BACHELOROPPGAVE:

BO22EH-01

ESTIMERING AV POSISJONSENDRING VED COMPUTER VISION

Bjørnar Årvik
Jan Steinar Stuvik
Eirik Holme Grutle

30.05.2022

Dokumentkontroll

Rapportens tittel: Estimering av posisjonsendring ved computer vision	*Dato/Versjon* 30.05.2022
	Rapportnummer: BO22EH-01
Forfatter(e): Bjørnar Årvik Jan Steinar Stuvik Eirik Holme Grutle	*Studieretning:* AUTYH_2019_HØST
	Antall sider m/vedlegg 63
Høgskolens veileder: Gisle Yngvar Romslo Kleppe	*Gradering:* Åpen
Eventuelle Merknader: Gruppen tillater at oppgaven kan publiseres.	

Oppdragsgiver: Kystdesign AS Eikeskogvegen 80 5570 Aksdal Norway +47 52 70 62 50 post@kystdesign.no	*Oppdragsgivers referanse:* Torbjørn B. Hansen
Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon): *Torbjørn B. Hansen* *hansen@kystdesign.no*	

Forord

Oppgaven er utført som avsluttende del av utdanninga for tre studenter fra ingeniørlinjen: Automatisering med robotikk på HVL campus Haugesund våren 2022. Oppgaven utgjør 20 av totalt 180 studiepoeng ved fulført studium.

Oppgaven ble presentert til HVL Haugesund av Torbjørn B. Hansen fra Kystdesign AS, i et ønske om å se på løsninger for bruk av «computer vision» sammen med eksisterende monokulære kamera på en ROV (Remotely Operating Vehicle).

Programmeringsspråket som er blitt brukt er Python, da dette er mye brukt sammen med OpenCV for bildeprosessering og «computer vision». «Computer vision» har ikke vært en del av pensum gjennom studiet. Studentene i gruppen hadde heller ingen erfaring med Python fra før. Derfor var det mye nytt og spennende å sette seg inn i, gjennom arbeidet med bacheloroppgaven.

Gruppen vil takke Torbjørn B. Hansen for en spennende, lærerik og utfordrende oppgave. Vil også takke for gode tips og innspill under møtene.

Gruppen vil også takke Gisle Romslo Kleppe for hjelp og innspill med den teoretiske delen av oppgaven og skriving av rapporten.

Sammendrag

Oppgaven består av å finne løsninger på hvordan en kan bruke «computer vision» til å tolke forflyttingen av et objekt gjennom eksisterende monokulære kamera på en ROV. Det skal da lages et program som implementerer valgte løsninger.

Det ble brukt mye tid på utforskning og testing av forskjellige typer metoder som kunne finne igjen et objekt i et bilde. Konklusjonen ble da at det ikke bare var én fullgod løsning på problemstillingen, men det var flere måter å løse det på. Det ble derfor bestemt å utvikle programmet slik at det kan kombinere flere forskjellige metoder. Bruker kan da velge hva som skal brukes og sammenligne dem.

Gruppens løsning ble dermed å lage et program som kan kjøre opp til tre «tracking»-algoritmer samtidig. Etter testing av forskjellige «tracking»-algoritmer ble de tre beste valgt etter deres prestasjoner. Det ble også valgt å legge til «tracking» ved bruk av «Aruco»-markør, da denne gir ut en estimering av posisjon og rotasjon av kameraet i forhold til markøren. Videre ble det laget et kalibreringsprogram for å eliminere eventuelle forvrenginger av bilde som kan oppstå med forskjellige typer kamera.

Den ferdige løsningen dekker de krav som kommer frem av oppgaveteksten.

1 Innhold

1 Innledning

1.1 Organisering av rapporten

Det blir først en presentasjon av oppdragsgiver, problemstilling og hovedidé for løsning. Neste del er en gjennomgang av teori for å gi en forståelse av hva som ligger bak prinsippene innenfor «computer vision». Deretter blir de forskjellige løsningsforslagene presentert med en konklusjon. Til slutt blir det endelige produktet og tester presentert.

1.2 Oppdragsgiver

KYSTDESIGN leverer et bredt spekter av produkter til ulike bransjer som olje & gass, kartlegging, fiskeoppdrett, havforskning, søk/redning og marinen.

De har et flerfaglig ingeniørteam med spesialister innen mekanisk design, hydrauliske systemer, elektronikk og programvareutvikling kombinert med omfattende offshore-erfaring som sikrer at produktene passer for operasjonelle utfordringer.

De leverer nøkkelferdige komplekse systemer innen ROV.

Kystdesign holder til i Aksdal, 15 km øst for Haugesund, halvveis mellom Stavanger og Bergen. Lokalene inkluderer kontorer, verksteder, kurs/trenings-fasiliteter, kantine og det største prøvebassenget i Nord-Europa.

1.3 Problemstilling

En ROV blir i dag hovedsakelig styrt manuelt fra en eller flere ROV-piloter som sitter i et kontrollrom på overflaten. Hvordan ROV-en oppfører seg i vannet er direkte avhengig av manøvreringer piloten utfører med hjelp av styrepinner i kontrollrommet. Det finnes enkelte hjelpemidler i dag som f.eks. gjør at ROV-en kan holde konstant høyde over havbunn ved bruk av ett altimeter. En INS (Inertial Navigation System) brukes til å måle endring i orientering og akselerasjon, denne vil kunne «drifte» over tid og trenger derfor en korrigerende tilleggsmåling. For å lokalisere den absolutte posisjonen til en ROV brukes ett USBL (Ultra-Short Baseline) system. Dette gir en tredimensjonal posisjon av en ROV i havet i forhold til supply-skip med en feilmargin på <1%. Via båtens DPS (Dynamic Positioning Systems) kan det da regnes ut absolutt posisjon av ROV. Dette systemet er dyrt og kan være unødvendig nøyaktig for mange operasjoner. Det er derfor ikke standard på en vanlig offshore ROV, men brukes mest på spesialiserte survey ROV-er. I enkelte operasjoner vil det være mer interessant og tilfredsstillende nok å vite posisjonen til ROV-en i forhold til et objekt som er i nærhet av ROV-en. Dette vil kunne forenkle prosessen hvis en ønsker å følge eller holde posisjonen til ROV-en konstant i forhold til det objektet, noe som ofte er tilfelle i en arbeidssituasjon.

Alle ROV-er blir levert med et kamerasystem til overvåking av arbeidsoperasjon eller inspeksjon. Kameraene er plassert på sentrale punkter på ROV-en som gjør at en har tilfredsstillende visuell kontroll. Det vil derfor være en gunstig løsning hvis en kan bruke de eksisterende kameraene til å måle en posisjonsendring mellom ROV og interessant objekt, da det ikke er særlig begrenset hvor objektet må befinne seg i forhold til ROV-en. Det kan da være mulig at ROV-en følger et bestemt objekt automatisk som f.eks. en modul som senkes ned på havbunnen av en kran, eller holde konstant posisjon i forhold til et objekt den skal utføre et arbeid på eller overvåke.

Ved å bruke kamera som allerede er montert på ROV-en for å gjøre denne jobben, vil dette bli en langt rimeligere løsning enn å måtte installere nye og mer avanserte instrumenter. Det er en løsning som i noen situasjoner kan erstatte behovet for de mest avanserte systemene for dagens posisjoneringssystem eller fungere som en ett supplement for å øke sikkerhet eller lette arbeidet for pilotene.

1.4 Hovedidé for løsningsforslag

Video fra kamera på ROV-en strømmes opp til kontrollrom på et supply-skip/rigg. Her kan bildestrømmen kobles mot gruppens programvare. Ved bruk av «computer vision»-algoritmer vil det være mulig å følge et ønsket objekt i en bildestrøm ved å analysere hvordan pikslene endres over tid fra bilde til bilde. Gjennom denne prosessen kan algoritmen gi ut hvor mye posisjonen til ROV-en i forhold til et bestemt objekt i bildet, endrer seg i horisontal-, vertikalt- og i dybde-retning (x, y og z). Disse parameterne kan da brukes i en regulator for posisjonering av ROV. Produktet skal helst kunne tilpasses de fleste typer kamera som er standard på en ROV i dag slik at det da ikke er noe problem å installeres opp mot eksisterende ROV-systemer.

2 Kravspesifikasjon

2.1 Oppgavetekst

Oppgavetekst er utarbeidet sammen med ekstern veileder:

«Lage en programvare som ved bruk av computer vision kan oppfatte en form, objekt eller farge som igjen kan brukes for å tolke forflytning av et objekt innenfor bildesynet til ett enkelt kamera på en ROV. Forflytning som skal tolkes er hovedsakelig den vertikale og horisontale forflytningen i bildet (x og y-retning). Det skal også vurderes mulige løsninger for å bedømme forflytning i z-retning i form av dybde i bildet.»

2.2 Programvare

- Programvare må være kompatibelt med flere typer kamera med forskjellige typer linser og oppløsning. Dette fordi produktet skal kunne brukes på mange forskjellige typer av ROV-er som er levert med forskjellige kamerasystemer.
- Programvaren må kunne gi informasjon som kan brukes til regulering av posisjon.
- Seriell, TCP/IP eller UDP tilbakemelding.

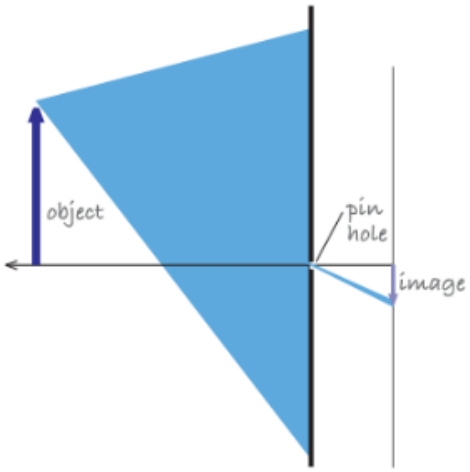
3 Forkortelser og ordforklaringer

AdaBoost	Adaptive boosting
ANN	Artificial Neural Network
ASEF	Average of Synthetic Exact Filters
CSR-DCF	Channel and Spatial Reliability of Discriminative Correlation Filter.
CSRT	Channel and Spatial Reliability Tracking
Descriptor	"Beskriver". Noe som inneholder informasjon for å identifisere ett særtrekk.
DNN	Deep Neural Network
DPS	Dynamic Positioning Systems
Features	Særtrekk som er lett gjenkjennbare
Forward-Backward Error	Metode for feildeteksjon ved tracking.
GOTURN	Generic Object Tracking Using Regression Networks
GUI	Graphical User Interface
HOG	History of Oriented Gradients
INS	Inertial Navigation System
KCF	Kernelized Correlation Filter
MOSSE	Minimum output sum squared error
MIL	Multiple Instance Learning
ONNX	Open Neural Network Exchange
Pinhole camera	Nåløye kamera
PSR	Peak to sidelobe ratio
R-CNN	Region-Based Convolutional Neural Network
ROI	Region of interest
ROV	Remotely Operating Vehicle
TLD	Tracking, learning and detection
Tracker	Algortime som følger gjenkjennbare sammensetninger av piksler.
USBL	Ultra-Short Baseline
VOT	Visual Object Tracking
SSD	Single-Shot Multi-box Detection
Tk	Kryssplatform widget verktøy
Tcl	Tool Command Language

4 Teori

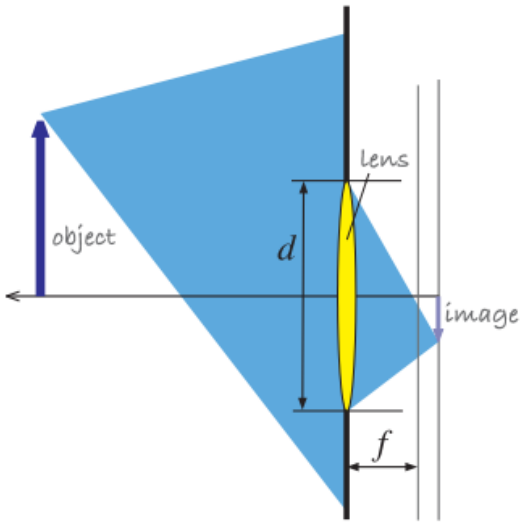
4.1 Kamera

Et digitalkamera fanger et bilde ved hjelp av elektronikk og optikk. Kameraet kan også ta opp video i form av en samling med bilder over tid. Disse bildene kan deretter bli behandlet av datamaskiner til blant annet «computer vision» som objektgjenkjenning og «tracking».



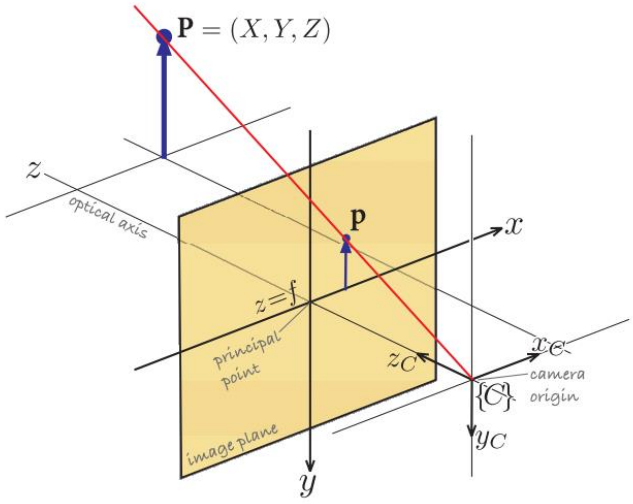
Figur 1 "Pinhole camera" [1, p. 320]

Et kamera består av et kamerahus med en irisblender og en bildebrikke, den kan også ha et objektiv som er bygd opp av to eller flere linser. Et av de enkleste kameraene kalles for et «pinhole camera», der kamerahuset bare er en lystett boks med en liten åpning i den ene siden. I denne kameramodellen vil lys fra objektet foran åpningen projisere et invertert bilde av objektet på motsatt side av boksen. Som vist i Figur 1 er høyden på det inverterte objektet bestemt av avstanden mellom åpningen og objektet. Siden «pinhole camera» ikke har fokusjusteringer vil derfor alle objektene være i fokus uavhengig av avstanden i landskapet.



Figur 2 "Pinhole camera" med linse [1, p. 320]

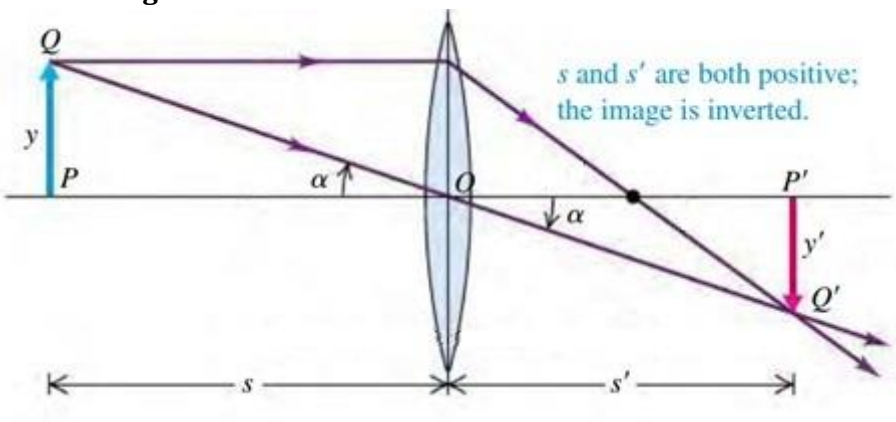
For å få et sterkere bilde må kameraet få inn mer lys fra objektet. Figur 2 viser en tynn konveks linse, denne gjør at kameraet får inn mer lys fra objektet over et større område og sender det inn i kameraet. Dette er en forenklet modell da kameraer som oftest har mer kompleks optikk.



Figur 3 Perspektivprojeksjonsmodell [1, p. 320]

Når et kamera tar et bilde, blir objektet projisert fra 3D til 2D. Dette gjør at bildet «mister» z-aksen og dybden «forsvinner». Figur 3 over viser perspektivprojeksjonsmodellen. Her kan en se at punkt $P = (X, Y, Z)$ i verdens koordinater blir projisert til koordinatene $p = (x, y)$ der $(x, y) = (f\frac{X}{Z}, f\frac{Y}{Z})$ som er et perspektiv.

4.1.1 Distansemåling med kamera



Figur 4 Sammenheng mellom avstand til objekt og bildet [2, p. 1128]

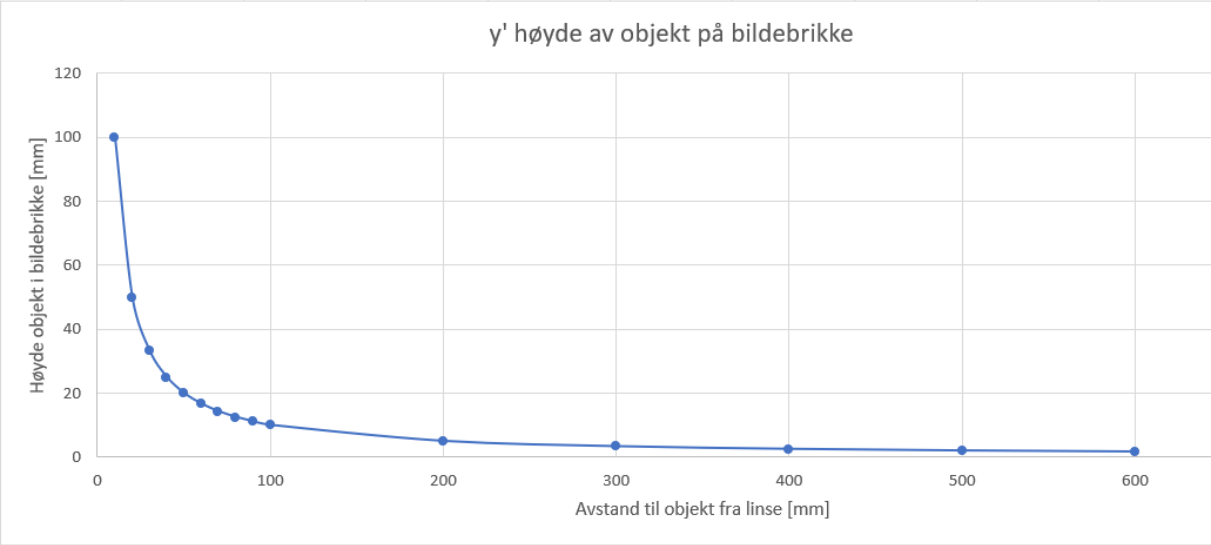
Figur 4 viser Figur 3 fra siden i z-, y-planet. For å finne distansen s fra linse til objekt i bildet kan en bruke sammenhengen:

$$\tan(\alpha) = \frac{y}{s} \Rightarrow s = \frac{y}{\tan(\alpha)}$$

For å kontrollere forholdet mellom distansen til et objekt som blir observert i bildet og høyden til objektet brukes følgende utrekning, setter da inn for $\tan(\alpha)$:

$$\tan(\alpha') = \frac{y'}{s'} \Rightarrow y' = \tan(\alpha) \cdot s' \Rightarrow y' = \frac{y}{s} \cdot s'$$

s' er avstanden fra linse til bildebrikke som er konstant.
s er avstanden til objektet fra linsen.
y' er avbildet høyde av objektet på bildebrikken.
y er høyden av objektet som er konstant.
α er vinkelen mellom senterlinjen som går gjennom linsen og linjen som går fra topp av objekt gjennom senter av linsen. Illustrert i Figur 4

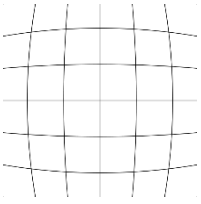


Figur 5 - Plot av distanseforhold mellom høyde av objekt og avstand til kamera

På Figur 5 kan en se y' plottet som funksjon av s. Den illustrer tydelig en ulineær sammenheng mellom avstanden til objektet fra kamera og registrert høyde av objektet på bildebrikken. I plottet er objektet satt til en høyde y = 100 mm og avstand mellom linse og bildebrikke s' = 10 mm.

