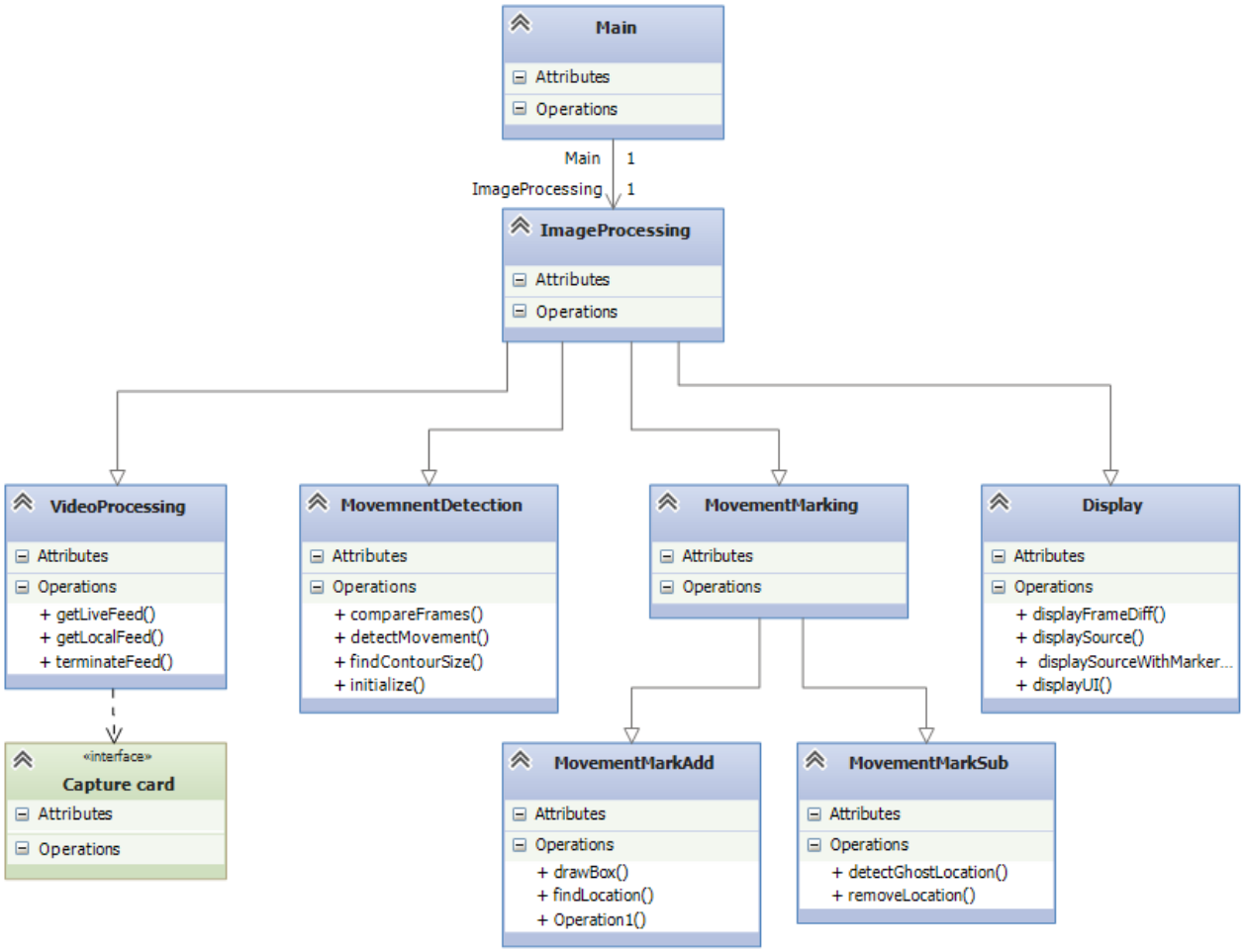
I Figur 8 kan man se en grov skisse av hvordan programmet arbeider. Hver av de tre boksene representerer en av hovedtrådene i applikasjonen. Den første, "RTS/RTSManager", tar seg av alt oppsettet. Den starter ved å lage en manager som initialiserer "DeckLinkDevice" tråden og setter opp hendelsessystemet som brukes for å si ifra om at det er nye bilder å prosessere. Deretter setter den opp og starter prosesseringstråden "VideoProcessing" før den venter på at disse skal fullføre arbeidet sitt. "DeckLinkDevice" tråden, etter at den er initialisert, går hele tiden igjennom videostrømmen den mottar og sender den videre i nødvendig format for videre bruk. Enkelt forklart sjekker denne tråden hele tiden om inngangssignalet endres slik at den må endre konverteringsmetoden sin eller om den bare kan fortsette å lese og behandle videostrømmen som før, og si ifra at en et nytt bilde er klart for behandling. Den siste tråden, "VideoProcessing", venter etter initialiseringen på hendelsen fra den andre tråden før den går igjennom bildet og prosesserer det og viser frem det endelige bildet. Når dette er gjort går den tilbake og venter på en ny hendelse.



Figur 8: Programmets flyt

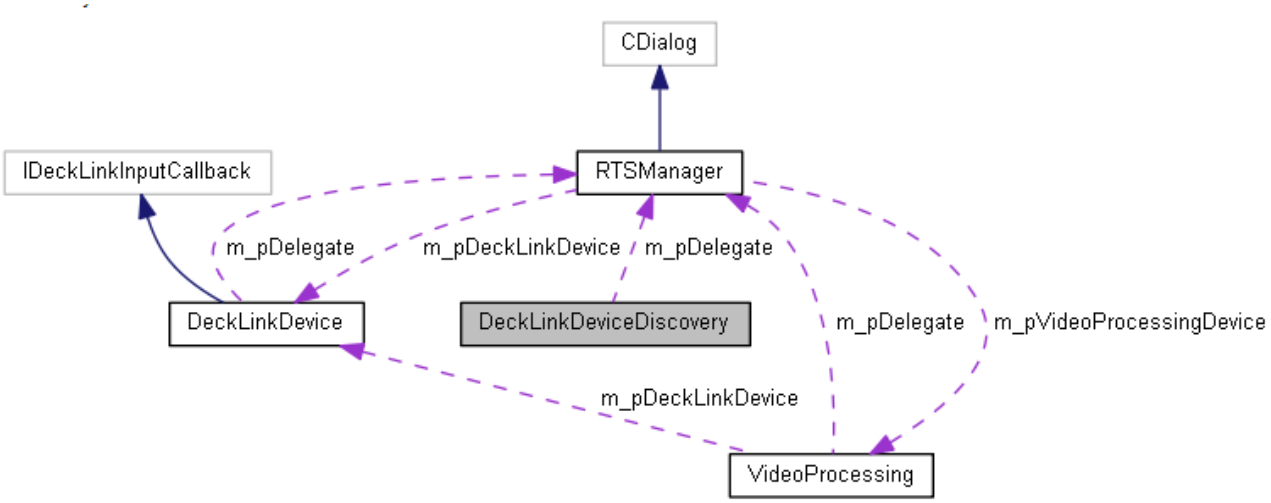
5.5 Software klasser

Det er slik, som i Figur 9, vi tenkte å strukturere programmet da vi begynte planleggingen av programmets funksjoner og struktur. Vi valgte dette utgangspunktet da vi tenkte dette ville gjøre programmet oversiktlig og enklere å forstå hvor de forskjellige tingene ble gjort. Da vi fikk begynt å lage det fant vi ut at dette designet i utgangspunktet var litt vel komplisert, samt at det var enkelte deler vi ikke trengte og noen nye som kom på grunn av enkelte avhengigheter. Vi ende også opp med å bruke andre navn under utvikling, disse samt en kort beskrivelse av klassene kan leses i seksjon 5.1 i dette dokumentet.



Figur 9: Klassediagram, Initial konsept

Fra Figur 10 kan den endelige klassestrukturen på programmet sees. "DeckLinkDevice" er den som har funksjonene som i Figur 9 ville vært tildelt "VideoProcessing", denne endte også opp med å måtte arve fra DeckLink sin SDK for å kunne ta opp video. "DeckLinkDeviceDiscovery" er en hjelpeklasse for å finne enheten til "DeckLinkDevice" og er en ekstra klasse som kom inn etter hvert. Den nye "VideoProcessing" klassen er en sammenslåing av de fire "Movement" klassene fra diagrammet i Figur 9, dette er fordi det viste seg at det var enklere for oss å gjøre disse tingene inline i prosesseringen og mindre komplisert enn om vi skulle ha flyttet dette ut i flere mindre klasser. "Display" klassen har også gått inn i denne klassen da vi valgte å bruke metoder fra det rammeverket som blir brukt der for å vise frem det prosesserte bildet det i stedet for å bygge inn enda et rammeverk kun for dette. Den gamle "ImageProcessing" klassen ble gitt et nytt navn, men har fortsatt hovedansvar for å koble sammen de andre klassene.



Figur 10: Endelig klassediagram

5.6 Systemets navngiving

Vi har valgt at variabel og metodenavn skal være beskrivende og følge en standard form. Denne standarden har vi satt som at alle metoder og variabelnavn starter med små bokstaver, deretter vil hvert av de påfølgende nye ord eller forkortelser i navnet starte med en stor bokstav. Den første bokstaven i et variabelnavn bør også beskrive hva slags type variabel det er, for eksempel “dDetteErEnDobbel” eller “nDetteErEtNaturligTall”. For metoder trenger de bare være beskrivende for hva metoden gjør, for eksempel “detteErEnMetode”.

6. Konseptvalg

6.1 Bildedeteksjon

Programmeringsspråket er C++ som må tas i betraktning, på grunn av at ikke alle bibliotek støtter dette. Hvorvidt andre programmeringsspråk støttes er ikke relevant for oss, fordi kravet som ble satt av oppdragsgiver kun gjelder C++. Det finnes flere biblioteker enn de som vi har sett på, men mange er "småprosjekter" gjort av enkeltpersoner/små grupper. Vi har derfor sett på et utvalg av de bibliotekene som vi følte var relevante, med nok dokumentasjon til at vi kunne få et overblikk over bruksområdene.

Tabell 28: Pugh matrise

		Bibliotek			
Kriterier	Vekt(1-3)	OpenCV	VXL	LTI	IPP
1. Brukervennlighet	3	1	0	-1	0
2. Installasjonsvennlighet	2	-1	0	0	1
3. Dokumentasjon av program	2	1	0	-1	1
4. Lisens	1	1	1	1	1
5. Bibliotekstørrelse/Bidrag	3	1	0	-1	-1
6. Modifiserbarhet	1	0	0	0	0
7. Oppdatert	3	1	0	-1	1
Sum		4	1	-3	3
Vektet sum		10	1	-10	5

1. Brukervennlighet beskriver hvor lett er det å forstå biblioteket og eventuelt hvor bra tutorials biblioteket har. Det er også fokus på antall brukere fordi sannsynligheten for å kunne søke seg fram til løsninger øker med antall brukere, fordi det er mulig å bruke forum og brukergenerert innhold til å løse problemer.

2. Installasjonsvennlighet handler om hvor lett det er å installere programmet i seg selv, men også eventuelle plugins/tilleggspakker som trengs. For eksempel til OpenCV trenger man codecer for å kunne spille av enkelte typer video, som er vanskelige å installere riktig uten internett.

3. Dokumentasjonen har mye å si for hvor lett det er å kunne forstå seg på algoritmene som brukes. Hvis vi forstå hva som gjøres, kan vi også lettere velge ut hvilken algoritme som passer for enhver situasjon.

4. Lisensen på biblioteket burde være "Royalty free" slik at kommersielt og akademisk bruk er gratis/lovlig.

5. Bibliotekstørrelsen er nesten synonymt med bidrag i vårt tilfelle. Antall brukere kobles også inn her, fordi i for eksempel OpenCV er det brukerne som lager/forbedrer mesteparten av det som eksisterer i bibliotekene.

6. Modifiserbarhet går på hvor lett det er å forandre algoritmene og deres virkemåte. Dette er ca. like vanskelig å få til i alle bibliotekene, og vi skal mest sannsynlig ikke forandre for mye på disse.

7. Hvorvidt biblioteket er oppdatert nok til å kunne brukes. Dette er et svært viktig poeng, siden algoritmene som brukes innenfor computer vision må være oppdaterte for å fungere godt nok.

6.1.1 Konklusjon

OpenCV kom best ut på nesten alle områder for oss, med unntak av installasjonsvennlighet. Hovedgrunnen til at vi valgte OpenCV var fordi antall brukere, dokumentasjon og mulighet for eksempelkode var utrolig god. OpenCV sitt bibliotek er uten tvil det beste innenfor computer vision, som i tillegg fyller alle behovene vi hadde for biblioteket vårt. Det er usannsynlig at vi kommer til å velge noe annet enn OpenCV sitt bibliotek grunnet resultatet av Pugh matrisen. OpenCV har enkelte av oss også litt erfaring med, noe som også trekker oss mot å anvende denne løsningen ovenfor de andre.

