

BACHELOROPPGAVE

Detekteringssystem for Project Hessdalen ved bruk av bildeanalyse

BO18-G09

Henrik Kronborg

Jonas Rolstad

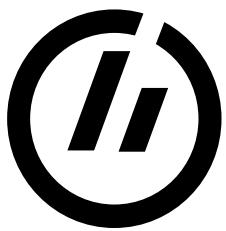
Mathias Jensen

16.05.2018

Ingeniørfag - data

Avdeling for informasjonsteknologi





HØGSKOLEN I ØSTFOLD

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

BACHELOROPPGAVE

Prosjektkategori: Utvikling	X	Fritt tilgjengelig
Omgang i studiepoeng: 20		Fritt tilgjengelig etter
Fagområde: Dataingeniør		Tilgjengelig etter avtale med oppdragsgiver

Tittel: Detekteringssystem for Project Hessdalen ved bruk av bilde-analyse	Dato: 15. mai 2018
Forfattere: Henrik Kronborg, Jonas Rolstad og Mathias Jensen	Veileder: Lars Emil Knudsen
Avdeling / Program: Avdeling for Informasjonsteknologi / bachelorstudium i Ingeniørfag - data	Gruppenummer: BO18-G09
Oppdragsgiver: HiØ - Project Hessdalen	Kontaktperson hos oppdragsgiver: Erling Petter Strand

Ekstrakt:

To timer sør for Trondheim finnes det et dalføre kalt Hessdalen. Her oppstår det hittil uforklarelige lysfenomener som siden 1981 har fanget vitenskapsmenns nysgjerrighet. Det er påvist at fenomenene inneholder mye lysenergi som er interessant med tanke på fremtidige, rene energikilder. Dette prosjektet går ut på å lage et system som skal detektere og dokumentere disse lysfenomenene i form av video og bilder.

3 emneord:

Hessdalsfenomenet
Bildeanalyse
Single-board computer

Forord

Dette er en prosjektrapport utarbeidet av tre dataingeniørstudenter ved HiØ (Høgskolen i Østfold). Gruppen valgte dette prosjektet fordi oppdragsgiver har vært deres foreleser i tidligere fag og stadig brukt Hessdalsfenomenet i sine eksempler. Så når sjansen bød seg for å faktisk arbeide med dette mystiske lysfenomenet fikk gruppen raskt låst oppgaven til seg. Ingen av gruppemedlemmene hadde tidligere erfaring med bildeanalyse, men så på det som en mulighet til å utfordre seg selv og lære noe nytt.

Sammendrag

To timer sør for Trondheim finnes det et dalføre kalt Hessdalen. Her oppstår det hittil uforklarelig lysfenomener som siden 1981 har fanget vitenskapsmenn nysgjerrighet. Det er påvist at fenomenene inneholder mye lysenergi som er interessant med tanke på fremtidige, rene energikilder. Dette prosjektet går ut på å lage et system som skal detektere og dokumentere disse lysfenomenene i form av video og bilder. Gjennom å studere eksisterende teknologier tilegnet gruppen seg nok kunnskap til å produsere en løsning.

Etter å ha satt seg inn i bildebehandlingsmetoder endte gruppen med å benytte detektering av bevegelser. Dette gjøres ved å fortløpende sammenligne bilder for å se etter forandringer. Bildeanalysen er en egen del av programmet, slik at den enkelt kan endres eller oppgraderes ved et senere tidspunkt.

Rapporten omhandler systemets funksjonalitet og virkemåte samt hvordan gruppen har jobbet med programmet gjennom prosjektet. Arbeidet har blant annet bestått av testing av systemet, hvor gruppen dokumenterer at det fungerer både på dagtid og kveldstid. Til slutt tas det for seg forslag til hvordan systemet kan utvikles videre.

Takk Til

Vi ønsker å takke vår veileder, Lars Emil Skrimstad Knudsen, for å ha bidratt med råd og veiling gjennom hele prosjektperioden. Uten svar på våre uendelige spørsmål angående dokumentstruktur hadde vi aldri endt opp med det resultatet vi sitter igjen med i dag. Vi ønsker også å takke Erling Petter Strand for å ha gitt oss muligheten til å arbeide med Hessdalsfenomenet som du snakket så mye om i alle fagene vi har hatt med deg opp gjennom årene.

Videre er det på sin plass å takke Ted Magnus Sørli for bistand når det kom til teknisk hjelp, og Lars Vidar Magnusson for å ha hjulpet oss i riktig retning når det kom til spørsmål angående bildeanalyse.

Innhold

Sammendrag	i
Takk Til	iii
Figurliste	ix
Tabelliste	xi
Kodeliste	xiii
Ordliste	1
1 Introduksjon	2
1.1 Prosjektgruppen	2
1.2 Veileder	3
1.3 Oppdragsgiver	3
1.4 Oppdraget	3
1.5 Formål, leveranser og metode	4
1.6 Rapportstruktur	6
2 Analyse	7
2.1 Systemoversikt	7
2.2 Teknologi og utstyr	8
2.3 Kamera og linse	11
2.4 Filformat	14
2.5 Codecser	15
2.6 Encoding	15
2.7 Programmeringsspråk og annen teknologi	16
2.8 API	18
2.9 Detektering	19
2.10 Bildeanalyser	20
2.11 Maskering av bildeområde	21
2.12 Innstillinger, parametere	21
2.13 Plassering	21
2.14 Kommunikasjonsprotokoller	22
2.15 Lagring av media	22
2.16 Alternativer	22
2.17 Systemoversikt - revisjon 1	23

3 Utforming	25
3.1 Valg av hardware	25
3.2 Valg av programvare	27
4 Implementasjon	33
4.1 Kamerahus	33
4.2 Kontrollpanel	34
4.3 Bildeanalyse	37
4.4 Hovedprogram	44
4.5 Systemoversikt - revisjon 3	49
5 Testing	51
5.1 Hensikt	51
5.2 Testlokasjoner	51
5.3 Utstyr	53
5.4 Testing i mørket	56
5.5 Stresstesting	66
5.6 Testing på dagtid	70
6 Diskusjon	73
6.1 Målene	73
6.2 Resultater	74
6.3 Vurdering av metoder	74
6.4 Oppgaven	75
6.5 Videre arbeid	77
7 Konklusjon	79
Bibliografi	82
Register	83
A Hvordan bruke systemet	83
B Pikselstørrelse for deteksjoner	86

Figurer

1.1	Henrik Kronborg	2
1.2	Jonas Rolstad	2
1.3	Mathias Jensen	2
1.4	Høgskolen i Østfold, avdeling for informasjonsteknologi [12]	3
2.1	Dagens system	7
2.2	Raspberry Pi [24]	9
2.3	NVIDIA Jetson TX1 [18]	10
2.4	NVIDIA Jetson TX2 Developer kit [19]	10
2.5	Speilreflekskamera [27]	11
2.6	Speilrefleks virkemåte [28]	11
2.7	Nikon D7500 [17]	11
2.8	Digitalkamera [9]	12
2.9	Sony Cybershot DSC-HX90V [26]	12
2.10	Overvåkningskamera [21]	12
2.11	D-Link DCS-7110 [4]	13
2.12	Industrikamera [14]	13
2.13	DFM 21BU04-ML [7]	13
2.14	DFM 31BU03-ML [8]	14
2.15	DFK 31BU03.H [6]	14
2.16	Python [23]	16
2.17	C++ [2]	16
2.18	HTML [11]	16
2.19	CSS [3]	17
2.20	JavaScript [15]	17
2.21	AJAX [1]	17
2.22	PHP [22]	18
2.23	OpenCV [20]	18
2.24	GStreamer [10]	19
2.25	Scikit-Image [25]	19
2.26	ImageMagick [13]	19
2.27	Systemoversikt - revisjon 1	23
3.1	Benchmarktest av Jetson TX2 og Raspberry Pi	25
3.2	Nåværende oppsett av kamera [5]	30
3.3	Systemoversikt revisjon 2	31

4.1	Oppsett	33
4.2	Kamera montert på skinne som føres inn i kamerahus	33
4.3	Innlogging til kontrollpanel	34
4.4	Konfigurasjonsfil for kontrollpanel	34
4.5	Fjerne tilgang til for uvedkommende fil på kontrollpanel	34
4.6	Hvordan kontrollpanel ser ut	35
4.7	Illustrasjon av at maske lagres med gjennomsiktig bakgrunn	35
4.8	Program på SBC kjører ikke	36
4.9	Tilkobling til SBC mislyktes	36
4.10	Program på SBC kjører	36
4.11	Pop-up	36
4.12	Innstillinger som kan endres på kontrollpanelet	37
4.13	Bildeanalyse - referansebilde	38
4.14	Bildeanalyse - ny frame som skal sammenlignes med referansebilde	38
4.15	Bildeanalyse - ny frame gjort om til gråskala	39
4.16	Bildeanalyse - gråskala frame blurret	39
4.17	Bildeanalyse - utregnet forskjell mellom ny frame og referansebilde	40
4.18	Bildeanalyse - thresholdet forskjellsbilde	41
4.19	Bildeanalyse - tegnet maske, etter thresholding, som skal ganges inn	41
4.20	Bildeanalyse - thresholdet bilde etter at maske er ganget inn	42
4.21	Bildeanalyse - thresholdet bilde etter at maske er ganget inn og hvite objekter har økt i størrelse	42
4.22	Bildeanalyse - originalt bilde med røde sirkler som markerer hvor det har vært endring	43
4.23	Illustrasjon som viser systemoversikt	49
5.1	Rom DU1-043	51
5.2	Område utenfor IT-avdelingen	52
5.3	Skjøtetrommel og Jetson	52
5.4	Utsikt fra lokasjon	52
5.5	Lokasjon ovenfor hovedinngang	52
5.6	Oppsett på D1-052	52
5.7	Detekterte ut vinduet	52
5.8	DFK 31BU03.H	53
5.9	Jetson TX2 med strømtilførsel	53
5.10	USB-hub med mus og tastatur	53
5.11	Skjerm	53
5.12	Lommelykt	54
5.13	Hodelykt	54
5.14	Laser	54
5.15	Mobil	55
5.16	LED	55
5.17	Papirfilter	55
5.18	Eksempel på lysskjær	56
5.19	Deteksjon på 30-60 cm	57
5.20	Deteksjon på 5-6 meter	57
5.21	Deteksjon på 10-15 meter	57

5.22 Deteksjon av hvitt lys	58
5.23 Deteksjon av gult lys	58
5.24 Deteksjon av rødt lys	58
5.25 Deteksjon av blått lys	59
5.26 Oppsett med fire LED	59
5.27 Fire LED pålyst	59
5.28 Alle farger detektert	59
5.29 Objektet	60
5.30 Objektets ferd	60
5.31 To objekter detektert	60
5.32 Objektenes ferd	60
5.33 Flere objekter	61
5.34 Detekterte objekter	61
5.35 Objektenes ferd	61
5.36 Objekt på fiktiv stjernehimmel	62
5.37 Objekt detektert på fiktiv stjernehimmel	62
5.38 Deteksjon 1	63
5.39 Deteksjon 2	63
5.40 Tidslinje	64
5.41 Maske laget ved hjelp av nettsiden	64
5.42 Lys ved 50% maske	65
5.43 Deteksjonsbilde ved 50% maske	65
5.44 Maskering	65
5.45 Maskerte områder	66
5.46 Temperaturmåling	67
5.47 Referansebilde	70
5.48 Neste bilde	70
5.49 Før maskering	70
5.50 Etter maskering	70
5.51 Kamera utsikt i Hessdalen [16]	71
5.52 Testing ovenfor hovedinngang	71

Tabeller

1.1	Original oppgavebeskrivelse	4
2.1	Eksisterende mot tiltenkt system	8
3.1	Overføringsmetoder som kommer til å bli brukt	29
4.1	Oppsummering av bildeanalyse	44
5.1	Intervaller på 60 sekunder	66
5.2	Intervaller på 2 minutter	67
5.3	Intervaller på 5 sekunder	68
5.4	Intervaller på 60 sekunder	68

Kodeliste

4.1	Leser en og en frame fra video	38
4.2	Gjør lest frame til gråskala	39
4.3	Blur lest frame for å fjerne støy	39
4.4	Regner ut forskjellen mellom referansebilde og lest frame	40
4.5	Forskjellsbilde thresholdes for å gjøre det svart-hvitt	40
4.6	Maske thresholdes for å gjøre den svart-hvitt	41
4.7	Maske ganges inn i thresholdet forskjellsbilde for å ikke inkludere uønskede områder	42
4.8	Hvite områder økes i størrelse	42
4.9	Finner og legger alle konturer på behandlet bilde i en array	43
4.10	Går igjennom arrayen med konturer og hvis konturen er både større og mindre enn satte verdier legges de i en annen array	43
4.11	Regner ut størrelse på sirkler og tegner på originalbilde der konturer er funnet . .	43
4.12	Sjekker om bildeanalySEN returnerer 1, altså at man har en deteksjon	46
4.13	Hvis bildeanalySEN returnerer 1 og man ikke er i deteksjonsfase	46
4.14	Hvis bildeanalySEN returnerer 1, man er i deteksjonfase og bildeanalySEN har returnert et brukerdefinert antall 1'ere på rad	46
4.15	Hvis bildeanalySEN returnerer 1, man er på siste frame i videoen og bildeanalySEN har returnert flere 1'ere på rad enn brukerdefinert antall	47
4.16	Hvis bildeanalySEN returnerer 1 og man er i deteksjonsfase	47
4.17	Hvis bildeanalySEN returnerer 0, man er i deteksjonsfase og detectStartCount er mindre enn brukerdefinert antall	47
4.18	Hvis bildeanalySEN returnerer 0, man er i deteksjonsfase og man har telt et brukerdefinert antall 0 på rad	47

Ordliste

Ord	Forkortning	Forklaring
Single-board computer	SBC	Se kapittel 2.2.1
NVIDIA Jetson TX2	TX2 / Jetson / datamaskin / SBC	Se kapittel 2.2.4
Deteksjonsvideo	Trim / utklipp	Forkortet videoklipp som kun inneholder video som dokumenterer fenomen
Deteksjonsbilde	Alarmsbilde	Bilde av noe som har trigget systemet
Maskering	Maske	Område som blir utelukket fra kameraets bilde
Fenomener		Lysfenomen som oppstår i Hess-dalen
Grayscale		Funksjon som gjør et bilde om til gråskala
Threshold		Funksjon som gjør et bilde binært, altså kun sort og hvitt
Kontrollpanel	Nettside	Nettside som inneholder diverse funksjonalitet som styring og parameterendringer av systemet
Application Programming Interface	API	Et programmeringsgrensesnitt i en programvare som gjør at spesifikke deler av denne kan kjøres fra en annen programvare.
Central Processing Unit	CPU	Hjernen i en datamaskin
Graphics Processing Unit	GPU	Dedikert CPU som kun behandler grafikkdata
OpenCV		Open Source Computer Vision
Secure Shell	SSH	SSH er både et dataprogram og en nettverksprotokoll som brukes å få tilgang til en kommandolinje (shell) på en annen maskin
Secure Copy	SCP	SCP er en protokoll og et program for sikker overføring av filer over SSH
Secure File Transfer Protocol	SFTP	SFTP er en nettverksprotokoll for sikker overføring av filer
Frame		En video består av mange frames
Rødt, grønt, blått	RGB	En fargemodell som brukes ved fargeblanding

Kapittel 1

Introduksjon

1.1 Prosjektgruppen

Gruppen består av tre dataingeniørstudenter som alle er på sitt 6. semester. Deltakerne har forskjellige bakgrunner og møttes allerede på sommerforkurset til ingeniørlinjen for TRESS og Y-vei. De har siden arbeidet sammen gjennom studiet, opparbeidet gode rutiner og har et godt samhold utenom studier.



Figur 1.1: Henrik Kronborg



Figur 1.2: Jonas Rolstad



Figur 1.3: Mathias Jensen

Henrik Kronborg, 22 år gammel dataingeniørstudent fra Drøbak. Studielivet begynte på Høgskolen i Østfold etter endt skolegang på Nesodden videregående. Erfaring innen programmering og nettsideutvikling fra sommerjobber og hobbyprosjekter. Store interesser for programmering og utvikling, slalåm og båt.

482 54 966 - henrik@hiof.no

Jonas Rolstad, 25 år gammel dataingeniørstudent. Født i Oslo, oppvokst i Bærum. Utdannet Telekommunikasjonsmontør med interesse for den tekniske verden. Glad i trening, natur og data.
481 43 208 - jonasr@hiof.no

Mathias Jensen, 25 år gammel dataingeniørstudent fra Sandane. Startet på Høgskolen i Østfold etter tjent førstegangstjeneste i Forsvaret. Har alltid hatt interesse innenfor data og fotball.
976 61 148 - mathiaj@hiof.no

1.2 Veileder

Veileder for prosjektet er Lars Emil Skrimstad Knudsen. Han har en master i informatikk og jobber per dags dato for Høgskolen i Østfold som høgskolelektor. Fagene han underviser i er androidprogrammering og objektorientert programmering.

1.3 Oppdragsgiver



Figur 1.4: Høgskolen i Østfold, avdeling for informasjonsteknologi [12]

Oppdragsgiver er HiØ/IT (Høgskolen i Østfold sin IT-avdeling). HiØ sin IT-avdeling er lokalisert i Halden og tilbyr fire ulike IT-studier. Skolen ble etablert i 1994.

Project Hessdalen er et prosjekt ved HiØ som observerer lysfenomener på himmelen i Hessdalen, Sør-Trønderlag. Prosjektet ble dannet sommeren 1983, og ved en feltaksjon i 1984 ble det gjort 53 visuelle observasjoner. I 1998 ble det opprettet en automatisk målestasjon i Hessdalen, for å dokumentere fenomenet ved hjelp av data og alarmbilder. Per dags dato er det tre personer som jobber aktivt på prosjektet, hvor de samarbeider med forskjellige institusjoner i både Frankrike og Hellas. Det gjøres ca. 20 observasjoner av ukjente lysfenomener hvert år. Fenomenet kalles "Hessdalsfenomenet".

Kontaktperson er Erling Petter Strand, som er en høgskolelektor ved avdeling for informasjonsteknologi på Høgskolen i Østfold. Han er en av grunnleggerne av Project Hessdalen, og er nå lederen av prosjektet.

1.4 Oppdraget

Project Hessdalen har siden 1988 dokumentert lysfenomener som oppstår i Hessdalen. På området er det satt opp en målestasjon som inneholder flere systemer som benyttes for å dokumentere fenomenene. Ett av disse systemene går ut på å detektere når fenomenene oppstår ved hjelp av bildeanalyse, og det er dette oppdragsgiver ønsker en oppgradering av.

Systemet skal detektere lysfenomener i Hessdalen ved hjelp av et kamera og en datamaskin. Bildene som tas skal analyseres av et program utviklet av gruppen, og ved deteksjon skal det tas et alarmbilde og video. Utover oppgavebeskrivelsen er det ønskelig å kunne lage en maske som utelukker ønskede områder fra bildeanalyisen. Videre utover oppgavebeskrivelsen er det ønskelig at video skal inneholde hendelsesforløpet rett før og etter et fenomen oppstår, slik at en kan se fenomenets livsløp. Alarmbildet og video skal så lastes opp til Project Hessdalen sin server.

Oppdragsgiver mener den nye løsningen er viktig for at det skal kunne fortsettes med å innhente detaljert dokumentasjon av fenomenene som oppstår. Fenomenet er interessant fordi det kan gi ny viden om vår verden, muligens en ny ren energikilde og gi morgendagens teknologi.¹

¹http://hessdalen.hiof.no/index_n.shtml

Oppdragsgiver har satt krav om at datamaskinen skal benytte operativsystemet Linux og at det skal være mulig å koble seg til eksternt via SSH. Utenom det står gruppen relativt fritt til å velge hvilken datamaskin og kamera som skal brukes.

Original oppgavebeskrivelse tilsa at oppgaven skulle løses med en Raspberry Pi, men som det kommer frem i senere kapitler ville ikke denne ha klart å utføre prosjektets oppgaver. Under er original oppgavebeskrivelse:

Tabell 1.1: Original oppgavebeskrivelse

Tittel: Bildeanalyse, med bruk av Raspberry Pi

I Hessdalen er det en automatisk målestasjon som inneholder forskjellige typer sensorer, som sender sine data til hessdalen.hiof.no. Det skal lages et nytt system, hvor det skal brukes Raspberry Pi. Et kamera skal tilknyttet Raspberry Pi, som analyserer bildestrømmen. Hvis noe uventet lys dukker opp, skal det tas et alarmbilde, og en kort videostrøm skal lagres. Dette skal overføres til hessdalen.hiof.no. – Det skal også være mulig å koble seg inn på denne Raspberry Pi via ssh. Studentene må finne et egnert kamera, velge en passe kraftig utgave av Raspberry Pi, og programmere denne. Det skal brukes Linux.

1.5 Formål, leveranser og metode

1.5.1 Formål

Hovedmål Prosjektet vil kunne gi grunnlag for et egenutviklet system som kan detektere all type lys som plutselig kommer til syne og dokumentere dette i form av bilde og video

Delmål 1 Gjøre det mulig å maskere områder på bilde slik at det ikke tas med i detekteringsprosessen

Delmål 2 Gjøre det mulig å endre innstillinger og kommunisere med single-board datamaskinen via SSH

Delmål 3 Gjøre det mulig å laste opp dokumentert fenomen til Hessdalen sin server

1.5.2 Leveranser

Ved prosjektslutt skal det leveres et fysisk oppsett hvor et kamera er tilkoblet en single-board datamaskin som skal lagre bilder av lysfenomen. Det skal være mulig for oppdragsgiver å koble seg til datamaskinen via SSH.