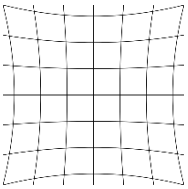
4.1.2 Kamerakalibrering

Noen kameraer kan lage en merkbar forvrenging på bildet. Dette kan digitalt korrigeres ved å kalibrere kameraet.
Forvrenginger som kan forekomme er som følgende:



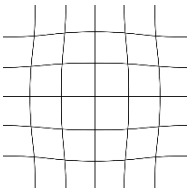
Tønneforvrengning:
Bildeforstørrelsen avtar med avstanden fra den optiske aksen, noe som fører til at rette linjer vil bli bøyd utover fra senter i bildet. Figur 6

Figur 6 Tønneforvrenging [3]



Puteforvrengning:
Bildefortørrelsen øker med avstanden fra den optiske aksen, noe som fører til at rette linjer blir bøyd innover mot senter i bildet. Figur 7

Figur 7
Puteforvrenging [3]



«Mustache»-forvrengning:
En blanding av tønne og pute forvrenging. Figur 8

Figur 8 «Mustache»-forvrengning [3]

Radiell forvrengning:

$$x_{forvrenging} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{forvrenging} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Tangentiell forvrengning:

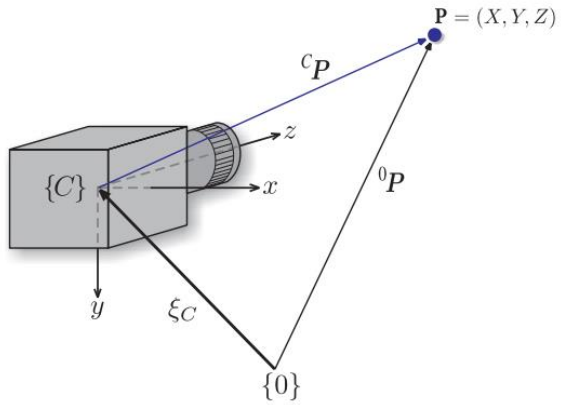
$$x_{forvrenging} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{forvrenging} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

Forvrengningskoeffisienter = $(k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$

4.1.3 Kameramatrisen

I homogene koordinater er et punkt i landskapet projisert til et bilde (Figur 9) ved den følgende matrise multiplikasjonen:

$$\tilde{P} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$



Figur 9 Globale kamera koordinater [1, p. 323]

Kameramatrise = $\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$

Kameramatrisen kan bli brukt til å fjerne forvrengning som kan forekomme på grunn av linsa til kameraet. Når kameramatrisen er regnet ut er den unik til sitt kamera. Denne er representert av en 3x3 matrise som sett under der (f_x, f_y) er linsens brennvidde og (c_x, c_y) er linsens brennpunkt. [1, pp. 322-325]

4.2 Bildeprosessering

4.2.1 Bilde som en funksjon

Et bilde kan bli representert som en funksjon av to variabler (x og y) som definerer et to-dimensjonalt område. Et digitalt bilde er bygd opp av et rutenett med piksler der hver piksel har en verdi som representerer hvilken intensitet av lys som forekommer i et gitt område(piksel) i bildet(rutenettet).

4.2.2 Modellering av piksel-lysstyrke

For å finne lysstyrken til bildet, er det 3 fenomen som avgjør dette.

Kamera responsen: Moderne kamera responderer lineært til moderat intensitet av lys, men er merkbart ulineær ved betraktelig mørkere og lysere belysning. Dette tillater kamera å lage et bredt dynamisk område av naturlig lys uten at det blir mettet.

For de fleste tilfeller kan en anta at kameraresponsen er lineær i forhold til lysintensiteten på overflateområdet. Hvor $\boldsymbol{X}$ representerer et punkt i rommet som projiseres til et punkt $\boldsymbol{x}$ i bildet, $\boldsymbol{I_{overflate}(X)}$ er intensiteten av overflateområdet i $\boldsymbol{X}$ og $\boldsymbol{I_{kamera}(x)}$ er kamera responsen i $\boldsymbol{x}$. Da får en denne modellen:

$$\boldsymbol{I_{kamera}(x) = kI_{overflate}(X)}$$

Overflaterefleksjon: Forskjellige punkt kan reflektere forskjellige mengder med lys, der mørke overflater reflekter mindre og lyse overflater reflektere mer.

Belysning: Mengden av lys som et begrenset område mottar, er avhengig av den totale lysintensiteten som påføres og geometrien til området. Den totale intensiteten kan bli forandret på grunn av lyskilder som kan bli overskygget eller ha sterke retningsbestemte komponenter som kan gi utslag.

[4, pp. 62 - 63]

4.2.3 Hvordan en datamaskin ser bilder

For en datamaskin er et digitalt bilde en 2D matrise bygd opp av hver piksels verdi. Verdien representerer intensiteten over fargespekteret. For et gråskalert bilde, er hvert element i matrisen lagret som en Byte. Hvert av disse elementene representerer intensiteten til pikselen i intervallet [0,255]. Hvor 0 er sort og 255 er hvit.

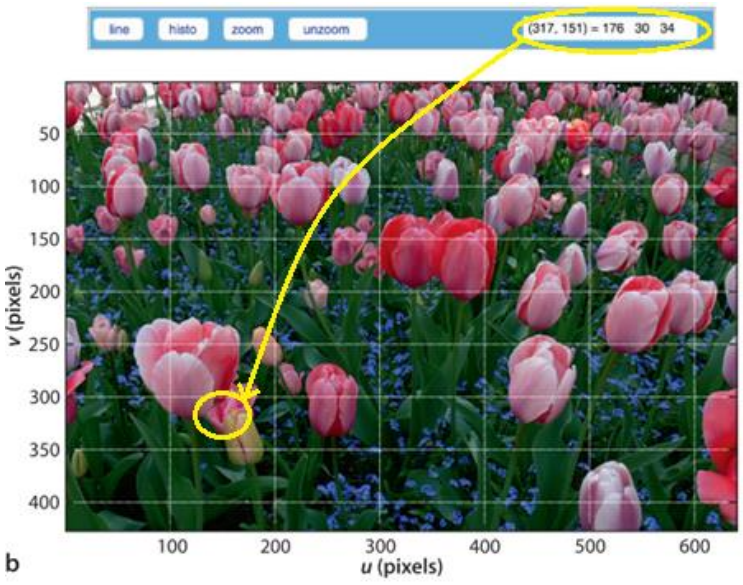
Figur 10 Gråskala bilde, [1]

Figur 10 viser en matrise med dimensjon 851 x 1280 med verdier fra 0 til 255. I punkt (335, 158) er intensiteten 202 som vist på høyre side. [1, pp. 359 - 363]

4.2.3.1 Fargebilder

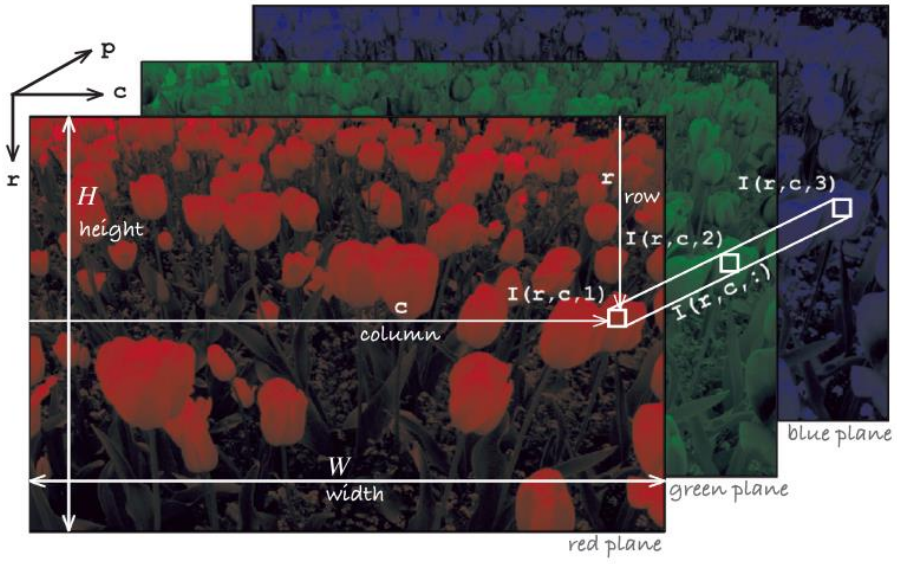
Gråskalerte bilder består bare av nyanser av gråtoner, mens fargebilder er bygd opp av RGB fargesystem. Her er bildet representert med 3 kanaler/plan av primærfargene: Rødt, Grønt og Blått. Disse blir lagt over hverandre og som med gråskala bilde, representerer verdiene: 0 – 255 intensiteten til fargene rødt, grønt og blått i hvert sitt plan. Sammen vil de lage et komplett bilde.

En kan se på Figur 11 at punktet (317,151) satt sammen av rødt: 176, grønt: 30 og blått: 34.



Figur 11- Farge bilde

Figur 12 viser de forskjellige farge-planene som blir brukt til å lage et fargebilde, der blir intensiteten av fargene lagt sammen i planene for hver piksel og vil da gi ut bildet i Figur 11. [1, pp. 359 - 363]

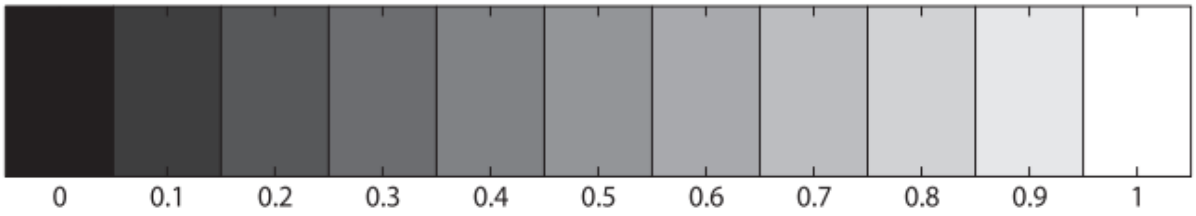


Figur 12 Farge bilde i 3 dimensjoner, [1]

4.2.3.2 Fra fargebilde til gråskala

En måte å omgjøre et bilde til gråskalaverdier, er å bruke prinsippet om fotometri eller kolorimetri som er et bredere spekter. En tar da i bruk lysstyrken i fargene og omgjør den til en gråtone med samme lysstyrke. Denne metoden vil da gjøre at begge bildene vil ha samme absolutte intensitet og blir målt i SI-enheten cd/m^2(candelas per kvadratmeter*)*.

Når en gjør bildet om til gråskala, så returneres det et lineært bilde der gråskalaverdiene er proporsjonale til lysstyrken i det originale bilde.



Figur 13 Lineær intensitetskala, [1]

Her blir fargene omgjort i henhold til intensitet der svart er svakest 0 og hvit er sterkest som blir 1, vist i Figur 13. [1, pp. 287 - 317]

4.3 «Feature» deteksjon

For å kunne gjenkjenne et objekt fra et bilde til et annet på en effektiv måte må det kunne detekteres distinkte trekk eller sammensetninger av piksler i objektet. Disse kalles «features». Ved å lokalisere dem, kan en enklere finne igjen de interessante objektene i bildet. Det er to hovedmåter å finne «features» på. Den første metoden er å finne en «feature» som lar seg nøyaktig bli sporet eller «tracket» ved bruk av en lokal søketeknikk som f.eks. korrelasjonsfiltrering. Den andre metoden brukes ved å individuelt detektere «features» i alle bildene for å så sammenligne de mot hverandre. Den første metoden egner seg på hurtige og nærliggende motiver som i en video, mens den andre metoden egner seg best på større motiver med lengre avstand som f.eks. når en skal sette sammen panoramabilder.

For å kunne finne en «feature» bør den inneholde følgende:

- Veldefinert posisjon i bildet.
- Stor kontrast/gradient i minst to forskjellige retninger.
- Repeterbarhet, det skal være lett å finne det igjen i neste bilde.
- Distinkt sammensetting av piksler som er avhengig av området rundt, men ikke for stort område.

[5, p. 183]

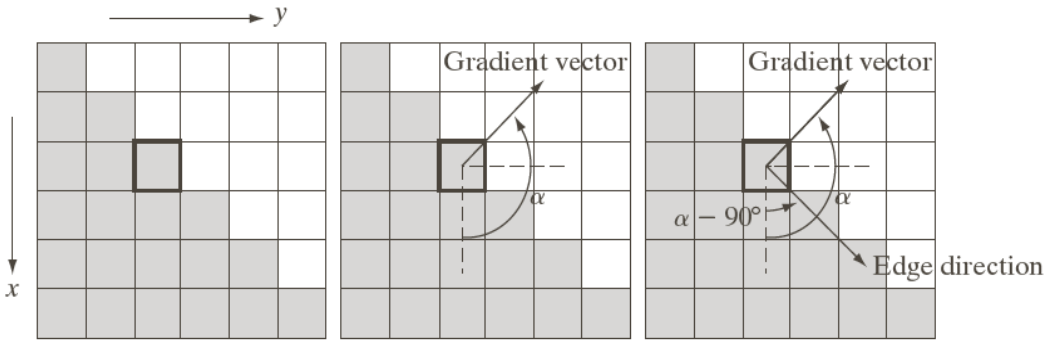
4.3.1 Gradient-operatorer

Den deriverte av en kontinuerlig funksjon er definert som:

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Et digitalt bilde er en diskret funksjon. Den deriverte er ikke definert for diskrete funksjoner, men en kan tilnærme den ved å la $h \geq 1$. I denne situasjonen er h pikselindeksen, som gjør at den ikke kan gå lavere enn 1 uten å bli 0. Det er mulig å finne gradienten ved å bruke de partiell-deriverte. I tillegg kan en finne retningen på gradienten samt magnituden som vist i Figur 14 nedenfor:

$$\nabla f = grad(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix}, \quad Retning = \alpha(x,y) = \tan^{-1}\left[\frac{g_y}{g_x}\right], \quad Magnitude = M(x,y) = \sqrt{g_x^2 + g_y^2}$$



Figur 14 diskret gradient [6]

For å tilnærme disse gradient-komponentene brukes da konvolusjon-filtre, en for hver komponent:

$$\frac{\partial f}{\partial x} = f(x+1, y) - f(x, y) \qquad \text{og} \qquad \frac{\partial f}{\partial y} = f(x, y+1) - f(x, y)$$

Dette gir følgende masker for å detektere horisontale og vertikale masker: $[-1 \quad 1]$ og $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$. Disse kan da føres over pikslene i bildet for å filtrere/detektere kanter. Noen populære tilnærminger for kantdeteksjon som er utvidet for å være mindre følsomme for støy er:

- **Prewitt:**

  Horisontal: $\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ Vertikal: $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ Diagonal: $\begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$ og $\begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$

- **Sobel:**

  Horisontal: $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$ Vertikal: $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ Diagonal: $\begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$ og $\begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$

4.3.1.1 Laplace-operator

Laplace-operatoren bruker den andrederiverte i masken for å detektere kanter. I motsetning til Sobel (1. deriverte) som gir én bred kant, gir Laplace to tynne kanter. Den er enda mer følsom for støy sammenlignet med Sobel. Det er derfor vanlig å bruke et Gauss-filter sammen med gradientoperatoren for å gjøre den mer robust og kalles da «LoG» («Laplace of Gaussian»).

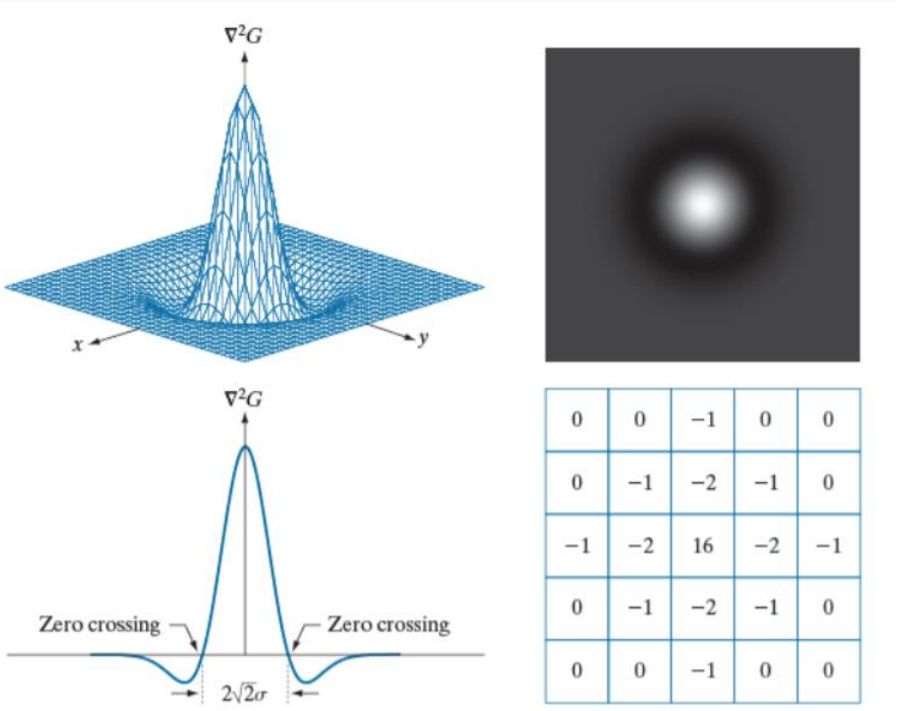
$$\text{Laplace tilnærming: } -\nabla^2 f = \frac{\partial^2 f}{\partial x^2} - \frac{\partial^2 f}{\partial y^2} \approx -f(i-1,j) + 2f(i,j) - f(i+1,j)$$

$$-f(i,j-1,j) + 2f(i,j) - f(i,j+1)$$

Dette kan da brukes ved å konvolvere med denne masken:

Horisontalt og vertikale kanter: $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

Diagonalt: $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$, dette filteret blir også brukt som punkt-deteksjon eller kontrastøkning.



Figur 15 LoG filter [7, p. 726]

$$LoG = -\nabla^2 G = \frac{1}{2\pi\sigma^4}\left(2 - \frac{x^2 + y^2}{\sigma^2}\right) e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Etter konvolvering med masken brukes nullkryssing for å identifisere kanter. I Figur 15 ovenfor vises en grafisk tilnærming av «LoG»-filteret og hvordan en 5x5 maske vil se ut.

4.3.2 Kantdeteksjon

Ved bruk av kantdeteksjon er en ute etter å finne betydelige endring i pikslers intensitetsverdi. Mye av informasjonen i et bilde kan finnes ved kantene til objekter/figurer. Det kan også brukes til å skjerpe ett bilde. Det skilles mellom horisontale, vertikale og diagonale kantlinjer. Kantdeteksjon inneholder stort sett følgende steg:

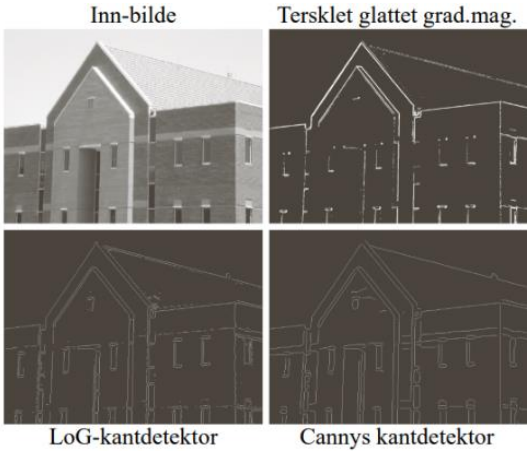
- **Støyreduksjon:** Kant deteksjon er følsom for støy. Derfor kreves det at en først må prosessere bildet med ett lavpassfilter. Målet er å fjerne så mye støy som mulig uten å glatte kantene for mye. Det brukes da ofte ett Gauss-filter.
- **Kantfiltrering:** Finner kandidater for kantpiksler ved hjelp av en lokal operasjon (konvolusjon). Her anvendes ofte en gradientoperatorer.
- **Kantlokalisering:** Prøver å finne de ekte kantpikslene ut fra de gitte kandidatene. Høypassfiltrering.