6.2 Algoritmer

I vårt tilfelle var det flere typer algoritmer som er interessante. Bevegelsesdeteksjon kreves for å kunne merke om flyet er i bevegelse eller om det står på bakken. Objektgjenkjenning kreves for at for eksempel fugler og biler, ikke blir markert på skjermen. Vi har også sett på estimeringsalgoritmer, algoritmer som estimerer posisjonen til fly om de går ut av syne for eksempel ved å fly bak skyer. “Optical Flow” kan være interessant for oss, fordi en slik algoritme gjør det mulig å se retningen til flyet. Det er viktig å merke seg at mange/alle av disse algoritmene kan kombineres, slik at det ikke er nødvendig med en Pugh matrise. Derimot er det lettere å konkludere med hvilken grad hver enkelt algoritme er anvendelig. [5]

6.2.1 Bevegelsesdeteksjon

Den letteste, praktiske og mest brukte måten å gjøre dette på er uten tvil “Background Subtraction”. [5] Hvis man kun har tilgang til bilder fra et stillestående videokamera er Background Subtraction svært treffsikker. Background Subtraction bruker bakgrunnen som referanse for å detektere forgrunnen. Fordi forgrunnen kan “foreldes” og bli del av bakgrunnen, er Background Subtraction veldig relevant for våre mål. I praksis vil aldring av forgrunn bety at stillestående objekter, som har stått stille en viss tid, bli regnet som bakgrunn. Det er også mulig å markere absolutt alt som er annerledes fra bakgrunnen, men i utemiljøer hvor forstyrrende objekter som for eksempel skyer, vær og vind spiller inn, er dette uaktuelt.

6.2.2 Objektgjenkjenning

“Feature Detection” er en av de mest populære måtene å kjenne igjen objekter på, fordi som navnet tilsier, detekteres enkelte trekk av et objekt. Disse trekkene kan defineres på flere måter av en Feature Detection, ved å se på forskjellige typer trekk. Hjørner, kanter og blobber (grupper) er de tre mest brukte Feature Detection algoritmene. Alene er Feature Detection ikke til å stole på, med mindre trekkene er svært unike eller miljøet kun inneholder en type objekt. Selv i disse situasjonene er Feature Detection ofte brukt sammen med noe som heter “Machine Learning”.

Machine Learning (maskinlæring) gjør det mulig å lære datamaskinen å kjenne igjen objektet vi ønsker. Dette gjøres ved at datamaskinen gis X antall bilder som den skal lære av, og desto flere bilder den får, desto bedre. For eksempel av 300 bilder, inneholder 270 av dem objektet eller et distinkt trekk, og 30 av de mangler den. I løpet av Machine Learning prosessen vil datamaskinen gå gjennom de totalt 300 bildene og sammenligne de. Machine Learning kan ta lang tid, avhengig av antall bilder og hvor kraftig datamaskinen er. Det er viktig å merke seg at kvaliteten på bildene

også spiller inn, og bildene som er positive (inneholder objektet), bør være feilfrie og helst i god oppløsning.

6.2.3 Optical Flow

Det engelske navnet er "Optical Flow", men et mer fornorsket og forenklet navn er "Retningsdeteksjon". Optical Flow brukes generelt til å detektere hvilken retning et objekt beveger seg. For at Optical Flow skal fungere riktig må man ha minst to bilder i riktig sekvens (det vil si bilde 1, 2, … n, n+1). Det er også mulig å bruke Optical Flow til å se orienteringen til et objekt, altså hvilke sider som befinner seg hvor – opp, ned, høyre og venstre. Sistnevnte høres kanskje ikke interessant ut for oss, fordi fly som regel ikke snur seg opp ned med det første. Grunnen til at orienteringen til flyet kan være interessant for vår del, er at vi kan bruke informasjonen til å se om et fly lander eller tar av.

6.2.4 Estimering

Estimering av hvor et objekt befinner seg, selv om det forsvinner for en gitt tid er det som menes med estimering i dette dokumentet. I vårt tilfelle vil det være hvis et fly forsvinner bak skyer og under tåkete forhold. Det er to hovedtyper med estimering som blir brukt, med kollisjon og uten kollisjon. For oss er kun uten kollisjon relevant, fordi fly aldri kommer til å kollidere, med kollidere her menes det da å sprette tilbake etter å ha truffet for eksempel en vegg.

Algoritmen vi har sett på i denne sammenheng heter "Kalman Filter". [5] Kalman Filteret gjør en gjetning/estimering på hvor objektet er når det ikke lenger er synlig. Kalman Filter fungerer kort sagt i flere totrinnsfaser. Første fase er å gjør en gjetning på hvor objektet er etter at det forsvinner. Andre fase er å korrigere en eventuell feil når objektet kommer tilbake, eventuelt registrere at gjetningen var riktig. Et av problemene med Kalman Filter er at det antar at systemet er lineært, noe svært få systemer er, spesielt virkelige systemer som inneholder tilfeldige variabler. Dette kan føre til enkelte feil når det gjøres gjetninger, men hvorvidt feilene er store nok til å være alt for misledende er ikke lett å si, spesielt uten at dette kan bli testet under realistiske forhold.

6.2.5 Konklusjon

Alle av algoritmene viser seg å ha et eller flere bruksområder, som kan brukes innenfor vår oppgave. Dette prosjektet ble løst ved å bruke bevegelsesdeteksjonsalgoritmen, da det ikke var verken tid, kunnskap eller nok bilder for å bruke Machine Learning algoritmen for å kjenne igjen objekter. Derfor diskriminerer ikke programmet på grunnlag av hvilken type objekt det er, den detekterer all bevegelse i et bilde. De resterende algoritmene kan også brukes i dette prosjektet som ekstrafunksjoner for å mer data og informasjon om et fly som kommer.

7. Software Moduler

7.1 Bildedeteksjon

Prosjektets hovedmål har gått ut på bildedeteksjon og å oppdage bevegelse i en videostrøm.

7.1.1 OpenCV

Systemet benytter seg av open-source bibilioteket OpenCV. OpenCV står for Open Source Computer Vision og inneholder programmer som omfatter bildedeteksjon og maskinlæring, slik at en datamaskin kan trenes opp til å gjenkjenne visse typer objekter. [5] Biblioteket har mer enn 2500 optimaliserte algoritmer, som kan brukes til å oppdage og gjenkjenne ansikter, identifisere objekter, klassifisere menneskelige handlinger i videoer, spore kamerabevegelser, spore objekter i bevegelse osv. I følge OpenCV sine egne sider har de over 47000 brukere og 9 millioner nedlastninger. Siste versjon av OpenCV kom ut i desember 2015 [6] og OpenCV blir oppdatert mest jevnlig av alle open source bibliotek i forhold til computer vision. Nettsidene deres har også en god organisering av dokumentasjon. De har også en rekke tutorials som beskriver hvordan funksjoner/programmer fungerer. Disse inneholder alt fra teorien bak, de matematiske formlene, kodeutklipp, osv.

7.1.2 Metoden

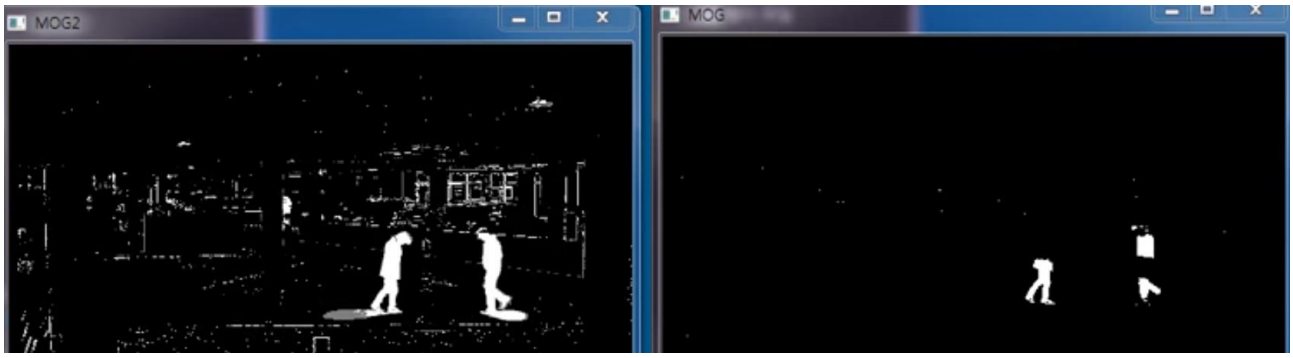
Målet med prosjektet var å lage et program som kunne kjenne igjen objekter, fortrinnsvis fly, men også forstå at for eksempel fugler og biler ikke skulle bli markert. Sluttprogrammet bruker hovedsakelig “Background Subtraction”, som i seg selv ikke kan kjenne igjen objekter. Derimot er det mulig å legge på filter som kan gjøre det, basert på noen av algoritmene nevnt i kapittel 5. Størrelse og form er noen av faktorene som spiller inn i de mest brukte algoritmene, men ved veldig små/fjerne objekter er dette ikke tilstrekkelig.

7.1.3 Sluttresultat

Som nevnt bruker programmet background subtraction, men grunnet startproblemer ble ikke løsningen utforsket godt nok i praksis. Løsningen viste seg å være velfungerende hvis objektet (flyet) var i samme størrelsesorden og form. Oppgaven ble løst, men ikke godt nok til at programmet kan anvendes i en situasjon med for mange forstyrrende elementer. Markering og fjerning av uønskede markeringer fungerer, men det vil være upraktisk å måtte fjerne hver enkelt feilmarkering for brukeren av programmet.

I dokumentet som forklarer metoder som kan brukes med background subtraction, [5] er det beskrevet om MOG (Mixture of Gaussians). Dette er metoden som har blitt brukt, selv om det finnes andre som for eksempel “Frame Differencing” osv. Det disse metodene gjør er å skille bakgrunn fra forgrunn, altså detekterer de bevegelse, men de skiller ikke mellom hva slags bevegelser som er relevante og irrelevante. Med andre ord er ikke dette en ekte måte å gjenkjenne objekter, men en måte å detektere forandring i et bilde.

Måten oppgaven ble løst på var ved å skille mellom størrelsen på objektene. Det vil si at programmet kunne skille mellom objekter som var for små til å være fly, uten at det var en maksimumsgrense på størrelse. Problemet med løsningen er at i likhet med Figur 8 (MOG2, venstre side), var for mye forstyrrelser i bildet. Grunnen til forstyrrelsene var forårsaket av “Compression artifacts”, det vil si forvrenginger og feil bildegjengivelse etter omgjøring av video. Dette gjorde at det dukket opp falske deteksjoner av bevegelse/forandringer i bildet. De falske forandringene som ble detektert gjorde det vanskelig å kunne detektere flyene lenge nok mens de var i bildet, sammenlignet med hvor lenge et menneske kunne se flyet selv.



Figur 11: MOG2 og MOG1

7.1.4 Konklusjon

Løsningen som ble brukt er optimal under klare forhold, det vil si at det eneste som beveger seg på bildet er enten fly, eller objekter som er for små til å detekteres/markeres av algoritmen. Hovedproblemet løsningen hadde var at den markerte for mye, neste steg hadde blitt å fjerne uønskede markeringer. Dette kunne blitt gjort etter veldig mye mer arbeid ved å bruke background subtraction, eller ved å bytte algoritmetype. Å bytte algoritmetype er mest sannsynlig den beste løsningen, fordi de fleste av algoritmene som nevnes i teknologidokumentet [5] ser på karakteristikker. Background subtraction gjør også dette, men i likhet med de andre algoritmene har vansker å bruke disse om objektet skifter både retning og størrelse.

En annen, men alt for tidkrevende og kompleks løsning, hadde vært å bruke “Machine Learning”. [5] Grunnen til at denne løsningen er for tidkrevende, er at en stor mengde bilder/video kreves av fly for å kjenne de igjen. Med andre ord måtte det bli tatt hundrevis av bilder av forskjellige fly, fra alle tenkelige vinkler. Det skal sies muligheten for å gjøre dette med kun ett fly for å bevise konseptet hadde vært en mulighet, men denne løsningen ville ikke vært holdbar på en ekte flyplass. Problemet er at det finnes for mange fly med forskjellige fasonger, størrelser, samtidig som disse kan komme fra forskjellige vinkler og er forskjellige avstander unna.

8. Hardware Moduler

8.1 Capture Card

For at systemet skal kunne ta inn en direkte videostrøm benytter vi et capture card av typen DeckLink Studio 4K. Dette gjør det mulig å ta imot direkte video fra både kamera og andre enheter som sender video ut via HDMI kabel og andre lignede kilder. [7]

Et capture card som er ment for datamaskiner kan kobles direkte inn i en PCIe-buss i maskinene. Kortet kan brukes til å ta opp video, men mer spesifikt brukes det i nesten alle sammenhenger med for eksempel redigering av høykvalitetsvideoer og filmer. Grunnen til dette er at et capture card gjør det mulig å jobbe med forskjellige typer videoformater og forskjellige typer oppløsninger. Fordi det er mulig å fange bilde/video rett fra enheter som sender et output signal, slik som kameraer, datamaskiner, spillkonsoler og andre videostrømmer, kan man modifisere/klippe denne videostrømmen direkte. [8]

Figur 12: DeckLink Studio 4K

Flere og flere kort kommer med verktøy som hjelper til å skape bedre visuelle effekter og fargekorreksjon. Dette er en av få hovedgrunner til at de brukes flittig i redigeringssammenheng. Den største fordelen med et capture card er at man jobber med den samme kvaliteten som blir tatt opp av kilden. Det vil si at en kilde som tar opp i 4K gjennom et kort vil da vise dette i samme kvalitet og oppløsning.