Leveranser:

- Gruppekontrakt
- Prosjektkontrakt
- Hjemmeside for prosjektet
- Forprosjektrapport
- Prosjektplakat
- Individuelle refleksjonsnotater
- Hoveddokument
- Prosessrapport

- Single-board datamaskin med tilhørende programkode
 - Oppkobling mot kamera
 - Bildeanalyse
- Utvalgt kamera montert i kamerahus

Leveranser utover original oppgavebeskrivelse:

- Funksjonalitet som gjør at deteksjonsvideoene inneholder et brukerdefinert antall sekunder før og etter fenomenet oppstår
- Maske som kan blokkere ønskede områder av bildeområde
 - Nettside/GUI som lar oppdragsgiver velge hvor maske skal blokkere
- Eventuell kode som automatisk viser fenomener på en ryddig måte på hessdalen.hiof.no

1.5.3 Metode

Gruppen har valgt å stykke opp prosjektet i tre faser:

- Startfase
- Produksjonsfase
- Sluttfase

I startfasen skal gruppen evaluere, undersøke og bestemme utstyr som må kjøpes inn for at systemet skal fungere optimalt. Her vil også all nødvendig planlegging gjennomføres for å være mest mulig produktive i neste fase. I produksjonsfasen vil det bli lagt vekt på å produsere den essensielle programkoden, montere og koble opp utstyr samt legge opp til å begynne testing i neste fase. Sluttfasen vil bestå av kontinuerlig testing av produktet hvor systemet vil gjennomgå flere iterasjoner.

Bildeanalyesen og maskering av bildeområde vil kunne jobbes på uavhengig av om et kamera er koblet opp. Montering av kamera og datamaskinen vil skje i fellesskap på et grupperom. Oppkobling og kommunikasjon mellom datamaskinen og kamera krever at gruppen undersøker og setter seg inn i kamera sitt API. Testing av kamera, bildeanalyse og maske vil skje på egnede testplattformer og i forskjellige miljøer. Dette for å forsikre at testene gir realistiske tilbakemeldinger slik at det kan gjøres utbedringer for å ende opp med best mulig sluttprodukt. Etterhvert som utviklingen av de forskjellige aktivitetene går fremover vil veileder bli oppdatert om dette på møtene slik at gruppen fortløpende får tilbakemelding på arbeidet.

Prosjektgruppen vil utføre følgende metoder for å komme i mål med leveransene:

- Litteratursøk rundt bildelagoritmer for å se hvordan de kan implementeres i prosjektet. På denne måten blir det fortgang i prosjektet, siden det ikke trengs å “finne opp hjulet på nytt”.
- Analyse av forskjellige programvare, metoder for å analysere bilder og utstyr som vil være nødvendig for oppdraget.

- Skissering og tavlebruk for hvordan det tenkes at systemet skal bygges opp. Ved skissering er det enklere å forklare visuelt hvordan gruppen tenker hvis de må rådføre seg med eksterne personer.
- Benytte personer på skolen som har erfaring innenfor bildebehandling eller andre relevante områder.
- Grundig testing av systemet, slik at gruppen oppdager eventuelle feil og mangler i den hen-sikt å gjøre sluttproduktet best mulig.
- Jevn arbeidsfordeling og mest mulig grupppearbeid slik at kommunikasjon mellom gruppe-medlemmene holder seg på et høyt nivå.

1.6 Rapportstruktur

I kapittel 2 startes det med å se på dagens system og hvordan gruppen ser for seg deres system blir. Det ses så på forskjellige teknologier som omhandler blant annet single-board datamaskiner, kameraer, filformater, programmeringsspråk, forskjellige metoder rundt bildeanalyse m.m. Over i kapittel 3 tas det for seg hvilke metoder og tekniske løsninger gruppen endte med ut i fra analysearbeidet som ble utført i kapittel 2. Det begrunnes her hvorfor gruppen valgte å gå videre med de forskjellige teknologiene. Kapittel 4 tar for seg hvordan gruppen i praksis utviklet produktet. Det blir så sett på systemets virkemåte og oppbygging i tillegg til en endelig illustrasjon av systemoversikten. Videre over i kapittel 5 tas det for seg hvordan testingen av systemet ble utført og dets resultater. Både hvor testene ble utført og hvilket utstyr som ble benyttet kan finnes her. Kapittel 6 omhandler hva gruppen har lært, erfart, forbedringspunkter og hvorvidt målene som ble satt er oppnådd. I slutten av kapittelet tas det for seg hva som kan gjøres i etterkant for å gjøre produktet enda bedre. I kapittel 7 avsluttes rapporten med en konklusjon som tar for seg sentrale deler av diskusjonskapittelet.

Kapittel 2

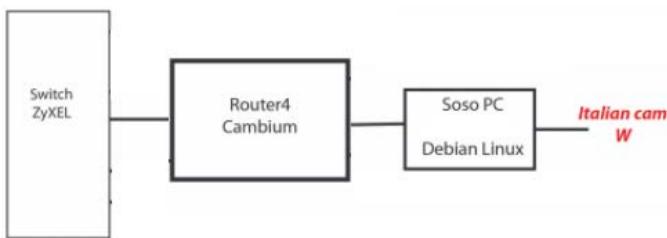
Analyse

Denne delen av rapporten tar for seg dagens situasjon, beskrivelse av tiltenkt system samt en oversikt over relevant teknologi og relatert arbeid som vil kunne være aktuelt å benytte i prosjektet.

2.1 Systemoversikt

2.1.1 Dagens system

Systemet som er i bruk i dag ble utviklet av en gruppe fra Italia og ble ferdigstilt 28. oktober 2016. Kameraet som benyttes er et analogt kamera kalt Mintron MTV-12V6HC-EX. Det er koblet opp til en PC som kjører Debian Linux hvor detekteringsprogrammet, Soso, ligger. Denne PCen er koblet til en router som igjen er koblet til en switch. Programmet reagerer når det skjer plutselige forandringer i bildene som tas. Deretter vil den lagre en kort video og sende dette samt et alarmbilde til Hessdalens server. Programmet avgjør om det er en deteksjon ved å se etter forandring i hver piksel mellom bildene.



Figur 2.1: Dagens system

2.1.2 Tiltenkt system

Oppdragsgiver ønsker et system tilsvarende det som er i drift i dag, men med teknologiske oppgraderinger. Oppgaven tilskir at en liten datamaskin skal kunne analysere bildestrømmen som kommer fra et tilkoblet kamera. Ved deteksjon av “uveitet lys” skal det tas et alarmbilde og en video som skal overføres til Hessdalens server. Muligheten for å koble seg til datamaskinen med SSH skal være til stede i tillegg til at Linux skal være maskinens operativsystem.

2.1.3 Dagens system mot tiltenkt system

Tabell 2.1 viser hva som blir forskjellig fra eksisterende system.

Tabell 2.1: Eksisterende mot tiltenkt system

Eksisterende system	Tiltenkt system	Fordel
Stasjonær PC	Single-board datamaskin	Plassbesparende, strømbesparende
Linux Debian	Linux Ubuntu 16.4	Oppdateres hver 6. måned, 5 års støtte
Analogt videokamera	Digitalt videokamera	Ingen omforming mellom analogt og digitalt, fysisk mindre, bedre videokvalitet
“Soso” bildeanalyseprogram levert av italienere	Modifisert versjon av en eksisterende bildealgoritme	Baserer seg på nyere teknologi, effektivt programmeringspråk
Analyserer kun mellom 20:30 og 07:00	Analyserer gjennom hele døgnet	Fenomener som skjer på dagtid vil bli tatt med

Funksjoner tiltenkt utover original oppgavebeskrivelse

- Mulighet til å forandre parametre/innstillinger for kamera og bildebehandlingsprogram
- Mulighet til å maskere bort deler av bildeområdet som ikke skal tas med i bildeanalyse
- Dersom det er tid nær prosjektslutt skal gruppen også presentere bildene på en visuell og ryddig måte på nettsiden

2.2 Teknologi og utstyr

2.2.1 Single-board computer (SBC)

Oppdragsgiver ønsker et plassbesparende og moderne system. Oppsettet i Hessdalen benytter en vanlig stasjonær PC, men med dagens teknologi er det ikke lenger nødvendig. Man kan få kraftige single-board datamaskiner på størrelse med et hovedkort i en vanlig stasjonær PC. En SBC¹ er en komplett datamaskin som er bygd på et mindre kretskort. Den består av en eller flere mikroprosessorer, RAM, I/O-enheter og andre nødvendige komponenter som en tradisjonell datamaskin behøver.

SBC-er var mulig å lage etter man klarte å øke tettheten av integrerte kretser. Som en følge av dette ble systemets kostnad redusert ved blant annet å eliminere kontakt- og busskretser som ellers ville bli brukt. Dens egenskaper gjør den mer relevant i scenarioer hvor en tradisjonell datamaskin ville vært overflødig i pris, kraft, strømforbruk og ikke minst fysisk størrelse. I dag er SBC-er mest brukt i industrielle sammenhenger. Single-board datamaskinene gruppen har vurdert benytter ARM-arkitektur i prosessorene sine.

¹https://en.wikipedia.org/wiki/Single-board_computer

2.2.2 ARM-arkitektur

ARM², kort for “Advanced RISC Machine”, tidligere “Acorn RISC Machine”, er en 32-bits prosessarkitektur som er kjent for dens lave strømforbruk. Dette gjør den godt egnet for mindre enheter som mobiltelefon, nettbrett, mikrokontroller og single-board datamaskiner.

2.2.3 Raspberry Pi

Raspberry Pi³ er en anerkjent single-board datamaskin som ble lansert 29. februar 2012. Den ble utviklet i den hensikt å lære skolebarn grunnleggende programmering, men det viste seg raskt at dens bruksområde slo godt an i industrien og hos hobby-entusiaster.

Innen 2017 var det blitt solgt mer enn 11 millioner Raspberry Pi-er.



Figur 2.2: Raspberry Pi [24]

Raspberry Pi 3 model B

Raspberry Pi 3 model B ble lansert februar 2016 og er tredje generasjonen av datamaskinen. Model B bringer med seg en 1,2 GHz ARM Cortex-A53 CPU, 1 GB RAM, trådløs LAN og bluetooth. Som sin forgjenger har den også blant annet 4 USB-porter, 40 GPIO pins, HDMI-port m.m. Lagringsplass må kjøpes separat i form av SD-kort. Strømforbruket er estimert å være mellom 750mA og 1.2 A. [33]

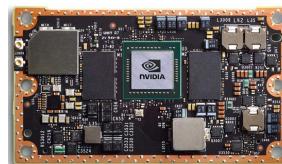
SBC-en kommer ikke med et operativsystem installert, så man står fritt til å velge om en vil benytte Linux (Raspbian), Windows (Windows IoT) o.l.

²[https://no.wikipedia.org/wiki/ARM_\(prosessorarkitektur\)](https://no.wikipedia.org/wiki/ARM_(prosessorarkitektur))

³https://en.wikipedia.org/wiki/Raspberry_Pi#History

2.2.4 NVIDIA Jetson

10. november 2015 annonserte NVIDIA at de ville slenge seg med på single-board datamaskinbølgen med deres Jetson TX1⁴. Det unike med TX1 var at det var den første av sitt slag som kom med GPU maskinvare, med visual computing og “on-the-fly-sensing” i tankene, for en plattform som bruker under 10 watt. De eksisterende produktene som Raspberry Pi og BeagleBone kjørte Linux godt, men de var ikke spesielt kraftige.



Figur 2.3: NVIDIA Jetson TX1 [18]

2.2.5 NVIDIA Jetson TX2 Developer Kit

TX2 ble lansert 14. mars 2017 og er storebroren til TX1. SBC-en kjører operativsystemet Linux, i form av distribusjonen Ubuntu. Enheten er laget for tung visuell databehandling, samtidig som den er strømbesparende. Developer kittet kommer ferdig installert med Linux Ubuntu og støtter NVIDIA Jetpack SDK.

NVIDIA Jetpack SDK støtter blant annet:

- Deep learning
- Deep neural network
- Computer vision
- Cuda (GPU accelerated application)

TX2 kommer med NVIDIAs 256-CUDA core Pascal GPU(1300MHz), to CPU'er; ARM Cortex-A57(quad-core) 2GHz og NVIDIA Denver2(dual core) 2GHz, 8GB RAM, trådløs, bluetooth, 2 USB porter; en USB3 og en USB2, samt 40 GPIO porter. I tillegg kommer den med 32GB lagringsplass. Strømforbruket er estimert å være mellom 810mA og 2.5A.[29]



Figur 2.4: NVIDIA Jetson TX2 Developer kit [19]

⁴<https://devblogs.nvidia.com/nvidia-jetson-tx1-supercomputer-on-module-%2drives-next-wave-of-autonomous-machines/>

2.3 Kamera og linse

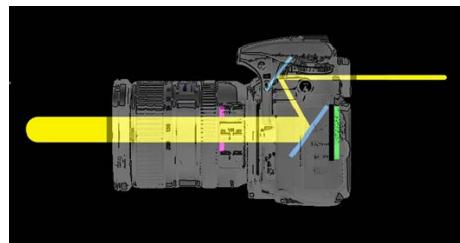
Det finnes en rekke forskjellige kameratyper å velge blant. Her er noen av de gruppen vurderte:

2.3.1 Speilrefleks

Speilreflekskamera bruker et flyttbart speil mellom lukkeren og linsen. Det fungerer slik at lyset trenger inn i frontlinsen, gjennom objektivet og treffer et speil foran lukkeren som reflekterer lyset direkte til søkeren. Det betyr at det som vises i søkeren, er mer eller mindre nøyaktig det samme som blir fanget av CCD-brikken når utløseren trykkes og speilene vippes opp.



Figur 2.5: Speilreflekskamera [27]



Figur 2.6: Speilrefleks virkemåte [28]

Nikon D7500

Kamera som kan ta bilder og video opp til 3840x2160 oppløsning, med 20.9 megapiksel. Kameraet har også innebygd WiFi og kan filme i 4K.



Figur 2.7: Nikon D7500 [17]

2.3.2 Digitalkamera

Moderne digitale kompaktkameraer er utstyrt med flere funksjoner, de kan ta film og gjøre lydoppdrag. Hos et digitalkamera føres lyset som danner motivet inn på en bildebrikke som er oppdelt i et antall kvadratiske piksler. Antall piksler utgjør kameraets oppløsning.



Figur 2.8: Digitalkamera [9]

Sony Cybershot DSC-HX90V

Kamera som kan ta bilder i 19020x1080 oppløsning, med 18.2 megapiksel. Det har innebygd solskjerm og en asfærisk linse.



Figur 2.9: Sony Cybershot DSC-HX90V [26]

2.3.3 Overvåkningskameraer

Kommer med masse ekstra funksjonalitet. Eksempler på dette er bevegelsesregistrering for å oppdage mennesker, vidvinkellinse, innebygd mikrofon og høyttalere, spotlight lys, oppkobling mot 110 og trådløst.



Figur 2.10: Overvåkningskamera [21]

D-Link DSC-7110

Et såkalt nettverksovervåkingskamera med 2MP bildesensor. Kan filme 1920x1080 med 30 fps. Kommuniserer via ethernet.



Figur 2.11: D-Link DCS-7110 [4]

2.3.4 Industrikameraer

Industrielle kameraer brukes til enkle overvåknings- og meteorologiske-oppgaver for kvalitetskontroll via bildebehandling. I motsetning til tradisjonelle kameraer har de ingen knapper på seg, som kamerautløser eller lignende. De industrielle kameraene er også ofte lysømfintlige.



Figur 2.12: Industrikamera [14]

DFM 21BU04-ML

Kameraet kan ta 60 bilder i sekundet, men har en oppløsning på 640x480. Kameraet er tilpasset mange oppgaver hvor det er begrenset plass for et kamera.



Figur 2.13: DFM 21BU04-ML [7]

DFM 31BU03-ML

Industrikamera som tar bilder og video i 1024x768 oppløsning. Kameraet har mulighet for å ta 30 bilder i sekundet. Kameraet er utstyrt med en CCD sensor, som gir god bilde kvalitet selv med skiftende lys forhold.


*LENS & HOLDER NOT INCLUDED

Figur 2.14: DFM 31BU03-ML [8]

DFK 31BU03.H

Kamera kommer med samme innmat og spesifikasjoner som foregående kamera, DFM 31BU03-ML. Forskjellen er at kameraet er dekket av et kamerahus for ekstra beskyttelse.



Figur 2.15: DFK 31BU03.H [6]

2.4 Filformat

2.4.1 AVI

AVI⁵ er et format introdusert av Microsoft som en del av deres ‘video for Windows’-programvare. Filformatet kan inneholde både video og lyd lagret.

2.4.2 MP4

MP4⁶ er stort sett brukt til å lagre video og lyd, men brukes også til å lagre data som undertekster og stillbilder. Filformatet lar deg strømme video direkte i nettleseren.

2.4.3 JPEG

JPEG⁷ er en vanlig metode brukt for komprimering av digitale bilder. Graden av komprimering kan justeres, slik at det kan velges mellom god kvalitet eller liten filstørrelse. JPEG støtter RGB og gråskala-bilder.

⁵https://en.wikipedia.org/wiki/Audio_Video_Interleave

⁶https://en.wikipedia.org/wiki/MPEG-4_Part_14

⁷<https://en.wikipedia.org/wiki/JPEG>

2.4.4 PNG

PNG⁸ er et grafisk filformat som støtter minimalt tap ved datakomprimering. Filformatet støtter RGB, RGBA og gråskala-bilder. RGBA betyr at det støtter Alpha channelen, som igjen betyr at man kan lagre bilder med gjennomsiktig bakgrunn.

2.5 Codecser

2.5.1 H264

H264⁹ er et videokomprimeringcodecs utviklet for bruk i systemer som HDTV, BluRay, HD DVD, men også lav-oppløsning systemer. I 2014 var filformatet det mest brukte formatet for opptak, komprimering og distribuering av video. Formålet med H264 var å lage en standard som kunne gi god bildekvalitet ved en lav bitrate.

2.5.2 WMV

WMV¹⁰ er en videocodecs utviklet av Microsoft. WMW ble opprinnelig utviklet for streaming til internett applikasjoner, men ble i stor grad utbredt i bruk på HD DVD og BluRay.

2.5.3 DivX

DivX¹¹ codecsen er kjent for sin evne til å komprimere lange videoklipp til små størrelse, samtidig som den opprettholder relativt høy visuell kvalitet.

2.6 Encoding

2.6.1 Bayer

Et Bayer filter¹² benyttes for å arrangere RGB fargefiltre på fotosensorer. Bayers spesielle ordning av farger blir brukt i de fleste enkelt-chip digitale bildesensorer som benyttes i digitalkameraer, videoopp takkere og scannere for å oppnå et fargebilde. Filtermønsteret består av 50% grønn, 25% rød og 25% blå.

2.6.2 YUV

YUV¹³ er et fargekodesystem som blir brukt som en del av fargebilde analyse. Det encoder et fargebilde eller video og tar det menneskelige aspektet med i betrakningen. Dette gjør at overføringsfeil eller komprimering blir bedre skjult fra den menneskelige oppfatningen enn ved å bruke en direkte RGB-representering.

⁸https://en.wikipedia.org/wiki/Portable_Network_Graphics

⁹https://en.wikipedia.org/wiki/H.264/MPEG-4_AVC

¹⁰https://en.wikipedia.org/wiki/Windows_Media_Video

¹¹<https://en.wikipedia.org/wiki/DivX>

¹²https://en.wikipedia.org/wiki/Bayer_filter

¹³<https://en.wikipedia.org/wiki/YUV>

2.7 Programmeringsspråk og annen teknologi

De siste 30 årene har det tiltrådt et mangfold av forskjellige programmeringsspråk for å dekke forskjellige behov og som spesialiserer seg på forskjellige aspekter.

2.7.1 Python

Python¹⁴ er et høynivå programmeringsspråk som blant annet baserer seg på såkalt objektorientert programmering hvor du på konseptnivå arbeider med ‘objekter’ som kan inneholde data i det som betegnes som objektets attributter.

Språket ble laget med en designfilosofi som legger vekt på kodelesbarhet og en syntaks som tillater programmerere å skrive kode med færre linjer. Som mange andre språk er Python inspirert av det grunnleggende språket “C”.



Figur 2.16: Python [23]

2.7.2 C++

Høynivå programmeringsspråket C++¹⁵ er også et utspring fra “C”. Det er et “general-purpose-programming language” som vil si at det er designet for å skrive software for forskjellige applikasjoner. Som Python er det objektorientert og tilbyr i tillegg lavnivå memory manipulasjon.



Figur 2.17: C++ [2]

2.7.3 HTML

HTML¹⁶ er standardspråket for å lage nettsider. HTML står for Hyper Text Markup Language og beskriver strukturen til nettsiden ved hjelp av tags. Nettlesere viser ikke disse HTML taggene, men leser dem og gjengir de som innhold på nettsiden.



Figur 2.18: HTML [11]

¹⁴https://en.wikipedia.org/wiki/Object-oriented_programming

¹⁵<https://en.wikipedia.org/wiki/C%2B%2B>

¹⁶https://www.w3schools.com/html/html_intro.asp

2.7.4 CSS

CSS¹⁷ beskriver hvordan HTML elementer skal vises på skjerm eller andre medier. CSS står for Cascading Style Sheets og kan kontrollere utformingen til flere nettsider samtidig.



Figur 2.19: CSS [3]

2.7.5 JavaScript

JavaScript¹⁸ er programmeringsspråket til HTML som kan få hendelser til å skje uten å måtte laste inn nettsiden på nytt. Det er et høynivåspråk som kjører på klienten, altså nettleseren til brukeren. JavaScript karakteriseres som dynamisk og gjør nettsider interaktive.



Figur 2.20: JavaScript [15]

2.7.6 AJAX

AJAX¹⁹ er ingen ny teknologi eller et språk, men benytter i stedet eksisterende teknologi på nye måter. Dette lar programmeren sende og hente data fra en server asynkront, noe som betyr at alt skjer i "bakgrunnen" uten at den besøkende ser siden jobbe.

Dette åpner muligheten for å oppdatere innholdet på en nettside uten å laste den inn på nytt. Det kan bes om, mottas og sendes data etter at siden er lastet inn.



Figur 2.21: AJAX [1]

¹⁷https://www.w3schools.com/css/css_intro.asp

¹⁸<https://en.wikipedia.org/wiki/JavaScript>

¹⁹[https://en.wikipedia.org/wiki/Ajax_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming))

2.7.7 PHP

PHP²⁰ er et serverskriptingspråk, og et kraftig verktøy for å lage dynamiske og interaktive nettsider. PHP skript blir kjørt på serveren, og kan dermed inneholde sensitiv informasjon som oppkobling mot database og innloggingssystemer.