4.3.2.1 Canny edge

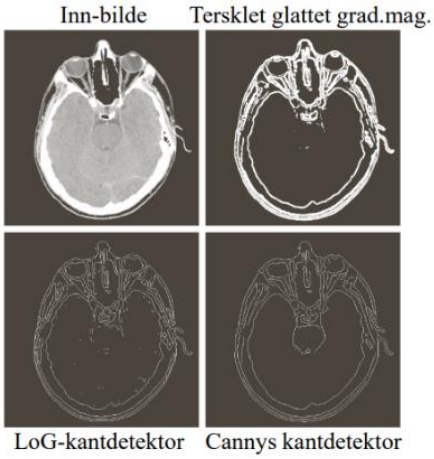
«Canny edge» en avansert algoritme for kantdeteksjon som bruker Sobel i et av sine flere steg. Den er mindre følsom for støy enn Sobel alene og er kanskje den algoritmen som er mest brukt for kantdeteksjon i dag. Figur 16 og Figur 17 viser forskjell på resultat av «LoG» og «Canny edge»-deteksjon. [7, p. 729]

«Canny edge» prosseseringsteg av et bilde:

1. Støyreduksjon: Lavpassfilter med 2D Gauss-filtre.
2. Finn gradient-magnituden og gradient-retningen.
3. Tynning av gradient-magnitude ortogonalt på kant.
4. Hysterese-terskling.



Figur 17 LoG mot Canny kantdetektor [7, p. 733]



Figur 16 LoG mot Canny kantdetektor [7, p. 734]

4.3.3 Hjørnedeteksjon

Hjørner er sentrale deler i oppbygningen av et bilde og de definerer ofte objekter eller figurer. Et hjørne av et objekt blir ofte brukt som et interessepunkt som har bratte gradienter i ortogonale retninger. De skiller seg ut som en sammenføying av to kanter. Disse punktene er særegne punkter som skiller seg ut. De er lettere å finne igjen fra en annen vinkel av samme bilde. Denne type deteksjon blir ofte viktige når en bruker «multi-view» tekniker som stereo og bevegelsesestimering. [1]

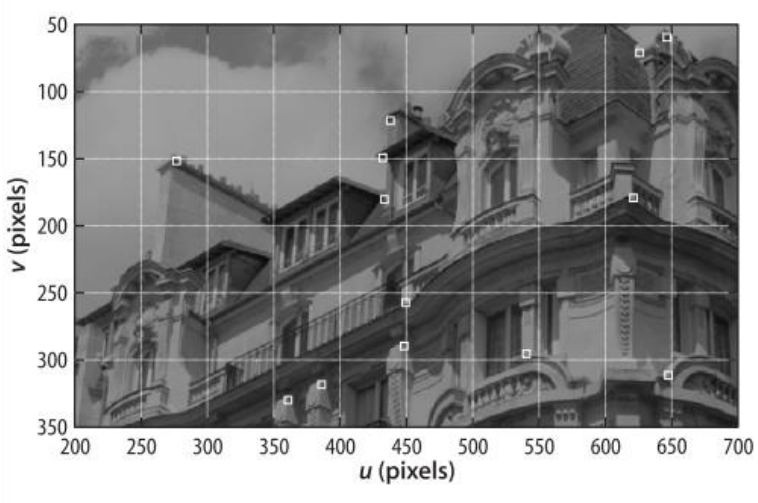
4.3.3.1 Harris

Harris er en mye brukt algoritme for hjørnedeteksjon som ble utviklet av Chriss Harris og Mike Stephens i 1988. Algoritmen skyver et vindu i form av en matrise (Kernel) over alle pikslene i bildet. Den gir så ut en verdi i forhold til hvor mye innholdet har endret seg, som vurderes av en satt terskel for å si om dette er et hjørne. Den er ikke påvirket av rotering, translasjon eller lysforhold. I Figur 18 nedenfor viser resultatet av en typisk hjørnedeteksjon med Harris-algoritmen.

Steg:

- Konverter bilde til gråskala.
- Gauss-filtrering for å redusere støy.
- Bruker Sobel operator for å finne gradientverdier.
- Finner Harris verdi.
- Terskel vurderer verdi for definering om det er interessant særtrekk/hjørne.

[8, p. 135]



Figur 18 Harris kantdeteksjon [1]

4.4 Objektdeteksjon

4.4.1 SIFT– Scale Invariant Feature Transform

SIFT ble utviklet av D. Lower i 2004. Dette er en populær algoritme for å gjenkjenne «features» i bildet. Den lar seg ikke påvirke av rotering, translasjon eller skalering av bildet, som kan være et problem med noen av disse algoritmene. Den er i tillegg robust i forhold til forandringer i lysforhold, støy og små endringer i synsvinkel. Ulempen er at den er regnemessig tung å utføre. Algoritmen er patentert og er derfor ikke tilgjengelig i OpenCV som er åpenkilde.

SIFT «features» finnes ved å beregne gradienten i en 16x16 maske rundt det detekterte nøkkelpunktet ved å bruke den tilhørende Gauss-pyramidegraden hvor hver «feature» ble funnet. Gradientmagnituden er nedskalert ved å bruke en Gauss-«fall out»-funksjon. Noe som reduserer hvor stor påvirkning gradientene som er langt fra senter av en «feature» har. For hver 4x4 kvadrant blir det etablert et «gradientorientert histogram» hvor det legges til den vektlagte gradientverdien. [8, p. 139]

4.4.2 ORB- Oriented FAST and Rotated BRIEF

ORB metoden ble utviklet på OpenCV Labs av Ethan Rublee, Vincent Rabaud, Kurt Konolige og Gary R. Bradiski i 2011, som et alternativ til SIFT som er patentert. Denne algoritmen bruker teknikker fra «FAST keypoint detector», «BRIEF keypoint descriptor» og til slutt «Brute-force matching».

- **FAST** (Features from Accelerated Segment Test) analyserer sirkulært 16 nærliggende piksler. Den markerer disse som lysere eller mørkere i forhold til en terskel som er definert av pikselen i senter. Den kan da kontrollere om sirkelen inneholder ett hjørne på en effektiv måte.
- **BRIEF** (Binary Robust Independent Elemantary Features) er en effektiv men avansert algoritme for å beskrive og gjenkjenne særtrekk i ett bilde.
- **Brute-force matching** er en «descriptor matcher» som sammenligner to sett med nøkkelpunktbeskrivelser og skaper et resultat som er en liste med treff. [8, p. 143]

4.4.3 HOG – History of Oriented Gradients

HOG er en viktig variant av SIFT. Det er en unik teknikk som teller tilfeller av gradientorientering i et spesifikt område på bildet. Den blir ofte brukt til gjenkjenning av mennesker i et bilde. Den interne mekanismen i HOG deler først opp bildet i celler, hvor den da regner ut gradienten på hver celle. Til sammen utgjør disse gradientene et histogram for hver celle. Dette vil være en mer effektiv måte å gjenkjenne en større del av et bilde kontra SIFT som konsentrerer seg på «features». [8, p. 170]

Med HOG så kan en gjenopprette kontrastinformasjon ved å telle gradientens orienteringer som blir vektet, der den reflekterer hvor betydningsfull en gradient er i forhold til omliggende gradienter i samme celle. Dette vil si at istedenfor å normalisere alle omliggende gradient-bidragsytere, så blir de bare normalisert med hensyn til nærliggende gradienter. Normaliseringen kan forekomme i et nett av celler som er forskjellige i forhold til orienteringssubnettet. En kan si at en enkel gradients posisjon kan bidra til flere forskjellige histogram, som er normalisert på forskjellige måter. Dette vil bety at en vil sannsynligvis ikke miste noen kanter som kan ha lave kontrastforhold. [4, pp. 189 - 190]

4.5 Objekt-«tracking»

Når en skal følge et objekt fra en bildestrøm eller video, så må det lagres en del informasjon om objektet fra første bilde, som informasjon om hvordan objektet ser ut og posisjonen til objektet i bildet. Ved å sammenligne plasseringen fra første bildet med det neste bildet, kan det dermed regnes ut hvilken hastighet og retning objektet har i form av antall piksler pr. tidsenhet. Det er da mulig å forutse hvor objektet kommer til å være i neste bilde. Dermed er det ikke nødvendig å søke gjennom hele bildet etter objektet, men lokalt rundt den estimerte posisjonen objektet trolig vil oppnå. Derfor vil prosessen med å «tracke» ett objekt foregå hurtigere enn ved objektgjenkjenning.

Problemer som kan oppstå ved objekt-«tracking» er f.eks. hvis objektet forsvinner bak en hindring eller forflytter seg for raskt slik at algoritmen mister objektet. Det hender også at algoritmen akkumulerer feil for posisjonen til objektet som gjør at avgrensingsboksen vil «drifte» bort fra objektet. Ikke alle algoritmer klarer å håndtere disse problemene like bra. Derfor finnes det et stort antall og versjoner av algoritmer for «tracking». Det er et fagfelt som er under rask utvikling. I «computer vision»-biblioteket til OpenCV er det tilgjengelig noen av de mest brukte typene av algoritmer for «tracking», hvor alle har sine styrker og svakheter.

4.5.1 Boosting:

AdaBoost (adapting boosting) ble først skapt av Yoav Freund og Robert Schpire i 1996. Boosting algoritmen til OpenCV er basert på en «online» versjon av denne AdaBoost-algortimen. Boosting klassifiserer positive og negative posisjoner av objektet i sanntid. Brukeren markerer interesseområdet eller har en algoritme for gjenkjenning av objektet som skal følges. Denne blir gitt en positiv verdi og bakgrunnen blir gitt negativ verdi. Ved neste bilde vil algoritmen klassifisere alle nærliggende piksler og gi dem en score. Den neste posisjonen til objektet er der hvor scoren er høyest (positiv). [9]

4.5.2 MIL (Multiple Instance Learning):

MIL bruker samme metode som Boosting. Men der Boosting bare har nåværende plassering av objektet som positiv verdi ser MIL på pikslene rundt objektet og danner flere mulige positive eksempler. En annen ulikhet er at i stedet for å spesifisere positive og negative punkt bruker MIL positive og negative grupper som inneholder små utsnitt av bildet i et samlet område, der gruppen må

minst ha et positivt utsnitt for å få positiv verdi. Det vil si at f.eks. når et interesseområde er valgt av bruker, vil algoritmen lage mange mindre rektangler med utsnitt fra dette området og legge de i en gruppe. I neste bilde vil algoritmen sammenligne den forrige gruppen med nye grupper fra dette bilde. Der den med høyest positiv verdi er der hvor det er størst sjanse for at det nye interesseområdet er. Dette gjør «trackeren» mer fleksibel og dermed vil den lettere kunne finne interesseområdet i neste bilde. [10]

4.5.3 KCF (Kernelized Correlation Filter)

KCF bygger på samme prinsipp som MIL, men KCF-algoritmen utnytter egenskapene til sirkulære matriser til å forbedre sporingshastigheten i forhold til mer tradisjonelle metoder som f.eks. Boosting. Filteret til KCF blir vektet opp mot interesseområdet, men også områder rundt. Dette gjør at i neste bilde kan KCF raskt finne neste posisjon av interesseområdet, siden det er størst sannsynlighet for at den vil være nærliggende forrige interesseområde. Samtidig blir alle mulige posisjoner vektlagt mot forrige posisjon. KCF samler så nærliggende mulige posisjoner i grupper som gjør algoritmen raskere. Posisjonen med høyeste verdi vil så bli valgt som neste interesseområde. [11]

4.5.4 CSRT (Channel and Spatial Reliability Tracking)

CSRT er OpenCV sin implementasjon av CSR-DCF (Channel and Spatial Reliability of Discriminative Correlation Filter). Det er en avansert algoritme som ved bruk av «spatial reliability map» kan endre filterstøtten til den delen av objektet som best er egnet for «tracking». Dette tillater algoritmen å øke søkeområdet og forbedrer sporingen av ikke-rektangulære objekter. Den tar i bruk metoder som «HOG» (4.4.3) og «Color names» (CN). [12]

4.5.5 TLD (Tracking, learning and detection):

Som i navnet er denne algoritmen delt inn i tre deler. «Tracking»: følger objektet fra bilde til bilde. «Detection»: detektor som lagrer hvor i bilde objektet opptrer og korrigerer «trackeren» hvis nødvendig. «Learning»: estimerer detektorens feil og oppdaterer den for å unngå disse. [13]

4.5.6 MOSSE (Minimum output sum of squared error):

MOSSE er en algoritme som ligner på ASEF (Average of Synthetic Exact Filters). ASEF introduserte en metode for å justere filtre for en spesiell oppgave. De tidligere metodene spesifiserte en «peak»-verdi mens ASEF-filter spesifisere hele korrelasjonsutgangen for hvert treningsbilde. MOSSE er en algoritme med et korrelasjonsfilter som er raskt og kan tilpasse seg blokkering og rotasjon av objektet. Den er også god på variasjoner i lysforhold, skalering og variasjon i positur. Når objektet blir helt eller delvis blokkert kan algoritmen fastslå status på «tracking» av objektet og oppdatere parameterne etter PSR- (Peak to sidelobe ratio) verdien. PSR måler styrken på korrelasjonen, kan detektere blokkerings- og «tracking»-feil. MOSSE trenger også færre treningsbilder enn ASEF. [14]

4.5.7 GOTURN (Generic Object Tracking Using Regression Networks):

GOTURN er en «tracking»-algoritme som er basert på dyplæring og CNN (konvolusjonalt nevralt nettverk). CNN er en metode som reduserer mengden treningsbilder som trengs. GOTURN har alle fordelene med CNN, men er mye raskere på grunn av «offline»-læring uten den finjusteringen som trengs med «online»-læring der algoritmen skal lære å gjenkjenne objektet i sanntid. GOTURN algoritmen er allerede trent på tusenvis av forskjellige video sekvenser derfor er den allerede ganske god på å forutse lokasjonen på objektet som skal følges fra bilde til bilde. [15, pp. 774-790]

4.5.8 Median Flow:

MedianFlow er basert på Lucas-Kanade sin «optical flow» «tracking» algoritme. Den bruker «Forward-Backward Error» og normalisert krysskorrelasjon til feedback. Lucas-Kanade algoritmen er en effektiv og mye brukt metode til å estimere bevegelse («optical flow») i utvalgte interessepunkt i et bilde. Dette gjør den ved å få en bevegelsesvektor (u, v) ved å sammenligne den på de to påfølgende bildene. For å få til dette antar Lucas-Kanade metoden to ting: At tiden (t) mellom to interessepunkt er konstant og at bildet inneholder en naturlig scene som inneholder objekter med forskjellige piksel intensitet. [16]

4.6 Objektgjenkjenning

For å detektere et objekt automatisk og gjenkjenne det for hva det er, brukes objektgjenkjenning. Hvordan et objekt ser ut, kan variere mye fra et bilde til et annet grunnet bevegelse og lysforhold. For å gjenkjenne et objekt må en først klare å distinktere det fra bakgrunnen som et eget objekt. Objektet må så klassifiseres ved å sammenligne særtrekk mot en klasse av kjente karakteristiske trekk. Det finnes ferdige utlærte klasser («offline») og klasser som kan læres direkte («online»). En «offline» klasse trenger kanskje tusenvis av eksempler av det kjente objektet for å være brukbar. En slik klasse læres opp ved å kjøre tusenvis av bilder gjennom en maskinlæringsalgoritme av et eller flere kjente objekter som da blir merket som positivt. Og like mange bilder uten det kjente objektet som blir merket negativt. Hvor nøyaktig klassifiseringen blir er avhengig av antall eksempler den blir matet med. Ved bruk av denne metoden kan en kontinuerlig få ut hvor sikker algoritmen er på at objektet som gjenkjennes er klassifisert rett. En klassifisering som utføres direkte kan læres av bare få eksempler hvor den så lærer seg selv videre. Dette vil være en mindre nøyaktig modell, men kan brukes til for f.eks. å gjøre en «tracker» mer tilpasningsdyktig.

Figur 19 – Objektgjenkjenning av en Iphone.

Figur 19 viser et forsøk gruppen gjorde av objektgjenkjenning der bildet blir prosessert mot en «offline» klasse som er ferdig opplært med bare 40 bilder av en Iphone. Metoden som ble brukt er YOLO, (4.7.4). Den klarer å gjenkjenne at det er en telefon med 66% sikkerhet. Dette er ganske bra med tanke på hvor få bilder som ble brukt, men den er da avhengig av at forholdene hvor testbildene ble tatt og hvor telefonen blir detektert er nok så like.

4.7 Maskinlæring

Målet med maskinlæring er å lære programmer noe de ikke kan fra før til å ta rette(re) beslutninger og forbedre automatikk ved hjelp av input i form av data over tid. Dette kan gjøres med en mengde forskjellige strategier, metoder og algoritmer innenfor maskinlæring. Maskinlæring er brukt i en mengde fagfelt og industrier som:

- Personalisering av reklame og søkemonitorer på internett.
- Programvare for detektering av unormal innlogging og transaksjoner i banker.
- Detektering av føflekkreft.
- Analyse av hjerneaktivitet for å bedømme om en pasient vil våkne fra koma.

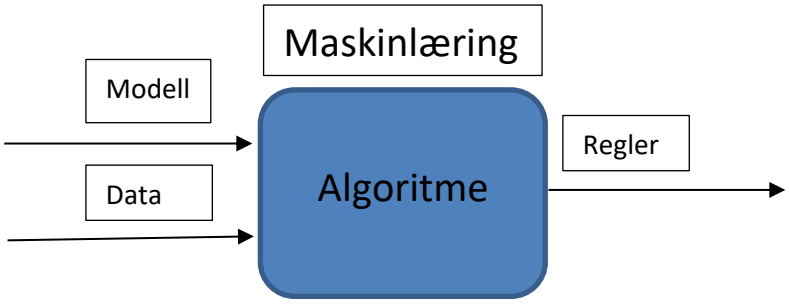
Maskinlæring spiller en stor rolle i autonome kjøretøy og roboter, som i en dynamisk verden må kunne ta egne avgjørelser på hvordan de skal reagere på input fra mange forskjellige kilder. [17, pp. 10-13]

Maskinlæring kan grovt sett, deles opp i læringsmetodene nedenfor:

- Overvåket læring
  - Maskinlæringen skjer med en forhåndslagd modell med eksempler av input og ønsket output.
- Uovervåket læring
  - Maskinlæringen skjer uten noe forhåndslagd modell og algoritmen må selv klare å finne en struktur på input og output.
- Forsterkende læring
  - Maskinlæringen foregår ved at algoritmen blir «belønnet» eller «straffet» for output som gjør at den kommer nærmere eller lengre bort fra ønsket output.

[8, pp. 290-291]

Maskinlæringsmetodene kan også være en blanding av disse «grunnmetodene» og målet til alle er å få laget et sett med «regler» som kan hjelpe algoritmen å nå en bestemt output, som vist i Figur 20 under.

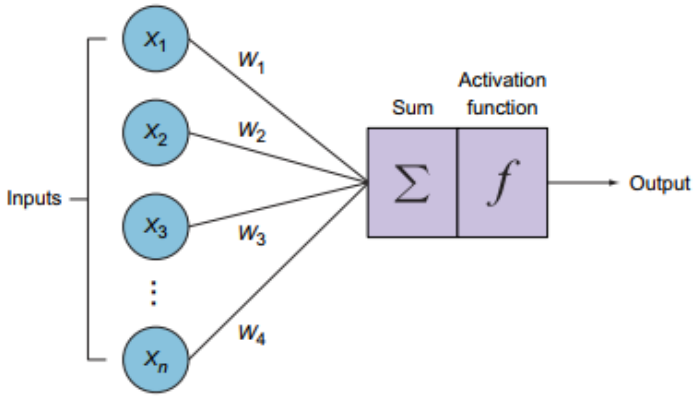


Figur 20 Overvåket maskinlæring

4.7.1 Artificial neural network (ANN)

Kunstige nevrale nettverk (ANN) er inspirert av biologisk nevralt nettverk, men med en forenklet modell. Den består av nevroner (noder) hvor hver nevron utfører en binærisk klassifisering. Hvert nevron får inn en input og produserer et enkelt output. Når et nevron får input fra et eller flere nevron blir det kalt for et «perceptron». Hver input et «perceptron» får fra hvert nevron, har en vekt assosiert

som sier hvor viktig den er for nevronets aktiveringsfunksjon. Aktiveringsfunksjonen prosesserer summen av påtrykket og en terskelfunksjon (Bias) blir brukt på aktiveringsfunksjonen for å gjøre den om til en binær klassifisering (0 eller 1). [8, pp. 291-292] [17, pp. 36-41]



Figur 21 Illustrasjon av input med tilhørende vekter, som blir sett inn i et nevron og summert [17, p. 40]

Sum av vekt (Figur 10):

$$z = \sum x_i \cdot w_i + b(bias) \qquad z = x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 + \cdots + x_n \cdot w_n + b$$

[17, p. 40]

4.7.2 OpenCV DNN – Deep Neural Network

OpenCV DNN [8] ble designet for at det skulle være enkelt å integrere eksisterende dyplæringsmodeller som allerede var opplært. Derfor ble denne arkitektur lagt til som en modul i OpenCV. Det er derfor mulig å finne ferdige modeller innenfor dyplæring for bildeklassifisering, objektgjenkjenning og bildesegmentering.