8.1.1 Hvorfor DeckLink?

Da det ble valgt capture card visste vi at vi kom til å trenge et kort som støtter både Windows og Linux operativsystemer, samt at det kan ta inn en 4K videostrøm. DeckLink Studio 4K kan installeres på forskjellige operativsystemer og det tar inn en 4K videostrøm på en HDMI-inngang.

Det ble gjort en del undersøkelser på dette temaet, da et kort som kan ta inn 4K vanligvis er dyrt, så det var viktig at det riktige kortet ble valgt. BlackMagic viste seg å være det beste merket at høyere kvalitet kort som funker på forskjellige operativsystemer.

8.1.2 Software Development Kit

I tillegg til selve capture cardet følger det også med et SDK (Software Devlopment Kit). Denne er også tilgjengelig på både Windows og Linux og er skrevet i C++. Det er lovlig å lage og selge modifiseringer av denne i forhold til Blackmagic Design sitt reglement. Dette gjør at vi står fritt til å disponere og bruke denne funksjonen i vårt system.

9. Verktøy

9.1 Microsoft Visual Studio

Prosjektet har benyttet Microsoft Visual Studio som er en IDE (Integrated Development Environment) fra Microsoft. Visual Studio brukes til å utvikle dataprogrammer for Microsoft Windows, samt nettsider, webapplikasjoner og webtjenester. Det kan produsere både egen kode og forvalte kode. Den har også en integrert debugger som fungerer både på kildekodenivå og på maskinnivå. Andre innebygde verktøy inkluderer en designer for å bygge GUI-applikasjoner, web designer, klasse designer og databaseskjema designer. Visual Studio støtter ulike programmeringsspråk og code editoren og debuggeren støtter nesten alle programmeringsspråk. Innebygd språk inkluderer C, C ++ og C ++ / CLI, VB.NET, C #, og F #. Støtte for andre språk som Python, Ruby, Node.js, og M blant annet er tilgjengelig via språktjenester som installeres separat. Den støtter også XML / XSLT, HTML / XHTML, Javascript og CSS. [9]

9.2 Mercurial(Hg)

Som versjonskontrollsystem har vi benyttet Mercurial(Hg), dette er et gratis versjonskontrollverktøy som håndterer flere versjonsfiler i både store og mindre prosjekter. [10] Mercurial er fordelt på alle klientmaskiner slik at hver utvikler har en lokal kopi av hele utviklingshistorien. På denne måten fungerer det uavhengig av nettverkstilgang og en sentral server. Mercurial fungerer på alle større plattformer, og er først og fremst et kommandolinjedrevet program, men det finnes også utvidelser for grafisk brukergrensesnitt. [11]

9.3 Doxygen

For å dokumentere kode, har vi benyttet programmet Doxygen. Dette er en dokumentasjons-generator, og er et verktøy for å skrive dokumentasjon og referanser til systemet eller programmet. [12] Genereringen av dokumentasjonen er fra kommentert C ++ kildekode, men den støtter også andre populære programmeringsspråk som C, Objective-C, C #, PHP, Java, Python, IDL, Fortran, VHDL, og Tcl. Det kan generer dokumentasjon i en nettleser i HTML og/eller lage en referansehåndbok ut ifra den dokumenterte kildekoden. Dokumentasjonen er hentet direkte fra koden, noe som gjør det mye enklere å holde dokumentasjonen oppdatert og i samsvar med kildekoden. [13]

10. Testrapport

Testrapporten beskriver resultatet av hver test som er gjennomført. Beskrivelsen inneholder resultatet av testen, samt en evaluering om det enkelte kravet er godkjent eller ikke.

10.1 Rapport Test ID 1

Systemet skal detektere bevegelse fra en videostrøm.

Resultat: Testen viser at alle objektene som er i bevegelse i videostrømmen blir detektert. Det kan være litt forsinkelser ettersom algoritmen finner bevegelsen ved å sammenligne med bakgrunnen, og ved oppstart må den finne ut hva som er bakgrunnen.

Konklusjon: Etter å ha sett på resultatet av testen kan vi fastslå at kravet om at systemet skal detektere all bevegelse fra en videostrøm er godkjent.

10.2 Rapport Test ID 12

Systemet skal ikke detektere værelementer.

Resultat: Testen viser at ingen værelementer i videostrømmen blir detektert og markert. Hadde testen vært utført med ekstremvær kan testen ha fått et annet resultat, men så lenge værelementene ikke beveger seg for mye i forhold til bakgrunnen vil ikke dette være et problem.

Konklusjon: Etter å ha kjørt testen på flere videoer og ingen værelementer blir detektert i disse kan vi si at kravet er godkjent.

10.3 Rapport Test ID 13

Systemet skal ikke detektere skygger.

Resultat: Testen viser at skyggen til objektet blir markert sammen med selve objektet.

Konklusjon: Etter å ha kjørt testen ser vi at skyggen til objektet er med i markeringen av objektet, og testen, samt kravet er derfor ikke godkjent.

10.4 Rapport Test ID 3

Systemet skal markere fly.

Resultat: Testen viser at fly både på bakken og i lufta blir markert.

Konklusjon: Etter å ha kjørt testen ser vi at alle fly blir markert så lenge de er i bevegelse. Kravet om markering av fly er derfor godkjent.

10.5 Rapport Test ID 14

Systemet skal markere fly med en grønn firkant rundt flyet.

Resultat: Testen viser at fly både på bakken og i lufta blir markert med en grønn firkant rundt flyet.

Konklusjon: Etter å ha kjørt testen ser vi at alle fly blir markert med en grønn firkant rundt flyet så lenge de er i bevegelse. Kravet om markering med en grønn firkant av fly er derfor godkjent.

10.6 Rapport Test ID 15

Systemet skal ikke markere fly som står stille etter landing.

Resultat: Testen viser at fly som står stille eller har beveget seg og stoppet, og stått stille en stund, ikke blir markert.

Konklusjon: Etter å ha kjørt testen ser vi at flyene ikke blir markert når de ikke har vært i bevegelse og stoppet. Kravet er dermed godkjent.

10.7 Rapport Test ID 16

Brukeren skal kunne skru av/på individuelle markeringer.

Resultat: Systemet gir objektene hver sin ID, og vi kan utfra disse velge å skru av eller på markeringene. Tasten "0" skrur av alle markeringer, og objektene med ID fra 1-9 blir skrudd av med tilsvarende talltast.

Konklusjon: Etter å ha kjørt testen ser vi at vi kan skru av og på markeringene så lenge vi er innenfor IDer fra 1-9 eller at alle blir skrudd av på en gang. Kravet er dermed delvis godkjent på grunn av at kun individuelle IDer fra 1-9 kan skrus av.

10.8 Rapport Test ID 5

Videostrømmen må være minimum 4096 x 2160(4K).

Resultat: Systemet tar inn videopakker på 4096 x 2160 (4K).

Konklusjon: Systemet oppfyller kravet og er godkjent.

10.9 Rapport Test ID 17

Systemet skal prosessere 5 bilder per sekund (5 fps).

Resultat: Systemet gjør alle nødvendige utregninger for å opprettholde 5 bilder per sekund, men på grunn av fremvisningsmetoden blir dette senket noe under kjøring.

Konklusjon: Systemet oppfyller kravet og er godkjent.

10.10 Rapport Test ID 18

Systemet skal prosessere på en direkte videostrøm.

Resultat: Systemet tar inn og prosesserer en videostrøm direkte.

Konklusjon: Systemet oppfyller kravet og er godkjent.

10.11 Rapport Test ID 6

Systemet skal kunne kjøre på operativsystemet Windows.

Resultat: Systemet fungerer på operativsystemet Windows.

Konklusjon: Systemet oppfyller kravet og er godkjent.

10.12 Rapport Test ID 9

Systemet må ha en playbackfunksjonalitet for testing.

Resultat: Systemet kan starte en lokal fil for testing, men har ingen GUI for å stoppe eller starte mens programmet kjører.

Konklusjon: Systemet oppfyller delvis kravet og er derfor delvis godkjent da man kan spille av en lokal fil hvis man spesifiserer den før man kompilerer programmet.

10.13 Krav som ikke er testet

ID 5: Systemet skal være skrevet i programmeringsspråket C++

- Hele systemet er skrevet med programmeringsspråket C++, og kravet er derfor verifisert og godkjent.

ID 5.2: Systemets kildekode skal dokumenteres med Doxygen.

- All kildekodedokumentasjon er beskrevet med programmet Doxygen, og kravet er derfor verifisert og godkjent.

ID 6: Systemet skal bruke versjonskontrollsystemet Mercurial(Hg).

- Under hele prosjektet har vi benyttet oss av Mercurial som versjonskontrollsystem, og kravet er derfor verifisert og godkjent.

11. Etteranalyse

Som resultat av dette prosjektet har vi et system som detekterer bevegelse og markerer fly med en grønn boks. Det er i tillegg en funksjon for å skru av og på markeringen.

Det var i utgangspunktet tenkt at utviklingen av systemet skulle skje på Linux maskiner, men etter mye utfordringer i forbindelse med installasjoner av forskjellig programvare, ble gruppa og oppdragsgiver enig om at vi gikk over til Windows for å få fortgang i prosessen.

Gjennom prosjektarbeidet fulførte vi mange av målene og kravene som var satt. Det endelige systemet detekterer og markerer fly direkte i en 1080p videostrøm.

11.1 Arbeidsprosessen

Som prosjektmodell valgt vi å bruke Unified Process, ettersom det er en iterativ og inkrementell utviklingsprosess. Den fokuserer på systemarkitekturen og de mest kritiske risikoene i prosjektets livssyklus. For oss ble det mye uforutsett arbeid i starten slik at vi kom litt etter planen. Vi hentet dette inn igjen i løpet av prosjektet. Vi oppdaterte hele tiden prosjektplanen slik at den alltid skulle reflektere hvor langt i prosessen vi var kommet. Siden vi var noe på etterskudd på planen i starten innså vi at vi måtte legge om litt og arbeide litt mer effektivt. Dette ble veldig synlig etter andre presentasjonen da vi innså at vi hadde lagt inn litt vel få timer så langt i prosjektet. Etter andre presentasjon økte vi derfor innsatsen slik at den skulle komme opp på litt riktigere høyde.

11.2 Samarbeid

Ettersom vi har vært en relativt liten gruppe har vi jobbet tett sammen, og vi har tatt de fleste avgjørelser nesten umiddelbart og som en gruppe. Avgjørelsene som ble tatt ble gjort på en demokratisk måte, hvor flertallet bestemmer. Om resultatet ble uavgjort, var det leder som avgjorde utfallet. Det har vært mange bra ideer hvor vi har diskutert rundt forskjellige løsninger, men vi har stort sett vært enig om sluttresultatet. Alle i gruppen har fått kommet med sine ideer og meninger, men alltid akseptert avgjørelsene som ble tatt.

11.3 Veiledere

Veilederne i prosjektet har blitt oppdatert underveis i prosjektet og vi har hatt ukentlige møter med både intern og ekstern veileder. De har også vært lett tilgjengelige på mail. Intern veileder, Sigmund Gudvangen er førsteamanuensis på høyskolen og har sitt kompetansefelt innen signalbehandling og audioteknologi. Han har vært til hjelp med prosjektprosessen, samt hva som forventes av dokumentasjon og presentasjonen. Han har hatt et stort fokus på rapporter og hvordan disse skal utformes. Ekstern veileder har vært Alexander Gosling fra Kongberg Defence Systems. Sammen med Kjetil Mikkelborg og Tore Langmoen har de vært til hjelp både med tekniske spørsmål og hva som var forventet av systemet.

11.4 Individuell Refleksjon

11.4.1 Heidi Troppen

I løpet av dette prosjektet har jeg lært mye om hvordan man styrer et prosjekt og veldig mye om hvordan man ikke styrer et prosjekt. Jeg har lært mye om hvordan man installerer forskjellig software på Linux og hvordan man installerer samme software igjen på Windows. I løpet av prosjektet har jeg lært mye om bevegelsesdeteksjon og hvordan man implementerer det med andre komponenter. Har også lært hvordan man bruker en versjonskontroll riktig. I

prosjektstyringsdelen av prosjektet har jeg lært mye hvordan man lager og opprettholder dokumenter gjennom et prosjekt, og hvordan man refererer korrekt i en stor publikasjon som denne. Med framføringene som har blitt gjort har jeg lært at man alltid skal trippelsjekke noe som blir vist offentlig og hvordan man holder en god presentasjon med andre. Gjennom alle utfordringene gruppen har hatt sammen har jeg lært mye hvordan man styrer en gruppe som har mistet noe av motivasjonen på grunn av mye motgang, og hvordan på få denne motivasjonen tilbake. Alt i alt har dette vært en lærerik prosess med mange nye elementer som man ikke har hatt tidligere i utdanningen.

11.4.2 Silje Ågren Aas

Prosjektet har vært en lærerik prosess, hvor jeg har lært mye om prosjektstyring, dokumentasjon og ikke minst bildebehandling. Jeg har fått en mye bedre forståelse for prosjektmodeller og hvordan man planlegger og gjennomfører et prosjekt. I forhold til dokumentasjonen har jeg lært mye om hvordan man skal skrive en rapport og hvor viktig det er å beskrive alle deler av et prosjekt. Vi har benyttet bildegjenkjenningsbiblioteket OpenCV, hvor jeg har lært hvordan jeg kan bruke biblioteket til å detektere bevegelse og ikke minst hvordan bevegelsen detekteres. I tillegg til dette har jeg lært om hvordan det er å jobbe som en gruppe mot et felles mål, opp mot en oppdragsgiver.