Språket er så kraftig, utbredt og fullstappet med funksjoner at Facebook benytter språket. Det kan generere dynamisk sideinnhold, håndtere filer, hente inn data fra skjemaer, opprette og håndtere cookies, håndtere databaser, kontrollere brukertilgang og kryptere data.



Figur 2.22: PHP [22]

2.8 API

Et Application Program Interface (API)²¹ er rutiner, protokoller og verktøy for å bygge programvare. Det definerer hvordan de ulike programvarekomponentene skal håndteres. APIer blir også brukt når en skal programmere et grafisk brukergrensesnitt (GUI). Et godt API gjør det enklere å utvikle programvare ved å tilby alle de ulike delene som trengs. En programmerer må så sette sammen disse delene til et komplett program.

Et typisk eksempel er Google Maps API, som lar programmeren enkelt hente ut kartdata og legge til ekstra funksjonalitet i form av eksempelvis piler, veimarkering og farger.

2.8.1 OpenCV

OpenCV²² er et bibliotek med programmeringsfunksjoner rettet mot bilde og videobehandling. Biblioteket har mer enn 2500 optimaliserte algoritmer. Disse algoritmene kan brukes til å detektere ansikter, objekter, fange bevegelser og mer. Det er utviklet for effektiv beregning med stort fokus på sanntidsapplikasjoner. Sammen med OpenCL utnytter det maskinvareakselerasjon fra maskinen som benytter biblioteket. OpenCV støtter Windows, Linux Mac OS, iOS og Android og er laget med med Python, C++ og Java grensesnitt.



Figur 2.23: OpenCV [20]

²⁰https://www.w3schools.com/php/php_intro.asp

²¹<https://www.webopedia.com/TERM/A/API.html>

²²<https://opencv.org/>

2.8.2 GStreamer

GStreamer²³ er et bibliotek for å håndtere komponenter som kamera og videofiler. Rammeverket tillater blant annet skapelsen av en rekke mediakomponenter, deriblant avspilling av lyd, video, streaming og redigering.



Figur 2.24: GStreamer [10]

2.8.3 Scikit-Image

Scikit-Image²⁴ er et bildebehandlings bibliotek for Python. Det inneholder algoritmer for segmentering, analyse, filtrering og mer. Den er designet for å samvirke med Pythons numeriske og vitenskapelige biblioteker NumPy og SciPy.



Figur 2.25: Scikit-Image [25]

2.8.4 ImageMagick

En programvarepakke for visning, konvertering og redigering av bilder²⁵. Programvaren består hovedsakelig av en rekke kommandolinjer for å manipulere bilder. For Unix-lignende operativsystemer er det et GUI kalt IMDisplay for gjengivelse og manipulering av bilder.



Figur 2.26: ImageMagick [13]

2.9 Detektering

Problematikken rundt det å oppdage lysfenomener på himmelen, er at det er flere ting enn fenomener som lyser. Fra foregående oppsett er flytrafikken er problem. Dette blir relativt vanskelig å unngå, med tanke på at det både beveger seg og gir fra seg lys når det passerer. En annen ting som kan skape problemer er månen, siden den kan oppstå i forskjellige former og posisjoner. På dagtid vil fugler kunne være en årsak for feilalarmer.

²³<https://gstreamer.freedesktop.org/>

²⁴<http://www.scipy-lectures.org/packages/scikit-image/>

²⁵<https://www.imagemagick.org/script/>

2.10 Bildeanalyser

Gruppen hadde tidlig i prosessen et møte med Høgskolens ekspert på bildebehandling, som foreleser om dette, Lars Vidar Magnusson. Han hadde tanker om hvordan gruppen burde gå frem angående detekteringen av fenomenene. Blant de foreslalte var Motion detection og Connected Components. Gruppen kom selv med forslagene Blob detection og FindContours.

2.10.1 Blob detection

Blob detection gjør at piksler som ligger intil hverandre og med omtrentlig lik RGB-farge blir slått sammen til et større objekt. Har man eksempelvis en hånd med spredte fingre vil hele hånden slås sammen til et stort objekt. Deretter vil man se etter objekter med en størrelse over en satt verdi. Blobs av vilkårlige størrelser kan markeres ved å forandre på størrelse parameteret.[31]

2.10.2 Motion detection

Ved motion detection [30] ses det etter bevegelse og forandring i et bilde fra et annet. Dette gjenomføres i praksis ved at man har et bilde, herav kalt referansebilde. Deretter blir alle nye bilder som tas sammenlignet med referansebildet for å se etter forandringer. Referansebilde kan eksempelvis oppdateres og byttes ut hvert femte sekund.

2.10.3 ConnectedComponents labeling

Labeling fungerer ved å skanne et bilde, piksel for piksel, fra topp til bunn og venstre til høyre, for å identifisere tilkoblede piksel samlinger.[34] En annen metode benytter tilsvarende logikk, men i tillegg ser den etter tilkoblinger diagonalt. Piksler som ligger i nærheten blir slått sammen til et objekt. Det er mulig for bruker å spesifisere hvordan det skal søkes etter denne tilkoblingen.

2.10.4 FindContours

FindContours er en innebygd funksjon i OpenCV som ser etter piksler som ligger nærmre hverandre og har samme farge eller intensitet. Disse nærliggende pikslene blir sett på som et objekt[32]. Alle objektene legges i en array med koordinatene de befinner seg på.

2.11 Maskering av bildeområde

Oppdragsgiver ønsket å kunne maskere områder slik at de ikke ble tatt med i analysen. Dette fordi kamera skal eksempelvis kunne rettet mot trær, bilvei eller lys som ikke skal gi utslag og detekteres som et fenomen. Det ble her diskutert hvordan dette enklest kan håndteres av bruker.

Noen forslag som kom opp var:

- Bruker laster ned et bilde, tatt i Hessdalen, til sin maskin. Bruker åpner så bilde i et program som MS Paint og tegner på de områdene som det ikke ønskes skal tas hensyn til i bildeanalyse. Etter at bildet er lagret på brukers lokale maskin, må det manuelt lastes opp til SBC-en.
- Bruker laster ned og installerer et program som en engangsprosedyre. Bruker starter programmet og får fremvist et bilde tatt i Hessdalen. Det kan så tegnes direkte oppå bilde inne i programmet. Til slutt trykkes det på en lagrekapp, og det ferdigtegnede bilde lastes automatisk opp til SBC-en.
- Bruker besøker en nettside. Her får personen automatisk fremvist et bilde tatt i Hessdalen. Det kan så tegnes direkte oppå bilde de områdene som ikke ønskes analysert. Til slutt trykkes det på en lagrekapp, og det ferdigtegnede bilde lastes automatisk opp til SBC-en. Enhver datamaskin kan benyttes til å besøke nettsiden og endre på masken.

2.12 Innstillinger, parametere

Gruppen anser det som relevant å kunne endre på innstillinger som

- Hvor mange sekunder før og etter en hendelse som skal beholdes på video
- Pikselstørrelse på objektene som skal detekteres
Minimum og maksimum
- Plassering og innhold på tekst som skal ligge oppå bilde og video

Innstillinger som vurderes, men ikke nødvendigvis er relevante å kunne endre

- Format
Om det skal være farger eller ikke på bilde og video
- FPS
Hvor mange bilder som skal tas hvert sekund
- Oppløsning
Hvor mange piksler et bilde skal inneholde

2.13 Plassering

Slik det foregående oppsettet har vært, er kameraet plassert i et kamerahus slik at det beskyttes mot vær og vind. Det må testes og vurderes om det er mulig å plassere SBC-en inne i kamerahuset sammen med kamera. Hvis ikke må SBC-en plasseres innendørs og deretter trekkes en kabel mellom de to husene.

2.14 Kommunikasjonsprotokoller

2.14.1 SSH

Secure Shell²⁶ er en nettverksprotokoll som normalt benyttes for å få tilgang til en annen maskins kommandolinje. SSH er å anse som sikkert da all data mellom de kommuniserende enhetene er kryptert.

2.14.2 SCP

Secure Copy²⁷ er en nettverksprotokoll for sikker overføring av filer mellom en lokal og ekstern enhet. Det er basert på SSH protokollen for overføring av data og bruker samme mekanisme for autentifikasjon.

2.15 Lagring av media

2.15.1 Lagring lokalt

Video og bilder lagres lokalt på datamaskinen. En flaskehals vil være den begrensede lagringsplassen de fleste SBC-er har.

2.15.2 Lagring eksternt(harddisk)

Hvis det viser seg at det er begrenset med plass på SBC-en, er det mulig å koble til en ekstern harddisk og lagre data på denne.

2.15.3 Lagring til server

Video og bilder lagres til server via en SCP forbindelse.

2.16 Alternativer

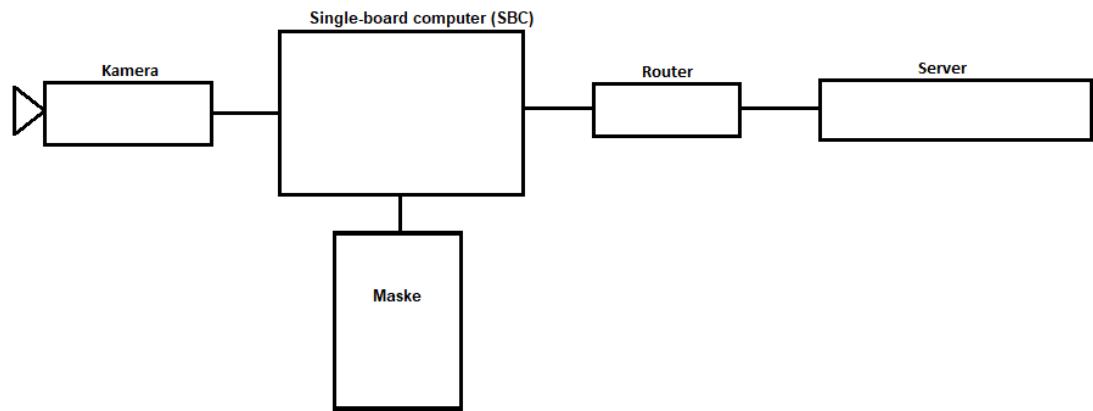
Dersom det skulle vise seg at enten kamera eller avgjørende programvare ikke vil fungere på Linux er det, ved diskusjon med oppdragsgiver, mulig å bytte operativsystem til Windows. Dette avviker fra oppdragsgivers ønske, men er i verste tilfelle et alternativ.

²⁶https://en.wikipedia.org/wiki/Secure_Shell

²⁷https://en.wikipedia.org/wiki/Secure_copy

2.17 Systemoversikt - revisjon 1

Ved oppstart av prosjektet samlet gruppen seg for å skisse opp hvordan oppsettet av systemet kunne se ut. Denne kan modifiseres over prosjektets løp.



Figur 2.27: Systemoversikt - revisjon 1

Kapittel 3

Utforming

Dette kapittelet tar for seg hvilke valg som har blitt gjort og hva slags utstyr gruppen har valgt å bruke i de forskjellige delene av systemet. Valgene som blir gjort i dette kapitlet vil være utgangspunktet for implementeringen som blir tatt for seg i neste kapittel.

3.1 Valg av hardware

3.1.1 Single-board datamaskin (SBC)

Da det kom til valg av SBC stod det mellom to typer; Raspberry Pi modell B og NVIDIA Jetson TX2. Førstnevnte var beskrevet ordrett i oppgavebeskrivelsen, men det ble tidlig reist tvil innad i gruppen om Raspberry Pi-en ville kunne klare å utføre bildebehandlingsoperasjonene oppgaven krever.

I starten av prosjektet lånte gruppen en Raspberry Pi, i påvente av å bestille en egen. Etter tre ukers arbeid med denne hadde gruppen et møte med Lars Vidar Magnusson, som underviser i bildebehandling på Høgskolen i Østfold. I dette møtet nevnte han NVIDIA Jetson TX2 som en mulig erstatter for Raspberry Pi. Et av argumentene var at TX2-en har en egen grafikkprosessor(GPU) som har tilgang til hele systemets RAM, noe som gjør at den kan gjennomføre oppgavene langt mer effektivt enn en Raspberry Pi.

Ut fra analysen i kapittel 2.2 ser man blant annet følgende fordeler med TX2:

- Bedre og flere CPU'er
- Egen GPU
- Mer RAM

I en benchmarktest¹ utført av hackaday.com viste TX2 seg hele 4 ganger raskere enn Raspberry Pi-en.



Figur 3.1: Benchmarktest av Jetson TX2 og Raspberry Pi

¹<https://hackaday.com/2017/03/14/hands-on-nvidia-jetson-tx2-fast-processing-for-embedded-devices/>

SBC-en denne oppgaven har behov for krever mye prosesseringsskraft for bildeanalyse og når TX2 har flere CPU-er og en dedikert GPU, står denne frem som beste alternativet. Etter å ha tatt opp dette med oppdragsgiver, var alle parter enige om at det var et smart valg å bytte fra Raspberry Pi 3 til NVIDIA Jetson TX2 for å ende opp med best mulig sluttresultat. Oppgaven gikk fra dette tidspunktet vekk fra original beskrivelse som beskrev bruk av RBi.

3.1.2 Kamera

DFK 31BU03.H

Oppdragsgiver hadde ingen spesifikke krav til hva slags kamera som skulle benyttes til oppgaven, og siden ingen av gruppemedlemmene har jobbet mye med kamera før måtte de lese seg opp på området.

Gruppen startet ved å se på speilreflekskamera, men oppdaget fort at denne typen ble unødvendig mye med tanke på funksjonalitet, utstyr og kostnad for dette prosjektet. Gikk deretter videre til å se på digitalkameraer, men også denne typen har funksjoner som ikke er nødvendig for vårt bruk siden det ikke er noen som skal fysisk operere kameraet. Det er heller ikke nødvendig med display siden kameraet skal stå i et kamerahus. Gruppen tittet så på overvåkningskameraer, men gikk vekk fra idéen siden det følger med veldig mye ekstra tilbehør og fordi det var usikkerhet om kameraet ville gi gode nok bilder til bildeanalyesen. Til slutt kom gruppen over industrikameraer, noe som virket veldig relevant for oppgaven.

Industrikameraene gruppen vurderte er mindre kameraer som ikke har normal funksjonalitet som utløserknapp og display. De har kuhirose-kontakt, eventuelt BNC-kontakt dersom en vil bruke ekstern utløser. Denne typen kamera virker optimale for å kobles og brukes gjennom en SBC.

Etter å undersøkt de analyserte kameraene, vurdert pris, egenskaper og diskutert hva som passet bruksområdet vårt best, bestemte gruppen seg for å skaffe et industrikamera av typen "DFK 31BU03.H". Avgjørelsen for å gå for dette kameraet kom etter telefonsamtaler til USA hvor gruppen var i kontakt med en salgsrepresentant fra "OEM Cameras" som svarte på forhåndspreparerte spørsmål angående kameraet. Han kunne bekrefte at kameraet støttet bruk av Linux, noe som var ett satt krav fra oppdragsgiver side. At det var lysømfintlig med en lux på 0.15, samt at det kom med ekstra beskyttelse, var også avgjørende. Da kameraet skulle bestilles fra USA, ble gruppen informert at dette kunne ta opp til 4 uker. Det ble derfor funnet en leverandør fra Tyskland som kunne levere samme kamera i løpet av to uker.

3.1.3 Linse

TCL 0616

Det var ikke nødvendig å gå i dybden på valg av linse i kapittel 2 før det var bestemt hvilken type kamera som skulle benyttes. Dette fordi digitalkamera og overvåkningskamera kommer med en fastmontert linse. Da valget falt på DFK 31 gikk gruppen i kontakt med en salgsrepresentant fra TheImagingSource². Han kom med forslag til hvilken linse som ville passe best for kameraet og oppgaven gruppen beskrev for han. Siden gruppemedlemmene ikke har noen erfaring med industrikamera, ble dette forslaget fulgt. Kameraet ble derfor bestilt med en tilhørende "TCL 0616" 5-megapikslers linse.

²<https://www.theimagingsource.com/company/about/>

3.2 Valg av programvare

3.2.1 Filformat

JPEG/MP4

Ut i fra de forskjellige filformatene gruppen vurderte falt valget på å bruke hovedsakelig JPEG og MP4 når det skal behandles film og video. JPEG benytter komprimering som betyr at unødvendig informasjon blir permanent slettet, noe som fører til liten størrelse på bildefilene. Årsaken til at valget falt på MP4 er at dette formatet tillater avspilling av video direkte i nettleseren, i tillegg er filstørrelsen til MP4 filer små i forhold til AVI-filer.

PNG

Ved lagring av masken blir filformatet PNG benyttet siden det tillater lagring av bilder med gjennomsiktig bakgrunn, som er en nødvendighet i dette tilfellet.

3.2.2 Codecs

H264

Etter å ha testet de forskjellige codecsene, ble gruppen enige om å bruke H264 på grunn av dens egenskaper for opptak og komprimering av video.

3.2.3 Encoding

Bayer filter

Gruppen bestemte seg for å bruke et bayer filter etter å ha testet både YUV og bayer. Årsaken er at bayer filteret produserer et klarere bilde ved filming. Filteret gjør også at det ikke er noen form for misfarging på bildet.

3.2.4 Programmeringsspråk

Gruppen har valgt å benytte Python som sitt primære programmeringsspråk, da det finnes API til Python som støtter bildebehandling og andre funksjoner prosjektet vil ha behov for. I tillegg har alle gruppemedlemmer arbeidet med språket fra tidligere prosjekter, slik at koden vil kunne bli skrevet mer ryddig og effektiv enn om gruppen skulle lært seg ett nytt.

JavaScript vil bli benyttet til all klientside-scripting, eksempelvis å muliggjøre tegning på en nettside. JavaScript kommer også til å kalle på ulike PHP filer ved hjelp av AJAX for å slippe å laste inn nettsiden på nytt når oppgaver skal utføres. Via PHP og SSH vil det kommuniseres mellom nettside og SBC.

3.2.5 API

OpenCV/Gstreamer

Gruppen begynte med ImageMagick, hvor det ble testet med forskjellige kommandolinjer. Dette var en programvare som ble brukt helt i starten av prosjektet, hvor gruppen prøvde å få kameraet til å operere optimalt. Programvaren fungerte greit for de små problemene, men det ble klart at dette ikke var noe gruppen kunne bruke videre siden det for det meste tok for seg konvertering og redigering av bilder.

Scikit-image ble testet ut i forhold til filtrering og analyse av en bildestrøm, men hadde ikke like mye funksjonalitet som OpenCV.

Gstreamer viste seg å være veldig effektiv når det kom til lagring av video, derfor har gruppen implementert det i koden. Ved å benytte Gstreamer kan det enkelt bestemmes hva slags parametere en video skal lagres etter. Senere kom gruppen over OpenCV, som er et bibliotek med funksjoner rettet mot video og bildebehandling. Siden det kan brukes for å detektere objekter og fange bevegelser, var dette veldig relevant for oppgaven. Gruppen tok i bruk funksjoner fra OpenCV når koden til bildeanalyesen skulle lages.

3.2.6 Bildeanalyse

Den første detekteringsmetoden gruppen testet ut var “blobDetection”. Her ble det sett etter pikselsamlinger på bildet som var lysere enn en satt verdi. Dersom samlingene var lik eller større enn gitt verdi ville de bli slått sammen og bli markert med en rød sirkel. Denne metoden fungerte bra da gruppen jobbet med å analysere to bilder med hverandre, men da det skulle analyseres video fungerte ikke denne type metode.

Gruppen gikk så over til å benytte “findContour” i analysen, noe som fungerte bedre ved analyse av film. Programmet klarte å følge et objekt mens det forflyttet seg med markering rundt selve objektet. Denne analysen fokuserer også på å detektere fenomener som er innenfor en gitt størrelse, siden dette tillot gruppen å luke ut eventuelle små objekter som stjerner eller store objekter som månen.

Etter grundig testing demonstrerte gruppen analysen for oppdragsgiver. Ved dette møte kom det frem at fenomenene kan oppstå mindre enn stjerner, noe som gruppen ikke hadde tatt høyde for. Oppdragsgiver ville også at det skulle implementeres motion detection.

Gruppen måtte derfor komme opp med en ny og endelig metode som kunne detektere fenomener som er mindre enn stjerner og som står helt stille. I forsøk på å løse dette, og for å tilfredsstille oppdragsgivers ønsker, gikk gruppen for motion detection som tar et referansebilde og ser etter forandringer. Mer om dette i kapittel 4.

3.2.7 Maske

Som nevnt i 2.10 hadde gruppen tre forskjellige forslag for hvordan tegning av maske kunne implementeres.

Alternativ 1 som foreslår å benytte Paint og manuell opp/nedlastning blir unødvendig tungvint og krever mye arbeid fra bruker. Et stort problem her er at masken vil lagres tegnet oppå et tilfeldig bilde fra Hessdalen, noe som gjør masken vanskelig å håndtere i programkode.

Alternativ 2 er heller ikke optimalt ettersom det krever at en laster ned programvare på sin datamaskin, noe som må gjøres på alle maskiner som skal endre på masken. Det er en nærliggende løsning for bruker.

Alternativ 3 er en helautomatisert løsning for bruker. Man kan fra enhver datamaskin besøke nettsiden og endre på masken. En stor fordel er at man kan få det til å se ut som man tegner masken på et bilde fra Hessdalen, men at masken i realiteten er et lag som ligger oppå bilde. Dermed lagres masken med en gjennomsiktig bakgrunn der det ikke er tegnet. Dette bilde bruker skal kunne tegne oppå. Gruppen valgte følgelig å gå for dette alternativet.

Ved opprettelse av en nettside ses det på som hensiktmessig å implementere passordbeskyttelse. Dette for å forhindre at uønskede personer kan endre på masken.

3.2.8 Innstillinger og parametere

Ettersom gruppen bestemte seg for å håndtere masken med en nettside falt det seg naturlig å la bruker kunne endre innstillinger på samme nettside. Med en nettside kan grensesnittet gjøres rent og brukervennlig ved at bruker kan endre på innstillingene med et museklikk i stedet for å måtte endre programkode.