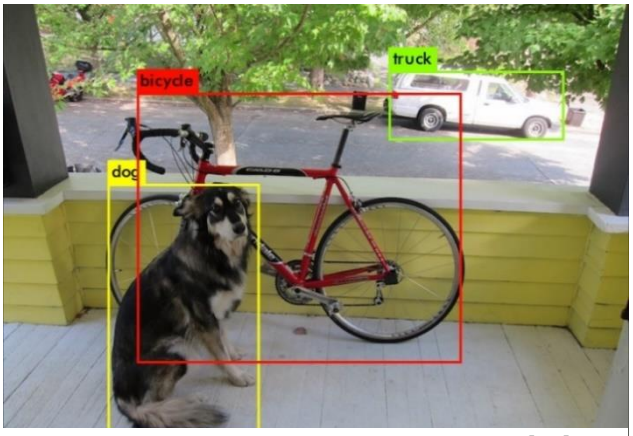
4.7.3 Maskinlæringsmodell

En maskinlæringsmodell er en fil med et sett av «regler» som kan bli brukt til å gjenkjenne et eller flere objekter. Modellen blir lagd/trent ved å behandle et sett med data for å gjenkjenne visse likheter og sammenhenger. En av flere alternativer til disse filtypene som er støttet av OpenCV, er ONNX (Open Neural Network Exchange).

ONNX er et åpent format til en kunstig intelligens modell for dyplæring og maskinlæring. ONNX ble utviklet av Microsoft i samarbeid med Facebook i 2017, der målet var å sammen lage en modell som kunne brukes på tvers av eksisterende og fremtidig rammeverk samt økosystem for kunstig intelligens [18].

4.7.4 YOLO – You Only Look Once

YOLO er objektgjenkjenning i sanntid. Dette er en av mange algoritmer som bruker ett-stegs nevrale nettverk for å detektere objekter i et bilde for så å klassifisere dem mot en database. Den markerer så alle klassifiserte objekter i bildet med hvert sitt rektangel. Lignende ett-stegs algoritmer er SSD (Single-Shot Multi-box Detection). Andre algoritmer for objektgjenkjenning er to-stegs typen R-CNN (Region-Based Convolutional Neural Network) som finnes i flere varianter. Det som skiller YOLO ut i forhold til disse er at den er mye raskere å prosessere samtidig som den kan være like nøyaktig i mange sammenhenger.



Figur 22 Objektgjenkjenning ved bruk av YOLO [19]

Ulempen er at den kan prestere dårlig i forhold til R-CNN når det skal detekteres små objekter som er plassert tett sammen. [19]

YOLO-algoritmen er avhengig av en modell som er opplært av testbilder fra objektene den skal detektere. Modellen blir lagret som en «vekt-fil» som algoritmen bruker for å klassifisere objektene mot i hvert bilde.

4.7.5 PyTorch – Neural network

Pytorch er et GPU-akselerert maskinlæringsrammeverk som tar i bruk tensorer. Den er utviklet av Facebook som åpen kildekode til bruk innen «computer vision» og naturlig språkprossesering. Det er hovedsakelig utviklet til Python, men har også et «C++»-brukergrensesnitt. Hovedfunksjonen til PyTorch er å ta i bruk regnekraften til en GPU for å raskere lære opp det nevrale nettverket til å utføre de oppgavene den blir satt til. Fordelen med en GPU i forhold til en CPU er at den kan kjøre mange parallelle prosesser samtidig, noe som er ideelt for et nevralt nettverk hvor det blir behandlet store mengder data. Funksjonaliteten kan lett utvides med å integrere en rekke kjente Python bibliotek for å håndtere matematiske funksjoner, vitenskapelig databehandling og gi C-lignende ytelse som NumPy, SciPy og Cython. [20]

5 Analyse av problemet

5.1.1 Vertikal og horisontal forflytning

For å kunne estimere forflytningen av et objekt i et bilde må en først skille ut et område med piksler som kan defineres som et objekt av interesse. Etterpå kan det brukes en algoritme for å gjenkjenne dette objektet i neste bildet i video/bilde-strømmen. Som gjennomgått i avsnitt 4, finnes det mange forskjellige typer algoritmer og prosesser for å utføre disse operasjonene. En del av problemløsningen blir da å få en oversikt over de forskjellige metodene som finnes og hvordan de kan brukes til å løse problemstillingen på en god måte.

5.1.2 Estimering av forflytning i bildets dybderetning

Ønsket fra bedriften er at estimering av posisjonsendring skal foregå gjennom et enkelt kamera. Problemet er da at det ikke finnes en naturlig invers fra et 2-dimensjonalt-bilde til en 3-dimensjonal tolkning av det som blir observert i et bilde. Innenfor vanlig «computer vision» blir det normalt brukt to kamera som gir stereo-syn eller et 3d-kamera for å kunne estimere dybde i bildet. Her må det da sees på om det finnes metoder eller algoritmer som kan klare å estimere en endring i avstand til objektet fra et 2D-bilde. Med tanke på at disse målingene skal brukes i en regulator er det mulig å forholde seg til forflytningen av objektet som differansen mellom kamera og et settpunkt.

5.2 Utforming av mulige løsninger

5.2.1 Felles løsning for alle løsningsforslag:

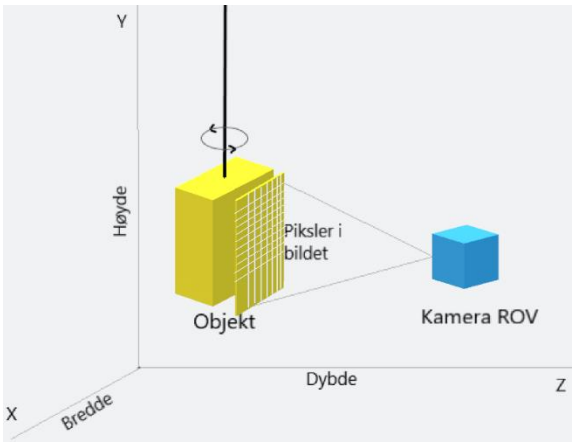
Etter samtale med veileder fra Kystdesign ble det enighet om at det skulle lages et grafisk brukergrensesnitt hvor det var mulig å bruke forskjellige typer «trackere». Dette pga. at det eksisterer flere ulike «tracking»-algoritmer som har forskjellige fordeler og ulemper. Fagfeltet innen «computer vision» og «tracking» er under rask utvikling, slik at det hele tiden kommer nye og forbedrede metoder å «tracke» på. Tanken med programmet er da å lett kunne sammenligne og velge den mest gunstige «trackeren» til formålet. Det vil også være lett å implementere nye «trackere» i programmet.

For at programmet skal fungere mot varierte kamera- og linse-typer kan det kjøres et kalibreringsprogram hvor en kan kalibrere output fra kamera mot en fastsatt testplakat for å detektere forvrengningen i bildet.

Ut fra programmet sendes en melding i form av UDP som beskriver endring av posisjon mellom objekt og valgt kamera i form av x, y og z som representerer forflytning i bredde, høyde og dybde i bildet. Verdiene vil representere målte pikselendringer fra senter av objektets startpunkt til hvor objektet befinner seg.

5.2.2 Løsningsalternativ 1 – Kombinering av OpenCV «trackere»

OpenCV har et bibliotek av tilgjengelige «trackere» som åpen kildekode. Disse kan brukes for å følge det interessante objektet i bildet. Det må først markeres i bildet hva som er det interessante objektet og hvor det befinner seg. Dette kalles interesseområdet. Det gjøres ved å dra en musepeker over objektet som da definerer det ved hjelp av et rektangel, hvor to av hjørnene i rektangelet er koordinater i form av pikselplassering i bildet. Ved å følge disse koordinatene vil det da være mulig å måle hvor mange piksler objektet har flyttet seg i bredde og høyde fra bilde til bilde.



Figur 23 - Objekt som blir nedsenket og følges av ROV

For å kunne måle en forflytning i dybderetning av bildet i dette alternativet, vil det med bakgrunn i den teoretiske modellen fra 4.1.1, måles endring i antall piksler som er innenfor rektangelet som definerer objektet. Et problem som kan oppstå da er hvis en følger ett svevende objekt (Figur 23) som f.eks. en modul som senkes ned i sjøen. Da vil størrelsen på objektet slik det blir observert i bildet kunne endre seg hvis det roterer rundt sin vertikale akse. Dette vil da kunne gi ett falskt inntrykk av at objektet har flyttet seg i dybderetning av bildet. For å ta hensyn til dette måles kun antall piksler i høyderetning av objektet slik at den i mindre grad blir påvirket av dens mulige rotering.

De inkluderte «trackerne» i OpenCV-biblioteket bygger på forskjellige bildeprosesseringsteknikker som har hver sine fordeler og ulemper. Ved bruk av objekt-«tracking» er det en risiko for at en feil-«tracker», mister objektet eller at interesseområdet som definerer objektet akkumulerer feil og

«drifter» av gårde. En kombinasjon av flere «trackere» som kjøres parallelt vil kunne sammenlignes mot hverandre og en har da mindre sannsynlighet for å miste objektet.

5.2.3 Løsningsalternativ 2 - Objektgjenkjenning

Ved å implementere en YOLO-modell med OpenCVs DNN-modul eller Pytorch kan programmet klare å gjenkjenne et objekt i hvert bilde i en bildestrøm, så lenge objektet er i kameraets synsfelt. Den vil også kunne finne objekt igjen hvis det kommer noe foran objektet eller at det forsvinner ut og kommer inn igjen i synsvinkelen til kameraet. Dette er fordi YOLO algoritmen «vet» hvordan objektet ser ut og derfor kan gjenkjenne det uansett hvor det kommer inn og befinner seg i bildet. For å kunne gjenkjenne objektet, må det som forklart i avsnitt 4.7.4, lages en modell med treningsbilder av objektene som algoritmen skal gjenkjenne. Programmet vil gi en usikkerhet fra 0.0 – 1 på deteksjonen av objektet den «tracker». Dette kan implementeres i programmet til å gi varsel og feildeteksjon til operatør ved en operasjon der det er mange lignende objekter i bilde.

5.2.4 Løsningsalternativ 3 – Estimert dybdeforhold i bildet med maskinlæring

Ett sentralt problem for oppgaven er å estimere et dybdeforhold i kamerabildet. En måte å løse dette på er å lage en maskinlæringsmodell som bruker bilder fra et stereokamera eller 3D-kamera og sammenligner disse mot et vanlig monokulært kamera fra samme perspektiv. En bruker da dybdekartet som blir produsert til hvert bilde og lærer opp en modell som kan estimere ett dybdeforhold i bildet mellom objekter og bakgrunnen.

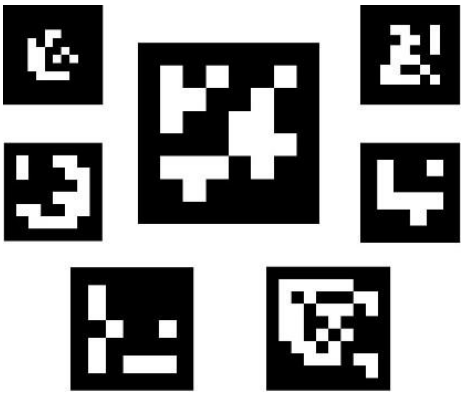
Ved testing av denne metoden ble det valgt å bruke en modell kalt «Midas», denne bruker en kombinasjon av flere forskjellige 3D-datasett som læringsmateriale for en mest mulig robust og allsidig modell.

En kan se på Figur 24 at output ved bruk av Midas gir ett dybdeforhold som vises i en fargeskala i pikslene som definerer om objektet er nær kameraet eller langt borte. Ideen med dette alternativet er å bruke dybdeforholdet til å estimere hvor nært kamera på ROV-en objektet som «trackes» er. Dette kan da brukes til å regulere forsterkningen i regulatoren for ROV-en.

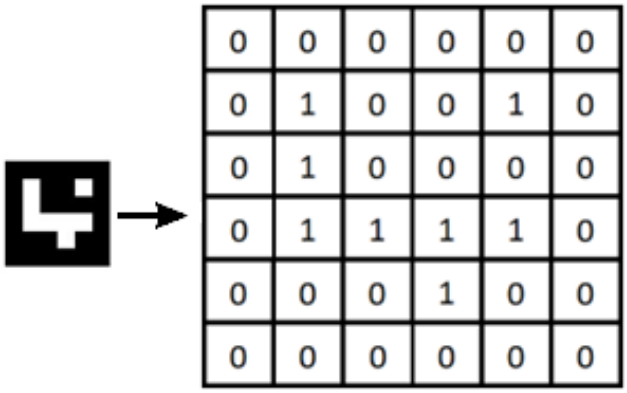
Figur 24 - [21] Output ved bruk av Midas

5.2.5 Løsningsalternativ 4 - Aruco

Denne metoden har et bibliotek med åpen kildekode for OpenCV som ble laget av forskningsgruppen «Application of artificial vision» på «University of Cordoba» i Spania. Den detekterer firkantede «fiducial»-markører i bildet ved å bruke et kamera. Så lenge de er synlig i bildet kan en regne ut distansen og rotering som kameraet har i forhold til markøren. Siden markørens dimensjoner er gitt kan også fysisk lengde i bildet bli regnet ut.

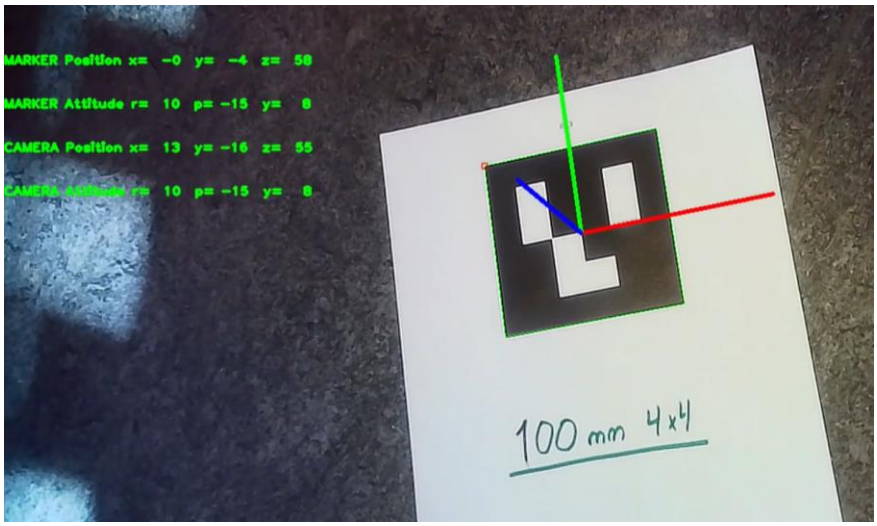


Figur 26 - Eksempler på Aruco markører [22]



Figur 25- Aruco Tag omgjort til Binær verdier

Hver markør er en vektor av fire punkt som er representert av hjørnene til markøren i bildet med et binært symbol som illustrert i Figur 26. Hver markør har en unik identitet som varierer i bit-størrelse. Hvert element i matrisen som representerer en markør, har verdien 1 hvis ruten er hvit og 0 hvis ruten er svart, illustrert i Figur 25. Translasjonen og rotasjonen av markøren er i forhold til senter av markøren og kamera. For å detektere en markør vil Aruco-programmet først se etter firkanter. Når den har funnet en eller flere firkanter vil den prøve å lese den binære koden som eventuelt er til stedet. Har firkanten en binær verdi er den en markør og programmet kan da gi ut en nøyaktig posisjon av markøren i forhold til kameraet. En markør med en mindre bit-verdi har lavere oppløsning på symbolet og er enklere å detektere for kameraet, men vanskeligere å skille fra andre markører. En markør med høyere bit-verdi er mer avansert og derfor lettere å skille fra andre, men til gjengjeld mer krevende for algoritmen å detektere. Figur 27 illustrerer hvordan kameraet fanger opp en Aruco-markør. [22]



Figur 27 - Aruco test

5.2.6 Software

5.2.6.1 OpenCV – Open Source Computer Vision

OpenCV har en åpen kildekode. Det betyr at det er et gratis rammeverk for alle å laste ned og videreutvikle på. Designere bruker ofte åpen kildekode som en måte å skape et utvikler-samfunn og høste tilbakemeldinger fra brukere over hele verden. OpenCV biblioteket har algoritmer og funksjoner som kan hjelpe til å lage programvare som gir datamaskinen «syn». Dette betyr at datamaskinen sammen med et kamera vil være i stand til å behandle og dels tolke bilder.

5.2.6.2 Python

Python er et programmeringsspråk som OpenCV kan brukes i. Python er et av de mest populære objektorienterte programmeringsspråk i utviklermiljøet. Det er mye brukt sammen med OpenCV noe som gjenspeiler den store tilgangen til dokumentasjon og veiledninger for akkurat dette. Objektorienterte språk minimerer re-arbeid ved at du kan referere til funksjoner mer enn en gang i løpet av et program. Dette betyr at en programmerere ikke ville måtte skrive kode for å kunne utnytte en funksjon som er opprettet tidligere i dokumentet. Python er opphavsrettslig beskyttet, men det er også en del av et åpent kildekode-miljø.

5.2.6.3 Tkinter

Tkinter er bindeledd mellom Python og «Tk GUI (Graphical User Interface)-tookit» biblioteket. «Tk GUI» er et grafisk brukergrensesnitt og åpen kilde som blir brukt i mange programmeringsspråk. Sammen med Tcl (som er en kommandotolk og et skriptbasert språk) kan Tkinter kalle opp kommandoer fra Tk biblioteket og gjøre det mulig å lage et brukergrensesnitt i Python. Tkinter sin største fordel er at den er rask og kan lett legges til i Python biblioteket. Originalt er dokumentasjonen til Tkinter litt svak, men det er mye ny dokumentasjon og brukermanualer som er laget i ettertid. [23]

5.2.6.4 Visual Studio Code

Visual Studio Code er en kildekode editor utviklet av Microsoft for Windows, Mac og Linux operativsystemer. Det ble sluppet tilbake i april 2015 og støtter en rekke programmeringsspråk samt Python. Den har praktiske funksjoner for testing og feilsøking. Gruppen har tidligere erfaring med Visual Studio Code og vil derfor bruke dette programmet til å skrive programkode.

5.2.6.5 NVIDIA Cuda (Compute Unified Device Architecture)

Cuda er en parallell databehandlingsplattform som gir programvare for vanlig databehandling tilgang til resursene til en GPU i maskinen. Dette brukes for å kunne kjøre mer krevende maskinlæringsmodeller på grafikkortet som er mer egnet for den type beregninger og vil da øke ytelsen.

5.2.7 Hardware

5.2.7.1 PC

Under utvikling og testing vil gruppen bruke sine private PC-er. «Tracking»-algoritmene varier i systemkrav etter hvor avanserte de er, men de aller fleste vil kunne bli kjørt med en helt ordinær bærbar PC. Ved bruk av maskinlæringsmodeller kan det være behov for å ha en dedikert GPU for å kunne håndtere prosesseringskraften algoritmen trenger.

Systemspesifikasjon:

- Windows 10 Home 64-bit Edition
- Intel(R) Core (TM) i5-8250U CPU @ 1.60GHz /1.80 GHz, Quadcore
- 8 GB 1600MHz LPDDR3 SDRAM
- USB 3.0
- Dual-band, trådløst 802.11ac-nettverk
- NVIDIA Geforce MX150 2GB GPU

5.2.7.2 Raspberry Pi

For å kunne gjøre løsningen kompakt og mindre kostbar vil det kontrolleres om programvaren kan kjøres på en Raspberry Pi som er en lavkost mini-PC. Den har kraften som skal til for enkel bildeprosessering og har de tilkoblingsmuligheter som er nødvendige.

Systemspesifikasjon:

- 1,5 GHz, Quadcore 64-bits ARM Cortex-A72 CPU
- 4GB LPDDR4 SDRAM
- Gigabit Ethernet
- Dual-band, trådløst 802.11ac-nettverk
- To USB 3.0 og to USB 2.0-porter
- Støtte for to skjermer med oppløsninger på opptil 4K via Micro-HDMI
- 4K p60-maskinvaredekoding av video

5.2.7.3 Kamera

Under utvikling og testing vil gruppen bruke eksternt webkamera som kamerakilde.