11.4.3 Dennis Aurbakken

Under dette prosjektet her jeg fått erfare mer hvordan det er å jobbe i et litt større prosjekt med en gruppe fra starten av utformingen til slutten hvor ting begynte å ta form. I løpet av perioden har jeg lært meg mer om hvordan bildebehandling virker og hvordan et capture card kan bli brukt i egne applikasjoner. Jeg har også fått sett litt mer hvordan man kan flette sammen flere forskjellige biblioteker for å jobbe sammen samt hvordan man kan buke tredjeparts verktøy for å hjelpe til med for eksempel dokumentering av kildekode.

11.4.4 Aleksander Lillevold

Prosjektet bygger videre på informasjon og erfaring som vi fikk via Systems Engineering, slik at vi hadde et grunnlag for hvordan vi skulle arbeide. Jeg har lært mye om hvordan man bør delegere tid og hvordan man bør starte et prosjekt. Samtidig har jeg også lært mye om hvordan man kan undersøke/researche informasjon på en riktig og effektiv måte, slik at denne informasjonen kan brukes i prosjektet på best mulig måte. Det var slik jeg lærte mer om biblioteket vi brukte, OpenCV, men også om konkurrentene og funksjonene som var tilgjengelig hos de forskjellige bibliotekene. Jeg har også lært at det er viktig å undersøke hva slags alternativer man har, og hva de forskjellige fordelene og ulempene alternativene har, slik at man kan ta et valg om hva som er best for en gitt situasjon.

11.5 Samlet refleksjon

Dette prosjektet har vært veldig lærerikt og har gitt gruppa en god pekepinn på hvordan et reelt prosjekt vil bli gjennomført. Gruppa har utviklet seg mye når det kommer til ferdigheter og kunnskap for generelt prosjektarbeid og samarbeid innad i gruppa. Prosjektet har bydd på forskjellige utfordringer underveis, hvor vi har måtte tilegne oss ny kunnskap og nye erfaringer. Gruppa er fornøyd med samarbeidet og resultatet av prosjektet, og alt i alt har det vært utfordrende, spennende og ikke minst lærerikt prosjekt.

12. Konklusjon

12.1 Utfordringer

Den største utfordringen til selve systemet er å kunne identifisere hvilke type objekt som blir markert. Dette er noe som både er vanskelig og tidkrevende å få til. Videre har gruppa gjennom prosjektet hatt ulike utfordringer rettet mot installasjon av programvare og biblioteker. [2] Ettersom mye av programvaren baserer seg på oppdateringer fra internett og i tillegg til at gruppa ikke hadde noe særlig erfaring med “standalone” installasjoner fra tidligere, førte dette til nye utfordringer.

12.2 Oppsummering

Prosjektets hovedmål var å markere fly visuelt i en videostrøm. Systemet som er utviklet detekterer og markerer fly i en videostrøm, men det markerer også andre objekter som ikke er fly. Mange av de opprinnelige kravene som ble stilt til systemet har blitt oppfylt, men det er allikevel en del som gjenstår får at systemet skal kunne skille mellom ulike objekter i videostrømmen.

For at systemet skulle ta inn en direkte videostrøm fra en 4K kilde, implementerte vi et capture card i systemet. I tillegg til dette gir systemet de forskjellige objektene i videostrømmen unike IDer. Dette implementerer funksjonen som gjør det mulig å skru av og på markeringen av objektene. Gruppa har ikke lyktes med å utvikle noe user interface ettersom det oppsto problemer med samarbeidet mellom Qt og OpenCV. [2] Dette løste vi ved å binde kommandoer direkte til tastaturet istedenfor knapper på skjermen.

Gjennomføringen av selve prosjektet ble for gruppen en læringsprosess hvor vi fikk erfare at det er viktig med en god plan og oversikt over arbeidet som skal gjøres. Dette hadde vi ikke nok erfaring med fra før og vi innså ut i prosjektet at vi ikke hadde lagt en detaljert nok plan for prosjektet. Tidsforbruket ble derfor kraftig økt etter presentasjon 2 da vi innså at vi ikke hadde lagt ned nok arbeid i prosjektet.

Ettersom vi hadde en del utfordringer knyttet til oppstartsfasen og installasjonen av de forskjellige programmene, kom vi derfor sent i gang med arbeidet. Til tross for dette kom vi sterkere tilbake og fikk lagt ned mer arbeid både når det gjaldt prosjektstyringen og utviklingen av systemet. Alt tatt i betraktning føler vi at vi har levert et godt produkt med tanke på tidligere kunnskap og erfaring.

13. Referanser

[1] H. Troppen, S. Å. Aas, D. Aurbakken og A. Lillevold, «Timelister G13-TL,» HSN, Kongsberg, 2016.

[2] H. Troppen, S. Å. Aas, A. Lillevold og D. Aurbakken, «Installasjonsdokument, G13-ID,» HSN, Kongsberg, 2016.

[3] KDS, «Bacheloroppgave Prosjekt Remote Tower Solution,» 2015-12-01.

[4] H. Troppen, S. Å. Aas, D. Aurbakken og A. Lillevold, «Kodedokumentasjon, G13-KD,» HSN, Kongsberg, 2016.

[5] H. Troppen, S. Å. Aas, D. Aurbakken og A. Lillevold, «Teknisk Dokument, G13-TD,» HSN, Konsberg, 2016.

[6] Itseez, «OpenCV Downloads,» [Internett]. Available: http://opencv.org/downloads.html. [Funnet 15 04 2016].

[7] Blackmagic Design, «DeckLink,» Blackmagic Design, [Internett]. Available: https://www.blackmagicdesign.com/products/decklink. [Funnet 04 05 2016].

[8] N/A, «TV Tuner Card,» Wikipedia, [Internett]. Available: https://en.wikipedia.org/wiki/TV_tuner_card#Video_capture. [Funnet 04 05 2016].

[9] N/A, «Microsoft Visual Studio,» Wikipedia, [Internett]. Available: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio. [Funnet 04 05 2016].

[10] A. Babenhauserheide, «Mercurial Source Control Management,» [Internett]. Available: https://www.mercurial-scm.org/about. [Funnet 04 05 2016].

[11] N/A, «Mercurial,» Wikipedia, [Internett]. Available: https://en.wikipedia.org/wiki/Mercurial. [Funnet 04 05 2016].

[12] N/A, «Doxygen,» Wikipedia, 29 12 2015. [Internett]. Available: https://no.wikipedia.org/wiki/Doxygen. [Funnet 04 05 2016].

[13] Doxygen, «Doxygen,» Doxygen, [Internett]. Available: http://www.stack.nl/~dimitri/doxygen/. [Funnet 04 05 2016].

[14] Itseez, «OpenCV About,» [Internett]. Available: http://opencv.org/about.html. [Funnet 15 04 2016].

[15] A. Kaehler og G. Bradski, Learning OpenCV, O'Reilly Media, 2008.

14. Vedlegg

14.1 Vedlegg 1: Aktivitetsliste

ID	Aktivitet	Beskrivelse	Krav ID	Ansvarlig
1	Møter	-	N/A	HT
2	Oppsett av utviklermiljø	-	N/A	HT
3	Prosjektplanlegging	-	N/A	HT
4	Kravspesifikasjon	-	N/A	SÅA
5	Testspesifikasjon	-	N/A	AL
6	Nettside	-	N/A	DA
7	Presentasjon	Arbeid med presentasjon	N/A	Alle
8	Dokumentasjon	-	N/A	SÅA
9	Filming	Filmet biler for å ha noe å teste programmet på	N/A	SÅA/DA
10	Research	-	N/A	Alle
20	Generell programmering	Programmering som ikke faller under en aktivitet	1-3	Alle
21	UML	Lag og bruk av UML	1-3	N/A
22	Struktur	-	1-3	N/A
23	UI	UI programmering	10	N/A
24	Algoritme-research	-	1-2	N/A
25	Biblioteksimplementering	Implementasjon av biblioteker	1-2	N/A
26	Algoritmer	Lag og bruk av algoritmer.	1-3	N/A
27	Feilsøking/fiksing	-	1-3	N/A
28	Kodedokumentasjon	Dokumentasjon av kode	N/A	N/A

14.2 Vedlegg 2: Testene

Tabell 29: Test 1

Test ID	**1**
Krav ID	1
Beskrivelse	All bevegelse skal detekteres
Akseptkriterier	All bevegelse skal kunne oppdages, selv om bevegelsen ikke er relevant for sluttresultatet (det vil si et fly i bevegelse). Bevegelsen(e) skal være fra en videostrøm
Testtype	Funksjonell
Status	Testet
Godkjent	Ja

Tabell 30: Test 12

Test ID	**12**
Krav ID	1.1
Beskrivelse	Vær og vind skal ikke markeres
Akseptkriterier	Værelementer som for eksempel skyer, regn og sol skal ikke bli markert.
Testtype	Funksjonell
Status	Testet
Godkjent	Ja

Tabell 31: Test 13

Test ID	**13**
Krav ID	1.2
Beskrivelse	Skygger skal ikke markeres
Akseptkriterier	Skygger, uansett hvilket objekt de tilhører, skal ikke markeres
Testtype	Funksjonell
Status	Testet
Godkjent	Nei

Tabell 32: Test 2

Test ID	**2**
Krav ID	2
Beskrivelse	Når et objekt detekteres, skal de bli klassifisert som en type av et objekt
Akseptkriterier	Programmet skal kunne skille mellom forskjellig type objekter. Det vil si at en gruppe objekter er for eksempel fugler, biler og mennesker
Testtype	Funksjonell
Status	Ikke testet
Godkjent	-

Tabell 33: Test 3

Test ID	**3**
Krav ID	2.1
Beskrivelse	Fly skal markeres
Akseptkriterier	Fly skal markeres visuelt slik at de blir lettere å få øye på
Testtype	Funksjonell
Status	Testet
Godkjent	Ja

Tabell 34: Test 14

Test ID	**14**
Krav ID	2.1.1
Beskrivelse	Se om markeringen av flyene er en grønn firkantet boks
Akseptkriterier	Flyet skal bli markert med en grønn firkant rundt
Testtype	Funksjonell
Status	Ikke testet
Godkjent	-

Tabell 35: Test 15

Test ID	**15**
Krav ID	2.3
Beskrivelse	Fly som ikke er i bevegelse/står i ro, skal ikke markeres
Akseptkriterier	Fly som ikke har vært i bevegelse på en stund skal ikke markeres
Testtype	Funksjonell
Status	Testet
Godkjent	Ja

Tabell 36: Test 16

Test ID	**16**
Krav ID	2.4
Beskrivelse	Brukeren skal kunne skru av/på individuelle markeringer
Akseptkriterier	Alle markeringer som er synlige, skal kunne skrus av/på. Dette gjelder for fly som blir markert, men også eventuelle falske positiver
Testtype	Funksjonell
Status	Testet
Godkjent	Delvis

Tabell 37: Test 5

Test ID	**5**
Krav ID	3
Beskrivelse	Oppløsning må være minst 4K
Akseptkriterier	Videostrømmen som kommer inn til applikasjonen skal være på 4096 x 2160 (4K)
Testtype	Funksjonell
Status	Testet
Godkjent	Ja

Tabell 38: Test 17

Test ID	**17**
Krav ID	3.1
Beskrivelse	Programmet prosesserer 5 bilder per sekund
Akseptkriterier	Alt fungerer korrekt samtidig som kun 5 bilder per sekund vises
Testtype	Funksjonell
Status	Testet
Godkjent	Ja

Tabell 39: Test 18

Test ID	**18**
Krav ID	3.2
Beskrivelse	Programmet skal fungere på en direkte videostrøm
Akseptkriterier	Vi skal kunne ta inn en direkte videostrøm og systemet skal prosessere denne.
Testtype	Funksjonell
Status	Testet
Godkjent	Ja

Tabell 40: Test 6

Test ID	**6**
Krav ID	4.1
Beskrivelse	Programmet skal fungere på Windows 7
Akseptkriterier	Alt som lages i forhold til systemet skal kunne kjøre på et Windows operativsystem
Testtype	Ikke-funksjonell
Status	Testet
Godkjent	Ja

Tabell 41: Test 9

Test ID	**9**
Krav ID	10
Beskrivelse	Sjekke funksjonaliteten til playbackfunksjonen
Akseptkriterier	Spille av, pause, og stoppet videoen, samt velge hvor i videoen avspillingen skal starte
Testtype	Funksjonell
Status	Testet
Godkjent	Delvis

KONGSBERG

Høgskolen i Sørøst-Norge

Teknisk Dokument

Remote Tower Solution

Rev	Endring	Dato	Utarbeidet av	Godkjent av	
0.1		10.03.16	Aleksander Lillevold		
1	Siste gjennomgang	19.05.16	Silje Ågren Aas		

Tabell 1: Endringslogg

Innholdsfortegnelse

Liste over tabeller

Liste over figurer

1. Introduksjon

Bildegjenkjenning blir mer og mer brukt i daglige scenarioer. Behovet for maskiner som kjenner igjen forskjellig type objekter eller mennesker øker ekstremt slik samfunnet er i dag. Derfor finnes det veldig mange forskjellige løsninger for å prosessere video og bilder. Disse løsningene har positive og negative sider, og for å kunne holde oversikt på løsningene, valgte vi å lage dette tekniske dokumentet. Undersøkelsene våre har aktivt blitt utført fra starten av prosjektet, og fortsatt slik at man alltid vet den mest optimale løsningen på eventuelle utfordringer man kan støte på.