Bruker vil ha muligheten til å forandre følgende innstillinger via nettsiden:

- Antall sekunder før og etter en deteksjon som skal inkluderes i trimmen
- Minimums- og maksimumsstørrelse på objekter som skal detekteres
- Antall frames før referansebilde skal oppdateres
- Antall frames som må inneholde en deteksjon før det startes å filme og tas et alarmbilde
 - Denne innstillingen er opprettet for å unngå deteksjon ved eksempelvis regn og snø som passerer kameraområde innenfor få frames
- Antall frames som ikke skal inneholde en deteksjon før det sluttet å filme
 - Denne innstillingen er opprettet for å slå sammen fenomener som skjer tett opp til hverandre
- Tekst som skal ligge oppå alle bilder og video som lagres ved deteksjon

3.2.9 Plassering

Hvor kameraet skal plasseres ble etter møte med oppdragsgiver, den 11.04.18, bekreftet å være på samme sted som dagens kamera³, med bildeområde over til andre siden av dalen.

Det stod mellom å plassere kamera og SBC sammen i kamerahuset eller hver for seg. Etter å fått tilgang til oppdragsgivers tiltenkte kamerahus ble det åpenbart at det ikke er plass til SBC-en. Det betyr at SBC-en må plasseres innendørs.

3.2.10 Overføring av deteksjoner til Hessdalen.hiof.no

Som det kommer frem i kapittel 1.4 ønsker oppdragsgiver at det skal være mulig å kommunisere med SBC-en via SSH. Ved hjelp av en SSH klient, som eksempelvis PuTTY, vil det være mulig å koble seg til SBC-en dersom en vet IP-adressen, brukernavnet og passordet. Slik gruppen tenker å designe programmet vil det være behov for å overføre bilde- og videofiler fra SBC-en til Project Hessdalen sin server. Dette vil antageligvis utføres ved hjelp av kommunikasjonsprotokollen SCP.

All håndtering og behandling av videofiler vil skje på SBC-en før det lastes videre opp til serveren. Dette for å unngå at unødvendig store videofiler blir lastet opp til serveren.

Tabell 3.1: Overføringsmetoder som kommer til å bli brukt

Bekrivelse	Metode
Fra SBC til Hessdalen.hiof.no	SCP
Fra nettside til SBC (lagre maske & innstillinger, hente bakgrunnsbilde)	SFTP
Fra nettside til SBC (starte og stoppe program, hente status på program)	SSH

³<http://hessdalen.hiof.no/stasjon/lokalisering.shtml>



Figur 3.2: Nåværende oppsett av kamera [5]

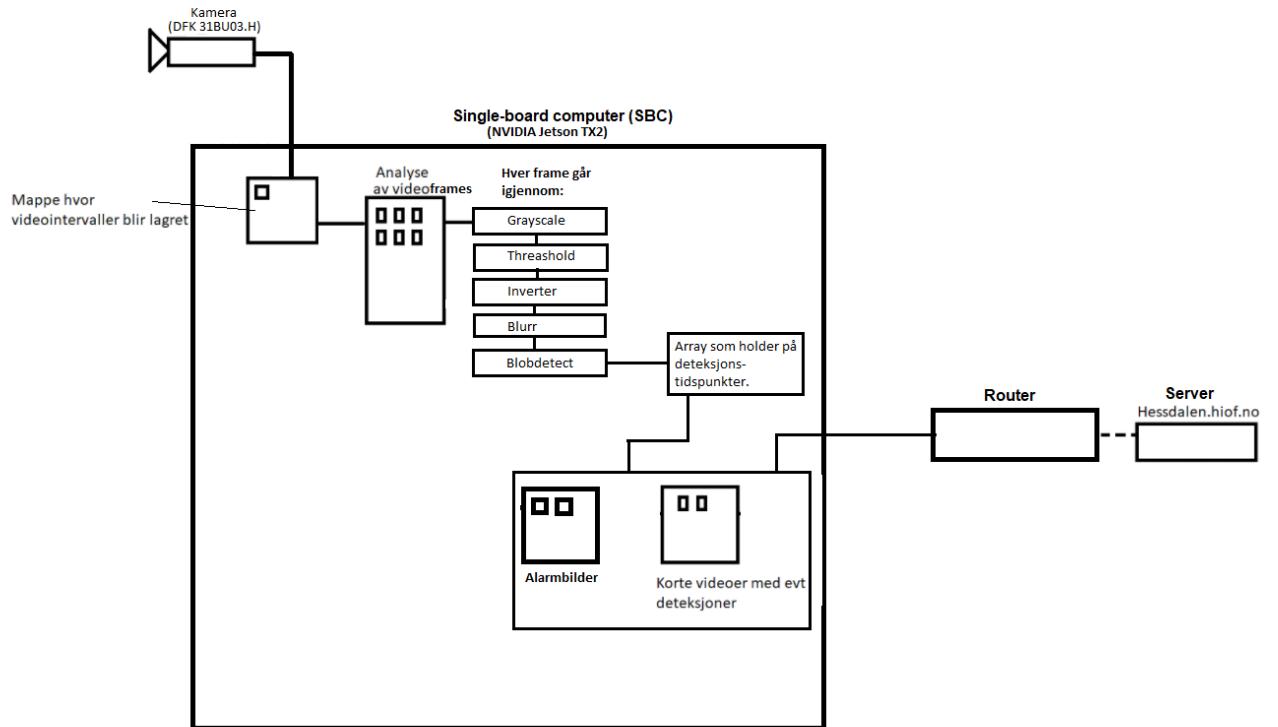
3.2.11 Lagring

Ut i fra lagringsalternativene nevnt i kapittel 2.15 kom gruppen frem til at lokal lagring vil være en tilstrekkelig løsning. Med Jetson TX2 sin lagringskapasitet på 32GB vil lokal lagring av video være mer enn godt nok til å ikke trenge en ekstern lagringseenhet. Hvorfor dette ikke blir et problem blir beskrevet i kapittel 4.4.2. Lagring til server er et lite optimalt valg da videoanalyse-prosessen ikke lengre ville skjedd på SBC-en.

3.2.12 Systemoversikt - revisjon 2

Endringer:

- Navngitt kamera, SBC og server
- Lagt til midlertidig algoritme i SBC



Figur 3.3: Systemoversikt revisjon 2

Kapittel 4

Implementasjon

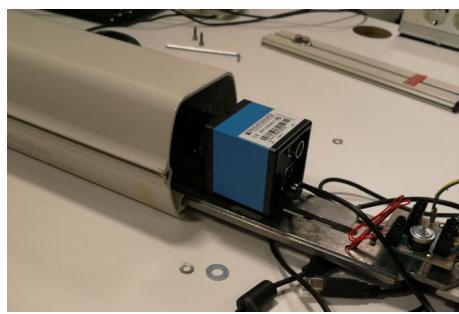
I dette kapittelet beskrives det hvordan gruppen gikk frem for å løse oppgaven i praksis, systemets oppbygging og virkemåte samt hvilke verktøy som ble brukt for å komme i mål med prosjektet.



Figur 4.1: Oppsett

4.1 Kamerahus

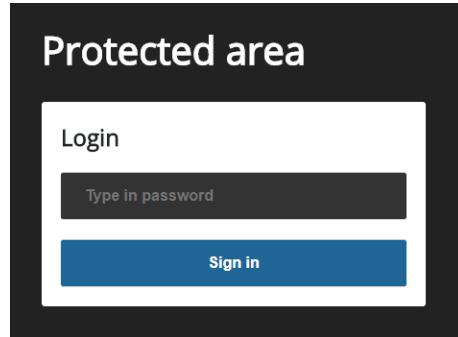
Oppdragsgiver overleverte tiltenkt kamerahus til gruppen den 16.04. Kameraet ble skrudd fast til en skinne som føres inn i kamerahuset etter ønske fra oppdragsgiver. Dette vil plasseres på en mast i Hessdalen.



Figur 4.2: Kamera montert på skinne som føres inn i kamerahus

4.2 Kontrollpanel

For at bruker av systemet enklest mulig skal kunne definere hvilket område som ønskes å maskeres bort er det utviklet en nettside. Denne nettsiden lar bruker både tegne maskeringsområde og endre relevante innstillingar som benyttes i bildeanalyesen og hovedprogrammet. Nettsiden er følgelig i praksis som et kontrollpanel for systemet.



Figur 4.3: Innlogging til kontrollpanel

Når nettsiden lastes inn er bruker nødt til å oppgi passordet oppdragsgiver har definert i config.php, som er en fil bestående av nettsidens innstillingar. Filen inneholder passord for å besøke nettsiden, IP adressen til SBC-en og dens brukernavn og passord. Informasjonen om SBC-en er nødvendig for å kunne kommunisere med den via nettsiden.

```
<?php
return [
    "jetson" => [
        "ip" => "158.39.188.196",
        "username" => "nvidia",
        "password" => "nvidia"
    ],
    "website" => [
        "password" => "Hess123"
    ]
];
?>
```

Figur 4.4: Konfigurasjonsfil for kontrollpanel

Siden innstillingsfilen inneholder sensitiv informasjon er den sikret og kun tilgjengelig for eier av systemet. Filen er sikret ved hjelp av en .htaccess fil som er satt til å avvise alle forsøk på å besøke filen.

```
1 <Files "config.php">
2 Deny from all
3 </Files>
```

Figur 4.5: Fjerne tilgang til for uvedkommende fil på kontrollpanel

Etter innlogging blir en tatt til siden vist på figur 4.6. Her har bruker mulighet til å tegne og viske ønsket maskeringsområde oppå et bakgrunnsbilde. Størrelsen på tegne- og viskeverktøyet

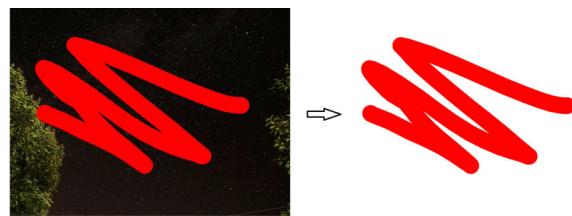
kan enkelt endres, og om ønskelig kan det hentes et nytt bakgrunnsbilde ved å trykke på “Get new background picture”. Det blir da automatisk hentet et bakgrunnsbilde fra kamera tilkoblet SBC-en, som så blir presentert for brukeren på nettsiden. Hvis bruker ønsker å lage et helt nytt maskeringsområde kan det trykkes på “Clear”, som fjerner alt som er påtegnet.

For å oppnå muligheten å tegne på en nettside er det benyttet JavaScript sammen med et canvasområde. Et canvasområde i HTML benyttes for å la utviklere lage sirkler, kvadrater, figurer osv. ved hjelp av kode. Det var dermed nødvendig å lage en løsning som tillot tegning ved hjelp av musen. Dette er løst ved å følge med på når musen beveges, klikkes og slippes. Ved et enkelt klikk på musen lages det en prikk der det ble trykket. Når musen klikkes ned og deretter beveges vil det fortløpende lages mange prikker, som resulterer i en “glatt” strek.



Figur 4.6: Hvordan kontrollpanel ser ut

Masken tegnes oppå et bakgrunnsbilde, men lagres med gjennomsiktig bakgrunn. Dette er oppnådd ved at masken tegnes i canvasområdet, mens bakgrunnsbilde er plassert bak canvasområdet ved hjelp av CSS. Dette er nødvendig siden masken skal legges oppå hver frame kamera leser inn. Man kan da ikke ha med bakgrunnsbilde som bakgrunn, siden dette ville overskrevet lest bilde fra kamera.

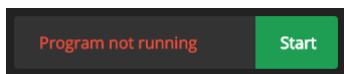


Figur 4.7: Illustrasjon av at maske lagres med gjennomsiktig bakgrunn

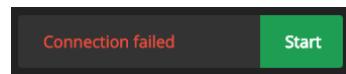
Når det trykkes på “Save” lagres påtegnet maske og innstillinger både lokalt på nettsiden og til SBC-en. Dette er en rask prosess som er gjort på under ett sekund. Det er nødvendig å lagre lokalt på nettsiden slik at informasjonen kan vises når nettsiden lastes inn senere. Bruker blir deretter

presentert med en melding om lagringen var vellykket eller ikke, noe som avhenger av om SBC-en er avskrudd eller ikke tilgjengelig. Om det skulle oppstå en feilmelding trengs det kun trykkes “Save” ved en senere anledning, når SBC-en er tilgjengelig, siden alt ligger lagret lokalt. På denne måten slipper bruker av systemet å måtte tegne masken om igjen dersom oppkoppling til SBC-en skulle feile.

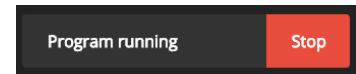
Brukere har mulighet til å starte og stoppe hovedprogrammet som kjører på SBC-en via nettsiden. Bruker vil også se en statusmelding til venstre for start/stopp knappen som forteller hvorvidt program kjører eller ikke, eller om tilkoppling mislyktes. Ved trykk på knappen “Stop” må bruker bekrefte en gang til at det faktisk ønskes å stoppe programmet, slik at man ikke kommer borti ved et uhell. Dette skjer i form av en pop-up figur 4.11.



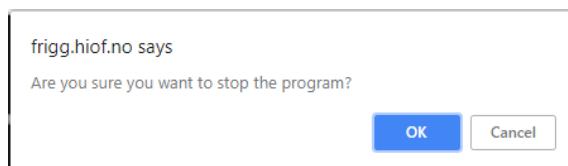
Figur 4.8: Program på SBC kjører ikke



Figur 4.9: Tilkobling til SBC mislyktes



Figur 4.10: Program på SBC kjører



Figur 4.11: Pop-up

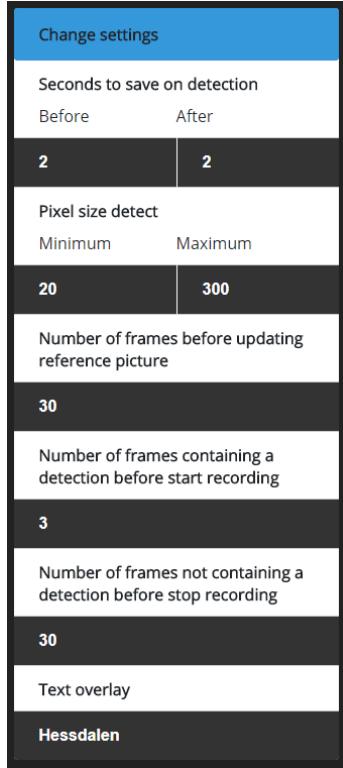
Ved å trykke på “Change settings” i figur 4.6 åpnes menyen som vist på figur 4.12. Her kan bruker bestemme følgende innstillingar for hovedprogrammet:

- Antall sekunder før og etter en deteksjon som skal inkluderes i trimmen
- Minimums- og maksimumsstørrelse på objekter som skal detekteres
- Antall frames før referansebilde skal oppdateres
- Antall frames som må inneholde en deteksjon før det startes å filme og tas et alarmbilde
- Antall frames som ikke skal inneholde en deteksjon før det sluttet å filme
- Tekst som skal ligge oppå alle bilder og video som lagres ved deteksjon

Gruppen har valgt å benytte AJAX slik at nettsiden aldri trengs å lastes inn på nytt når en oppgave skal utføres. Dette skjer når:

- Maskeringsområdet og innstillingar skal lagres
- Referansebilde skal hentes
- Det trykkes start/stopp for å kontrollere programmet

Dette fungerer ved at informasjon som ligger i JavaScript-variablene sendes til ulike PHP filer ved hjelp av AJAX. Det er laget en PHP fil for hver av oppgavene nevnt ovenfor.



Figur 4.12: Innstillinger som kan endres på kontrollpanelet

For å kunne kommunisere med SBC-en benyttes biblioteket phpseclib¹, som tilgjengeliggjør bruk av SFTP og SSH tilkobling til en ekstern enhet i PHP kode. Når maskeringsområde og innstillinger lagres, og når bakgrunnsbilde skal hentes er det brukt SFTP for å laste opp/ned filene. Når programmet skal startes eller stoppes er det brukt SSH for å sende kommandoer til SBC-en. Sistnevnte oppgave er gjort slik at det åpnes et terminal vindu hos SBC-en i stedet for at kommandoene utføres direkte i nettleseren. Å utføre det direkte i nettleseren ville ført til at programmet stoppet så fort nettleseren ble lukket.

4.3 Bildeanalyse

Bildeanalysen som er implementert i programmet benytter prinsippet å se etter bevegelse. Kamera tar 30 bilder i sekundet, som kalles frames. Programmet har en justerbar variabel, men per standard vil hver 30ende frame lagres som et referansebilde. De neste 29 framene blir så sammenlignet med referansebildet for å se etter endringer som kan ha oppstått i bildeområdet.

Analysen ligger i en funksjon som returnerer en tallverdi i form av en integer. Returnerer funksjonen “1” har man et utslag på at noe er detektert, returnerer funksjonen “0” er det ingen deteksjon på analysert frame. Det er ved bruk av OpenCV sitt bibliotek gruppen har laget bildeanalysen.

I programkoden finner man bildeanalysen i funksjonen *analyse(frame, frameNumber)*. Som parametere tar funksjonen inn en frame samt en tallverdi som forteller hvor mange frames som er lest så langt.

¹<http://phpseclib.sourceforge.net>

4.3.1 Analysen

Programmet har tatt et referansebilde for 30 frames siden, altså et sekund tilbake i tid. Avbildet er et av gruppens medlemmer for å demonstrere hvordan analysen går frem.



Figur 4.13: Bildeanalyse - referansebilde

1. Ny frame leses fra kamera og lagres i en variabel, slik at man unngår å lagre alle bildene på disk og opppta lagringsplass. Koden nedenfor leser frames fra en eksisterende videofil som er lagret av kamera.

Kode 4.1: Leser en og en frame fra video

```
capture = cv2.VideoCapture("videonavn.mp4")
while(capture.isOpened()):
    ret, frame = capture.read()
```

Gruppens medlem har rukket å bevege seg 1-2m lenger bort på gangveien.



Figur 4.14: Bildeanalyse - ny frame som skal sammenlignes med referansebilde

2. Lest frame blir gjort om til gråskala. Parameterne sier hvilket bilde som skal endres farge på og at det skal gjøres til gråskala.

Kode 4.2: Gjør lest frame til gråskala

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```



Figur 4.15: Bildeanalyse - ny frame gjort om til gråskala

3. Lest frame gjøres uklar ved hjelp av metoden GaussianBlur, slik at man unngår “støy” i bildet. Dette betyr i praksis at man fjerner veldig små elementer som man ikke ønsker skal detekteres som en forandring i bildet. Parameterne sier hvilket bilde som skal blurrer, størrelse på blurren og avvik i plassering.

Kode 4.3: Blur lest frame for å fjerne støy

```
gray = cv2.GaussianBlur(gray, (3, 3), 0)
```



Figur 4.16: Bildeanalyse - gråskala frame blurret

4. Hver frame som leses er representert som en array i programkode. Absdiff-funksjonen regner ut den absolutte forskjellen mellom to arrayer, altså frames. Resultatet er dermed en ny array som inneholder de stedene det er forskjell, herav kalt “forskjellsbildet”.

Kode 4.4: Regner ut forskjellen mellom referansebilde og lest frame

```
frameDelta = cv2.absdiff(previousFrame, gray)
```

Det blåste mye da bildene ble tatt, noe som førte til at tretoppene beveget seg. Dette resulterte i at absdiff regnet en forskjell på plasseringen til tretoppene, avbildet i figur 4.17.



Figur 4.17: Bildeanalyse - utregnet forskjell mellom ny frame og referansebilde

5. Forskjellsbildet blir thresholdet, som betyr at det gjøres svart-hvitt. Dette blir gjort ved at hver piksel i bildet sjekkes for fargene rød, grønn og blå. En piksel kan ha en fargeverdi mellom 0 og 255, hvor 0 er svart og 255 hvit. Gjennomsnittet av de tre fargene regnes ut for å bestemme om pikselen skal bli hvit eller svart. Hvis utregnet gjennomsnitt er over 25 blir pikselen hvit, og er den under 25 blir den svart. I gruppens system fører dette til at veldig mørke piksler, nærmest svarte, ikke blir tatt hensyn til. Dette for å unngå deteksjon på eksempelvis en mørk himmel. Følgende formel blir brukt: $(Rød + Grønn + Blå) / 3$.

Parameterne sier hvilket bilde som skal behandles, minimumsgrense for hva som skal bli hvitt, maksimumsgrense for hva som skal bli hvitt, og at det skal benyttes vanlig thresholding. Alternativt kan det benyttes “invertert svart-hvitt”.

Kode 4.5: Forskjellsbilde thresholdes for å gjøre det svart-hvitt

```
ret, thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)
```



Figur 4.18: Bildeanalyse - thresholdet forskjellsbilde

6. Masken gjøres om til gråskala og thresholdes ved oppstart av program. Masken vil aldri endre seg så lenge programmet kjører, så det er unødvendig bruk av ressurser å gjøre disse operasjonene hver gang bildeanalyesen kjøres. Masken gjøres svart-hvitt ved bruk av invertert thresholding, som betyr at det blir seende ut motsatt av bilde i steg 5. Det røde i masken blir svart, mens resten hvitt.

Kode 4.6: Maske thresholdes for å gjøre den svart-hvitt

```
thresholdMaskTemp = cv2.threshold(grayMask, 0, 255, cv2.THRESH_BINARY_INV)
```

Siden det blåste mye denne dagen var det ikke ønskelig å inkludere tretoppene. Det er derfor tegnet en maske som ekskluderer dette i eksempelet. Trær er ikke et problem når analysen skal benyttes i Hessdalen, siden trærne der vil være flere hundre meter unna.