Spesifikasjon:

- USB 2.0
- HD 1280x720
- Windows 7,8,10
- Mac OS 10,11 ++

5.3 Drøfting av løsningsalternativ

5.3.1 Alternativ 1 - Kombinering av OpenCV «trackere»

En kombinasjon av flere «trackere» kan være en god ide, men det har vist seg å være vanskelig å vekte de forskjellige algoritmene opp mot hverandre da de bruker forskjellige metoder til å «tracke» objektet. Dermed har de ikke en felles verdi som kan si hvilken algoritme som er mest rett. Det er heller ikke mulig å sammenligne de med et felles referansebilde for å eventuelt detektere «tracking»-feil. Dette på grunn av at «tracking»-algoritmene har hvert sitt unike referansebilde. Referansebildet til hver «tracker» vil også bli litt forandret over tid. Da de fleste av algoritmene finner nåværende objekt med å sammenligne det med tidligere bilder. Dermed vil alle over tid ha et ulikt bilde på hvordan objektet ser ut. Det vil være mulig i programmet å fange opp hvis «trackere» tar raske og uventede forflytninger. Det kan tyde på feil da objekter i en undervannsoperasjon, normalt beveger seg sakte. Ved å kjøre flere «trackere» samtidig, kan operatør velge den som observert klarer å følge objektet best. En mulighet ved kombinering av tre «trackere», er å sammenligne de mot hverandre for å kontrollere om en skiller seg ut fra resten, som kan tyde på at den har problemer. Hvis flere «trackere» brukes, er det mulig å ta ut et gjennomsnitt av de aktive «tracker»-posisjonene. Det kan gi en bedre estimering av posisjonen i motsetning til hvis det bare brukes en «tracker» og den akkumulerer mye feil i estimering av posisjon. Det vil også gi en ekstra sikkerhet i form av at hvis en «tracker» mister objektet, så fortsetter de andre «trackerne» å følge objektet.

5.3.2 Alternativ 2 – Objektgjenkjenning YOLO

YOLO er god på å definer avgrensningen rundt objektet som blir gjenkjent. Her gjør objektgjenkjenning en bedre jobb enn en «tracker», da den i hvert bilde klassifiserer objektet mot den opplærte modellen. Dette gjør den mer robust mot feil-«tracking» over tid og programmet har en god indikator på om den klarer å følge objektet som ønsket. At avgrensningen er nøyaktig vil også innebære en sikrere estimering av forflytning i bildets dybderetning, hvis en bruker samme metode for dette som i alternativ 1. Sett fra det ståstedet alene, vil YOLO være et bedre alternativ for å følge et objekt enn alternativ 1.

Et problem med å bruke YOLO, er at det må lages en modell med treningsbilder på objekter den skal detektere. Derfor kan den kun gjenkjenne de objektene den er trent opp på og vil ikke kunne detektere nye objekt den ikke har «sett» før. Det kreves mye forarbeid å lage en slik modell. Den må være bygget på tusenvis av bilder av objektet i forskjellige omgivelser, med tilhørende data om posisjon i bildet. Bildene bør være fra de rette omgivelsene for å være sikker nok å bruke. For å håndtere dette kan det lages en rutine for å samle treningsbilder av interessante objekter under undervannsoperasjoner. Det kan da lages modeller av objektene som kan brukes ved fremtidige operasjoner.

Denne metoden kan være akseptabelt å utføre for utstyr som skal følges mange ganger, men det vil være for tidkrevende å utføre på nytt utstyr som kanskje bare skal følges en gang. Grunnet omfanget ved å lære opp en slik modell og begrensningene den har, blir ikke denne tatt med i programmet videre. Det kan likevel være en god metode å implementere ved en senere anledning for å følge standard utstyr som brukes ofte.

5.3.3 Alternativ 3 – Estimert dybdeforhold i bildet med maskinlæring

Etter testing av denne metoden ble det observert at dybdeforholdet fungerte bra på enkeltbilder, men i en videostrøm hvor en er interessert i avstandsforholdet mellom kamera og et bestemt objekt som «trackes» var det et problem. Dybdeforholdet som registreres er mellom kamera og alt som blir

observert i bildet. Hvis det da dukker opp ett nytt objekt som f.eks. en fisk eller verktøy i bildet mellom kamera og objektet som «trackes», endres hele dybdeforholdet. Det vil derfor bli problematisk å bruke denne verdien som en estimering av dybde hvis skaleringen plutselig vil kunne endres. Det ble vurdert om en kunne avgrense området som blir prosessert av denne metoden til kun interesseområdet rundt objektet, men da vil ikke algoritmen ha nok informasjon til å skape et dybdeforhold. Gruppen velger derfor å gå bort fra alternativ 3.

5.3.4 Alternativ 4 – Aruco

Aruco var den metoden som under testing viste seg å være mest nøyaktig. Når en markør blir gjenkjent, kan programmet hente ut informasjon om posisjon og rotasjon mellom markør og kameraet. Et problem med Aruco er at den er avhengig av at en markør er festet på objektet for å kunne spore det. Det innebærer at markøren må festes i forkant av en operasjon eller at ROV-en har markøren med seg som f.eks. en magnet som kan festes på objektet under operasjonen. Dette ser gruppen likevel på som å være en enklere, mer fleksibel og mer nøyaktig metode en alternativ 2. Det er heller ingen begrensing på størrelsen av Aruco-markøren, derfor vil denne kunne justeres etter hvilket arbeidsområde ROV-en skal operere i.

5.3.5 Software

Etter møte med ekstern veileder den 09.03.22, ble det diskutert hvordan et brukergrensesnitt bør se ut og ulike måter å lage dette til programmet. De ulike metodene som ble diskutert var: webgrensesnitt i HTML og grafisk brukergrensesnitt i f.eks. C# eller Python. Gruppen satt av en uke for å velge metode. Etter å ha lest om og testet de forskjellige metodene ble grensesnitt i C# valgt bort fordi OpenCV ikke var direkte kompatibelt med C#. Webgrensesnitt ble også valgt bort da gruppen ble enige at det var for tidkrevende og dermed ikke var sikkert de kunne få lagd et fullverdig brukergrensesnitt med den tiden som var til rådighet. Valget falt dermed på et grafisk brukergrensesnitt i Python kodet med Tkinter [5.2.6.3].

5.3.6 Hardware

Under testing av forskjellige «tracking»-modeller ble det observert at flere av de mer avanserte algoritmene var krevende i forhold til regnekraft på testmaskinen. Maskinlæringsmodellene var enda mer krevende og krevde bruk av GPU gjennom Pytorch for å få en brukbar sanntidsprestasjon. Når det gjelder å teste bruk av en Raspberry Pi til å drive programmet ble det konkludert med at det ikke ville ha en gevinst for rapporten. Spesifikasjonene for testmaskinen er omtrent de samme, bortsett fra en svakere GPU som ikke er kraftig nok til å drive maskinlæringsmodellene. Det ble derfor ikke sett på som hensiktsmessig å overføre programmet til en Raspberry Pi.

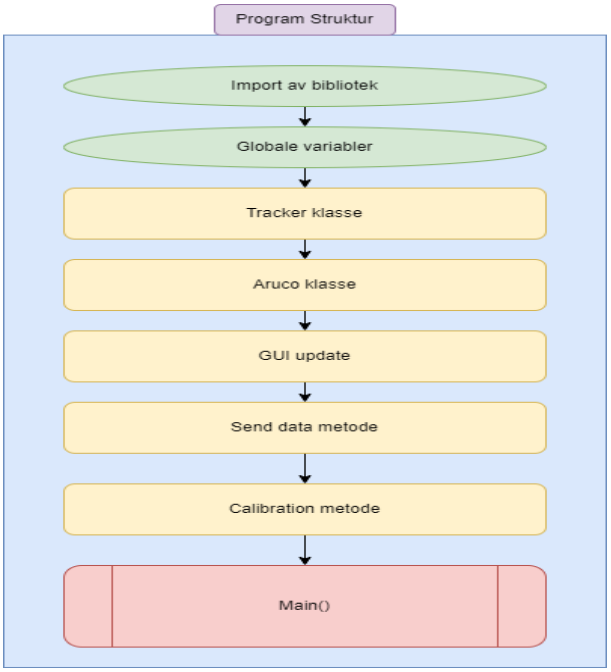
5.4 Konklusjon

Etter testing og drøfting har gruppen kommet fram til at en kombinasjon av Løsningsalternativ 1 – Kombinering av OpenCV «trackere» og Løsningsalternativ 4 - Aruco kan være en god løsning for programmet som skal lages. En har da kombinasjonen av fleksibiliteten til OpenCV «trackerne» og nøyaktigheten til Aruco ved behov. Programmet vil bli skrevet i Python [5.2.6.2], og brukergrensesnittet i Tkinter [5.2.6.3]. Det vil bli implementert en egen metode for å kalibrere forskjellige typer kamera.

6 Realisering av valgt løsning

6.1 Program kode

6.1.1 Struktur



Strukturen til programmet er som vist i Figur 28. Det starter øverst med import av bibliotek, etterfulgt av deklarering av variabler. Deretter kommer selve programmet som er delt inn i Tracker, Aruco, GUI, Calibration, Send data og Main. Der hver metode håndterer sin del av programmet. Rammeverk for det grafiske brukergrensesnittet er satt opp av deklareringer i Main().

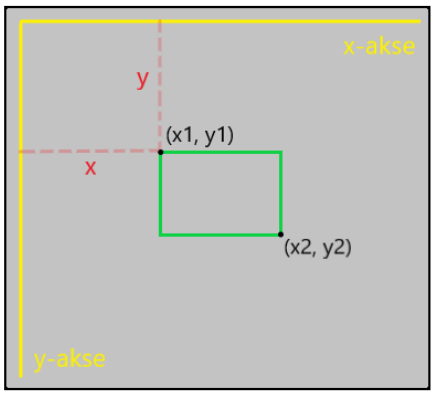
Figur 28 Programmets struktur

6.1.1.1 Tracker

Ved å lage en egen objektklasse for «trackerne» blir det enklere når flere skal kjøres samtidig. Konstruktøren tar som input hvilken type tracker som skal brukes, koordinater for interesseområdet i bildet, bildekilde og fargeverdi for markering av interesseområdet. Her opprettes tilgangsmedlemmene for klassen og det kontrolleres om det mottas en bildestrøm.

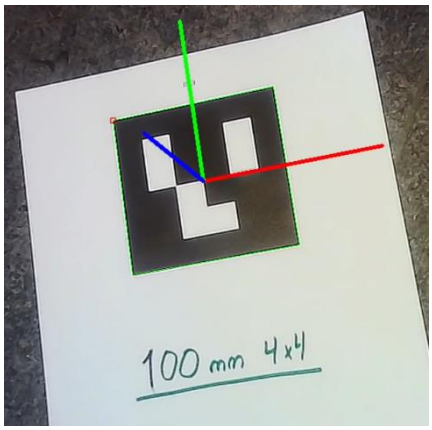
Metoden «Run()» oppretter «trackeren» ved bruk av valgt interesseområde(ROI) (Region of interest) og første tilgjengelige bilde fra bildestrømmen. Koordinatene fra valgt interesseområdet lagres i en variabel som referanse for måling av offset. For hver gjennomkjøring av metoden leser «trackeren» et nytt bilde fra bildestrømmen og oppdaterer koordinatene til interesseområdet hvor objektet er detektert. Koordinatene brukes i tillegg for å markere interesseområdet på bildet for å visualisere til bruker hva «trackeren» sporer. Metoden blir kjørt kontinuerlig med et intervall fastsatt av variabelen «tms» så lenge bruker har aktivert «Start Tracker»-knappen.

Det er laget to metoder for måling av posisjonsendring i klassen. En som måler fra senter av bildet, som brukes hvis en ønsker å regulere ROV til å holde interessant objekt i senter skjerm. Den andre metoden måler fra senter av det markerte interesseområdet, som brukes når ROV skal reguleres til å holde objekt på samme punkt i daværende bilde. Likt for begge er at interesseområdet blir lagret som to punkt som representerer to av hjørnene som danner en firkant. Det måles da antall piksler i x- og y-retning fra hjørne øverst til venstre i bildet og ned til disse punktene.



Figur 29 - Måling av punkt i bildet

6.1.1.2 Aruco



Figur 30 - Aruco markør gjenkjent

Aruco koden er satt inn som en egen objektklasse for å lett kunne kalles opp i «Main()» når Aruco-markører skal brukes til tracking av objektet. Selve Aruco-«trackingen» skjer i «Aruco_run()» metoden inne i Aruco-klassen. Denne tar inn bildestrømmen fra kameraet og finner Aruco-markøren i bildet. Når markøren er funnet, tolkes data fra markøren som ved bruk av metoden «aruco.estimatePoseSingleMarkers()», kalkulerer posisjon og rotasjon til markøren og deler dette opp i to matriser; translasjonsmatrisen «tvec» og rotasjonsmatrisen «rvec». Disse brukes videre for å tegne x-, y- og z-akser på markøren, som vist i Figur 30 - Aruco markør gjenkjentFigur 30. Til slutt blir posisjonene x, y, z og vinklene for «roll, pitch, yaw» til kameraet i forhold til markøren validert og kalkulert. Oppsett for å kunne kjøre Aruco er hentet fra Tiziano Fiorenzanis github [24].

6.1.1.3 GUI Update

GUI Update delen av koden er et sett av metoder som blir brukt for å oppdatere det grafiske brukergrensesnittet til programmet. De viktigste delene av koden er «Show_frames_one()» som leser bildekilden og viser bildet «live» til bruker. Hvis en «tracker» kjører, leses informasjon om referanseområdet som bruker har definert. Deretter markeres referanseområdet med en rød avgrensningsboks på bildet. Samtidig vil hver «tracker» som er aktiv få sin egen avgrensingsboks sammen med navn og tilhørende farge tegnet på bildet.

Metoden som håndterer hva som sendes ut om posisjonsendring, er «Output_control()». Den leser x-, y- og z verdier fra de «trackerne» som kjøres. De blir sendt til tekstboksen «Output» som vises til bruker. Hvis flere «tackere» kjøres samtidig, regner den et gjennomsnitt av verdiene. Den tar ikke med «z» fra Mosse, da den ikke kan estimere forflytning i dybderetning. Disse verdiene blir også brukt av metoden «SendData()».

«Error_detection()»-metoden brukes for enkel feilsøking av «trackere». Hvis posisjonen av objektet endres med mer enn en absoluttverdi på 100 piksler i løpet av 500 millisekund, trigger den en boolsk variabel «warning», som indikerer at noe kan være galt. Denne indikasjonen blir så brukt i «Update_indicators()», som brukes for å oppdatere indikatorene til hver «tracker». Den leser av om «trackeren» kjører som normalt som gir grønn farge. Hvis «warning» er aktivert vises gul farge. Hvis «tracking» har mistet objekt, vises rød farge. Den styrer også indikator om Aruco er aktiv.

6.1.1.4 Send data metode

I start av programkoden opprettes en sokkel for å sende posisjonsdata fra programmet som et UDP-telegram. Sokkelprogrammering er en måte å koble sammen to noder på et nettverk ved hjelp av maskinens innebygde nettverkskort. Sending av utdata skjer i «SendData()»-metoden. Her blir x-, y- og z-verdier hentet fra «trackerne». Hvis Aruco er brukt, blir også «roll»-, «pitch»- og «yaw»-verdier satt inn i variabelen «telegram». For å kunne håndtere om «tracker» mister objektet eller markør, sendes det info om dette i variablene «Warning» og «Error» som 1 eller 0 for hver tracker til slutt i strengen. Videre blir «telegram» omgjort til bytes og sent ut som et UDP-datagram.
Standard format for telegrammet er:
$TRACKX0000Y0000Z0000W000E000# (Ved bruk av «tracker» metode)

$ARUCOX0000Y0000Z0000RO0000PI0000YA0000E0# (Ved bruk av Aruco)
$ - Telegram data start
TRACK – Tracker data
ARUCO – Aruco data
X – Posisjons data X-akse ([piksler] for TRACK og [cm] ved bruk av ARUCO)
Y – Posisjons data Y-akse ([piksler] for TRACK og [cm] ved bruk av ARUCO)
Z – Posisjons data Z-akse ([piksler] for TRACK og [cm] ved bruk av ARUCO)
RO – Rotasjons data Roll [grader]
PI – Rotasjons data Pitch [grader]
YA – Rotasjons data Yaw [grader]
W – «Warning», advarsel unormal oppførsel
E – «Error», tracking mislykkes eller finner ikke markør.
- Telegram data slutt

6.1.1.5 Calibration

Kalibreringsprogrammet blir kalt opp gjennom metoden «Cal_Click», som regner ut hvor stor forvrengningen av bildet er til det valgte kameraet. Den lager matriser som brukes for å korrigere bildet før «trackingen» av et objekt starter.

Først i programmet brukes «start_cam_cal» som leser bildestrømmen. Operatør kan da se om testplakat med rutemønster er innenfor kameraets synsfelt. Ved å kjøre «take_picture»-funksjonen lagrer den det siste bilde i bildestrømmen i en lokal mappe. For hvert bilde som er lagret, så fungerer «Pic_taken»-metoden som en teller hvor den skriver ut hvor mange bilder mappen inneholder. Dette er for å kunne vite hvor mange bilder som er tatt og kan da starte «start_Calib»-fuksjonen når grensen på 25 er nådd. Hvis det er mindre enn 25 bilder ved start av kalibreringen, kommer det opp en melding om at det er behov for flere bilder.

For hvert bilde som er blitt tatt, leter metoden etter et spesifikt rutemønster i bildet ved å bruke «findChessboardCorners», denne velger kun ut de bildene hvor det spesifikke mønsteret er gjenkjent. «DrawChessboardCorners» lager hjørnepunkter i bildene som gjør at de kan bli brukt til kalibrering. «CalibrateCamera»-metoden vil da gjøre forvrengningsverdien lavere for hvert bilde som blir prosessert. Når forvrengingsverdien kommer under 0.1, er forvrengingen av bildet liten nok til at forvrengningskoeffisientene og kameramatrisen lagres, som kan brukes for å korrigere bildet fra det spesifikke kamerat ved fremtidig bruk.

6.1.1.6 Main

Main styrer hvordan programmet skal utføres under oppstart og gir en oversikt hvordan programmet er strukturert. Her styres hvilke metoder som skal kjøres i hvilken rekkefølge.

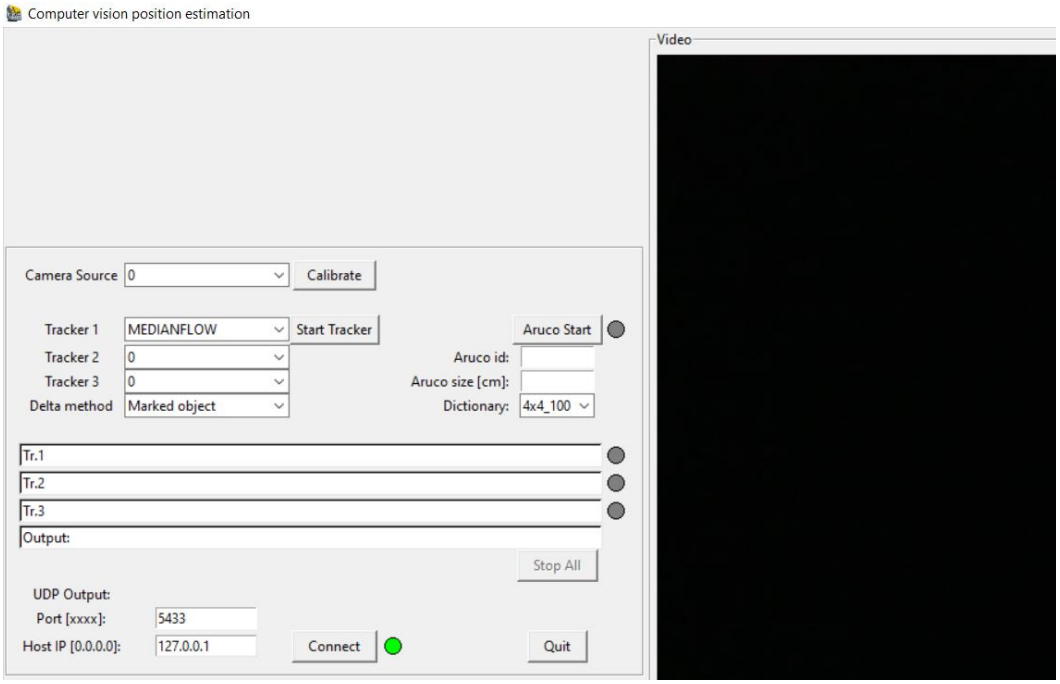
Først startes «Tk()» som starter klassen som gir tilgang til Tkinter biblioteket. Med dette blir brukergrensesnittet bygget opp i form av menyer, knapper, tekstvinduer og bildevindu som er knyttet mot bestemte metoder og verdier.