1.1 Dokumenthistorie

0.1: Samlet alle tekniske dokumenter til ett hoveddokument med felles innledning og konklusjon.

1: Siste gjennomgang før innlevering.

2. Maskinlæring og Algoritmer

2.1 Maskinlæring

I forhold til algoritmene som kommer til å bli brukt og vurdert, må de læres opp. Hovedmålet med maskinlæring er å gjøre om data til informasjon. Dette gjøres ved at det er en stor kolleksjon av data, for eksempel 3000 “positive” bilder og 300 “negative” bilder. Hvis maskinen skal læres opp til å se fly, vil de 3000 positive bildene inneholde fly, mens de 300 andre ikke vil ha det.

Maskinen kan også læres opp til å kjenne igjen forskjellige attributter ved objektet. For eksempel er det mulig for algoritmen å se forskjell på type fly, selskapet som eier flyet (gitt at de for eksempel har en logo på flyet) og lignende. Dette krever derimot forskjellige typer klassifikatorer som har forskjellige bruksområder. I vårt tilfelle er det ikke relevant hvilket selskap eller hva slags type fly det er, kun om objektet som blir markert er et fly.

Under maskinlæringsprosessen vil maskinen bestemme hvilke trekk som er relevante for å finne objektet, eller detaljen i objektet vi leter etter. Hvis klassifikatorene ikke klarer å se en stor nok forskjell på for eksempel fly og fugler, kan vi legge til trekk i dataen vår eller bruke en annen klassifikator.

2.1.1 Haar-Like Features

Navnet kommer av “Haar Wavelets” som på norsk betyr Haar bølger/sekvenser, som originalt er en matematisk sekvens av omskalerte firkantformede funksjoner, som tilsammen danner en basis. Disse firkantformede funksjonene er selve grunnlaget for hvordan "Haar-Like Features" brukes til bildegjenkjenning.

Bildegjenkjenning er svært dyrt i forhold til kalkulasjoner som skal gjøres av en datamaskin, spesielt hvis man hovedsakelig arbeider med farger og intensiteten til fargene på bildet. Grunnen til dette er at hver enkelt piksel får en tallverdi, og hvis alle disse pikslene skal sammenlignes, kreves det veldig mye tid og kalkulasjoner av datamaskinen. "Haar-Like Features" gjør denne jobben veldig mye lettere ved å lage rektangulære områder så store som ønskelig, og tar heller fargeintensiteten på områder og sammenligner disse. Enkelte områder står også lett ut på noen objekter, som for eksempel ansikter hvor øynene er svært annerledes fra andre trekk som for eksempel kinnene våre.

Selv om det er mange fordeler relatert til hastighet med "Haar-Like Features", har den svært store ulemper: Algoritmer som bruker "Haar-Like Features" har vansker for å lære og ofte kreves mange av disse "Haar-Like Features" (Haar “områder” /rektangler) for å beskrive et objekt godt nok. Disse ulempene har mye å si når objektet er langt unna, eller om bildekvaliteten gjøres lav for å øke hastigheten på programmet. I vårt tilfelle kjører vi kun 5 bilder per sekund, men bildekvaliteten er 4K, som har sine positive og negative sider. Det mest negative med 4K er at formatet er vanskelig å arbeide med, samtidig som det krever med prosesseringskraft av datamaskinen. Derimot blir resultatet mer treffsikkert av at vi bruker 4K, fordi bildene er tydeligere/klarere enn det lavere oppløsning ville vært. Sist, men ikke minst er det vanskeligere å beskrive og sammenligne objekter som er skrå over en vinkel av 45 grader. I vårt tilfelle er ikke dette for relevant, siden fly sjeldent skrår mer enn 45 grader med mindre det er noe alvorlig galt. [1]

2.1.2 Local Binary Patterns

Som navnet tilsier, bruker "Local Binary Pattern" binærmønstre for å kunne sammenligne og gjenkjenne objekter i bilder. Dette gjøres ved at et bilde blir delt opp i celler, som igjen er delt opp i piksler. Pikslene blir sammenlignet med sine (lokale) naboer, og fordi pikslene blir delt opp i

rutenett, har alle pikslene 8 nærmeste naboer. Om pikselen i sentrum har høyere verdi enn en nabo, skrives 0, og hvis naboen er høyere skrives 1. Dette danner en byte (8 bit) med informasjon, for eksempel “0000 1000” hvis 7 av naboene har lavere verdi og en har høyere.

Hvis vi har en celle som er for eksempel 16x16 piksler, vil vi ende opp med en 256-dimensjonal vektor fordi vi må ha en vektor per piksel. I "Local Binary Pattern" brukes trekk som gjør følgende for å kutte ned på vektorstørrelsen: Hvis binærmønsteret har to overganger, det vil si alle utenom ett tall er 0, eller 1, (eks 0000 0001, 1111 0111) kalles dette mønsteret for uniformt. Uniforme og ikke-uniforme mønstre deles så opp i hver sin gruppering, og disse gruppene gjør det mulig å redusere vektorene fra 256, helt ned til 59 stykker.

"Local Binary Pattern" lærer svært raskt fordi den anvender "Integers", noe som blant annet "Haar-Like Features" ikke gjør. "Local Binary Pattern" har også en del utvidelser som enten gjør beskrivelsen raskere eller mer treffsikker. Ulempen med å gjøre beskrivelsene raskere er at den blir mindre treffsikker og motsatt hvis utvidelsen gjør den mer treffsikker. [2]

2.2 Algoritmetyper

Det finnes flere måter algoritmene kjenner igjen objekter på, men felles for algoritmene er at de kan deles inn i en egen gruppe. Gruppene har forskjellige måter å kjenne igjen bilder på, som forklares under. Merk at det er flere algoritmer i nesten hver eneste gruppe, men at forskjellene går på matematiske formler og hvilket bruksområde de har. Merk også at de fleste bildene er tatt i optimale situasjoner, noen ganger uten bevegelse i bildet. Algoritmene er innenfor hovedområdet som kalles “Feature Detection”. To av hovedgruppene ("Affine Invariant Feature Detection" og "Feature Description") er mindre relevante for oss, sammenlignet med de andre, så disse har vi valgt å ignorere. [3]

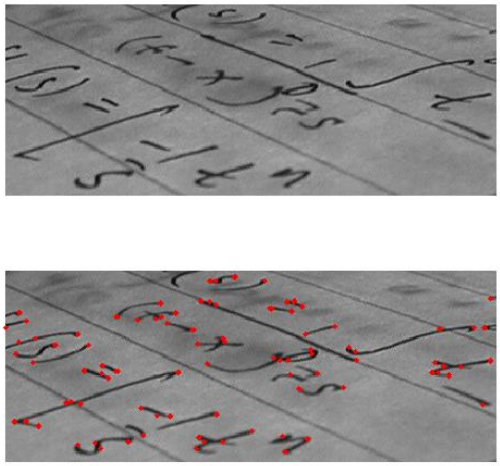
2.2.1 Kantdeteksjon

Kantdeteksjon ser etter skarpe forandringer i farge, eller at fargen stopper opp. På eksempelbildet er dette svært synlig, men hvis objektet beveger seg kan denne algoritmen ha enkelte problemer. Støy i form av lys og skygge spiller også en omfattende rolle, siden det kan føre til at algoritmen finner falske kanter. [4]

Figur 1: Kantdeteksjon

2.2.2 Hjørnedeteksjon

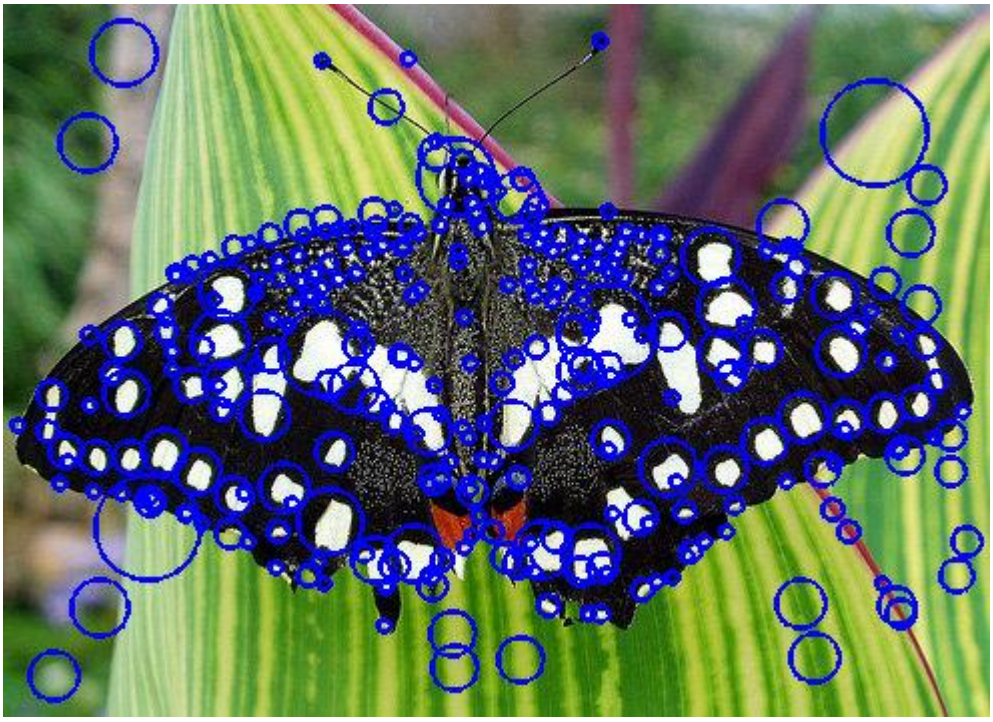
Hjørnedeteksjon som navnet tilsier, leter etter hjørner i bilder. For eksempel vil bokstaven I ha to hjørner ved bruk av denne logikken, ett hjørne på start og sluttpunktet (øverst og nederst). Denne algoritmen kan kjenne igjen blant annet bokstaver, men krever ofte at den blir lært opp og/eller at den har en referanseliste på hva den skal kjenne igjen. Denne algoritmen brukes ofte i kombinasjon med parkeringshus for eksempel for å kunne kjenne igjen nummerskiltene / kjennemerkene til biler. [5]



Figur 2: Hjørnedeteksjon

2.2.3 Blobbdeteksjon

Blobbdeteksjon ser etter regioner (grupper, "blobber") hvor egenskapene i bildet er konstante, eller nesten konstante. Det er mulig å justere hvor store blobber algoritmen skal se etter, noe som øker/minsker treffsikkerheten. Dette bildet har 20 forskjellig størrelser på blobber, og det gjør det lett å se at største blobbene er de minst treffsikre. [6]



Figur 3: Blobbdeteksjon

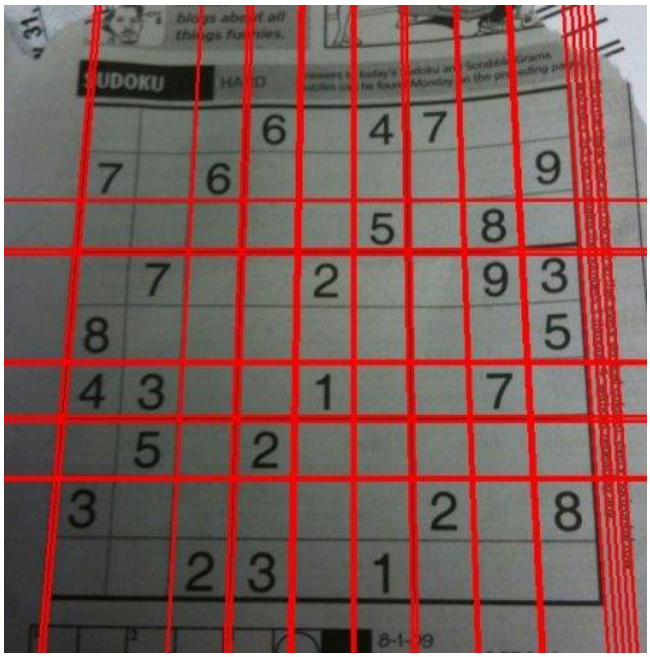
2.2.4 Ridge Detection

"Ridge Detection" leter etter kanter, for så å detektere “midten” / “forhøyningen” i disse. Dette er kanskje best forklart ved hjelp av det bilde. Algoritmen høres svært lik kantdeteksjon men fungerer i praksis litt annerledes, noe som kan sees av områdene som markeres. Bruken av "Ridge Detection" er ikke like utbredt som kantdeteksjon, spesielt i bilder tatt fra kun en vinkel, av ett kamera. Grunnen til dette er simpel; Det er svært vanskelig å se dybde fra kun ett bilde, samtidig som datamaskiner ikke “oppfatter” dybde på samme måte som mennesker gjør. [7]