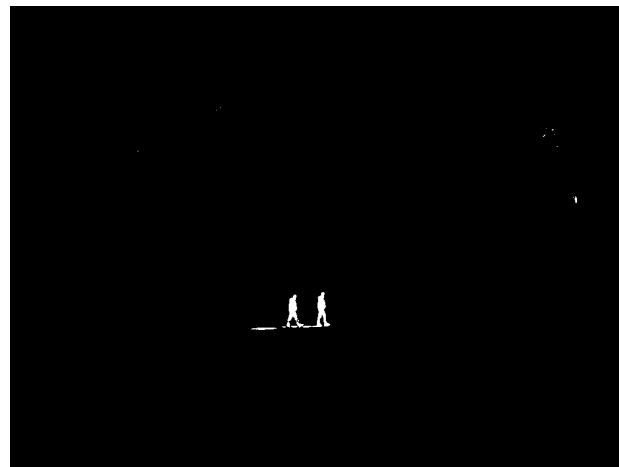


Figur 4.19: Bildeanalyse - tegnet maske, etter thresholding, som skal ganges inn

Masken ganges så bitvis inn i forskjellsbildet, som betyr at hver piksel på samme posisjon i bildene ganges med hverandre. En svart piksel har verdien 0, og en hvit verdien 1. Siden bitwise_and er multiplikasjon vil piksler som er sorte i ett av bildene, resultere i en svart piksel i utregnet bilde. På denne måten vil alle steder det er tegnet en maske gjøres til svart, og vil dermed ikke føre til et deteksjonutslag.

Kode 4.7: Maske ganges inn i thresholdet forskjellsbilde for å ikke inkludere uønskede områder

```
thresh = cv2.bitwise_and(thresh, thresholdMask.value)
```



Figur 4.20: Bildeanalyse - thresholdet bilde etter at maske er ganget inn

7. Alle hvite områder på utregnet bilde fra steg 6) blir økt i størrelse. Dette gjøres ofte etter at bilder er blurret, siden blur fjerner støy og noen ganger gjør objekter mindre. Ved å benytte funksjonen dilate blir da eventuelle hull i de hvite konturene fyldt med hvitt. Parameterne sier hvilket bilde som skal behandles og hvor mye det skal forstørres.

Kode 4.8: Hvite områder økes i størrelse

```
thresh = cv2.dilate(thresh, None, iterations=2)
```



Figur 4.21: Bildeanalyse - thresholdet bilde etter at maske er ganget inn og hvite objekter har økt i størrelse

8. Det kalles så på en innebygd funksjon i OpenCV som finner alle konturer i bildet fra steg 7) og lagrer dem i en array. Parametrene til funksjonen nedenfor sier hvilket bilde som skal behandles, at det kun skal ses på overordnede sirkler og ikke alle innenfor, og komprimere konturene.

Kode 4.9: Finner og legger alle konturer på behandlet bilde i en array

```
- , contours , _ = cv2 . findContours ( thresh . copy () , cv2 . RETR_EXTERNAL ,
cv2 . CHAIN_APPROX_SIMPLE )
```

Det går så igjennom arrayen med en for-løkke. Hvis konturen eksempelvis er større enn 50 og mindre enn 500 blir den lagt i en ny array, contourList. Dette gjøres for å luke ut små og store elementer som det ikke ønskes å inkludere.

Kode 4.10: Går igjennom arrayen med konturer og hvis konturen er både større og mindre enn satte verdier legges de i en annen array

```
for contour in contours :
    if cv2 . contourArea ( contour ) > 50 and cv2 . contourArea ( contour ) < 500 :
        contourList . append ( contour )
```

Hvis arrayen contourList er tom ved slutten av funksjonen, returneres det en 0. Hvis det finnes elementer i arrayen betyr det at det har skjedd en deteksjon, og det returneres 1.

Hvis det returneres 1 blir arrayen contourList gått igjennom i en annen funksjon for å tegne røde sirkler for å markere hvor detektert fenomen er på det originale bilde.

Kode 4.11: Regner ut størrelse på sirkler og tegner på originalbilde der konturer er funnet

```
(x , y) , radius = cv2 . minEnclosingCircle ( contour )
cv2 . circle ( frame , ( int ( x ) , int ( y )) , int ( radius ) , ( 0 , 0 , 255 ) , 2 )
```

Til slutt blir det lagt til tekst nederst til venstre på bildet som inneholder tekst spesifisert i innstillingene via kontrollpanelet, dato, tidssone og klokkeslett. Dette gjøres i etterkant for å unngå at bildeanalyesen slår ut på forandring i klokkeslett.

I eksempelet reagerer analysen både på at gruppens medlem har beveget seg og at skyggen hans er borte fra bakken, derav de to sirklene.



Figur 4.22: Bildeanalyse - originalt bilde med røde sirkler som markerer hvor det har vært endring

Ved oppstart av programmet har man ikke et referansebilde. Derfor blir første framen som leses fra hver video gjort om til gråskala, blurret og lagret som referansebilde. Ingen videre analyse av framnen er nødvendig, og funksjonen returnerer følgelig 0. Referansebilde blir deretter oppdatert hver 30 frame, altså hvert sekund.

4.3.2 Oppsummering

Tabell 4.1: Oppsummering av bildeanalyse

Steg	Funksjon	Beskrivelse
1	cv2.cvtColor	Gjør bildet til gråskala
2	cv2.GaussianBlur	Fjerner støy fra bildet i form av små elementer
3	cv2.absdiff	Regner ut om det er noe forskjell mellom referansebildet og tatt bilde
4	cv2.threshold	Gjør bildet til svart-hvitt. Regner gjennomsnittet av fargene rød, grønn og blå. Hvis gjennomsnittet er over en bestemt verdi blir pikselen hvit, og hvis ikke svart
5	cv2.bitwise_and	Ganger bitvis maske med thresholdet bildet. Tilsvarende piksel i bildene multipliseres med hverandre, hvor man kun har verdiene 0 og 1
6	cv2.dilate	Gjør de hvite områdene på bildet større
7	cv2.findContours	Finner alle konturer i bildet og legger de i en array
8	cv2.contourArea	Sjekker størrelsen til en enkelt kontur i findContours arrayen
9	cv2.circle	Tegner en sirkel rundt en enkelt kontur i findContours

4.4 Hovedprogram

4.4.1 Gstreamer

Gstreamer er et API som benyttes for å håndtere kameraet. API-et fungerer slik at man kaller på en pipeline, noe som er en oppstartslinje med ønskede kommandoer. Man må i pipelinen definere kilden, altså om det skal være en direkte visning av kamera eller en videofil, oppløsning, antall bilder per sekund, hvor output skal vises og mer. Den valgte codecen, fargefilteret og filformatet som ble bestemt i kapittel 3.2 blir også satt i pipelinen.

Gruppen har ved hjelp av Gstreamer klart å optimalisere kameraet til å håndtere 30 bilder i sekundet i farger. Standardformatet som kameraet benytter gir en lavere framerate og en feil representasjon av farger, hvor eksempelvis hudfarge blir vist som grønn.

4.4.2 Programforklaring

Ved oppstart av programmet leses innstillingene som er satt i kontrollpanelet fra config-filen som ligger lagret lokalt på SBC-en. Videre leses masken bruker har tegnet på nettsiden, som deretter gjøres til gråskala og thresholdes slik at den er klar til bruk. Deretter blir alle filnavn på videoer som finnes i "Videos"-mappen lagt inn i en kø. En kø er en datastruktur hvor man kun har tilgang til elementet som ble lagt inn først. Køen benytter altså et først inn, først ut prinsipp. Dette gjøres for å klargjøre eksisterende videofiler fra tidligere filminger for behandling. Eksisterende videofiler vil finnes fra f.eks. når programmet holder på å behandle en video, men programmet blir stanset.

Ved å implementere denne logikken vil videofilen bli behandlet neste gang programmet startes i stedet.

Det blir så kalt på *record*-funksjonen som filmer i intervaller. Grunnen til at det filmes i intervaller i stedet for å håndtere bildene direkte er for å få til sekundene før og etter en deteksjon. Funksjonen starter en pipeline med ønskede innstillinger for kameraet, hvor det blant annet spesifiseres hvor videofilen skal lagres og navnet på filen. Filnavnet settes til tidspunktet det ble startet å filme ved hjelp av Pythons innebygde `datetime.now()`. Når funksjonen er ferdig med å filme legges filnavnet i den tidligere nevnte køen.

I *read*-funksjonen sjekkes det kontinuerlig om køen er tom. Hvis køen ikke er tom henter den det videoavnet som først ble lagt inn og legger det i en variabel. Programmet vil så åpne videofilen med samme navn som variabelen og lese den frame for frame. Hver frame blir sendt gjennom bildeanalyesen forklart i kapittel 4.3. Hvis bildeanalyesen returnerer 1 går det inn i "Tilfelle deteksjon", og hvis den returnerer 0 går det inn i "Tilfelle ikke deteksjon". Programmet har også en variabel som sier hvorvidt man er i deteksjonsfase, med verdien true eller false.

Oppdragsgiver kan definere verdier for følgende innstillingar på forhånd via nettsiden:

- Antall frames før referansebilde skal oppdateres, herav kalt *setting[4]*
- Variabler som lagrer hvilken frame i videoen programmet står på og som blir benyttet for trimming av videoen i form av hvilke frames det skal trimmes fra og til
 - starttid
 - stopptid
- Antall sammenhengende frames som må inneholde en deteksjon før det startes å filme og tas et alarmbilde, herav kalt *setting[5]*
- Antall sammenhengende frames som **ikke** inneholder en deteksjon før det sluttet å filme, herav kalt *setting[6]*
- Tekst som skal bli påtrykt alle bilder og video som lagres ved deteksjon, herav kalt tekststempel
- Variabel som teller antall frames på rad som returnerer 1, herav kalt *detectStartCount*
- Variabel som teller antall frames på rad som returnerer 0, herav kalt *detectEndCount*

I listen på neste side forklares ett if, if-else oppsett. Dette betyr at hvis punkt 1) skjer, så kan ikke punkt 2) skje samtidig.

Tilfelle deteksjon

Kode 4.12: Sjekker om bildeanalyesen returnerer 1, altså at man har en deteksjon

```
if analyse(frame, frameNumber) == 1:
```

Hver gang “tilfelle deteksjon” inntreffer øker detectStartCount med en.

1. Hvis man ikke er i deteksjonsfase, altså at variabelen er false, beholdes lest frame i en variabel og starttiden blir satt til den framden programmet står på. Variabelen deteksjonsfase settes så til true.

Kode 4.13: Hvis bildeanalyesen returnerer 1 og man ikke er i deteksjonsfase

```
if detectionPhase == False:
    startFrame = frame
    startTime = frameNumber
    detectionPhase = True
```

2. Hvis man er i deteksjonsfase og bildeanalyesen har returnert *setting[5]* antall 1’ere på rad så skal:

- Framen som ble beholdt i 1) lagres i “Pictures”-mappen som to bilder, ett originalt og ett som marker hvor på bildet det ble oppdaget forskjeller mellom referansebildet og framden programmet nå er på.
- Tekststempel, klokkeslett og dato, bli påtrykt nederst i venstre hjørne av bilde. Dette gjøres to ganger for å få en svart ramme rundt teksten. Den første gangen skrives teksten i svart, og den neste skrives teksten i hvit og noe mindre størrelse.

Siden bildeanalyesen baserer seg på motion detection legges teksten på i etterkant i stedet for mens det filmes. Dette fordi bildeanalyesen ellers ville reagert på forandringer i klokkeslettet for hver frame som håndteres.

Kode 4.14: Hvis bildeanalyesen returnerer 1, man er i deteksjonfase og bildeanalyesen har returnert et brukerdefinert antall 1’ere på rad

```
elif detectionPhase == True and detectStartCount == int(settings[5]):
    frameTime = datetime_object + timedelta(seconds=frameNumber / 30)

    cv2.putText(startFrame, settings[7] + " " + frameTime.strftime("%d/%m
        /%Y %H:%M:%S"), (2,762), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0,0,0),
        2)
    cv2.putText(startFrame, settings[7] + " " + frameTime.strftime("%d/%m
        /%Y %H:%M:%S"), (2,762), cv2.FONT_HERSHEY_SIMPLEX, 0.4,
        (255,255,255))
    cv2.imwrite(filepath + "Detections/Pictures/" + str(frameTime.
        strftime("%d-%m-%Y_%H-%M-%S")) + ".jpg", startFrame, [int(cv2.
        IMWRITE_JPEG_QUALITY), 100])

    for contour in contourList:
        (x, y), radius = cv2.minEnclosingCircle(contour)
        cv2.circle(startFrame, (int(x), int(y)), int(radius), (0, 0,
        255), 2)
    cv2.imwrite(filepath + "Detections/Pictures/" + str(frameTime.
        strftime("%d-%m-%Y_%H-%M-%S")) + "_marked.jpg", startFrame, [int(
        cv2.IMWRITE_JPEG_QUALITY), 100])
```

3. Hvis man er i deteksjonsfase, på siste frame i videoen og bildeanalyesen har returnert flere eller likt antall 1'ere på rad enn det bruker har bestemt i *setting[5]*, så settes stopptiden til framnen som programmet nå behandler. Deretter lagres et utklipp av videoen med starttiden fra 1) og stopptiden ved hjelp av funksjonen trim. Dette tilfelle er nødvendig for å dokumentere hendelser som skjer på slutten av videoen. Uten denne logikken vil trimmen bli korrupt og ikke mulig å avspilles

Kode 4.15: Hvis bildeanalyesen returnerer 1, man er på siste frame i videoen og bildeanalyesen har returnert flere 1'ere på rad enn brukerdefinert antall

```
elif detectionPhase == True and frameNumber == (filmDuration * 30) - 2 and
    detectStartCount >= int(settings[5]):
    stopTime = frameNumber
    trim(startTime, stopTime)
    detectionPhase = False
```

4. Hvis ingen av de andre testene over har slått ut og man er i deteksjonsfase så settes detectEndCount variabelen til 0

Kode 4.16: Hvis bildeanalyesen returnerer 1 og man er i deteksjonsfase

```
elif detectionPhase == True:
    detectEndCount = 0
```

Tilfelle ikke deteksjon

1. Samme logikk som punkt 3) i “tilfelle deteksjon”. Nødvendig tilfelle for å håndtere videoen korrekt om man står på siste frame i videoen.
2. Hvis man er i deteksjonsfase..

- og detectStartCount er mindre eller lik *setting[5]* resettes detectStartCount og deteksjonsfasen settes til false.

Kode 4.17: Hvis bildeanalyesen returnerer 0, man er i deteksjonsfase og detectStartCount er mindre enn brukerdefinert antall

```
if detectStartCount <= int(settings[5]):
    detectStartCount = 0
    detectionPhase = False
```

- og detectEndCount er lik *setting[6]* kaller funksjonen på trim som lager et utklipp(trim) av videoen og deteksjonsfasen settes til false. detectStartCount og detectEndCount nullstilles.

Kode 4.18: Hvis bildeanalyesen returnerer 0, man er i deteksjonsfase og man har telt et brukerdefinert antall 0 på rad

```
if detectEndCount == int(settings[6]):
    stopTime = frameNumber - int(settings[6])
    trim(startTime, stopTime)
    detectionPhase = False
    detectStartCount = 0
    detectEndCount = 0
```

Funksjonen *trim* går gjennom videoen som ble hentet ut av køen, med start og sluttid som parametre. Funksjonen trim blir kalt på for hver deteksjon innenfor en video. Det blir her lagt til antall sekunder før og etter en deteksjon basert på hva bruker har satt som innstillinger i kontrollpanelet. Resultatet av funksjonen trim er altså at hvert fenomen, fra originalvideoen, blir lagret som en egen, kort video. Filnavnet på trimmen blir satt til tidspunktet fenomenet oppsto.

Etter at *read*-funksjonen har lest gjennom hele videoen kalles det på *upload*-funksjonen. Her vil alle alarmbilder og trims av fenomen lastes opp til Hessdalen sin server via SCP. Hvis opplasting var vellykket, blir bildene og utklippene deretter slettet. Dette betyr at hvis den ikke klarer laste opp bildene og utklippene, vil de heller ikke slettes. På denne måten vil aldri dokumentasjon av fenomener gå tapt selv om SBC-en mangler internetttilgang.

På Project Hessdalens server skal alle alarmbilder og trims, som skjer samme dag, ligge samlet i en mappe med datoen som mappenavn. Programmet lagrer alarmbilder og trims med dato og klokkeslett som filnavn og kan på denne måten finne mappen med tilsvarende dato. Mappen blir kun opprettet hvis det har skjedd en deteksjon den dagen og mappen ikke eksisterer.

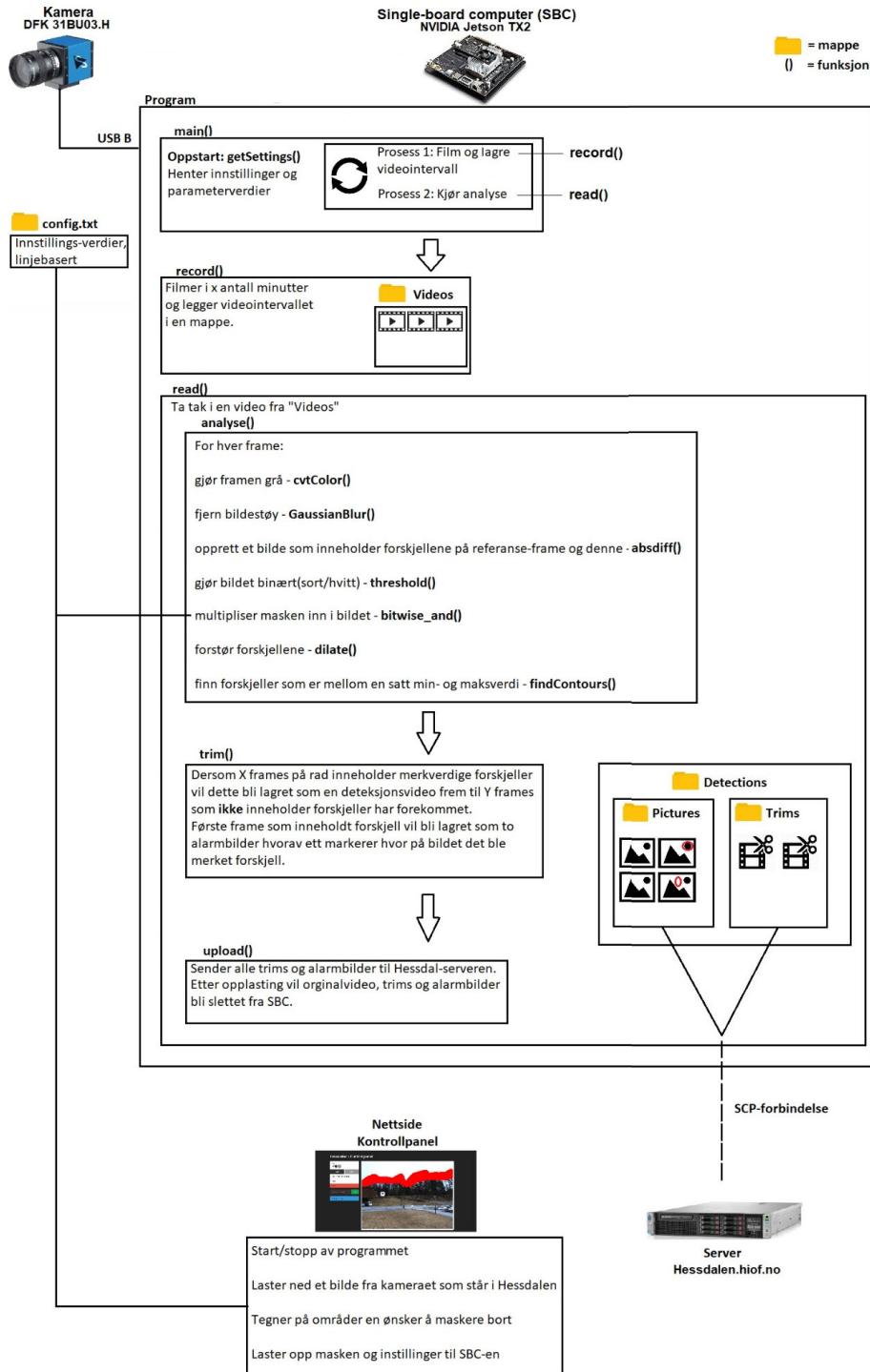
Programmet er laget for å ikke skulle krasje ved feilmeldinger. Er det en videofil som av en uforutsett grunn skulle bli korrupt og dermed ikke mulig å behandle vil programmet slette videofilen og gå videre. Mangler SBC-en nettilgang, og den ikke klarer å laste opp fenomener til Hessdalen sin server, vil programmet kjøre videre og forsøke å laste opp senere i stedet.

Det benyttes multiprocessing for å la ulike deler av programmet kjøre samtidig. I Python-programmet er det laget én funksjon for å filme og én for å behandle videoer. I gruppens tilfelle brukes multiprocessing til å la disse to funksjonene kjøre om hverandre som to uavhengige systemprosesser. Ved å separere disse to funksjonene og benytte en kø kan det filmes raskere enn programmet rekker å behandle videoene, og det er ikke et problem at størrelsen på køen øker.

Er det ingen deteksjoner vil bildeanalysen gjennomgå bildene raskere. Er det mange deteksjoner hvor fenomener eksisterer i lengre perioder vil det kunne oppstå kø.

4.5 Systemoversikt - revisjon 3

Følgende oversikt er en forenklet versjon og har som hensikt å gi et forklarende, visuelt bilde av systemet.



Figur 4.23: Illustrasjon som viser systemoversikt

Kapittel 5

Testing

I dette kapittelet vises det til hvordan testing av systemet ble utført og dets resultater. Hvor testingen har blitt utført og hvilke redskaper som ble brukt kommer også frem.

5.1 Hensikt

Hensikten med å utføre grundig testing av systemet var for å sikre et best mulig sluttprodukt for oppdragsgiver ved overlevering. Når produktet er overlevert vil ikke gruppen være tilgjengelig for å utbedre systemet dersom det skulle vise seg å være uforutsette feil. Systemet er ikke testet i omgivelsene det skal stå i, så følgelig må systemet være tilstrekkelig testet slik at det har gode forutsetninger for å fungere optimalt når det settes i bruk i Hessdalen.

Det ble testet i stor grad mer enn det som forekommer i dette kapittelet, men det ville vært overflødig å ha med alt i dette dokumentet. Det ville heller ikke ha bidratt med mye nytt.

5.2 Testlokasjoner

Alle tester ble utført på området til *Høgskolen i Østfold, Halden*.