Videre opprettes det to lister: «tList[]» og «aList[]». «tList[]» opprettes for å holde kontroll på de valgte «trackerne». Denne oppdateres mot «dropdown»-menyene for «trackere» hver gang bruker trykker på «Start Tracker». For hver «tracker» som er aktiv blir det også lagt inn data om x-, y- og z-koordinater

som knyttes opp til sin «tracker». «aList[]» blir brukt hvis Aruco-metode kjører. Her legges inn posisjonsdataene x-, y-, og z sammen med «roll», «pitch» og «yaw».

6.2 Brukergrensesnitt

Brukergrensesnittet er bygget opp slik at bruker har full kontroll over hva som «trackes», hvilken metode som brukes og hvilke verdier som sendes ut av program. Bildestrømmen fra kameraet som er i bruk på ROV-en og hva som spores, vises i vinduet «Video» som vist til høyre i bildet i Figur 31. Hvilket kamera som er i bruk på ROV-en velges fra «dropdown»-meny «Camera Source».



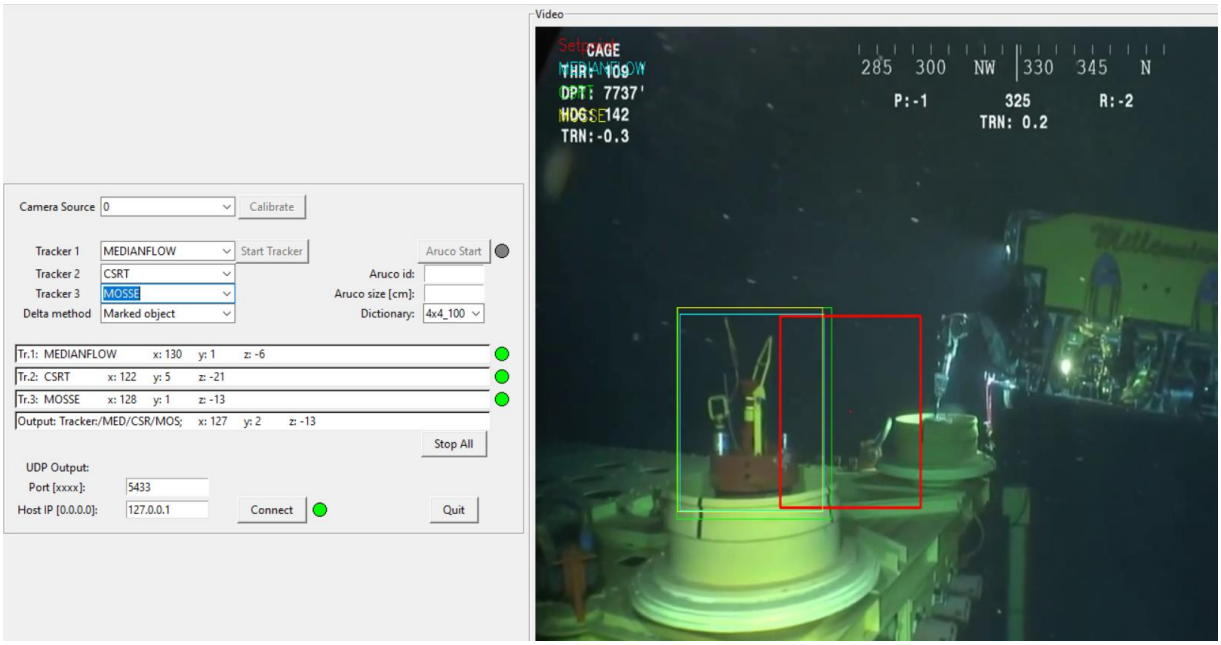
Figur 31 - Brukergrensesnitt

Hvilken port og IP som skal brukes under UDP oppkobling er satt til en standardverdi når programmet starter, men hvis bruker ønsker, kan dette endres i tekstboksene nede til venstre i programmet under «UDP Output», se Figur 31. Den nye oppkoblingen skjer når «Connect»-knapp betjenes. Indikator til høyre for «Connect» indikerer om programmet klarer å sende informasjon eller ei.

6.2.1 Kombinering av «trackere»

En kan se på Figur 32, programvaren i bruk på en testvideo fra en undervannsoperasjon med to ROV-er. Perspektivet i bildet er fra ROV-en som skal overvåke operasjonen og som skal reguleres til å forholde seg i ro mot et objekt.

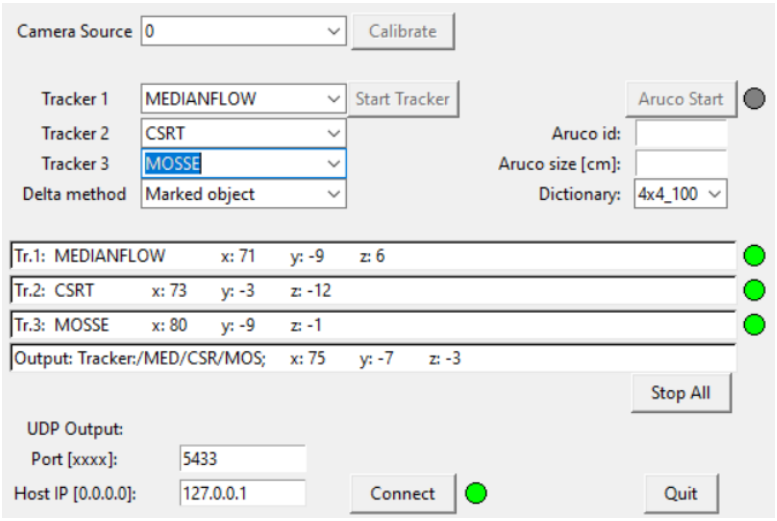
Det er her valgt kombinering av tre forskjellige «trackere». Når programmet da startes ved å klikke «Start Trackere», får bruker mulighet til å markere hva i bildet som skal «trackes» ved å markere det med en firkant som dras over objektet. Det er her valgt å «tracke» den røde ventilen til venstre i bildet. En ser at den er markert av en grønn, gul og turkis firkant som indikerer interesseområdet for «trackerne».



Figur 32 - Trackere i bruk

På Figur 32, ser en at bruker har valgt offset måling fra markert objekt i «Delta method». Programmet vil da sette opp en rød firkant hvor objektet ble først markert som indikerer ønsket posisjon av objektet(ventilen). Den gule, grønne og turkise firkanten viser hvordan «trackerne» klarer å spore objektet (ventilen) fra bilde til bilde. Det måles kontinuerlig antall piksler som er i offset mellom settpunkt (rød firkant) og «trackernes» firkanter. De flyttes etter objektet som «trackes», samtidig tilpasses størrelse av interesseområdet (firkanten). Regulator vil da kunne automatisk regulere ROV-en mot «tracker-boksene» slik at den røde firkanten matcher over de andre fargede firkantene, både sentrering og størrelse. Slik kan ROV-en holdes i ro i forhold til ventil.

På Figur 33 vises det et utsnitt av brukerpanelt i programmet. De tre nederste tekstboksene viser de valgte «tracker»-metodene og offset-verdien de har i form av piksler. Det er en indikator til høyre for hver tekstboks som viser om «trackingen» er gyldig i form av indikasjon i grønn farge. Indikatoren viser også om «trackeren» er usikker på om den er korrekt og trenger godkjenning eller korrigering av operatør, i form av indikasjon i gul farge. Ved rød farge har «trackeren» mistet objektet og brukes ikke lenger til måling.

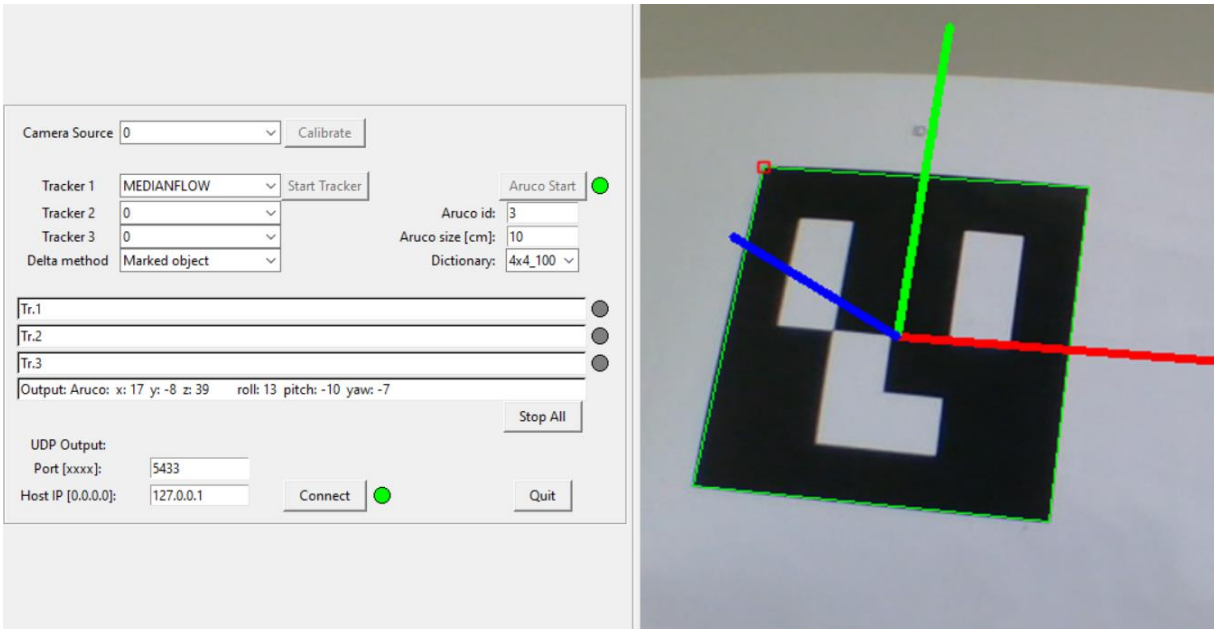


«Output»-tekstboksen viser resultatet av kombinasjonen av de valgte «trackere» og hvilken som er med i målingen.

Figur 33 - Utsnitt brukerpanel

6.2.2 Bruk av Aruco

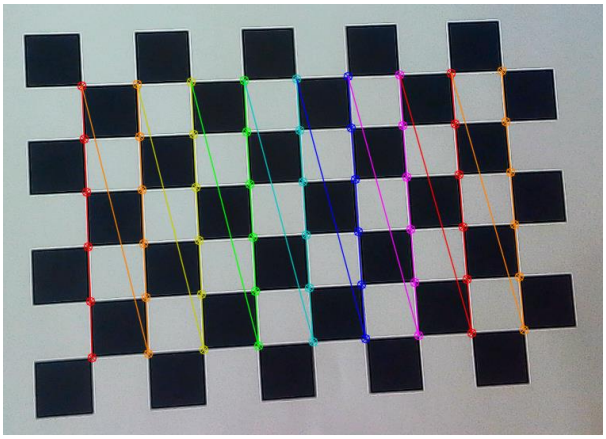
For bruk av Aruco-metoden, betjenes «Aruco Start»-knapp vist i Figur 34. I tekstboksene under knappen kan en taste inn hvilken type Aruco-id som er interessant, samt størrelse på markør i form av bredde eller høyde (de er kvadratiske). I tillegg kan det velges mellom tre forskjellige bibliotek av markører som skal brukes. De målte verdiene vises i samme tekstboks som brukes ved kombinering av «trackere». Det er en indikator til høyre for «Aruco Start»-knapp som lyser grønt når programmet kjører og søker etter markører.



Figur 34 - Bruk av Aruco

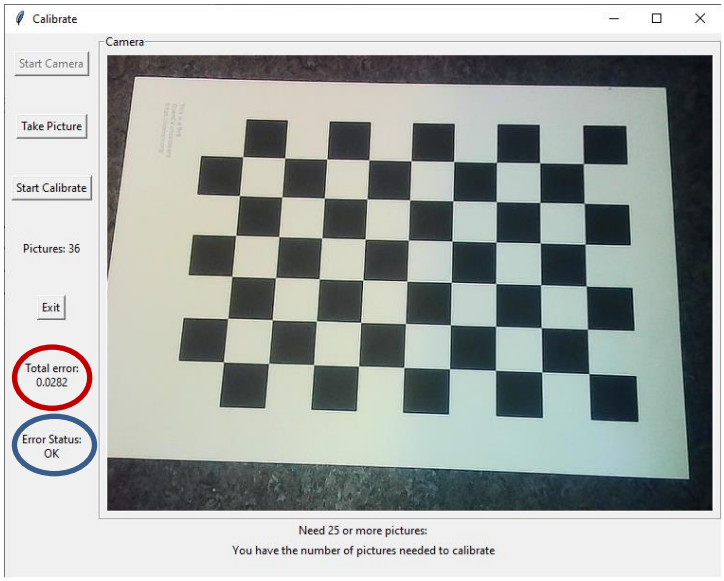
6.2.3 Kalibrering

For å kunne kompensere for en eventuell forvrenging av bildet, brukes et kalibreringsprogram. Dette må kjøres hver gang programmet blir brukt sammen med et nytt kamera. Hvis en bruker et kamera som allerede er kalibret, brukes matrisene for dette kameraet som ligger i mappen for kalibreringsmatriser. Grunnen til at sjakkbrettmønsteret blir brukt, er fordi det er distinkt og lett å gjenkjenne. Hjørnene gjør at den er ideell til lokalisering da det er to tydelige gradienter i hver sin retning. Hjørnene er også knyttet sammen i skjæringspunktene mellom sjakkbrettlinjene som blir illustrert i Figur 35.



Figur 35 – Sjakkbrettkalibrering

Algoritmen søker etter alle punkt hvor kanter møtes som blir brukt til kalibrering. Bildene blir lagret i en mappe som ligger lett tilgjengelig med programfilene. For å kunne kjøre kalibreringen, må det tas minst 25 bilder i forskjellige vinkler og posisjoner for å kunne kjøre programmet. Etterpå blir forvrengningskoeffisientene laget og det regnes ut en verdi som brukes til å måle forvrengningen. Om denne verdien ikke er lav nok, tyder det på at det fortsatt er en del forvrenging av bildet. Hvis det blir brukt et kamera med mye forvrengning som f.eks. et kamera med «fish-eye» linse, må det tas mange flere bilder for å få lav nok verdi. Forvrengningsverdien er vist som «Total error» i brukergrensesnittet, illustrert i *Figur 36.* Slik får bruker en indikasjon på hvor mye forvrengningen er i bildet.



Figur 36 - Kalibrering brukergrensesnitt

Det er satt en grense i programmet på 0,1 for «Total error», programmet vil fortsette å kreve bilder under kalibrering frem til ønsket nivå er nådd. Det blir vist hvor mange bilder som er tatt og lagret i bildemappen. Når verdien er lav nok, vil «Error»-statusen gi en ok melding illustrert i Figur 36, slik at en vet at den er god nok til å kunne bli tatt i bruk.

7 Testing

Det ble utført tester for å finne ut hvilke «trackere» som egnet seg best til bruk i programmet. Tester som ble utført var å se hvordan «trackerne» presterte i forhold til horisontale, vertikale og dybde-forflytning. Testene ble utført med et webkamera som var kalibrert med kalibreringsalgoritmen i programmet med følgende matriser:

Kameramatrise [3X3]:

$$\begin{bmatrix} 1.2930804425248006 74e+03 & 0 & 6.7112100296646565 31e+02 \\ 0 & 1.2709138680762468 99e+03 & 3.8802071659048630 18e+02 \\ 0 & 0 & 1 \end{bmatrix}$$

Forvrengningskoeffisienter [5X1]:

$$[1.9326443782458765 93e-01 \quad -6.9838237201959185 72e-01 \quad 2.6894523720755788 43e-03$$

$$-9.0048296468825950 18e-03 \quad 9.4959945971601944 50e-01\,]$$

Målet med testene er:

- Er det en lineær sammenheng mellom forflytning av objekt og målt pikselendring etter kalibrering.
- Kontrollere sammenhengen mellom målt forflytning i bildets dybderetning og endringen av piksler i høyderetning av «tracket» objekt.
- Hvor godt klarer «tracker» å følge et objekt i aktuelle omgivelser.

Objektet som ble «tracket» i testene var en hvit rektangulær eske som resulterte i god kontrast mot den mørke bakgrunnen, som var en svart tavle. Esken hadde bilder og tekst i front som ga «trackeren» interessepunkt som «features» for å sikre god sporing.

7.1 Test av «trackere» og kontroll av kalibrering

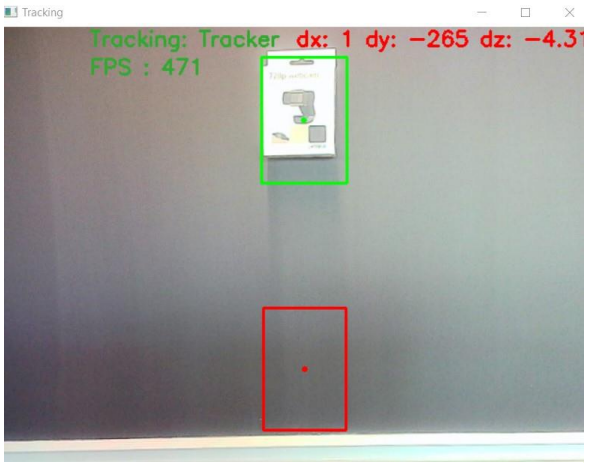
Poenget med den første testen var å få en oversikt over hvordan de forskjellige «trackerne» presterte opp mot hverandre i sanntid under kontrollerte forhold. En ønsket da å kunne velge ut de «trackerne» som kanskje var uaktuelle å bruke videre i en tidlig fase.

I tillegg fikk gruppen sjekket at etter kalibrering av kamera, så skal det var en lineær sammenheng mellom forflytning av objektet i horisontal- og vertikal-retning og målt pikselendring. Det skal altså her fanges opp om det er noe avvik som følge av forvrengning av bildet.

Det ble testet med forskjellige «trackere» som er tilgjengelige i OpenCV. De som ble testet var Boosting, KCF, TLD, Medianflow, CSRT, MOSSE. MIL ble ikke tatt med da KCF er en ny og forbedret utgave av MIL. Objektet som skulle «trackes» var festet midt på en sort tavle som vist på Figur 37. Tavlen er montert på skinner som da sikrer lik forflytning og samme startposisjon for hver test. Kameraet ble plassert 1 m fra objektet.

Objektet ble forflyttet med en iterasjon på 35 mm for hver måling. Etter at hele distansen er målt, ble objektet flyttet tilbake til start for å sjekke om senterpunket for «trackingen» av objektet hadde akkumulert opp feil under målingen. Det er utført flere målinger på forflytningene for hver «tracker» i begge retninger som det til slutt er regnet et gjennomsnitt av pga. det vil være litt variasjoner i hver måling.