Figur 4: Ridge Detection

2.2.5 Hough Transform

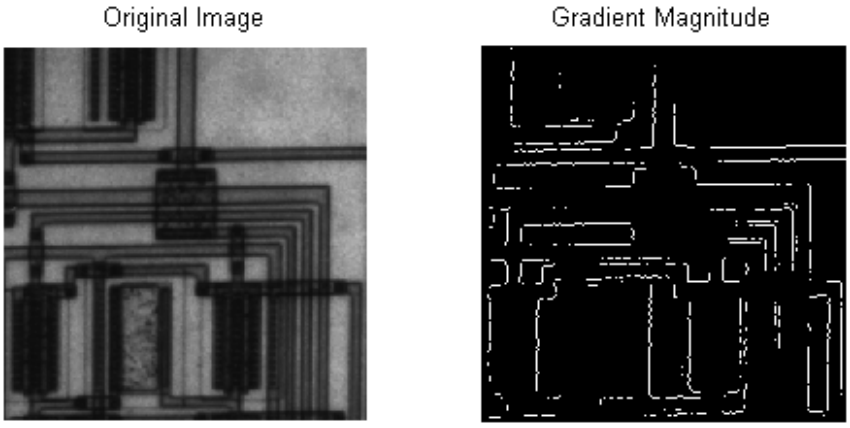
"Hough Transform" brukes til å finne rette linjer i et bilde, for eksempel hvis et A4 ark er tatt bilde av, vil algoritmen finne alle sidene (gitt at arket ikke er krøllet o.l.). Algoritmen kan også se forskjellige former, men linjer, sirkler og ellipser er mest brukt. [8]



Figur 5: Hough Transform

2.2.6 Structure Tensor

"Structure Tensor" som navnet hinter til, brukes denne til å se strukturen til objekter. Med dette menes at den lettere ser mønstre som for eksempel fingeravtrykk, kretskort, osv. Algoritmen ser etter linjer som har en logisk sammenheng, hvor den logiske sammenhengen kan defineres av klassifikatoren(e), som igjen kan defineres av den som lager programmet. Logisk sammenheng kan med andre ord bety “Alle L former” eller for eksempel spiralformede linjer. Også denne algoritmen kan minne om kantdeteksjon, men "Structure Tensor" bruker gradientinformasjon (“kantinformasjon”) for å finne områdene den leter etter. [9]



Figur 6: Structure Tensor

2.2.7 Scale Space

"Scale Space" anvendes for å kunne danne seg et bilde av et objekt fra en viss avstand. For eksempel hvis objektet er langt unna, kan objektet se ganske annerledes ut enn hvis det er rett foran kameraet. Ved å bruke "Scale Space" kan et originalbilde bli transformert til å se ut som det er mer uklart, som blir realiteten hvis objektet i bildet er lengre unna. Dette er derfor nyttig hvis man kommer til å få uklare bilder, som for eksempel fly - hvor formen blir mer og mer uklar desto lengre unna det er. Variabelen t er en parameter som bestemmer hvor langt unna objektet skal være. [10]

Figur 7: Scale Space

3. Background Subtraction

3.1 Background Subtraction

Også kjent som forgrunnsdeteksjon, fordi algoritmene prøver å detektere alt som ikke er en del av bakgrunnen. Algoritmene som faller under denne kategorien er egnet til å bli brukt i forbindelse med overvåkning og situasjoner hvor videokameraet står helt stille. Grunnen til dette er simpel: En bakgrunn som hele tiden forandrer seg er ikke særlig lett å holde styr på, fordi det blir oppfattet som en total forandring av bildet. Når noe skjer i bildet kan for eksempel kameraet bli skrudd på, objektet blir markert eller tiden blir markert som en hendelse. Fordi vår løsning skal brukes i et kontrolltårn blir det aldri aktuelt å skru av/på kameraene, og markering av tiden er neppe for interessant. Derimot bruker vi delen som gjør det mulig å oppdage og markere bevegelse.

3.1.1 Utfordringer

Et relevant og normalt problem for de fleste av disse algoritmene er at de har problemer utendørs. Disse problemene omhandler for eksempel regn, snø, fugler, trær, sol, osv. All bevegelse eller rask forandring av lys/skygge kan føre til falske positiver. Det finnes innstillinger som gjør det mulig å minimere noen av disse faktorene, men disse innstillingene gjør det også vanskeligere å se objekter som er langt unna. Objektene som er langt unna blir sett på som stillestående, på grunn av at de ikke beveger seg tilstrekkelig nok til at de blir oppdaget.

Et annet eksempel gjelder noe som kan kalles bakgrunnsskifter, eller enklere sagt, langvarige forandringer i bakgrunnen. Hvis et fly har landet og står stille, skal det fortsatt regnes som en forandring sammenlignet med bakgrunnen, eller skal det bli regnet som bakgrunn? Begge disse to er bra løsninger, men i vår situasjon har KDA bedt oss om å fjerne markeringen etter en viss tid i slike situasjoner. Vi ser derfor på løsninger som bruker bildehistorien til å danne et bakgrunnsbilde. Det vil si at bakgrunnen oppdateres hvert N'te bilde. Ved å gjøre dette unngår vi at stillestående objekter av alle slag blir tatt for å være i bevegelse. Fordi vi har 5 bilder per sekund, vil for eksempel

en historikk på 50 bilder tilsvare 50/5=10 sek uten bevegelse for at et objekt blir ansett som stillestående.

3.1.2 Eksempelbilde

Figur 8: Biler i bevegelse (venstre er originalbilde, høyre er etter Background Subtraction ved hjelp av Mixture of Gaussians metoden)

På figur 8 ser vi et enkelt eksempel av hvordan algoritmen fungerer. Merk at dette eksempelet er under optimale forhold, det vil si hverken vær eller vind spiller merkbart inn, som gjør at det ikke er noen falske positiver. [11]

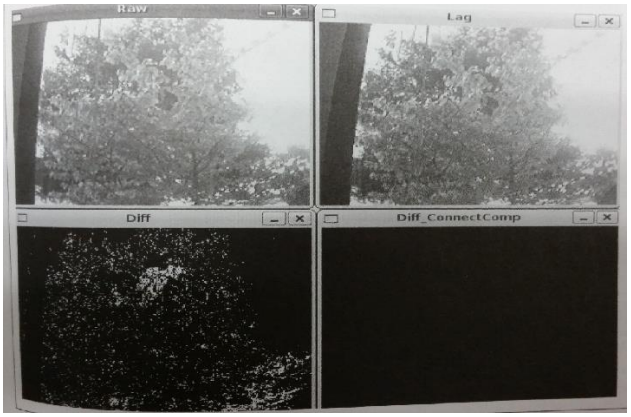
Det er kun to biler som er i bevegelse – dette kan ikke sees på originalbildet, fordi vi ikke med sikkerhet kan si at de parkerte bilene står stille. Derimot er det åpenbart hva som er i bevegelse på "Mixture of Gaussians" bildet, fordi bilene blir oppfattet som en forandring i bildet. Trærne beveger seg også litt, men de blir ikke markert på grunn av en satt terskel for bevegelse. Hva dette betyr skal vi se nærmere på i neste kapittel som forklarer de forskjellige algoritmene.

3.2 Konsepter

3.2.1 Bakgrunn/Forgrunn

I forhold til bakgrunnen, er det flere måter å skille mellom hva som er bakgrunn og hva som er forgrunn. Blant annet kan man skille mellom ny/gammel bakgrunn og ny/gammel forgrunn. Ny/gammel bakgrunn går ut på at objekter flyttes inn eller innenfor bildet, for eksempel en stol blir flyttet på, og at objektet etter hvert blir oppfattet som bakgrunn igjen. Det samme gjelder også for ny/gammel forgrunn – hvis objektet som er i forgrunn står stille for lenge, blir det regnet som gammel forgrunn. Hvorvidt gammel forgrunn har lik betydning som "Bakgrunn", er opp til programmereren. Det er mulig å skille mellom for eksempel siste bevegelse, mest relevante bevegelse (i forhold til fart og størrelse for eksempel), osv.

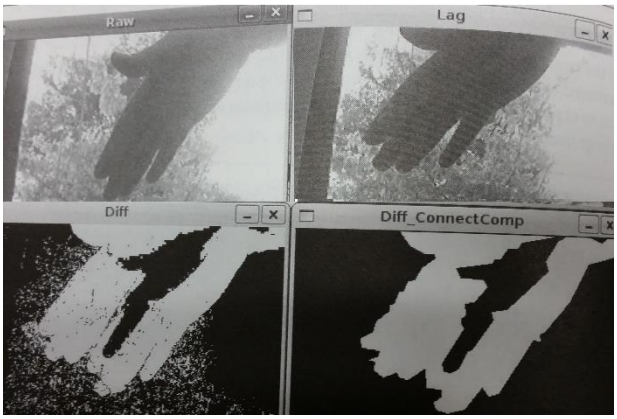
3.2.2 Connected Components



Figur 9: fra [12]

På figur 9 er det relativt lett å forstå hva terskel er i sammenheng med "connected components". Venstre øverst er bildet i nåtid, høyre øverst er fra fortiden, og venstre nederste er forskjellen mellom de to. Det siste bildet som er høyre nederst viser hvordan "connected components" fungerer, ved å fjerne alt som ikke er koblet sammen som en forandring. "Connected components" er vanskeligere å forklare på en annen måte, så det er derfor lettere å referere til figur 10. Forskjellen som treet utgjør blir fjernet, og forskjellen som sitter igjen er håndbevegelsen. Håndbevegelsen er altså sammenhengende nok til at den vises som en forandring.

Det er viktig å merke seg at det kun er to algoritmer som er brukt i disse bildene, total forandring (venstre nederst) og Diff_CC (høyre nederst). Det betyr at både nåtiden og fortiden blir registrert som forandring.



Figur 10: fra [12]

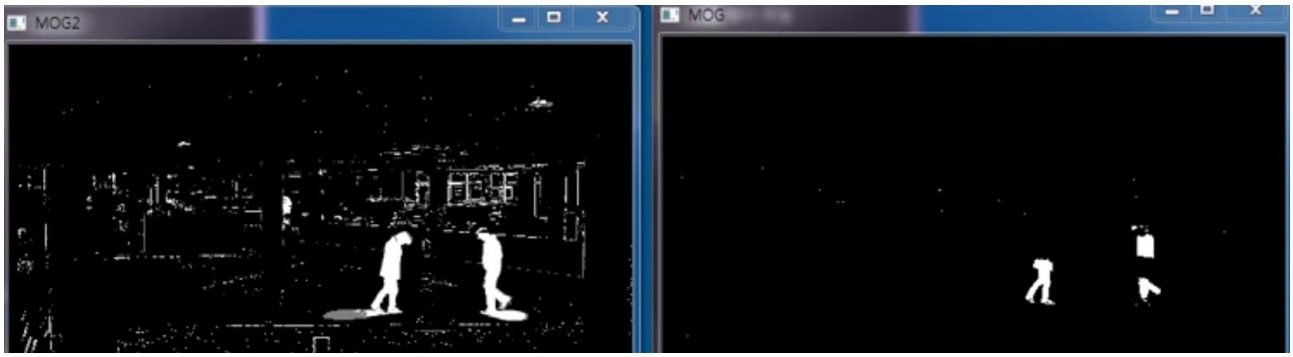
3.2.3 Nedskalering

Et stort og vedvarende problem vi har hatt, er at programmer prosesserer for få bilder per sekund. Dette er til tross for at vi kun skal ha fem bilder per sekund, noe som er relativt få bilder per sekund. Grunnen til dette er at videostrømmen er i 4K oppløsning som er en krevende/stor mengde med data å prosessere. Derfor valgte vi å nedskalere prosesseringen av videoen, slik at prosesseringen ble gjort på en full HD versjon av videoen. Ved å gjøre dette har vi fortsatt 4K oppløsning på bildet som vises, men alle kalkulasjoner blir gjort på et bilde som er full HD. I forhold til ytelse er dette gunstig, men det kan føre til lavere presisjon. Vi har ikke hatt noe problem med presisjon til nå, noe som tyder på at dette er en bærekraftig løsning om hardwaren er svak.

3.2.4 Forskjell mellom Mixture of Gaussians 1 og 2

"Mixture of Gaussians" er en statistisk metode som brukes i matematikken. Metoden brukes i denne teksten alltid i programmeringssammenheng, det vil si at algoritmen som brukes anvender "Mixture of Gaussians" for å utføre kalkulasjonene sine. Metoden er en undergruppering av det som kalles "Mixture Models", hvor forskjellige statistiske modeller blandes. "Mixture of Gaussians" anvender som navnet tilsier, gaussiske fordelinger som også er kjent under navnet normalfordelinger. I programmeringssammenheng går "Mixture of Gaussians" ofte under akronymet MOG. [13]

På figur 11 er det tydelig hvilken av de to som er mest treffsikker og inneholder flest detaljer. "Mixture of Gaussians" 2 viser skyggene til personene, og viser begge personene like tydelig. Personen til venstre går tregt, og vises så vidt på "Mixture of Gaussians" 1, men vises helt tydelig på "Mixture of Gaussians" 2. Det eneste som er negativt med "Mixture of Gaussians" 2 i dette tilfellet er at omgivelsene blir plukket opp som forandringer, selv om det kun er lyset. Derimot er forandringene ikke godt nok gruppert for at de vil bli detektert som en bevegelse av en algoritme, slik at dette ikke er et problem. "Mixture of Gaussians" 2 har mer funksjonalitet, slik som skyggene som ble nevnt, noe som også kan skrus av hvis dette er irrelevant/hvis det ønsker bedre ytelse. "Mixture of Gaussians" er muligens litt raskere enn "Mixture of Gaussians" 2, men for vår del var dette en forskjell som ikke var merkbar stor. [14]



Figur 11: Venstre er Mixture of Gaussians 2 og høyre er Mixture of Gaussians 1

3.2.5 Den mest optimale algoritmen

I de fleste tilfeller vil det alltid være en "best tilpasset" algoritme for enhver situasjon. Innenfor "Background Subtraction" derimot er det få tilfeller hvor en algoritme alltid vil være best. Det finnes flere grunner til dette, og som både vi og flere kilder [15] (Merk at rapporten er basert på flere kilder og har flere forfattere) har erfart:

1. Enkelte algoritmer arbeider kun med gråskalabilder, i ulikhet med algoritmer som arbeider med farger. Gråskalaalgoritmene vil i de fleste tilfeller aldri være mer presis enn algoritmene som arbeider med farger.