Rom DU1-043, “Design-lab”

Rommet i seg selv er avlangt så det var mulig å utføre tester på varierende avstander. Taklyset kunne både dimmes og skrus av.



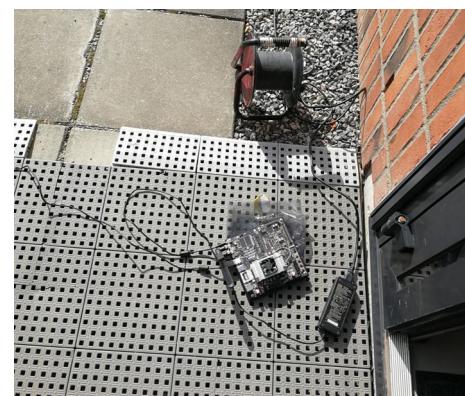
Figur 5.1: Rom DU1-043

Utenfor IT-avdelingen

Strøm var tilgjengelig ved hjelp av en skjøtetrommel, men det trådløse nettet var ustabilt.



Figur 5.2: Område utenfor IT-avdelingen



Figur 5.3: Skjøtetrommel og Jetson

Åpent landskap ovenfor hovedinngangen

Utsikt over fotballbane, nordlig parkeringsplass, bakre åsside samt himmel.



Figur 5.4: Utsikt fra lokasjon



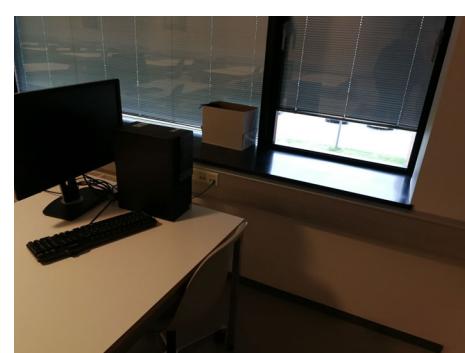
Figur 5.5: Lokasjon ovenfor hovedinngang

Rom D1-052

Utsikt over sørlig parkeringsplass og åpent område.



Figur 5.6: Oppsett på D1-052



Figur 5.7: Detekterte ut vinduet

5.3 Utstyr

Kamera: DFK 31BU03.H med USB-B kabel



Figur 5.8: DFK 31BU03.H

Single-board computer: NVIDIA Jetson TX2 med powersupply, antenner og ethernetkabel



Figur 5.9: Jetson TX2 med strømtilførsel

USB-hub med mus og tastatur



Figur 5.10: USB-hub med mus og tastatur

Skjerm: HP ZR2440W med HDMI-inngang



Figur 5.11: Skjerm

Lyskilder

Lommelykt - LED LENSER PS



Figur 5.12: Lommelykt

Hodelykt - Northpole Power LED 10W



Figur 5.13: Hodelykt

Laser



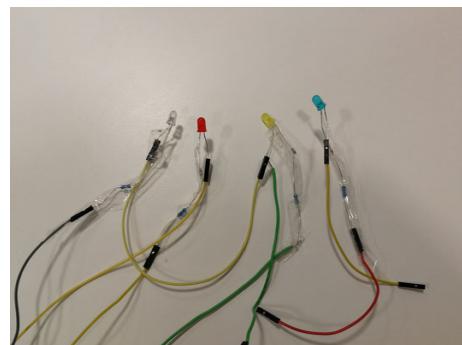
Figur 5.14: Laser

Mobil - Huawei P10



Figur 5.15: Mobil

LED - loddet sammen med motstander på 180ohm



Figur 5.16: LED

Blåfarget papirfilter



Figur 5.17: Papirfilter

For å forhindre at lysene forekom som gigantiske lyskuler på videobildene var det nødvendig å teipe et papirfilter foran lyskildene. Dette gjorde det lettere å justere størrelsen på dem samt eliminere lysskjær som eksempelvis vises på figur 5.18.



Figur 5.18: Eksempel på lysskjær

5.4 Testing i mørket

Det var forventet at disse testene ville gi best resultater og færrest feildeteksjoner grunnet mindre elementer i bildeområdet som trær i vind, forbipasserende fugler o.l.

De fleste testene ble utført på en slik måte at det ikke skjedde en deteksjon før noen sekunder inn i filmingen. Dette ble gjort for å være sikker på at systemet, i oppstart, ikke gav falske resultater ved at det detekterte objekter før lyset var påskrudd. Eksempelvis der det ble deteksjon fra første frame på grunn av dårlige parameterinnstillinger. En annen grunn var for å teste kodelogikken nevnt i kapittel 3.2.8, som innebar å ta med øyeblinket rett før og etter en deteksjon, som ikke ville vært mulig å få testet dersom fenomenet oppstod fra starten av.

I test 1-3 var målet å se på hvordan systemet fungerte med et lysobjekt på forskjellige avstander. Varierende avstander av lyset ville føre til forskjellige pikselstørrelser som systemet burde klare å detektere. Test 1-22 ble utført på DU1-043, "Design-lab".

Test 1 - Deteksjon på nært hold

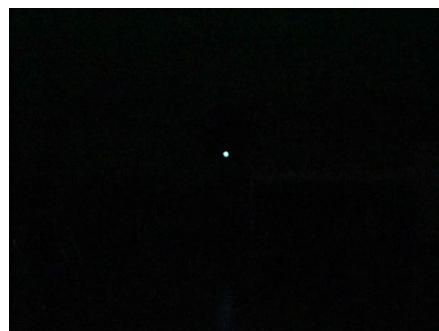
Utførelse: Lommelykt ble skrudd på 30-60 cm fra kamera, ca. 2 sekunder etter påstartet filming.
Resultat: Deteksjon. Alarmbilde og deteksjonsvideo ble opprettet på korrekte tidspunkter.



Figur 5.19: Deteksjon på 30-60 cm

Test 2 - Deteksjon på middels hold

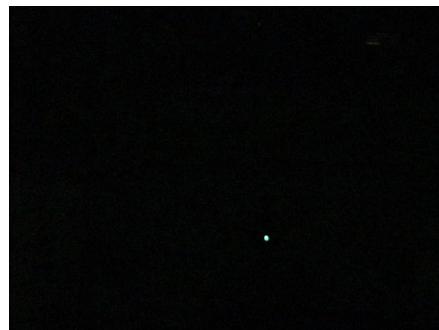
Utførelse: Lommelykt ble skrudd på 5-6 meter fra kamera, ca. 2 sekunder etter påstartet filming.
Resultat: Deteksjon. Alarmbilde og deteksjonsvideo ble opprettet på korrekte tidspunkter.



Figur 5.20: Deteksjon på 5-6 meter

Test 3 - Deteksjon på lengre hold

Utførelse: Lommelykt ble skrudd på 10-15 meter fra kamera, ca. 2 sekunder etter programmet startet.
Resultat: Deteksjon. Alarmbilde og deteksjonsvideo ble opprettet på korrekte tidspunkter.



Figur 5.21: Deteksjon på 10-15 meter

I test 4-8 ble det benyttet LED koblet opp mot TX2 sine GPIO-pinner. Avstand fra kamera var 1,5 meter. Utførelse: Benyttet GPIO pinne: 1(3.3V DC), 2(5.0V DC), 3(5.0V DC), 17(3.3V DC). Det ble benyttet 180-ohms motstander. Dersom alle LED fikk tildelt 5.0V var hvit og blå desidert de mest dominante(lysstyrke-messig) og ble i stedet tildelt 3.3V for å få alle LED på samme intensitetsnivå.

Hensikten med å teste deteksjon av forskjellige farger var for å se om programmet reagerte på ulike fargespekter. Som det forklares i kapittel 4.3.1 punkt 5, har piksler en fargeverdi mellom 0 og 255. Dersom pikslene på de ulike lysene var lavere enn en satt thresholdverdi ville de følgelig ikke tas med. Programmet har gitt deteksjonsbildene under et rødt omriss for å vise nøyaktig hva det reagerte på.

Test 4 - Deteksjon av farge: Hvit

Resultat: Deteksjon av hvitt lys.



Figur 5.22: Deteksjon av hvitt lys

Test 5 - Deteksjon av farge: Gul

Resultat: Deteksjon av gult lys.



Figur 5.23: Deteksjon av gult lys

Test 6 - Deteksjon av farge: Rød

Resultat: Deteksjon av rødt lys



Figur 5.24: Deteksjon av rødt lys

Test 7 - Deteksjon av farge: Blå

Resultat: Deteksjon av blått lys



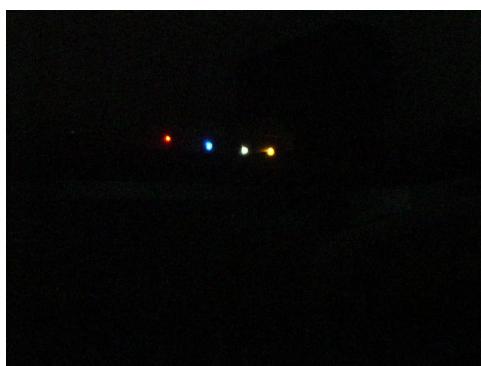
Figur 5.25: Deteksjon av blått lys

Test 8 - Deteksjon av 4 farger

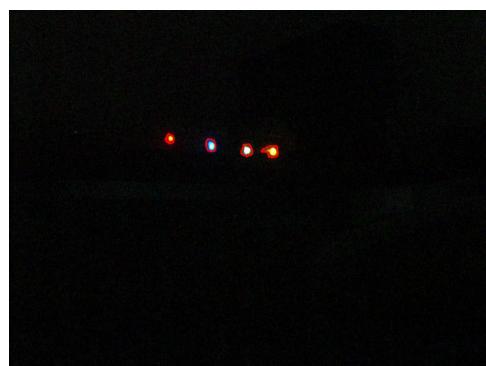
Figur 5.26: Oppsett med fire LED

Utførelse: Ut fra en pappeske ble det laget en testplattform hvor gruppen kunne plassere lysdiodene slik at de sto stødig og fikk avstand fra hverandre. Testen ble først utført med alle LED belyst fra starten av filmingen og deretter en test hvor en og en slo ble påslått. Farger som ble benyttet: rød, hvit, gul og blå.

Resultat: Ved første test hvor alle LED'ene var belyst ble det oppdaget fenomener umiddelbart. Alle farger ble detektert uten problemer. Programmet oppdaget også alle LED'ene som ble påslått på forskjellige tidspunkter i den andre testen.



Figur 5.27: Fire LED pålyst

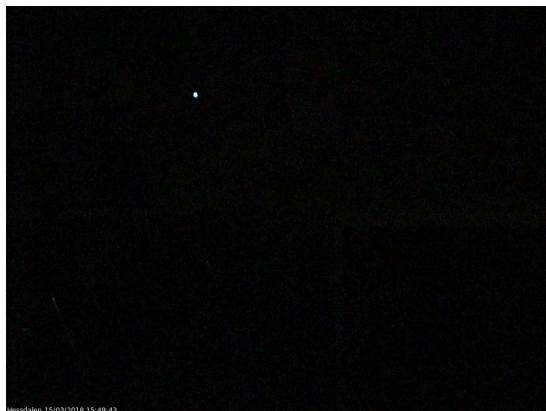


Figur 5.28: Alle farger detektert

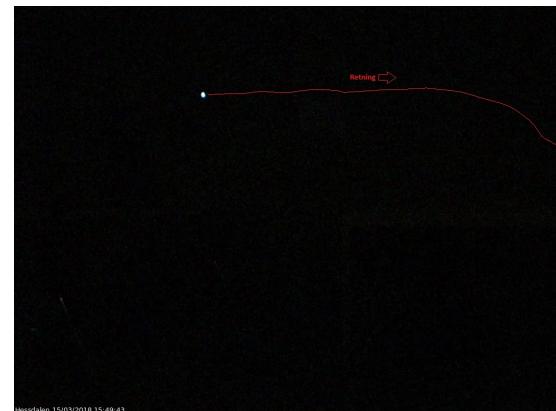
Test 9 - Objekt i bevegelse

Utførelse: Benyttet lommelykten til å simulere et lysobjekt i bevegelse, fra venstre til høyre. Objektet ble påskrudd ca. 2 sekunder inn i filmen hvor det beveget seg i et tregt tempo i starten og økte fortløpende til det forsvant ut av bildeområdet.

Resultat: Deteksjon. Programmet detekterte objektets ferd problemfritt uavhengig av dens hastighet. Deteksjonsvideo ble trimmet korrekt.



Figur 5.29: Objektet

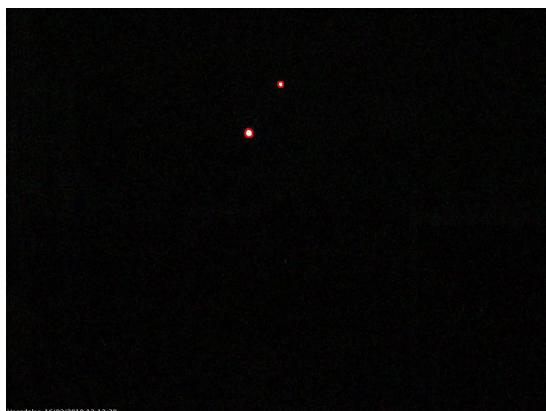


Figur 5.30: Objektets ferd

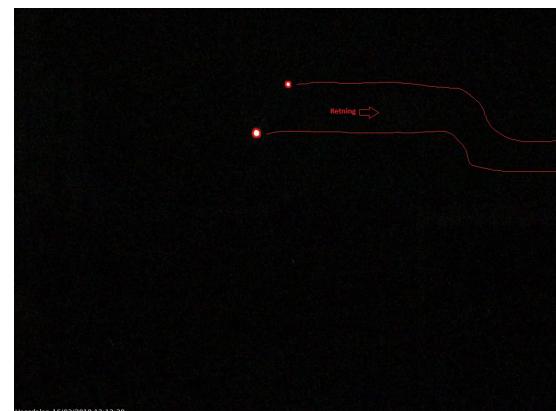
Test 10 - To objekter i bevegelse

Utførelse: Benyttet lomme- og hodelykt til å simulere to lysobjekter i bevegelse (fra venstre til høyre). Lysene ble påskrudd ca. 2 sekunder inn i filmen, med forskjellige plasseringer, hvor de beveget seg i et tregt tempo i starten og økte fortløpende til de forsvant ut av bildeområdet.

Resultat: Deteksjon. Programmet detekterte objektenes ferd problemfritt uavhengig av deres hastighet. Deteksjonsvideo ble trimmet korrekt.



Figur 5.31: To objekter detektert



Figur 5.32: Objektenes ferd

Test 11 - Flere objekter i bevegelse

Utførelse: Benyttet lommelykt, hodelykt, mobil og laser til å simulere lysobjekter i bevegelse. Objektene ble påskrudd ca. 2 sekunder inn i filmen hvor de beveget seg i et tregt tempo i starten og økte fortløpende til de forsvant ut av bildeområdet. Laseren var ikke med i bildeområdet i starten, men kom inn fra høyre side litt senere og holdt seg i bildet til etter at de andre lysene var borte.

Resultat: Deteksjon av samtlige objekter. Programmet detekterte objektenes ferd problemfritt uavhengig av deres hastighet. Deteksjonsvideo ble trimmet korrekt ved at den varte fra deteksjon av de første lysene og helt til laseren forsvant. *Denne videoen kan sees på vedlagt minnepenn.



Figur 5.33: Flere objekter



Figur 5.34: Detekerte objekter

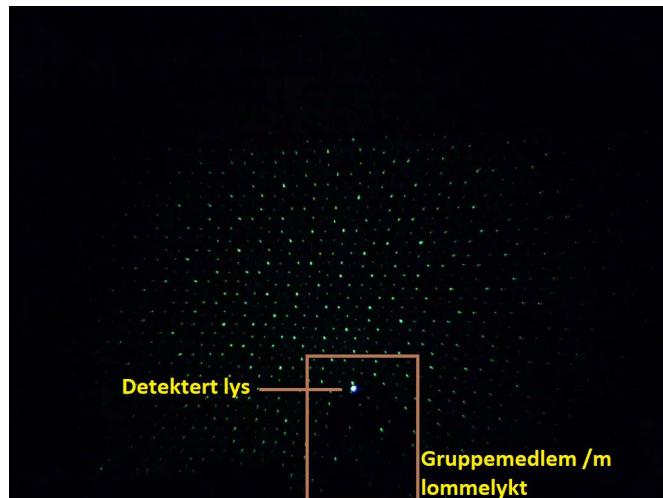


Figur 5.35: Objektenes ferd

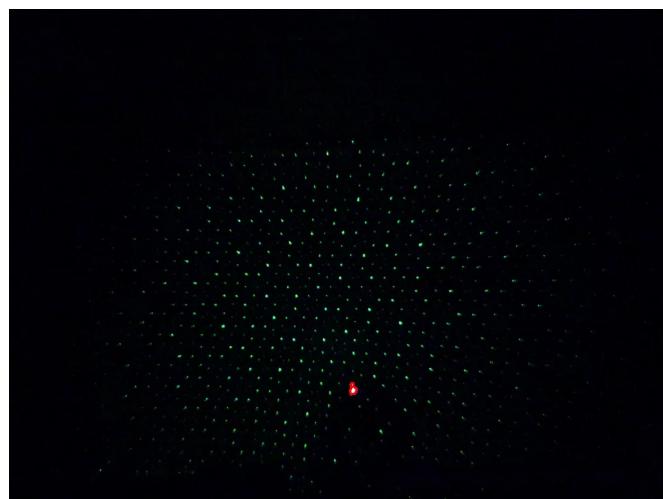
Test 12 - Fiktiv stjernehimmel

Utførelse: Benyttet laseren med et påskrudd "stjernefilter", figur 5.36, for å simulere en stjernehimmel. Justerte programmet slik at det ikke reagerte på "stjernenes" størrelse. Lommelykten ble slått på ca. 2 sekunder inn i filmen og fungerte som et lysfenomen som oppstår på en stjernehimmel.

Resultat: Programmet ignorerte "stjernene" og detekterte lysobjektet uten problemer. Videotrim og alarmbilder ble korrekt generert.



Figur 5.36: Objekt på fiktiv stjernehimmel



Figur 5.37: Objekt detektert på fiktiv stjernehimmel

Test 13 - Kort blink

Utførelse: Blinket raskt én gang med laser for å se om programmet oppdager fenomener som bare er synlige i en kort periode (mindre enn 0,3 sekunder).

Resultat: Programmet detekterte lyset til tross for dets korte varighet. På grunn av logikk implementert i gruppens program vil dette likevel ikke dokumenteres som et fenomen, siden så korte forekomster ofte er et fly som blinker.

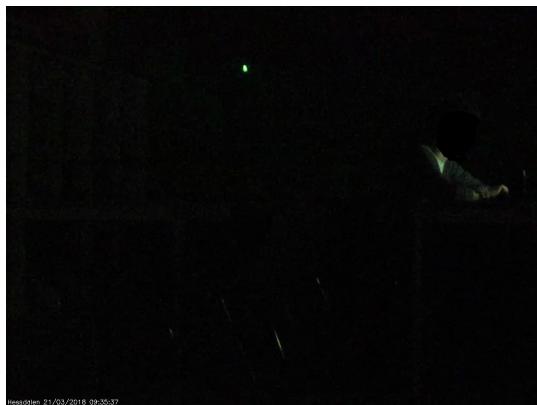
Test 14 - Korte, gjentagende lysblink

Utførelse: Blinket regelmessig med en lyskilde for å simulere et lysfenomen som blinker, eller forsvinner og så dukker opp igjen gjentatte ganger. Det som skulle unngås var at det ble opprettet en deteksjonsvideo og alarmbilder for hver gang det blinket.

Det ble filmet et videointervall på 10 sekunder. Fra sekund 1-5 ble det blinket syv ganger med laser før det var en pause til åttende sekund hvor det ble blinket to ganger før filmslutt. Forventet resultat var at det ble lagret kun to deteksjonsvideoer og to alarmbilder selv om det totalt var 9 blink.

Resultat: Suksess. Ved ferdig prosessering satt gruppen igjen med to deteksjonsvideoer med korrekt innhold og 2 alarmbilder (fra første og åttende sekund). Grunnen til at det ble to forskjellige deteksjoner var fordi parameteret, som styrer hvor mange kontinuerlige frames som ikke inneholder deteksjon før det anses å være slutt på deteksjonen, var satt til 30 frames (1 sekund). Så mellomrommet mellom sekund 5 og 8, som var på tre sekunder, oversteg dette parameteret og anså første fenomen som avsluttet.

Eksempelvis hadde de syv blinkene i sekund 1-5 et tidsmellomrom på mindre enn 30 frames mellom seg som førte til at disse ble ansett som et langt, sammenhengende fenomen. Systemet fungerer følgelig som ønsket.



Figur 5.38: Deteksjon 1

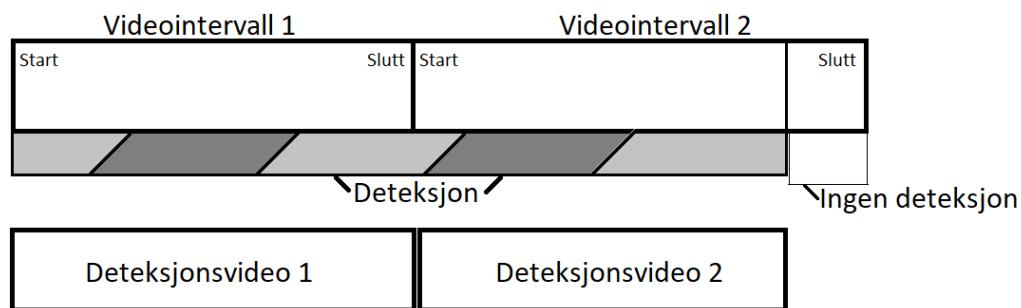


Figur 5.39: Deteksjon 2

Test 15 - Lysintervall som fortsetter utover innspillingsintervall (og inn i neste)

Utførelse: Lot det være et lysobjekt i bildeområdet så det konstant var deteksjon. Lot lyset være på i ca 4/5-deler inn i neste videointervall. Det ble filmet i intervaller på 5 sekunder.