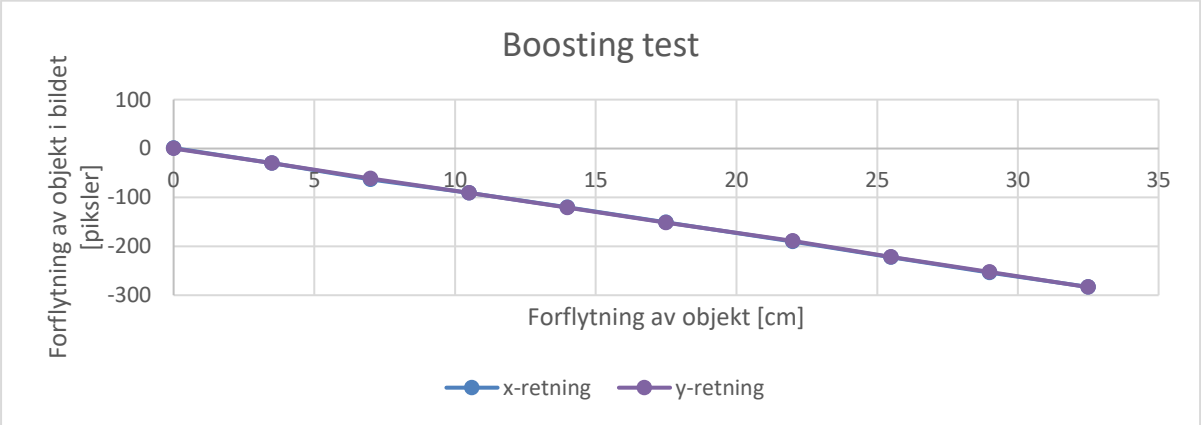
Til slutt ble objektet gradvis tildekket for å se hvordan «trackeren» håndterer blokkering.



Figur 37 - Bilde av test mot tavle

7.1.1 Boosting:

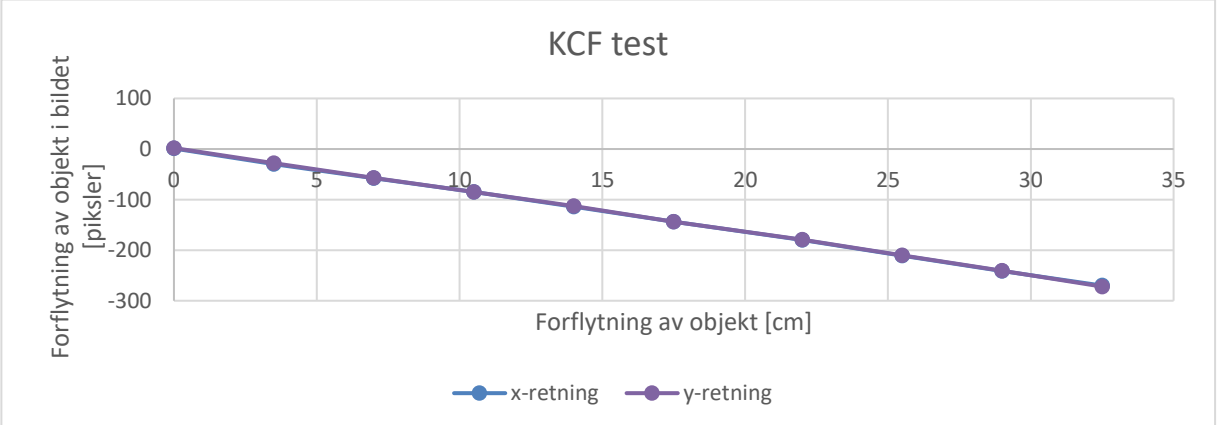
Under testforholdene fungerer Boosting på tilfredsstillende måte til oppgavens formål selv om dette er en gammel og utdatert «tracking»-metode. Forholdet mellom pikselendring og forflytning var lineært, se Figur 38. Den presterte derimot dårlig når det kommer til blokkering, da den mistet objektet allerede ved 50 % tildekning.



Figur 38 - Test Boosting "tracker"

7.1.2 KCF

KCF «tracket» tilfredsstillende til forventningene, se Figur 39. Når objekt ble flyttet tilbake til startposisjon kunne det observeres en offset på 10-12 piksler, noe som indikerer noe feil som blir akkumulert underveis. Ved 50 % blokkering klarte «tracker» å gjenspore objektet med offset på 22-24 piksler. Den klarte ikke å håndtere 75 % tildekking.



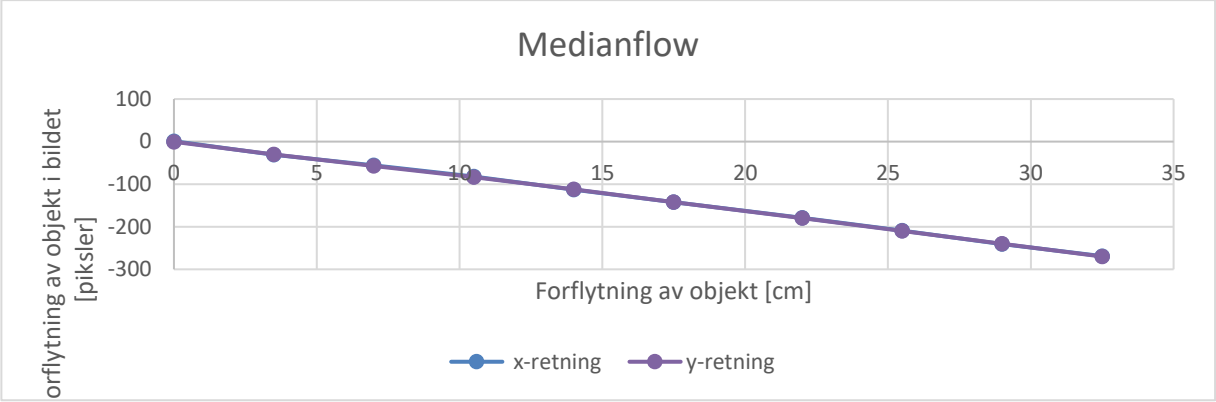
Figur 39 - Test KCF "tracker"

7.1.3 TLD:

Til «tracking» fungerer ikke TLD til oppgavens formål. Den er for ustabil og fant nye punkt/objekt å spore hele tiden. Det ble derfor ikke mulig å få til en kontrollert måling av denne «trackeren». En ting denne metoden var god på, var å finne igjen objektet ved okklusjon da den hele tiden sammenligner med første referansebilde.

7.1.4 Medianflow

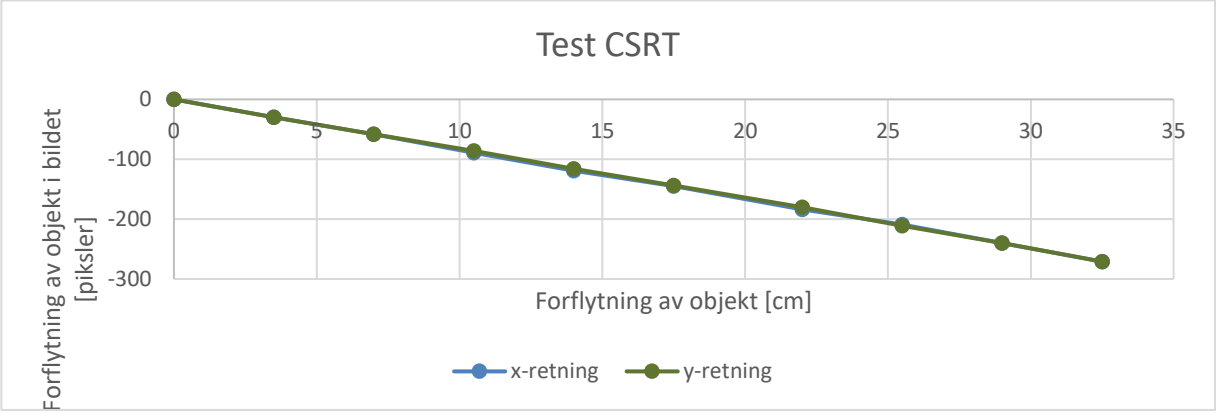
Tilfredsstillende «tracker» under testforholdene, se Figur 40. Feil ved tilbakestilling var 8-15 piksler. Ved 50 % tildekking av objektet klarte «tracker» å gjenspore objekt med offset fra 1 til 6 piksler. Ved 75 % tildekking fra 0 til 60 piksler. Ved 100 % tildekking mistet den objektet og ga tilbakemelding om det.



Figur 40 - Test Medianflow "tracker"

7.1.5 CSRT

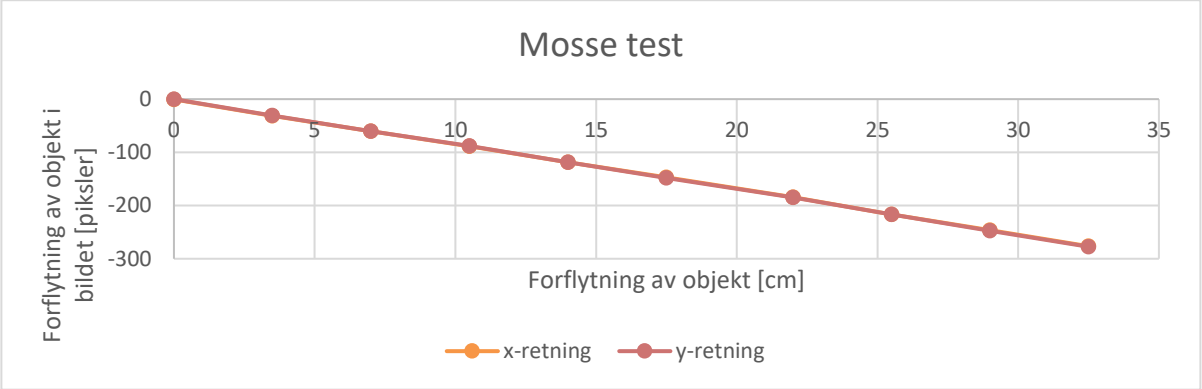
Figur 41 viser tilfredstillende «tracking» ved forflytning av objekt. Feil ved tilbakestilling var gjennomsnittlig 14 piksler. Ved 50 % blokkering av objektet var offsetten ved gjensporing 0-25 piksler. Ved 75 % Blokkering er offset fra 3-28 piksler. Ved 100 % blokkering fant ikke «tracker» objektet igjen. Etter blokkeringer ble det observert at interesseområdet til tracker ble endret noe da denne metoden tilpasser ROI-rektangelen til objektet kontinuerlig.



Figur 41 - Test CSRT "tracker"

7.1.6 MOSSE:

Det observeres at det er en rask «tracker» som er effektiv i forhold til regnekraft da den gir høy FPS, opp mott 800. Tilfredsstillende «tracking». Se Figur 42. Feil ved tilbakestilling var 5-7 piksler. Ved 50 % tildekking er offset ved gjensporing 3-5 piksler. Det samme ved 75 % blokkering. Klarte ikke å gjenspore ved 100 % tildekking.



Figur 42 - Test Mosse "tracker"

7.1.7 Konklusjon

En kan først og fremst konkludere med å se på plottet av de forskjellige «trackerne» at sammenhengen mellom forflytning av objekt og endring i målt pikselforflytning er lineær og at det da ikke er noe forvrenging i bildet. Det er vanskelig å se forskjell på x- og y-plott da de er nesten identiske og ligger over hverandre.

Alle «trackerne» klarte å spore objektet tilfredsstillende under testforholdene. Det som skillet mest mellom de forskjellige algoritmene i testen er hvor mye feil de akkumulerte og hvordan de håndterte blokkering. TLD var så ustabil at den blir sett på som irrelevant etter testen.

Boosting og KCF kom dårlig ut ved blokkering av objektet, grunnet Boosting ikke håndterte noe blokkering og KCF kun klarte 50 % blokkering, som medførte en stor offset av interesseområdet.

Medianflow og CSRT klarte seg bra ved 50 % blokkering, noe høyere offset ved 75 % blokkering. Begge hadde lav offset ved tilbakestilling.

Mosse er trackeren som kommer best ut i denne testen med lavest offset etter tilbakestilling i måling og lavest offset etter blokkering av objekt. Den er i tillegg den som gir høyest bilde per sekund, som indikerer at denne er raskest å prosessere.

7.2 Test av bevegelse i bildets dybderetning

I testen ble det kontrollert sammenhengen mellom pikselendring som følge av forflytning av et objekt i bildets dybderetning og tangensfunksjonen. Dette ble gjort ved å sammenligne den teoretiske modellen fra 4.1.1 og målingene som ble utført. De beregnede målingene viser hvilken høyde objektet ville hatt på bildebrikken, mens de målte verdiene viser pikselendringen av høyden til objektet på bildebrikken.

Objektet som er 70 mm høyt, ble plassert 2 m fra kameraet. For hver måling ble objektet forflyttet 200 mm mot kameraet, frem til en avstand på 1 m. Deretter ble det tatt målinger for hver 100 mm. Det er kun CSRT og Medianflow av de valgte «trackerne» som tilpasser avgrensningen av interesseområdet rundt objektet, som igjen muliggjør det å måle en forflytning i bildets dybderetning. Derfor ble kun de to brukt i denne testen.

7.2.1 Teoretisk modell:

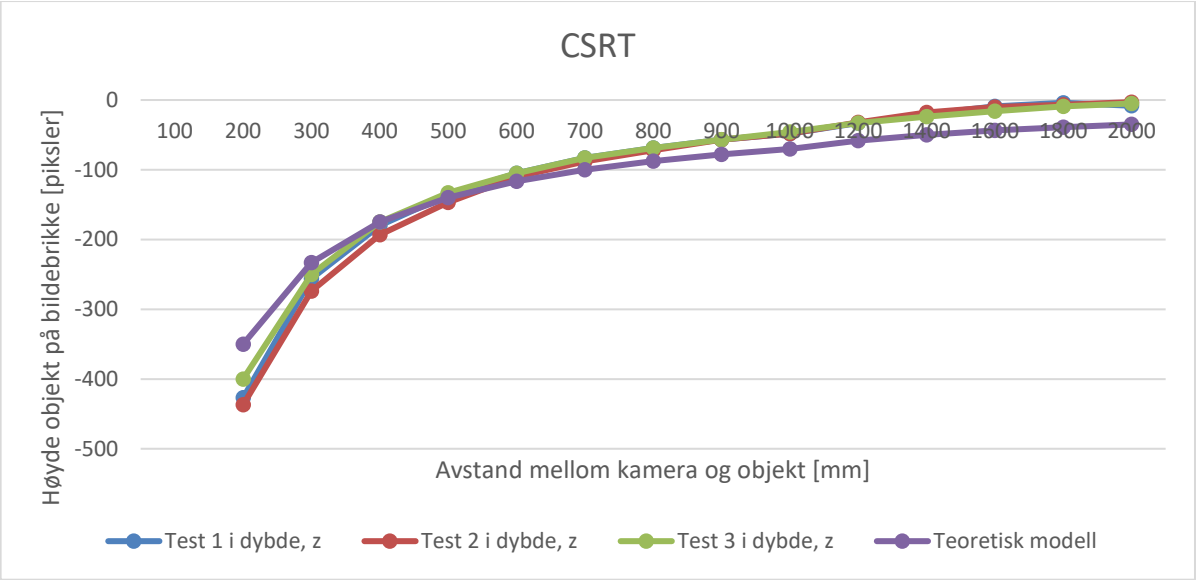
I den teoretiske modellen som ble vist i 4.1.1, kom det frem at avbildet høyde på bildebrikken kunne regnes ut med formelen:

$$y' = \frac{y}{s} \cdot s'$$

For sammenligning med måleresultatene ble denne formelen brukt for å lage et plot. Formelen regner ut høyde i millimeter, mens programmet måler endring av antall piksler i høyden. Resultatene ble derfor etter beregning av teoretisk modell, skalert med 100 for å kunne sammenlignes med målingene fra testen.

7.2.2 CSRT:

Størrelsen på objektet i bildet var ganske liten når avstanden var på 2m. Det medføre at CSRT fikk litt problemer med «trackingen» på første måling, som resulterte med en offset på -5piksler fra start. Når objektet flyttes nærmere kamera, klarer den å stabilisere seg ved at «features» blir lettere å detektere. Plottet på Figur 43 viser en at den målte høyden av objektet ved CSRT-«trackingen», ligner det forventede resultatet i den teoretiske modellen.

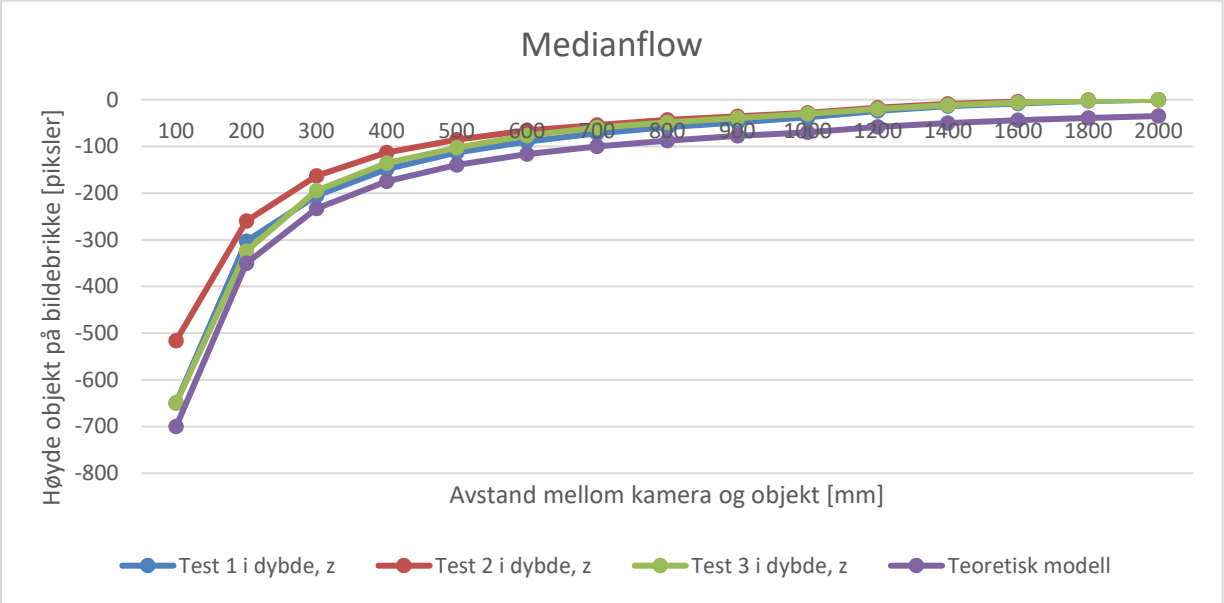


Figur 43 - Test dybde CSRT

Da avstanden mellom objekt og kamera ble redusert til 200 mm klarte den ikke å tilpasse interesseområdet rundt objektet. Årsaken til dette var at objektet ble så stort at deler av det kom utenfor kantene i bildet.

7.2.3 Medianflow:

Medianflow klarte å holde seg stabil med liten offset av interesseområdet fra start. Resultatet av målingene former en kurve som er tilnærmet lik den teoretiske modellen, vist i Figur 44. En kan se at Medianflow klarer å «tracke» objektet helt til det er 100 mm fra kameraet. Da får «trackeren» problemer ved at objektet havner utenfor bildet og klarer derfor ikke å finne de «features» som ble sporet.



Figur 44 - Test Medianflow dybde

7.2.4 Konklusjon

CSRT metoden gir like resultater gjennom alle testforsøkene. Den kan få litt problemer ved å finne «features» hvis objektet er på en distanse som gjør at det blir for lite i bildet eller når det kommer for nært kameraet, som gjør at deler havner utenfor bildet. Derimot er den mer stabil enn Medianflow i avstandene mellom ytterpunktene.

Mediaflow kom bedre ut i starten av testen ved at den bedre klarer å «tracke» et lite interesseområde. Den hadde litt mer varierte målinger fra test til test, men økningen mellom hver måling etter forflytning av objekt er forholdsvis lik, noe som gjør at denne har mest likhet med den teoretiske modellen. Medianflow klarer å «tracke» litt lenger enn CSRT, da den klarer å kjøre «trackingen» selv om den mister noen av «features» som ble brukt når deler av objektet havner utenfor bildet.

CSRT og Medianflow klarer å måle endring av høyden til objektet tilfredsstillende lik den teoretiske modellen. Dette styrker teorien om at denne metoden kan brukes til å estimere forflytning av et objekt i bildets dybderetning.