2. At en algoritme er mer sofistikert/tar hensyn til flere faktorer, betyr ikke alltid at algoritmen gir mer presise resultater. Problemet med en bestemt algoritme er at den enten tar for mye eller for lite hensyn til støy i bakgrunnen.

3. CPU og minne vil alltid være en faktor, spesielt i tilfeller hvor det benyttes "embedded systems", det vil si små enheter med relativt begrenset CPU og minne.

Dette er de faktorene som er mest relevante for oss. Andre faktorer eller observasjoner har vært mindre betydningsfulle for oss.

Et sluttpoeng som også trekkes hos noen [15] er at selv om det ikke finnes en algoritme som har både bra prosesseringskraft, presisjon og lavt forbruk av CPU/minne, er det mulig å forbedre resultatet. Resultatet kan bli forbedret ved å implementere et filter som forbedrer resultatet (for eksempel fjerner for små eller for store detekterte blobber/områder) eller ved å bruke en algoritme som har best mulig kompromiss mellom effektivitet, hastighet og simplisitet.

3.2.6 Bildedifferensiering

$$P[F(t)] = P\,[I(t)] - P[B]$$

Figur 12: Likning 1

Bokstavene står for følgende: t er tid, I(t) er bilde i en gitt tid t, B er bakgrunnsbilde.

Dette er den letteste måten å detektere bevegelse på, men absolutt alt som ikke er slik det var ved t=0 vil være en forandring. Likningen i figur 12 egner seg derfor svært dårlig med mindre vi er ute etter å se om bilde har forandret seg siden t=0. Generelt vil dette være en dårlig måte å detektere bevegelse på, med mindre bakgrunnen aldri forandrer seg.

$$|\,P[F(t)] - \,P[F(t+1)]\,| > Threshold$$

Figur 13: Likning 2

I ligningen i figur 12 bruker vi et filter for å forbedre ligningen i figur 13. Det ligningen i figur 13 gjør er å filtrere bilde n med bilde n+1, for å kunne forstå hva som er bakgrunn og hva som er forgrunn.

Figur 14: Eksempel filtrering

Den enkleste måten å forklare filtrering på, er å vise til figur 14. Filteret bruker fargeforandringer til å danne seg en forståelse av hva som er bakgrunn, og hva som er forgrunn. Svart blir oppfattet som forgrunn, mens hvitt er bakgrunn. Der det skjer plutselige forandringer i farge, f. eks mellom den hvite himmelen og trærne, blir markeringen svart. Derimot er det mulig å se at dette ikke er en helt feilfri måte, fordi fargeendringen ikke er skarp nok, blant annet i de markerte områdene på figur 15.

Figur 15: Fargeendringer som ikke er skarpe nok

4. Kalmanfilter

4.1 Kalmanfilter

Det som er spesielt for denne algoritmen er at det som blir observert (objektet), kan inneholde tilfeldige variasjoner i form av støy og unøyaktigheter. I praksis betyr det hvis objektet går fra en rett bane til å ta en høyresving eller plutselig går oppover for eksempel, regnes dette som støy/unøyaktighet. For fly er det mindre sannsynlig at banen endres radikalt, men derimot kan det hende at farten oppfattes som tregere/kjappere på grunn av avstanden. Kalmanfilteret oppdaterer i disse tilfellene den forventede posisjonen etter å ha observert resultatet. Under optimale forhold vil Kalmanfilteret aldri ta feil, men disse tilfellene vil i realiteten aldri skje på grunn av støyen og unøyaktighetene som er i den ekte verden. Derfor vil Kalmanfilteret alltid prøve å forbedre beregningene sine, etter å ha observert at posisjonen til objektet ikke stemmer med beregningene som ble gjort.

Fordi algoritmen er rekursiv og fungerer i sanntid, betyr det at den kan bruke input som er synlig i dette øyeblikket, samt det forrige øyeblikket en kalkulasjon ble gjort. Med andre ord trenger Kalmanfilteret kun å ha sett objektet og dets fysiske egenskaper (hastighet, akselerasjon) minst en gang for å kunne gjøre en gjetning. Denne gjetningen blir ikke like treffsikker som hvis det har skjedd flere observasjoner, men det gjør algoritmen godt egnet for våre formål. Under værforhold hvor det er svært overskyet, trenger flyet kun å bli oppdaget for en kort periode før Kalmanfilteret prøver å beregne posisjonen om det forsvinner ut av syne. [18]

4.1.1 Kalmanfilter konsepter

Kalman Gain

På norsk kalles dette en Kalmanforsterkning. Dette er en variabel som er med i mange av formlene som omfatter korrigeringsfasen i algoritmen. Den bestemmer hvorvidt man skal stole på den informasjonen man allerede vet (tidligere beregninger/resultater) eller ny informasjon (nåtid). Hvorvidt den skal stole på tidligere beregninger ser den ut ifra om beregningene har vært riktige eller feil. Å ha en høy Kalmanforsterkning betyr at algoritmen legger høyere vekt på observasjoner i nåtid, mens lav forsterkning betyr at tidligere observasjoner er viktigere. Kalmanforsterkningen bestemmes ut ifra en funksjon, og funksjonen blir i de fleste tilfeller forenklet til fordel for ytelse/hastighet over treffsikkerhet.

Predict/Update

Som forklart tidligere, er det en beregningsfase og en korrigeringsfase. Sammenlignet med andre typer algoritmer, for eksempel "Batch estimation", trenger ikke Kalmanfiltre å samle opp en historie før den gjør beregningene sine. Kalmanfilteret bruker en "X-1 og X" løsning for å finne ut hva som kommer til å skje, det vil si en fortid og en nåtid. X-1 og X blir brukt som variabler under kalkulasjonene for å kunne prøve å beregne hvor objektet er innenfor den neste gitte tidsperioden.

Sammenlignet med "batch estimation" har Kalmanfiltre en bedre ytelse/prosesseringshastighet, men i noen tilfeller også dårligere treffsikkerhet.

4.1.2 Fordelene med Kalmanfilter

På grunn av at vi bruker 4K bildeoppløsning, spiller ytelse veldig mye inn i bildet. Desto mer kraftig/treffsikker en algoritme er, desto mer kostbar er den i forhold til hvor mye den tar opp av datamaskinen. Kalmanfiltre har fordelen at de justerer seg selv etter at de har gjettet feil, slik at treffsikkerheten øker. En "batch estimation" derimot bygger opp en historie som øker i treffsikkerhet desto lengre historien er satt til å være. Og med en lengre historie kommer en tregere algoritme som koster mye, men som til gjengjeld er veldig treffsikker.

Treffsikkerhet er svært viktig i vår oppgave, men fordi objektet (fly) er relativt stort sammenlignet med andre objekter i nærheten, som f. eks biler, fugler, osv, vil treffsikkerhet spille mindre inn. Derfor handler det for oss om å finne en riktig balanse mellom at algoritmen fungerer raskt nok på 4K, samtidig som den er treffsikker nok til å kunne skille mellom fly og ikke-fly.

4.1.3 Eksempel i praksis

Figur 16: Eksempel, Kalmanfilter (video tatt fra 0:48 til 0:57 fra [19])

På første bilde ser vi objektet (en person) helt tydelig, derav vises en grønn ramme rundt personen for å indikere at objektet er fullt synlig. I neste bilde har personen forsvunnet bak treet, men algoritmen bruker hastigheten og retningen personen holdt før han forsvant til å gjette lokasjonen. I tredje bilde er bena til personen så vidt synlige, og litt av skjorten. Vi ser at gjetningen er nesten 100% presis, men at den er litt feil med tanke på at rammen ikke starter i hodehøyde. I det fjerde bildet hopper rammen rett tilbake til hodehøyde, og rammen blir grønn igjen fordi personen er synlig. Personen er bare borte for ca. 2 sekunder, men for fly kan tiden som går være mye lengre, spesielt hvis det er veldig overskyet. Det som er viktig om vi skal bruke et Kalmanfilter, er at vi må justere tiden før algoritmen "gir opp" å lete etter individuelle objekter. I tilfelle det dukker opp falske positiver må vi også ha en "av/på" funksjon for hver enkelt ramme.

5. Konklusjon

Ved å sette seg inn i de forskjellige metodene som finnes av løsninger på å gjenkjenne bevegelse og objekter kan man mer effektivt lage et produkt. En finner da potensielle løsninger og hindringer som kan da virke negativt og positivt på utviklingen av produktet. Gjennom utviklingen av dette produktet ble det brukt mye tid på å undersøke hvile løsninger som var de optimale før de ble implementert i produktet. Dette gjorde at produktet ble så effektivt som det kunne ganske tidlig i prosessen. Dette dokumentet har hjulpet prosjektet enormt og det er ekstremt lærerikt hvis man ikke vet noe om bildebehandling fra før av.

6. Referanser

[1] OpenCV, «Cascade Classifiers,» OpenCV, [Internett]. Available: http://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html. [Funnet 17 Mars 2016].

[2] N/A, «Local Binary Patterns,» Wikipedia, [Internett]. Available: https://en.wikipedia.org/wiki/Local_binary_patterns. [Funnet 17 Mars 2016].

[3] N/A, «Feature Detection,» Wikipedia, [Internett]. Available: https://en.wikipedia.org/wiki/Feature_detection_(computer_vision). [Funnet 17 Mars 2016].

[4] N/A, «Edge Detection,» Wikipedia, [Internett]. Available: https://en.wikipedia.org/wiki/Edge_detection. [Funnet 17 Mars 2016].

[5] N/A, «Corner Detection,» Wikipedia, [Internett]. Available: https://en.wikipedia.org/wiki/Corner_detection. [Funnet 17 Mars 2016].

[6] N/A, «Blob Detection,» kixor, [Internett]. Available: http://www.kixor.net/school/2008spring/comp776/assn1/. [Funnet 17 Mars 2016].

[7] N/A, «Ridge Detection,» Wikipedia, [Internett]. Available: https://en.wikipedia.org/wiki/Ridge_detection. [Funnet 17 Mars 2016].

[8] OpenCV, «Hough Transform,» OpenCV, [Internett]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html. [Funnet 17 Mars 2016].

[9] S. Arseneau, «Structure Tensor,» Carnegie Mellon University, 21 9 2006. [Internett]. Available: https://www.cs.cmu.edu/~sarsen/structureTensorTutorial/. [Funnet 17 Mars 2016].

[10] N/A, «Scale Space,» Wikipedia, [Internett]. Available: https://en.wikipedia.org/wiki/Scale_space. [Funnet 17 Mars 2016].

[11] OpenCV, «Background Subtraction Methods,» OpenCV, [Internett]. Available: http://docs.opencv.org/3.1.0/d1/dc5/tutorial_background_subtraction.html#gsc.tab=0. [Funnet 11 April 2016].

[12] A. Kaehler og G. Bradski, Learning OpenCV, O'Reilly Media, 2008.

[13] N/A, «Mixture of Gaussians,» [Internett]. Available: https://en.wikipedia.org/wiki/Mixture_model. [Funnet 18 April 2016].

[14] J. Kim, «Background Subtraction Comparison,» Youtube, 24 April 2014. [Internett]. Available: https://www.youtube.com/watch?v=T-L9FoH3D9w. [Funnet 18 April 2016].

[15] Y. Benezeth, P.-M. Jodoin, B. Emile, H. Laurent og C. Rosenberger, «Comparative study of background subtraction algorithms,» [Internett]. Available: https://hal.inria.fr/inria-00545478/document. [Funnet 9 Mai 2016].

[16] N/A, «Kalman Filter,» Wikipedia, [Internett]. Available: https://en.wikipedia.org/wiki/Kalman_filter. [Funnet 5 April 2016].

[17] S. Sarwar, «Visual Object Tracking, Kalman Filtering,» Youtube, 22 1 2013. [Internett]. Available: https://www.youtube.com/watch?v=whwsLjLjEiY. [Funnet 6 April 2016].

KONGSBERG

Høgskolen i Sørøst-Norge

Installasjonsdokument

Remote Tower Solution

Rev	Endring	Dato	Utarbeidet av	Godkjent av	
0.1		10.05.16	Heidi Troppen		
1	Revisjon 1	13.05.16	Silje Ågren Aas		
1.1	"Fornorsket" dokumentet	18.05.16	Heidi Troppen		
2	La til DeckLink installering	20.05.16	Heidi Troppen		

Innholdsfortegnelse

Liste over figurer

1. Introduksjon

Dette dokumentet beskriver hva man trenger å installere, samt ting man må være oppmerksom på underveis for at systemet skal fungere. Det blir også diskutert hva som ble gjort lite optimalt i begynnelsen av prosjektet med installeringer i et Linuxmiljø og fortsettelsen i et Windowsmiljø.

1.1 Dokumenthistorie

0.1: Første versjon.