Resultat: "Deteksjonsvideo 1" ble trimmet fra første frame det ble detektert til siste frame i intervallen. "Deteksjonsvideo 2" startet med deteksjon fra første frame og holdt på til ca 4/5-deler inn, men ble lagret som en egen trim.



Figur 5.40: Tidslinje

Test 16 - Maskering 50%

Utførelse: Testen gikk ut på å dekke 50% av bildeområdet med masken for å bekrefte at programmet ignorerte alt av lys og bevegelser i det maskerte området. En lysstråle ble ført gjennom bildeområdet fra venstre til høyre. Dersom gruppens logikk var implementert riktig ville deteksjonsvideoen kun inneholde frames så lenge lyset var til venstre for masken og stoppe så fort det gikk innenfor det maskerte området.



Figur 5.41: Maske laget ved hjelp av nettsiden

Resultat: Deteksjonsvideoen varte fra framden med første deteksjon og frem til lyset gikk inn i maskeområdet.



Figur 5.42: Lys ved 50% maske



Figur 5.43: Deteksjonsbilde ved 50% maske

Ut i fra bildet til høyre, figur 5.43, kan man også se at det kun er venstre halvdel som blir analysert av programmet.

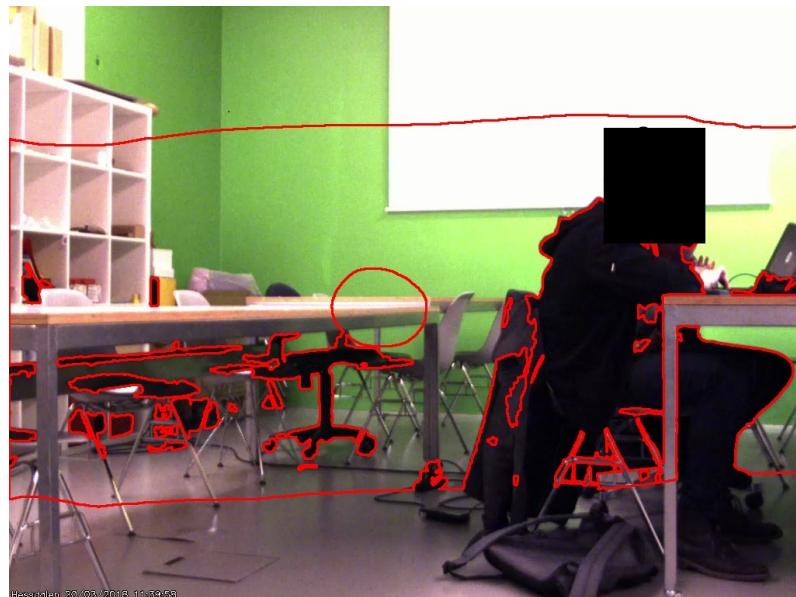
Test 17 - Maskering på topp, bunn og spesifikt punkt

Utførelse: Simulasjon hvor det ble maskert vekk en fiktiv bilvei nederst i bildeområdet, en eventuell måne øverst og eksempelvis et hus i midten. Masken laget på nettsiden:



Figur 5.44: Maskering

Resultat: Som figur 5.45 viser, tegnet programmet rundt områdene den fant piksler over en gitt threshold verdi. Det har suksessfullt ignorert topp, bunn og sirkelpunktet i midten. Andre markeringer på bildet som ikke er inkludert er objekter som er under thresholdverdien. Med andre ord ble mørke objekter bevisst ikke detektert.



Figur 5.45: Maskerte områder

5.5 Stresstesting

Hensikten med testene var å se hvordan SBC-en taklet et worst-case scenario hvor det konstant er deteksjon og det kontinuerlig må lages detektsjonsvideoer, samtidig som det filmes. Det som var interessant å få med seg her var om analysen og SBC-en klarte å henge med i tillegg til å teste SBC-ens kjøleegenskaper. Følgende to tester tar for seg dette.

I følge dokumentasjonen til TX2 har den følgende, nevneverdige egenskaper:
Dersom CPU når 95.5 grader eller GPU 93.5 grader vil TX2 redusere klokkehastigheten. Dersom CPU eller GPU når 101 grader vil TX2 slå seg av for å forhindre skade.

Test 18 - Stresstest (1 time)

Utførelse: Lot programmet kjøre i én time med deteksjon i hver eneste frame. Filmet i intervaller på 60 sekunder.

Resultat: Ved hjelp av terminalkommandoen “watch sensors” ble det notert CPU- og GPU temperaturer før og på slutten av testen.

Tabell 5.1: Intervaller på 60 sekunder

	Tidspunkt	CPU (grader)	GPU (grader)
Start	13:53	32	38.5
Slutt	14:59	51	57.5

```

tor Detections Pictures
nvidia@tegra-ubuntu: ~
Every 2.0s: sensors                                     Tue Mar 20 13:50:06 2018

BCPU-therm-virtual-0
Adapter: Virtual device
temp1:      +32.5°C  (crit = +101.0°C)

MCPU-therm-virtual-0
Adapter: Virtual device
temp1:      +32.5°C  (crit = +101.0°C)

GPU-therm-virtual-0
Adapter: Virtual device
temp1:      +38.5°C

A0-therm-virtual-0
Adapter: Virtual device
temp1:      +32.5°C  (crit = -40.0°C)

Tboard_tegra-virtual-0
Adapter: Virtual device
temp1:      +29.0°C  (crit = +107.0°C)

Tdiode_tegra-virtual-0
Adapter: Virtual device

```

Figur 5.46: Temperaturmåling

CPU var oppe i 53 grader, men gikk ned og holdt seg stabilt rundt 50 grader. På et tidspunkt i testen skrudde TX2 selv på kjøleviften. Konklusjonsvis økte CPU- og GPU-temperaturen med 19 grader under kontinuerlig filming og trimming. SBC-en har følgelig ingen problemer med å jobbe tungt over en lengre periode. Ettersom trimming av video tar lengre tid enn filming hoper det seg opp en del i deteksjonskøen. Dette vil ikke være et realistisk problem med tanke på at systemet alltid vil ha lengre perioder uten deteksjoner hvor det får hentet seg inn.

Test 19 - Testing av systemet over en dag

Utførelse: Plasserte kameraet rettet mot en trafikkert gang. Lot programmet kjøre fra 09:00 til 16:00. Filmet i intervaller på 2 minutter.

Resultat: Testen resulterte i en god del deteksjonsvideoer og alarmbilder ettersom gangen viste seg å ha mer aktivitet enn først antatt. I de travleste periodene med tilnærmet konstant deteksjon hoper det seg opp med originalvideoer i køen som ventet på å bli behandlet. I roligere perioder som i lunsjpausen rundt 12:00 og på slutten av dagen hentet programmet seg inn og fikk behandlet videoene som ventet i kø.

Det er ikke forventet at det vil være på langt nær så høy aktivitet i Hessdalen, og spesielt på kveldstid vil programmet enkelt hente seg inn ettersom det er færre elementer som kan slå ut som deteksjoner.

Tabell 5.2: Intervaller på 2 minutter

	Tidspunkt	CPU (grader)	GPU (grader)
Start	09:43	35.6	41
Slutt	15:21	51.5	57.5

Som tabell 5.1 og tabell 5.2 viser holder temperaturene seg stabile enten det kjøres i en time eller en dag.

Test 20 - Automatisk opplasting og sletting av video/bilder

Utførelse: Lot programmet lage to videointervaller på 15 sekunder med minst én tilfeldig deteksjon i hver så det ble opprettet tilhørende alarmbilder og deteksjonsvideoer. Testen var veldig godt dersom følgende skjedde for hver håndterte video:

- Deteksjonsvideo og alarmbilder ble overført til Project Hessdalen sin server
- Originalvideo, deteksjonsvideo og alarmbilder ble slettet automatisk etter suksessfull overføring

Resultat: Etter endringer i koden fungerte overføring og sletting som ønsket. Dersom opplasting til serveren ikke var mulig på tidspunktet gikk den videre uten å slette deteksjonsvideo og alarmbilder. Originalvideo kunne trygt bli slettet da det interessante allerede hadde blitt trimmet ut.

Test 21 - Håndtering av TX2 via SSH

Utførelse: Benyttet SSH-terminalprogrammet PuTTY til å koble en laptop opp mot SBC-en. For tilkoblingen trengtes det SBC-ens IP samt dens brukernavn og passord. Målet med testen var å kunne koble seg til eksternt for å utføre vedlikehold, gjøre endringer o.l. Testet med SBC tilkoblet både trådløst og kablet.

Resultat: Veldig godt tilkobling begge gangene. Alle normale/vanlige terminalkommandoer fungerer som om en jobbet direkte på SBC-en.

Test 22 - Tap av tid

Utførelse: Testen gikk ut på å sjekke hvor mye tid som gikk tapt mellom hvert videointervall. Det å stoppe og starte filming med kameraet har vist seg å ta en viss tid.

Det ble plassert en mobil med en stoppeklokke foran kamera i den hensikt å se på hvilket tidspunkt et videointervall sluttet på og hva neste startet med. Det ble filmet med intervaller på 5 og 60 sekunder.

Resultat: I tabellene 5.3 og 5.4 er tiden oppgitt som minutt : sekund : millisekund

Tabell 5.3: Intervaller på 5 sekunder

Tidspunkt på slutten av video	Tidspunkt på starten av neste video	Tid tapt / ikke filmet (sekund)
00:05:96	00:06:57	0.61
00:11:30	00:12:15	0.85
00:16:99	00:17:73	0.74

Tabell 5.4: Intervaller på 60 sekunder

Tidspunkt på slutten av video	Tidspunkt på starten av neste video	Tid tapt / ikke filmet (sekund)
01:01:10	01:01:72	0.62
02:01:63	02:02:25	0.62
03:02:28	03:03:02	0.74

Som resultatet ovenfor viser tar det rundt 0.6 til 0.8 sekunder før neste video blir påbegynt. Lengden på intervallene hadde ingen innvirkning hvor mye tid som gikk tapt på å starte og stoppe kameraet. Konklusjonsvis vil man ved intervaller på 5 sekunder tape mer tid i løpet av en dag enn om det filmes i for eksempel 30 minutters intervaller.

Eksempel med 0.70 sekunder tap mellom hvert intervall: 24 timer i døgnet som tilsvarer 48 halvtimer.

Tid tapt over 24 timer med intervaller på 1800 sekunder(30 minutter):

$$0.70 * 48 = 33.6 \text{ sekunder med tapt video over 24 timer}$$

Hvor mange ganger kan ett minutt splittes opp i 5 sekunder?

$$60/5 = 12 \text{ i ett minutt}$$

$$12*60 = 720 \text{ i en time}$$

$$720 * 24 = \mathbf{17280} \text{ i et døgn}$$

Tid tapt over 24 timer med intervaller på 5 sekunder:

$$0.70 * 17280 = 12096 \text{ sekunder med tapt video/tid over 24 timer}$$

$$12096 / 60 = 201.6 \text{ minutter}$$

$$201.6 / 60 = 3.36 \text{ timer med tapt video over 24 timer}$$

Følgelig ser man at det lønner seg å filme i lange intervaller for å oppleve minst tap. Dette er fordi kameraet bruker ca. ett sekund på å starte opp filming mellom intervallene.

5.6 Testing på dagtid

Systemet ble grundig testet i reelle scenarier på dagtid, men majoriteten av testene ga ikke nye resultater og er av den grunn ikke inkludert i rapporten.

Test 23 - Testing av objekt i bevegelse, utendørs

Testlokasjon: Rom D1-052

Utførelse: Et gruppemedlem bevegde seg i bildeområdet for å se om programmet oppdaget forskjellene fra forrige referansebilde.

Resultat: Før parameterne ble justert kom det deteksjoner både fra personen i bevegelse samt tretoppene som blåste i vinden. Etter justering av parameterne og maskering av trærne var resultatet deteksjon kun av personen som bevegde seg og hans skygge.



Figur 5.47: Referansebilde



Figur 5.48: Neste bilde



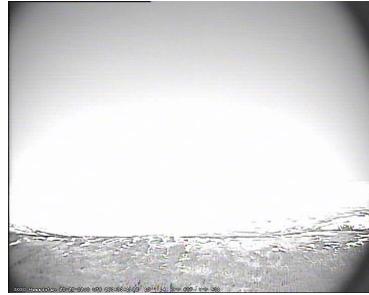
Figur 5.49: Før maskering



Figur 5.50: Etter maskering

I følge oppdragsgiver vil det nye kameraet overvåke samme område som dagens kamera gjør. Dette betyr at det ikke kommer til å være nærliggende, bevegelige elementer som trær i bildeområdet som ellers ville ført til falske deteksjoner.

Dagens bildeområde:



Figur 5.51: Kamera utsikt i Hessdalen [16]

Trærne på andre siden av dalen vil ikke kunne slå ut som deteksjoner ettersom de eventuelle bevegelsene deres ikke vil bli oppdaget av programmet.

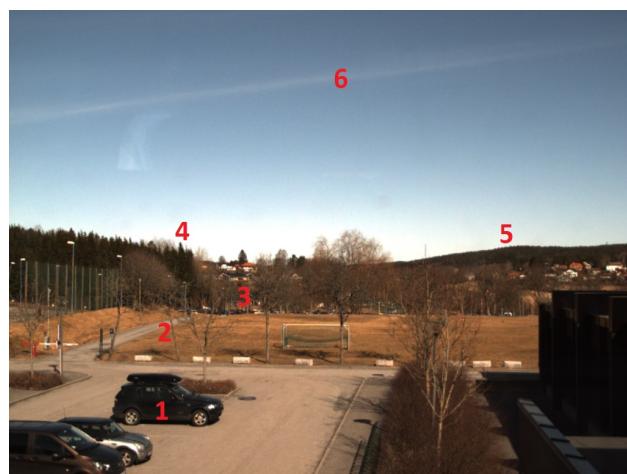
Test 24 - Testing av objekter i bevegelse, utendørs

Testlokasjon: Åpent landskap ovenfor hovedinngangen

Utførelse: Testen gikk ut på å detektere objekter på forskjellige avstander. På figur 5.52 er det påtrykt følgende fokusområdet:

1. Parkeringsplass, ca. 35 meter
2. Gangsti, ca. 110 meter
3. Fjernparkering, ca. 180 meter
4. Skogholtt, ca. 350 meter
5. Horisont, ca. 1600 meter
6. Himmel, varierende avstander

Resultat: Gjennom testens løp klarte systemet å oppdage biler, trær og mennesker i bevegelse på punkt 1-4, samt fugler som fly i punkt 6. Ettersom gruppen ikke hadde tilgang på en drone var det vanskelig å iscenesette deteksjoner på forskjellige, lengre avstander. Resultatene varierte ettersom gruppen justerte på parametrene.



Figur 5.52: Testing ovenfor hovedinngang

Kapittel 6

Diskusjon

Dette kapittelet starter med å drøfte i hvilken grad målene for prosjektet har blitt nådd. Videre kommer en evaluering av prosjektets resultater og metoder. Deretter tar gruppen for seg selve oppgaven, i tillegg til de positive og negative aspektene ved prosjektet. Til slutt tar kapittelet for seg muligheten for videreutvikling av produktet.

6.1 Målene

6.1.1 Hovedmål

Oppdragsgiver ønsket et modernisert oppsett av det som allerede opererer i Hessdalen. Det var også ønske om ny funksjonalitet som å maskere bildeområdet og få øyeblikket rett før og etter et fenomen på video. Hovedmålet med prosjektet var å lage et produkt som kunne detektere fenomener på himmelen og dokumentere dette i form av bilde og video.

Gruppen har produsert en løsning som detekterer, håndterer og dokumenterer fenomener. Som det er vist i kapittel 5, har systemet gjennomgått en rekke testing for å dokumentere at programmet fungerer som ønsket. Programmet er bygd opp slik at all kode vil fungere likt selv om bildeanalySEN byttes ut i senere tid. Det er også implementert maskering av bildeområde, gjort så enkelt som mulig for oppdragsgiver å endre. Løsningen gruppen har produsert benytter moderne teknologi og bringer ny funksjonalitet, noe som gjør at målet anses som oppnådd.

6.1.2 Delmål 1 - Maskere bort områder

Gruppen mener dette målet er nådd ved at det er produsert en nettside hvor oppdragsgiver har mulighet til å både tegne og viske bort områder som ikke skal tas hensyn til i analysen. Selv om dette ikke var en del av oppgavebeskrivelsen skjønte gruppen, etter samtale med oppdragsgiver, at dette var en sterkt ønsket funksjonalitet. Ved å ha denne funksjonaliteten kan oppdragsgiver enkelt utelukke områder i kameraets synsvinkel hvor man vet at det kan forekomme forstyrrelser som for eksempel billys.

6.1.3 Delmål 2 - Endre innstillinger og kommunisere med SBC via SSH

Det å kommunisere med kontrolleren via SSH var et viktig punkt, siden det gir oppdragsgiver muligheten til å operere systemet uten å fysisk måtte reise til Hessdalen. Utover dette har gruppen laget det slik at også nettsiden kommuniserer direkte med SBC-en via SSH og SFTP. Som nevnt i kapittel 3.2.8 er det lagt inn innstillinger på nettsiden som det anses relevant å kunne endre på.

Oppdragsgiver kan fra nettsiden også starte og stoppe programmet som analyserer bilder. Ved å ha implementert både SSH og muligheten for å styre programmet fra nettsiden, ser gruppen på dette delmålet som oppnådd.

6.1.4 Delmål 3 - Laste opp fenomen til Hessdalen

Oppdragsgiver ønsket at dokumenterte fenomener automatisk skulle lastes opp til Hessdalen sin server, slik at det ikke var nødvendig å manuelt måtte koble seg til SBC-en og laste opp materialet. Som det kommer frem i kapittel 3.2.10 blir dette gjort ved hjelp av SCP. All håndtering og analysering av videofilene blir gjort på SBC-en før det lastes opp. På denne måten unngås det å laste opp unødvendige store filer til serveren.

6.2 Resultater

Prosjektgruppen hadde som mål å levere et produkt bestående av en SBC og et kamera som kunne ta bilde og video av lysfenomener som oppstår på himmelen. Gruppen imøtekommerte disse kravene, med et produkt som inneholder en bildeanalyse som kan oppdagte fenomener som forekommer på himmelen og deretter dokumentere disse.

En del av leveransekravene var å levere en rapport som beskriver prosjektet og dets utvikling. Denne hovedrapporten imøtekommerte dette kravet ved å følge HiØ sin mal for bacheloroppgaver og viderebygging av denne. Utover oppgavebeskrivelsen så gruppen for seg, i tabell 2.1, å levere en måte å fremvise fenomenene på en ryddig måte på Hessdalens server. På grunn av at tiden ikke strakk til, fikk gruppen tatt tak i dette.

6.3 Vurdering av metoder

6.3.1 Litteratursøk

Ved å gjøre litteratursøk fikk gruppen utforsket utstyr og relevant teknologi, samt fikk de et overblikk av hva som fantes av løsninger fra før. Dette ga også gruppen ideer om hvordan oppgaven kunne løses.

Det er lest mye på programmeringsforumer, som Stackoverflow¹, for å lære og finne løsninger på problemer underveis. Det er også lest dokumentasjon og manualer for Python² og OpenCV³

6.3.2 Analyse

I kapittel 2 Analyse har gruppen vurdert utstyr, programmeringsspråk og relevant teknologi som ble avdekket ved litteratursøket. På denne måten har gruppen kommet frem til hvilke teknologier som passet best for oppgaven, og satt gruppen på riktig spor i forhold til en løsning.

6.3.3 Testing

Gjennom testing fikk gruppen mange gode resultater, men det ble også avdekket mangler ved programmet. Ved testing kom det frem nye ideer for hvordan programmet kunne forbedres. Første

¹<https://stackoverflow.com>

²<https://docs.python.org/3>

³[https://docs.opencv.org/3.4.1\]](https://docs.opencv.org/3.4.1)

gang programmet ble testet i mørket ble det avdekket at den daværende bildeanalyesen fungerte særdeles dårlig.

Det ble også funnet ut at det var nødvendig å implementere innstillingen som lar oppdragsgiver bestemme hvor mange frames som ikke lenger skal inneholde en deteksjon før det sluttet å filme. Hvis denne innstillingen ikke hadde blitt implementert ville fly som passerte kameraområde blitt detektert hver gang det blinket. Det ville da blitt lagret et bilde og video for hvert eneste blink. Ved å implementere innstillingen vil det kun bli lagret ett bilde og en video, slik at det enkelt kan fjernes ved feildeteksjon. Det fører også til at fenomener som forsvinner og kommer fort tilbake blir dokumentert i samme video.

6.4 Oppgaven

Oppgaven var beskrevet på en måte slik at gruppen har stått fritt til å velge utstyr og hvordan en endelig løsning skulle se ut. Oppdragsgivers ønsker og krav står beskrevet i kapittel 1.3. Prosjektet har ført til varierte arbeidsoppgaver som programmering, håndtering av både kamera og SBC, opprettelse av nettside. Det er gjort mye undersøking for å finne de enhetene som ville passe prosjektet best. Ved bestilling har gruppen forhørt seg med oppdragsgiver, for å forsikre seg om at enhetene hadde de riktige spesifikasjonene. Med gruppens begrensede erfaring rundt emnet bildebehandling har det vært et krevende, men interessant prosjekt.

6.4.1 Problemer underveis

Prosjektets art krevde omfattende bildeanalyse-kunnskap fra samtlige gruppemedlemmer. Dette er noe gruppen ikke hadde arbeidet med tidligere, slik at det ble brukt mye tid på lesing og opparbeidelse av kunnskap rundt emnet.

Det desidert største, uforutsette problemet var endringen av oppgavebeskrivelsen da det ble klart at en Raspberry Pi 3 ikke ville klare å utføre arbeidsoppgavene gruppen hadde tenkt å påtrykke den. Gruppen hadde som nevnt i kapittel 3.2.1 da allerede arbeidet med Raspberry Pi-en i 3 uker og konsekvensene av dette ble å reformulere det tekstlige arbeidet i hoveddokumentet samt å forkaste visse deler. Det var også nødvendig å:

- Lese seg opp på NVIDIA-s BC-er for å vurdere hvilken som best passet prosjektet
- Finne en leverandør med kort leveringstid og rabattmuligheter

Ved forsøk på å kontakte NVIDIA-Norge ble gruppen raskt klar over at outsourcing er en røl ting da gruppen, etter å ha ringt et norsk nummer, blir svart av en engelsk-indisk stemme. Personen kunne ikke svare på om det etterspurte produktet var tilgjengelig på europeiske lagre, leveringstider eller om det var muligheter for akademisk rabatt. Selve leveringen av SBC-en tok også lengre tid enn forventet, noe som resulterte i purringer til NVIDIA.