1: Rettet feil, endelig versjon.

1.1: Gjort om forskjellige engelske uttrykk til norske uttrykk, lagt til figurtekst til figur 1.

2: La til installasjonsprosedyrer for DeckLink Studio 4K og så over for innlevering.

2. Installering av IDE og kompilator

For å kunne kjøre programmet og bygge OpenCV trengs det en kompilator og en IDE som støttes. I dette prosjektet har det blitt brukt Microsoft Visual Studio 2015 Update 1 som IDE og Visual Studio 14.0 64-bit som kompilator. Dette var ganske rett fram å installere da installeringsprosedyren er å gjøre hva installeringsprogrammet sier. En feil som ble gjort da det ble installert første gang var at installeringen ikke var satt til å ta med C++ biblioteket. Siden denne IDE'en var originalt lagd for C/C# må man spesifisere at man også vil installere C++.

2.1 Viktig å huske på

- Legge til C++ bibliotekene under installasjonen

3. Installere OpenCV og CMake.

Måten man installerer OpenCV på er å bygge kildekoden selv, eller bruke en forhåndsbygd versjon. Dette prosjektet har bygd fra kilde. Måten man gjør dette på er å bruke CMake [1] for å starte byggingen av OpenCV-biblioteket og fortsette den i MSVC 2015.

3.1 Installere CMake

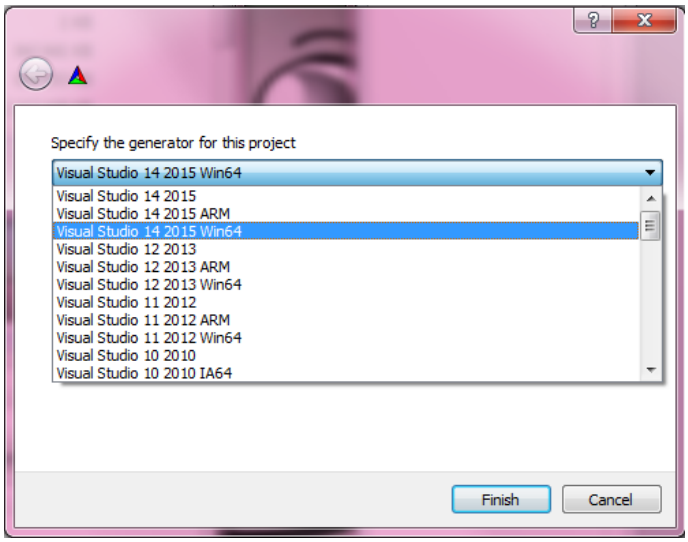
1. Last ned en "Binary distribution" fra "Latest Release" delen i nedlastningssidene til CMake.
2. Kjør filen og følg installasjonsprosedyren.
    a. Hust å huke av for å legge til CMake i "System Path" når du får valget om dette
    b. Pass på at CMake er i "System Path" ved å åpne "Control Panel" også gå til "Systems" videre til "Advanced system settings". Her går du inn i "Environment Variables…". I den nederste boksen navngitt "System variables" sjekker du at "Path" inneholder fillokasjonen til CMake. Hvis det ikke er noe med CMake må du legge til dette ved å legge til lokasjonen til "CMake/bin".

3.2 Installere OpenCV

1. Last ned ønsket versjon av OpenCV [2]
2. Kjør .exe filen som pakker ut filene og pakk den ut til C:/OpenCV
3. Kjør CMake
4. Legg til "Source" mappen til OpenCV i feltet som spør "Where is the source code:"
5. Lag en ny mappe i C:/OpenCV som heter CMakeBuild, legg til denne lokasjonen i feltet som spør om "Where to build the binaries:"
6. Klikk "Configure", finn ønsket kompilator og klikk "Finish".
    a. Som vist på Figur 1 ble det brukt Visual Studio 14 2015 Win64 som da gir et 64-bit build til MSVC 2015
7. Hvis man kjører dette på en maskin uten internett, blir det høyst sannsynlig en feilmelding som sier at det er noe feil med prosjektfilene. Dette er fordi CMake prøver å laste ned filer online, og når dette går feil, feiler CMake og den tror det er noe feil med prosjektfilene.
    a. For å fikse dette kan man fjerne de filene som må lastes ned i CMake konfigurasjonen
    b. Fjern "WITCH_FFMPEG" og "WITH_IPP".
    c. Fjern "BUILD_PERF_TESTS" og "BUILD_TESTS" fordi det er noen byggefeil senere i MSVC 2015 som blir skapt av disse.
    d. Klikk "Configure" igjen.
        i. Nå kan man eventuelt legge til flere "includer" til OpenCV, slik som Qt. Man huker da av for "WITH_QT" flagget og klikker "Configure" igjen, da vil man få

opp flere Qt flagg som man kan huke av. Dette krever da at Qt installasjonen som blir brukt er lagt til i "System Path'en".

8. Klikk "Generate" og åpne mappen der bygget er lagret.
9. Åpne OpenCV.sln filen i MSVC 2015
10. Pass på at Bygg konfigurasjonen er satt på "Debug" og "64" og bygg prosjektet.
11. Bytt Bygg konfigurasjonen til "Release" og bygg prosjektet.



Figur 1: Valg av generator

4. Sette opp prosjekter som bruker OpenCV

For å sette opp et prosjekt som bruker OpenCV i koden så må man endre diverse "Properties" i prosjektet.

1. Åpne "Properties" ved å velge prosjektet du vil bruke og høyreklikke og velge "Properties" nederst i listen.
    a. Pass på at "Configuration" og "Platform" øverst i vinduet som kommer opp er satt til enten "Active(Debug)" og "Active(x64)" eller "Debug" og "x64"
2. Velg "VC++ Directories" og i "Include Directories" og klikk "Edit" og legg til filstiene til
    a. "$path$/bin/Debug" som inneholder .dll filer
    b. "$path$/lib/Debug" som inneholder .lib filer
    c. "$PathTilSource$/opencv/build/include" som inneholder "include" filer, denne mappen er der hvor du satte kilden til OpenCV i CMake.
3. Velg "C/C++" og "General" og legg til
    a. "$path$/bin/Debug" som inneholder .dll filer
    b. "$PathTilSource$/opencv/build/include" som inneholder "include" filer, denne mappen er der hvor du satte kilden til OpenCV i CMake.
4. Velg "Linker" og sett "Enable Incremental Linking" og "Use Library Dependency Inputs" til "Yes(/INCREMENTAL)" og "Yes". Legg til disse filstiene i "Additional Library Directories"
    a. "$path$/3rdparty/lib/Debug"
    b. "$path$/bin/Debug" som inneholder .dll filer
    c. "$path$/lib/Debug" som inneholder .lib filer
5. Velg "Input" i "Linker" og i "Additional Dependencies" legg til:
    a. opencv_calib3d310d.lib
    b. opencv_core310d.lib
    c. opencv_features2d310d.lib
    d. opencv_flann310d.lib

e. opencv_highgui310d.lib
f. opencv_imgcodecs310d.lib
g. opencv_imgproc310d.lib
h. opencv_ml310d.lib
i. opencv_objdetect310d.lib
j. opencv_photo310d.lib
k. opencv_shape310d.lib
l. opencv_stitching310d.lib
m. opencv_superres310d.lib
n. opencv_video310d.lib
o. opencv_videoio310d.lib
p. opencv_videostab310d.lib
q. IlmImfd.lib
r. libjasperd.lib
s. libjpegd.lib
t. libpngd.lib
u. libtiffd.lib
v. libwebpd.lib
w. zlibd.lib

6. I filmappen til prosjektet må man også legge til ekstra .dll filer for å ikke få feilmelding som sier "$FILE$ does not seem to exist in this project"
    a. I prosjektmappen, naviger til "x64/Debug" og legg til alle .dll filer fra "$path$/bin/Debug»

For å få prosjektet i "Release" må disse stegene gjentas med "Configuration" satt til enten "Active(Release)" eller "Release". Stegene ovenfor blir da lagt til med "Release" ekvivalenten til "Debug" versjonen.

5. DeckLink Studio 4K: oppsett og informasjon

5.1 Installere DeckLink Studio 4K i PC

1. Sett inn kortet i et tomt PCIe-buss
2. Se i [3] hvilken HDMI-port som er ut og inn for bruk i HDMI-forlengeren
3. Sett så HDMI-forlengelsen i en tom åpning bak for enklere tilgang til kortets inngang og utgang.
    a. Smart å markere hvem som er inn og ut for framtidig bruk.
4. Last ned nyeste programvare/driver fra [4] og installer disse.
5. Åpne "Blackmagic Desktop Video Utility" og sett video "input" til HDMI.
    a. Dette gjøres ved å klikke på den runde knappen under "DeckLink Studio 4K".
    b. Dette steget er viktig for å kunne kjøre programmet, da programmet ikke setter "input" selv.

5.2 Sette opp Desktop Video SDK

Dette er installert på en Windows maskin med Visual Studio 14, installeringsprosessen kan være noe annerledes i en Linux maskin med en annen IDE. Utenom denne installeringsprosedyren står det i [5] hvordan man installerer på forskjellige plattformer.

1. Last ned nyeste "Desktop Video SDK" fra [4] og pakk ut .zip filen.
2. Legg til filene i "Win/include" i en mappe i prosjektet ditt.
3. Kompiler "DeckLinkAPI.idl" slik at "DeckLinkAPI_i.c" blir generert i prosjektets hovedmappe.
    a. Legg denne filen inn i prosjektet via Visual Studios metoder for å legge til eksisterende filer.

4. Gå inn på "Properties" til "DeckLinkAPI_i.c" og velg "C/C++" og deretter "Precompiled Headers"
    a. Sett "Precompiled Headers" til "Not Using Precompied Headers"

6. Utfordringer under RTS oppsett og utvikling

RTS skulle originalt bli utviklet med et Linux miljø. Det skulle også være på PC'er som ikke hadde tilgang til internett. Siden Linux bruker internett for å hente de fleste pakker og slikt for installasjoner, var dette ekstremt vanskelig å tilpasse seg til. Likevel fikk vi installert Qt og OpenCV på disse maskinene, testprogrammer klarte å vise bilder med OpenCV, men ikke videoer. Det viste seg da at det var manglende mediakodeker.

PC'ene som vi fikk hadde ikke noen mediakodeker installert. Å installere disse viste seg å være ganske avansert uten internett for ingeniørstudenter som hadde lite erfaring med å bruke Linux. For å kunne vise video i Linux trenger man mediakodeker, så dette var ekstremt viktig å få installert.

Måten å installere ting uten internett på Linux med RedHat versjonen vi brukte var å bruke "rpm'er". Disse lastet man ned på en annen PC med internett og flyttet de over på RedHat-maskinen for å pakke ut og installere. En "rpm" er et filformat som har programvarepakke inn i seg, og er en pakkemanager i seg selv. [6] Når man installerer en "rpm" kan det hende at man ikke har installert flere pakker som "rpm'en" trenger for å bli installert. Man må da legge til det som ikke er installert, som høyst sannsynlig igjen trenger noe annet. Slik går det lenger og lenger ned i "rpm'er".

Etter mye strev med dette var troen at mediakodekene skulle funke, noe de ikke gjorde. Det var uvisst om dette var at kodekene ikke var de riktige, eller bare installert feil. Gruppen fikk hjelp fra personer på KDS, men dette løste dessverre ikke problemet. Det ble da avgjort at det var enklere å bruke Microsoft Windows, da dette OS'et kommet med mediakodeker som standard.

Samme problemet oppstod her, mediakodekene var ikke de riktige som trengtes. Men etter å ha installert nyere versjon av kodekene som allerede fantes, og installert et par nye kodeker funket det fint.

Så skulle alt som hadde funket på Linux installeres på Windows. Siden Windows ikke kommer med en kompilator i OS'et måtte Microsoft Visual Studio 2015 installeres for å ha en fungerende kompilator.

For å bruke Qt med MSVC 2015 måtte man da bruke en "Release Candidate" for Qt som da hadde MSVC 2015 støtte. Tidligere versjoner av Qt brukte tidligere versjoner av MSVC 2015. Qt var installert, så var det bare å installere OpenCV med Qt kompatibilitet. For å ha mest mulig ytelse på programmet var valget tatt om å bruke 64-bit versjoner av forskjellig programvare. OpenCV installeringsprosessen ble startet, men da biblioteket skulle bygges i MSVC 2015 kom det en del feil, som da tydet på at OpenCV fungerer bare med Qt i 32-bits versjoner. Siden denne feilen ikke hadde en løsning etter mye undersøkelser, ble det gått bort fra å bruke Qt og heller bare bruke MSVC 2015.

7. Refererte dokumenter

[1] CMake, «CMake Download,» [Internett]. Available: https://cmake.org/download/. [Funnet 10 05 2016].

[2] Itseez, «Downloads OpenCV,» [Internett]. Available: opencv.org/downloads.html. [Funnet 11 05 2016].

[3] Blackmagic Designs, «Desktop Video Operational Manual,» 2016.

[4] Blackmagic Designs, «Support Center,» [Internett]. Available: https://www.blackmagicdesign.com/support/family/capture-and-playback. [Funnet 20 05 2016].

[5] Blackmagic Designs, «Blackmagic DeckLink SDK,» 2016.

[6] Wikipedia, «RPM Package Manager,» [Internett]. Available: https://en.wikipedia.org/wiki/RPM_Package_Manager. [Funnet 12 05 2016].