Et annet uforutsett problem gruppen støttet på var at SBC-enes strømbesparende ARM64-prosessorarkitektur ikke støttet diverse programvarer og biblioteker som ellers ville fungert på en tradisjonell datamaskin. Dette førte til at gruppen hadde begrenset med API alternativer samtidig som det meste som fantes på internett ikke støttet ARM64. En direkte følge av dette var at mye av programlogikken måtte utvikles selv fra bunnen av.

TX2-en bydde også på problemer da den skulle settes opp for første gang. Gruppen var nødt til å installere operativsystem på den tre ganger før den til slutt samarbeidet. De to første gangene møtte gruppen på mye feilmeldinger, manglende funksjoner og manglende rettigheter. Eksempelvis manglet hele skrivebords-menyen ved den første installasjonen.

Når det kom til hvordan å detektere et fenomen på himmelen, både dag og natt, viste dette seg å være et ultimatum. Siden oppdragsgiver ønsket at det skulle bli benyttet motion detection, trenger programmet et referansebilde for å se om det har skjedd endringer i bildeområdet. Ut ifra hvor ofte en velger å oppdaterer referansebildet vil det ha noe å si på hva som kan detekteres. Ved hyppig oppdatering vil stillestående eller ekstremt saktegående fenomener ikke bli oppdaget siden de ikke anses som forandringer når referansebildet blir byttet ut ofte. I gjengjeld vil falske alarmer fra for eksempel raske, lavtflyvende skyer og andre ting bli drastisk redusert.

Ved tregere oppdatering av referansebildet vil saktegående fenomener bli tatt med, men falske alarmer vil oppstå oftere.

6.4.2 Det positive

I løpet av prosjektets løp har gruppen tilegnet seg mye ny kunnskap innenfor bildebehandling, Python, multiprosessering, Linux, SBC-er, web, SSH og andre kommunikasjons- og overføringsprotokoller. Til tross for begrensete forkunnskaper ble det overlevert et velfungerende system som oppfylte oppgavens krav og ønsker, i tillegg til ekstra funksjoner og tjenester. Andre positive ting er erfaringen som gruppen har fått gjennom prosjektets løp, og muligheten til å arbeide med ny teknologi.

Gruppen har klart å opprettholde progresjonen takket være god planlegging. Eksempelvis var det perioder gruppen måtte vente på leveranse av kamera og SBC. På grunn av måten gruppen la opp prosjektet, var det mulig å jobbe med blant annet bildeanalysen samt maskeringstjenesten, uavhengig om nevnt utstyr var tilgjengelig.

Noe som gjorde oppgaven lettere var at gruppen kom frem til at det ikke var nødvendig å analysere og behandle video i sanntid. Video kunne dermed filmes i intervaller og analyseres i etterkant da det ikke er kritisk for oppdragsgiver å ‘få vite på sekundet’ om det har skjedd en deteksjon.

Overgangen fra Raspberry Pi til Jetson TX2 viste seg, til tross for nevnte problemer i 6.4.1, å gi en langt mer effektiv og pålitelig datamaskin som klarte å utføre alle de krevende operasjonene gruppen ‘kastet på den’. Python-koden, som ble skrevet for Raspberry Pi-en, var også i en viss forstand mulig å videreføre til TX2-en. Et annet positivt aspekt var at gruppen fikk 2000kr avslag på SBC-en etter å ha tilsendt leverandør bevis på at gruppen bestilte produktet for akademisk bruk.

6.4.3 Hva kunne vært gjort annerledes

Som i de fleste prosjekter er det alltid ting som kunne vært gjort annerledes og eventuelt bedre. Men dette kommer gjerne i form av etterpåklokskap eller når en ikke lengre har tid igjen til å fikse på det. Følgende punkter lister hva gruppen på slutten av prosjektet kunne tenkt seg annerledes:

- Det at den originale oppgavebeskrivelsen ordrett sa “bruk av Raspberry Pi” gjorde at gruppen ikke vurderte andre alternativer til single-board datamaskinen annet enn forskjellige modeller av Raspberry Pi.
- Gruppen brukte tid og penger på å ringe til USA-baserte selskaper, som solgte relevant utstyr, i forsøk på å bestille til Norge. Disse viste seg å ha lange, usikre leveringstider. Fokuset skulle fra starten av ha vært på europeiske selskaper.
- Dersom ett av gruppemedlemmene hadde tatt valgfaget “Bildebehandling og mønstergjenkjenning” parallelt med prosjektet hadde det blitt brukt vesentlig mindre tid på å lese seg opp på området. I tillegg kunne bildeanalysen blitt mer avansert.

- Dersom gruppen hadde hatt mer tid og gjerne bedre forkunnskaper, ville den optimale metoden for å analysere bilder innehårt machine learning. Med dette ville programmet blitt 'lært opp i hva et fenomen er eller hvordan det ser ut' så det kunne skille mellom for eksempel et fenomen og en fugl.
- I etterkant reflekterte gruppen over dialogen med oppdragsgiver. Det burde ha blitt satt opp regelmessige møter for å få konstant tilbakemelding på produktet under utvikling.

6.5 Videre arbeid

Gruppen har laget et produkt som kan settes opp i Hessdalen, klart til bruk. Programmet er laget på en slik måte at det er mulig for fremtidige grupper å fortsette arbeidet. Nedenfor listes det punkter som gruppen ønsket å ta tak i, men hvor tiden ikke strakk til. Følgelig er dette forslag til hvordan produktet kan videreføres.

Ved strømbrudd må noen fysisk trykke på SBC-ens startknapp for starte opp datamaskinen. I følge NVIDIA's hjelpeforum og SBC-ens manual er det mulig å få til at den skrur seg på av seg selv når den får strømtilførsel igjen. Det gjøres ved å utføre visse modifikasjoner, blant annet loddning og innlegging av motstander, på brettet.

6.5.1 Email-varsling

En mulig ekstrafunksjon til programmet er å sende oppdragsgiver en varslingsepst og/eller SMS ved deteksjon. Gruppen valgte å ikke prioritere dette da det ikke er kritisk for oppdragsgiver å vite om deteksjoner øyeblikket de skjer og på grunn av falske alarmer ville det blitt mye spam.

6.5.2 Sette opp systemet i Hessdalen

Det har ikke vært planlagt i dette prosjektet å reise til Hessdalen for å sette opp systemet. Målet har vært å utvikle og klargjøre et system som kan benyttes. Oppdragsgiver eller en senere bachelorguppe må derfor stå for å flytte systemet til Hessdalen.

Det som trengs gjøres:

1. Plassere kameraet på ønsket lokasjon
2. Trekke en USB-B ledning mellom SBC-en og kameraet
3. Gi strøm og internett til SBC-en
4. Lese av SBC-en sin IP-adresse og legge denne i konfigurasjonsfilen til nettside kontrollpanelet

6.5.3 Videreføring

Ved bruk av motion detection vil det kunne oppstå feilalarmer, noe det også gjør på det eksisterende systemet i Hessdalen. Det er åpent for at bildeanalysen enkelt kan oppgraderes eller byttes ut i programkoden av en eventuell senere bachelorguppe.

Bildeanalysen ligger inne i en funksjon kalt analyse(). Hvis denne funksjonen returnerer 1 tolker programmet det som at et fenomen er detektert, og 0 hvis ikke. Dette er den eneste logikken som må videreføres inn i en ny bildeanalyse ved utbytting.

Kapittel 7

Konklusjon

Prosjektets hovedformål var å levere Project Hessdalen et fungerende deteksjonssystem for dokumentering av lysfenomen som oppstår i Hessdalen. Gruppens delmål innebar å utvikle funksjonalitet som gjorde det mulig å kunne maskere bort visse deler av bildeområdet som ikke var ønskelig å ta med i bildeanalyesen. I tillegg skulle det være mulig å kommunisere med systemet eksternt, og endre på innstillinger ved hjelp av SSH. Dokumenterte deteksjoner skulle automatisk bli lastet opp til Project Hessdalen sin server og gjøres tilgjengelig for allmennheten.

Alle disse målene ble nådd og gruppen er godt fornøyd med sluttresultatet med tanke på hvor lite forkunnskaper de hadde innen bildebehandling. Gruppens valg av metoder og resultatene disse har gitt har også vært tilfredsstillende. Av metodene var det litteratursøk og analyse som ga størst utbytte i form av oversikt og forståelse rundt eksisterende teknologier.

Systemet endte opp med å bestå av kun to fysiske komponenter, et kamera og en single-board datamaskin. I motsetning til dagens eksisterende løsning har det nyutviklede systemet mulighet til å dokumentere fenomener både dag og natt, samt ta med øyeblikket rett før og etter deteksjonene. I bildeanalyesen ses det etter forandringer i bilder, og hvis det oppdages et nytt objekt dokumenterer programmet objektet i form av alarmbilder og en videosnutt med tilhørende klokkeslett. Videre har gruppen opprettet en nettside hvor oppdragsgiver har mulighet til å maskere ønskede områder som ikke skal tas hensyn til i bildeanalyesen. Nettsiden ble videreutviklet til å bli et kontrollpanel hvor innstillinger for programmet kan bestemmes. På denne måten har brukere av systemet enkelt mulighet til å endre på parametrene uten å måtte endre på koden. Basert på resultatene fra testingen, utført i kapittel 5, vil programmet kunne utføre de oppgavene som oppdragsgiver krevde i tillegg til de ønsker utover den originale oppgavebeskrivelsen.

Bibliografi

- [1] Ajax. http://stacksol.com/wp-content/uploads/2017/12/AJAX_logo_by_gengns.svg_.png. Hentet 13.03.2018.
- [2] C++. https://upload.wikimedia.org/wikipedia/commons/1/18/ISO_C%2B%2B_Logo.svg. Hentet 12.03.2018.
- [3] Css. https://www.planet-source-code.com/vb/2010Redesign/images/LangugeHomePages/HTML5_CSS_JavaScript.png. Hentet 16.03.2018.
- [4] D-link dcs-7110. <https://www.komplett.no/img/p/1200/098ee7db-3661-db01-6c6d-1dd60ecbc2ae.jpg>. Hentet 06.03.2018.
- [5] Dagens oppsett. <http://www.hessdalensorg/pict/station/ams-new1-big.jpg>. Hentet 14.03.2018.
- [6] Dfk 31bu03.h. <https://www.avsupply.com/images/items/the-imaging-source-cameras/med-res/dfk-31bu03-h.jpg>. Hentet 15.03.2018.
- [7] Dfm 21bu04-ml. <https://www.avsupply.com/images/items/the-imaging-source-cameras/hi-res/dmm21bu04ml.jpg>. Hentet 09.03.2018.
- [8] Dfm 31bu03-ml. http://www.oemcameras.com/pub/media/catalog/product/cache/c687aa7517cf01e65c009f6943c2b1e9/i/m/imaging_usb_ccd-800x800_1_1.jpg. Hentet 09.03.2018.
- [9] Digitalkamera. <https://n2.sdlcdn.com/imgs/a/i/t/Sony-CyberShot-DSC-W830-20-SDL275938723-4-7c89a.jpg>. Hentet 05.03.2018.
- [10] Gstreamer. https://blogs.s-osg.org/wp-content/uploads/2015/05/1024px-Gstreamer-logo.svg_.png. Hentet 08.03.2018.
- [11] Html. https://www.planet-source-code.com/vb/2010Redesign/images/LangugeHomePages/HTML5_CSS_JavaScript.png. Hentet 12.03.2018.
- [12] Høgskolen i Østfold. <https://www.hiof.no/assets/plugins/design-templates/illustrations/hiof-logo-nor-black-subbranding-it-v1.0.0.jpg>. Hentet 06.04.2018.

- [13] Imagemagick. https://i0.wp.com/www.brianlinkletter.com/wp-content/uploads/2015/12/logo_liquid-60.gif. Hentet 09.03.2018.
- [14] Industrikamera. http://www.tattile.com/wp-content/uploads/2013/10/TAG5_Tattile1.jpg. Hentet 06.03.2018.
- [15] Javascript. https://www.planet-source-code.com/vb/2010Redesign/images/LanguageHomePages/HTML5_CSS_JavaScript.png. Hentet 13.03.2018.
- [16] Kamera lokasjon. http://hessdalen.hiof.no/files/2018_05_05/soso/2018_05_05_50_04_hess.jpg. Hentet 06.04.2018.
- [17] Nikon d7500. <https://www.komplett.no/img/p/1200/be2a9e09-13c7-0ece-6453-819f7465433b.jpg>. Hentet 05.03.2018.
- [18] Nvidia jetson. https://developer.nvidia.com/sites/default/files/akamai/embedded/images/jetsontx2/TX2_Module_170203_0017_TRANSPI_2000px.png. Hentet 10.03.2018.
- [19] Nvidia jetson tx2 developer kit. https://elinux.org/images/9/9c/NVIDIA_Jetson_TX2_Module_Devkit.png. Hentet 05.03.2018.
- [20] Opencv. https://jayrambhia.files.wordpress.com/2012/06/opencv_hor_900_1.jpg. Hentet 08.03.2018.
- [21] Overvåkningskamera. https://images-na.ssl-images-amazon.com/images/I/41SztnQ6%2BUL._SX425_.jpg. Hentet 05.03.2018.
- [22] Php. <https://proappsoft.com/blog/wp-content/uploads/2014/12/PHP7-white-bg.png>. Hentet 10.03.2018.
- [23] Python. <https://www.raspberrypi.org/documentation/usage/python/images/python-logo.png>. Hentet 12.03.2018.
- [24] Raspberry pi. <https://upload.wikimedia.org/wikipedia/commons/thumb/e/e6/Raspberry-Pi-3-Flat-Top.jpg/1200px-Raspberry-Pi-3-Flat-Top.jpg>. Hentet 13.02.2018.
- [25] Scikit. http://scikit-image.org/_static/img/logo.png. Hentet 10.03.2018.
- [26] Sony cybershot dsc-hx90v. <https://www.komplett.no/img/p/800/5e65c2f9-eb16-4704-b135-b12b5dd49c0c.jpg>. Hentet 05.03.2018.
- [27] Speilreflekskamera. https://images-na.ssl-images-amazon.com/images/I/81Ev7DpPvvL._SX355_.jpg. Hentet 05.03.2018.
- [28] Virkemåte. <https://teknoprod2010.wikispaces.com/file/view/tekx2.png/163444245/506x291/tekx2.png>. Hentet 10.03.2018.
- [29] Tanya Amert, Nathan Otterness, Ming Yang, James H Anderson, and F Donelson Smith. Gpu scheduling on the nvidia tx2: Hidden details revealed. In *IEEE Real-Time Systems Symposium (RTSS)*, 2017.

- [30] Ross Cutler and Larry S. Davis. Robust real-time periodic motion detection, analysis, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):781–796, 2000.
- [31] Stefan Hinz. Fast and subpixel precise blob detection and attribution. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 3, pages III–457. IEEE, 2005.
- [32] Andrew Morgan, Zachary Jones, Richard Chapman, and Saad Biaz. An unmanned aircraft see and avoid algorithm development platform using opengl and opencv. *Journal of Computing Sciences in Colleges*, 33(2):229–236, 2017.
- [33] Vladimir Vujović and Mirjana Maksimović. Raspberry pi as a sensor web node for home automation. *Computers & Electrical Engineering*, 44:153–171, 2015.
- [34] Apachetechnology: Works, How It. Connected components labeling.

Tillegg A

Hvordan bruke systemet

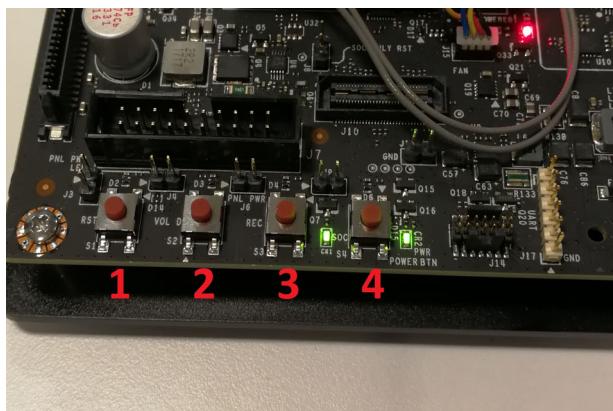
Sette opp system

For å sette opp systemet må det være tilgang på et 230V uttak og trådløst/kablet nettverk. Det er også nødvendig å koble til en skjerm(HDMI), mus og tastatur. OBS: Det er nødvendig med en USB-splitter når mus og tastatur skal tilkobles grunnet TX2 bare har en USB inngang.

1. Plugg inn nettverkkabel, strømtilførsel, HDMI, mus og tastatur og USB-B kabelen som går fra kameraet til SBC-en
2. Start Jetson TX2-en ved å trykke på Power button avbildet nedenfor
3. Få tak i IP-en til SBC-en ved å åpne terminal og skrive *ifconfig*
4. Legg inn funnet IP-adresse i config filen til nettsiden. Denne finnes i inc/config.php
5. Plasser kamera i ønsket retning

TX2 har følgende klikkbare knapper:

1. Reset
2. User defined
3. Force recovery
4. Power button



Programforklaring

- Utviklet system ligger på skrivebordet til Jetson TX2-en. Fullstendig filsti er `/home/nvidia/Desktop/Bachelor`
- For å starte programmet uten å bruke nettsiden åpne mappen kalt Bachelor som ligger på skrivebordet. Inne i mappen ligger en fil `main.py` som må kjøres. For å gjøre dette åpne et terminal vindu i mappen og skriv inn `python3 main.py`. OBS: ikke åpne terminalvinduet fra Desktop
- For å starte programmet mens man er tilkoblet via SSH, ved hjelp av et program som PuTTY, gjør følgende:
 1. Skriv inn kommandoen `export DISPLAY=:0` slik at programmet starter på Jetson TX2-en og ikke klienten som er tilkoblet
 2. Naviger til Bachelor mappen som ligger på skrivebordet
 3. Start programmet ved å skrive inn `python3 main.py`
- Det er viktig at mappestrukturen i Bachelor mappen *ikke* røres eller at mappenavn endres.
- Det blir kontinuerlig filmet og lagret videoer i mappen Bachelor/Videos. Disse videoene er rå-materiale og ligger der kun for å bli behandlet. Disse blir fortløpende slettet, og er ikke dokumentasjonen av fenomenene.
- Ved deteksjon av fenomen dokumenteres det i to mapper. Dokumentasjonen kan sees i følgende mapper, men blir automatisk lastet opp og slettet fortløpende.
 - Mappen Bachelor/Detections/Pictures inneholder alle bilder, både markerte med røde sirkler og originalbilder, som er dokumentert av et fenomen.
 - Mappen Bachelor/Detections/Trims inneholder alle videoer som er dokumentert av et fenomen.

Kontrollpanel innstillinger

1. Seconds to save on detection
Hvor mange sekund som legges til på start og slutt av videotrim
2. Pixel size detect
Minimum og maks størrelse på objektene som skal detekteres
3. Number of frames before updating reference picture
Hvor mange frames som tas før referansebilde oppdaterer seg
4. Number of frames containing a detection before start recording
Antall frames som må inneholde en deteksjon før systemet starter å lagre video
5. Number of frames not containing a detection before stop recording
Antall frames som ikke skal inneholde en deteksjon før programmet stopper å lagre video
6. Text overlay
Teksten som skal legges til i tillegg til dato og klokkeslett på hvert bilde og video.

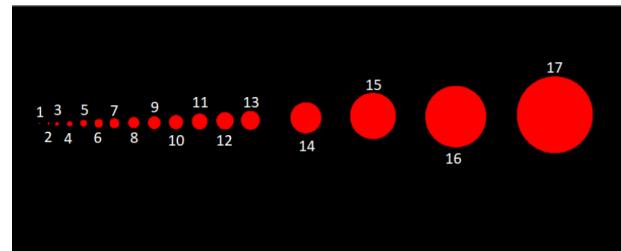
Hvordan bruke nettside

1. Gå inn på nettsiden til programmet, trykk på **Change settings** og sett de ønskede parametrerne som systemet skal operere med
2. Etter at kameraet er plassert i ønsket posisjon, kan det hentes fram et bakgrunnsbilde å tegne masken på ved å trykke på **Get new background picture**
3. Det kan nå tegnes området som ikke er ønskelig å få med i analysen. Dette gjøres ved å klikke ned venstre museknapp og bevege musen inne i den hvite firkanten på høyre side av nettsiden.
4. Når dette er gjort trykk på **Start**
5. For å avslutte programmet trykk på **Stopp**, hvor det vil det komme en pop-up som må bekreftes

Tillegg B

Pikselstørrelse for deteksjoner

Ved hjelp av bildet og tabellen under kan brukere av systemet vurdere hvilke min/max piksel-verdier som burde settes i kontrollpanelet ut i fra hvilke størrelser en ønsker å detektere.



Sirkelnummer	Pikselstørrelse	Lest konturstørrelse (fra program)
1	1	7
2	2,5	15
3	5	39
4	7,5	75
5	10	117
6	12,5	174
7	15	234
8	17,5	296
9	20	379
10	22,5	465
11	25	565
12	27,5	684
13	30	807
14	50	2121
15	75	4642
16	100	8158
17	125	12642

Eksempelvis hvis det kun er interessant å detektere mindre fenomener/objekter, men eliminere sol/måne-deteksjoner, så kan dette gjøres v.h.a “Pixel size detect”-parametrene på kontrollpanel-nettsiden.

I eksemplet under ble det kun sett etter objekter med konturstørrelse mellom 115 og 600. Alt over eller under ble ignorert. Som det leses fra tabellen er sirkel 5 til 11 innenfor disse parametrene. Bildet under er fra et testprogram som markerte objekter som var innenfor grensene med en grønne sirkler.