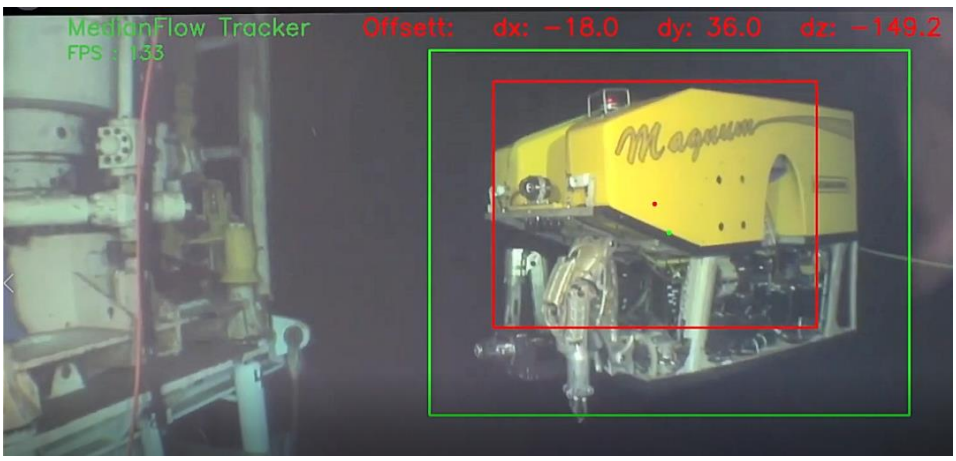
7.3 Test av «tracking» i aktuelle omgivelser

For å teste de forskjellige «trackerne» i de omgivelser de skal brukes i, er det brukt videofiler av undervannsoperasjoner fra perspektivet til en ROV. Det er ikke noe referansepunkt eller kjente mål i videoene fra før, så for å kontrollere «trackerne» mot hverandre er videoene satt sammen av to klipp. Et klipp som viser operasjonen som normalt, påfølgende av samme video i revers. På denne måten starter og stopper objektet som skal følges i samme posisjon. En kan da måle om «trackeren» har driftet eller akkumulert feil underveis ved å måle offset mellom start- og sluttpunkt for interesseområdet til «trackeren».

Figur 45 viser en undervannsoperasjon der bildet er fra perspektivet til en ROV som overvåker arbeidet som blir utført av en annen ROV på modulen. Det ble satt et fast interesseområde over ventilen i bildet som offset blir målt ut fra, i form av et rødt rektangel når video starter, se Figur 45. Det grønne rektangelet viser interesseområde som oppdatertes av CSRT «tracker». Bildet i videoen beveger seg rundt ventilen og ROV-en, men holder dem i bildet hele tiden.

Figur 45 - Testvideo 1 med CSRT

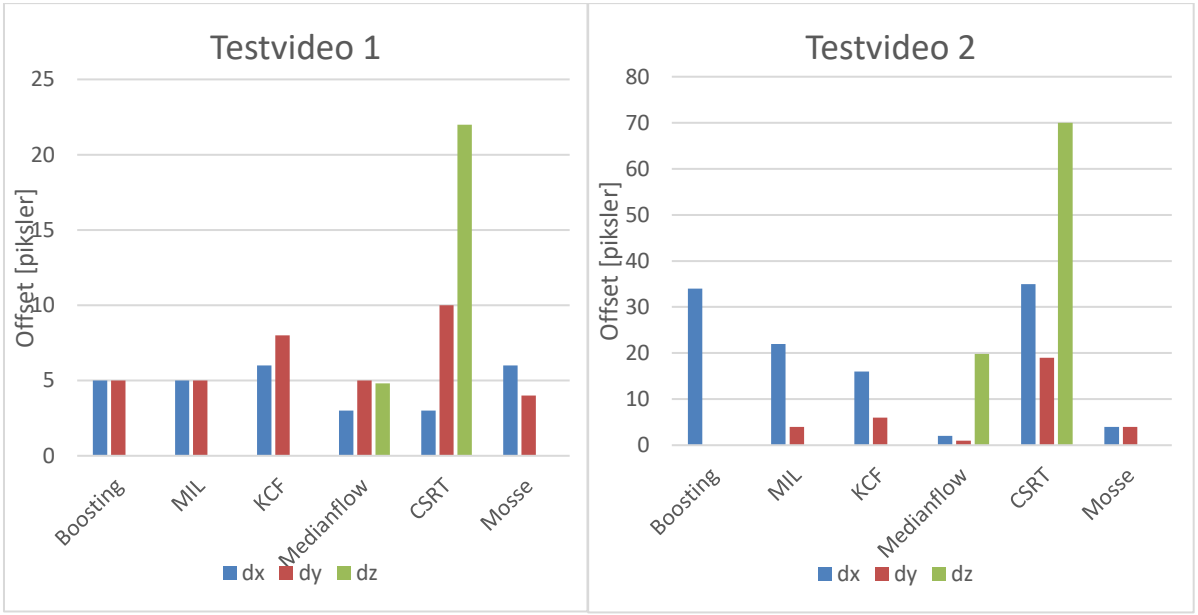
I Figur 46 ble interesseområdet markert over en ROV som kjører ut av et nedsenkningsbur og kjører mot en modul. Her blir også offset målt fra det røde rektangelet som ble markert i start av video rundt ROV-en. Det grønne rektangelet viser hvordan Medianflow-«tracker» tilpasser interesseområdet rundt ROV-en.



Figur 46 - Testvideo 2 med Medianflow

7.3.1 Resultat

«Trackerne» som var mest interessante i denne testen var CSRT og Medianflow, da det kun var de som tilpasset seg interesseområdet ved endring av posisjon i bildets dybderetning. De andre «trackerne» ble likevel tatt med for å teste hvordan de «tracket» i bredde og høyde retning i bildet.



Figur 47 - Resultat fra testvideo 1 og 2

I Figur 47 er dx og dy målt som offset fra senterpunkt av objekt når objektet er tilbake til startpunkt. De bør være minst mulig da de indikerer om «trackerne» har klart å følge objektet uten å akkumulere feil på strekningen. Den målte offseten av posisjon i bildets dybderetning, vises i dz-søylen. En kan se at i bildets dybderetning så er «trackeren» Medianflow bedre enn CSRT med lavere piksel-offset. Det ble observert at CSRT ble påvirket negativt når det var gjenskinn fra objektet. Det samme gjelder når lyskilden til ROV-en som ble «tracket» i testvideo 2, lyste direkte inn mot kameraet. Det medførte at interesseområdet ble forskjøvet. Dette gjenspeiles i den høye offseten i målt i Figur 47, men den «tracket» objektet bra ellers.

En kan se i Figur 47 at Medianflow er den «trackeren» som har klart seg best i både x- og y- og z-retning. Mosse peker seg ut i denne testen som en god kandidat for «tracking» i x- og y-retning. De resterende «trackerne» presterer noenlunde likt med litt høyere feilakkumulering.

8 Gjennomføring av fremdriftsplan

Fremdriftsplanen ble satt opp av gruppen etter endt forstudie. Den har blitt fulgt med noen små endringer og unntak underveis. Veiledningsmøtene har ikke blitt utført helt som først planlagt. Det ble planlagt totalt syv veiledningsmøter fordelt mellom intern- og ekstern-veileder. Dette ble fulgt i starten av prosjektet, men fra uke 7 ble møtene utsatt på grunn av arbeidsmengden, og fordi gruppen ikke hadde noe konkret å ta opp på det tidspunktet. Det var planlagt totalt fire møter med internveileder, men bare to ble utført. Grunnen til dette har vært at når gruppen har hatt spørsmål, har det ofte blitt løst uformelt med å ta en tur innom internveileders kontor på skolen eller via e-post. Ved start ble det også planlagt at gruppen skulle jobbe med rapporten gjennom hele semesteret, men på grunn av arbeidsmengden som måtte til for å lage et fungerende program med et grafisk brukergrensesnitt, har dette ikke vært mulig. Derfor ble hovedmengden av rapportskrivingen utsatt til etter påske. Videre har programvare-delen blitt endret ved at arbeid med dybdeestimering i bilde ble startet i uke 6 istedenfor uke 14. Det ble besluttet i gruppen at utviklingen av programmet var kommet så langt i uke 15 at arbeid med programmering og kamerakalibrering ble stoppet for å kunne skrive på bachelorrapporten. Ferdigstilling av programmet skulle fortsette i uke 21. Derfor ble også test av programvare flyttet fra uke 13 til uke 21. Etter diskusjon i gruppen og med eksternveileder, ble det enighet om å ikke utføre test av Raspberry Pi med programvaren, den er derfor ikke relevant i fremdriftsplan.

9 Endelig konklusjon

Når det gjelder Løsningsalternativ 1 – Kombinering av OpenCV «trackere» som er brukt i programmet, er et av problemet at den akkumulerer feil over tid uten noen løsning for å kunne automatisk korrigere denne. Den vil derfor ikke kunne bli brukt som en helt autonom løsning, da den er avhengig av manuell korrigering fra operatør. Likevel kan det være et godt hjelpemiddel for en ROV-pilot til å holde ROV-en i konstant posisjon i forhold til et objekt, istedenfor å gjøre dette manuelt, men da samtidig som ROV-pilot overvåker prosessen. Den «trackeren» som kom best ut av testene som ble utført, er Medianflow. Den er derfor satt som standard i programmet. Hvordan den presterer i virkeligheten under andre forhold er uvisst. Derfor kan muligheten for å bruke andre «trackere» eller kombinering av disse være en god løsning.

Med tanke på estimering av posisjonsendring i bildets dybderetning fra løsningsalternativ 1, viser resultatene fra test 7.2, at det er en ulineær sammenheng mellom avstand til objektet og målt høyde på objektet som opptrer likt som den teoretiske modellen i 4.1.1. Det kan skape problemer hvis objektet som «trackes» kommer for nært kamera, da det oppfattes av programmet som en større forflytning i bildets dybderetning enn hva det egentlig er. Dette må tas hensyn til i regulering av ROV-ens posisjon, slik at pådraget ikke skal bli for stort når objektet er nært kameraet eller for lite hvis det er langt borte. Dette kan f.eks. gjøres manuelt av operatør ved å justere forsterkningen på regulator med hensyn til hvor langt ROV er fra objekt. Eventuelt hvis en har en måte å måle avstanden til objektet, kan det lages en algoritme som justerer forsterkning automatisk. Det er viktig å påpeke at dette stort sett vil skape problemer utenfor normalt arbeidsområde til en ROV.

Alternativ 4 – Aruco, er en mer sikker løsning for nøyaktighetskrevende operasjoner, da denne metoden ikke akkumuler feil over tid. Så lenge sikten er god nok og programmet klarer å gjenkjenne Aruco-markøren, kan en være sikker på translasjonen mellom ROV og objekt. Det vil kunne oppstå situasjoner der markør kan bli blokkert fra synsfeltet til kamera, som må håndteres av et eventuelt subprogram for regulering av ROV-ens posisjon. Da kan f.eks. input for posisjonsregulering bli koblet mot INS for å holde ROV stasjonært til ROV-pilot overtar. Dette vil også være relevant for bruk av alternativ 1.

Det endelige programmet tilbyr to løsninger på problemstillingen som ble fremstilt av Kystdesign. Den største fordelen ved å bruke «tracker»-metodene fra OpenCV er fleksibiliteten. Alle relevante objekter i bildet kan brukes som et settpunkt for regulering av posisjon. Hvis en har mulighet til å feste en Aruco-markør på objektet som skal «trackes», tilbyr Aruco-metoden en mer nøyaktig løsning på problemet.

9.1 Mulige forbedringer

Slik programmet er nå, håndteres ikke lagring av kalibreringsdata (forvrengingskoeffisient og kameramatrise) for flere kameraer, derfor må kameraet kalibreres på nytt hvis det byttes kamerakilde. Dette kan løses med å endre programkode slik at det kan legges til kamera-ID som kan knyttes opp mot kalibreringsdata.

«Computer vision» er under rask utvikling, derfor vil det komme nye og bedre «tracking»-algoritmer i fremtiden som kan implementeres i koden. Det er en årlig konkurranse «VOT (Visual Object Tracking) Challange» hvor det blir testet ut nye og innovative «trackere». Hvis ønskelig, kan disse «trackerne» implementeres i programkoden og eventuelt erstatte utdaterte «trackere».

Objektgjenkjennings-metoden med maskinlæring som YOLO, har vist seg å være en god metode for å følge et objekt i bildet. En av styrkene med denne metoden er at i motsetning til OpenCV-«trackerne», gir denne en indikasjon på hvor sikker den er på om objektet er gjenkjent. Det kan derfor tenkes at dette vil være en bedre metode å bruke i en eventuell autonom prosess. Ulempen er at den krever en del mer regnekapasitet og forarbeid med å lage en opplært maskinlæringsmodell, som må læres opp for alle objekt den skal kunne gjenkjenne. Hvis en kan sette av tid til dette, vil denne metoden være et godt supplement til programmet.

Videre arbeid som gjenstår er optimalisering og videreutvikling av programmet for bedre ytelse og responstid på «trackerne», da programmet nå krever litt mer regnekraft en det som er optimalt. Etter hvert som program ble utviklet og kode ble mer avansert, ble det avdekket at Tkinter hadde sine begrensinger med tanke på effektivitet i programutførelse når flere operasjoner skal kjøres samtidig. Ved å bruke et mer moderne rammeverk enn Tkinter, kan det lages et program som er mer effektivt. Det kan også være et godt alternativ å gå over til et webgrensesnitt, som vil gi lettere tilgang til programmet for bruk og feilsøking.

10 Referanser

[1] P. Corke, Robotics, Vision and Control, 2.edition, Springer Tracts in Advanced Robotics, 2017.

[2] H. D. Young, R. A. Freeman og A. L. Ford, Sears and Zemansky's University Physics with mordern physics 14th Edition, Santa Barbara: Pearson, 2016.

[3] Wikipedia, «Distortion (optics),» 16 april 2022. [Internett]. Available: https://en.wikipedia.org/wiki/Distortion_(optics). [Funnet 29 april 2022].

[4] D. A. Forsyth og J. Ponce, Computer Vision A Modern Approach, Pearson Education Limited, 2012.

[5] R. Szeliski, i *Computer Vision*, Springer, 2011.

[6] F. Algbretsen, «https://www.uio.no/,» 28 02 2018. [Internett]. Available: https://www.uio.no/studier/emner/matnat/ifi/INF2310/v16/forelesninger/inf2310-2016-07-filtrering-ii.pdf.

[7] R. E. W. Rafael C. Gonzalez, «Digital Image Processing,» i *Digital Image Provessing*, Pearson, 2017.

[8] J. M. Joseph Howse, Learning OpenCV 4 Computer Vision with Python 3, Packt Publishing Limited, 2020.

[9] M. G. H. B. Helmut Grabner, «Real-Time Tracking via On-line Boosting,» British Machine Vision Conference, Edinburgh, 2006.

[10] B. M.-H. Y. S. B. Babenko, «Visual tracking with online multiple instance learning,» IEEE Conference on computer vision and Pattern Recognition. IEEE, Miami, FL, USA, 2009.

[11] S. S. P. Yadav, «Understanding tracking methodology of kernelized correlation filter,» IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). IEEE, Vancouver, BC, Canada, 2018.

[12] T. V. L. C. Z. J. M. M. K. Alan Lukezic, «Discriminative Correlation Filter with Channel and Spatial Reliability,» Proceedings of the IEEE conference on computer vision and pattern recognition., 2017.

[13] Z. K. M. J. M. Kalal, «Tracking-learning-detection.,» IEEE transactions on pattern analysis and machine intelligence 34.7 (2011): 1409-1422., 2011.

[14] J. R. B. B. A. D. Y. M. L. David S. Bolme, «Visual object tracking using adaptive correlation filters,» Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, San Francisco, CA, USA, 2010.

[15] S. T. S. S. David Held, «Learning to Track at 100 FPS with Deep Regression Networks,» European Conference on Computer Vision, SpringerLink, Amsterdam, Netherlands, 2016.

[16] R. Rojas, «Lucas-Kanade in a Nutshell,» [Internett]. Available: http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/tutorials/Lucas-Kanade2.pdf. [Funnet 29 april 2022].

[17] M. Elgendy, Deep Learning for Vision Systems, Manning Publications, 2020.

[18] Community project, «ONNX: Open standard for machine learning interoperability,» 10 oktober 2017. [Internett]. Available: https://github.com/onnx/onnx. [Funnet 25 april 2022].

[19] J. Redmon, «YOLO: Real-Time Object Detection,» 9 mai 2016. [Internett]. Available: https://pjreddie.com/darknet/yolo/. [Funnet 25 april 2022].

[20] Community project, «PyTorch,» 1 oktober 2016. [Internett]. Available: https://github.com/pytorch/pytorch. [Funnet 26 april 2022].

[21] K. L. D. H. K. S. a. V. K. Rene Ranftl, «http://vladlen.info/papers/midas.pdf,» 2020. [Internett]. [Funnet 03 02 2022].

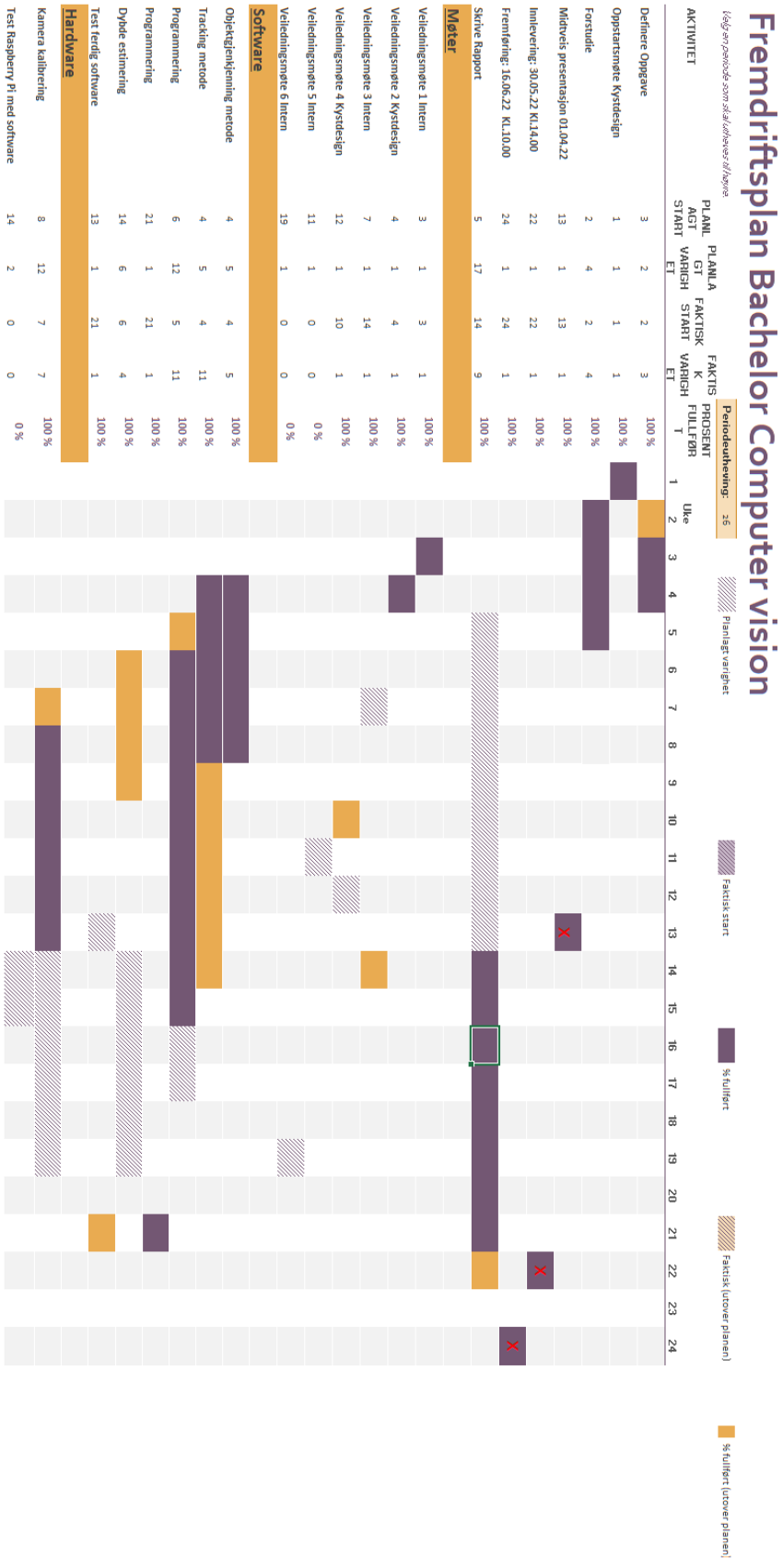
[22] F. J. Romero-Ramirez, R. Muñoz-Salinas, R. Medina-Carnicer, F. J. Madrid-Cuevas og S. Garrido-Jurado, «ArUco; http://www.uco.es/investiga/grupos/ava/node/26;,» [Internett]. Available: https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit.

[23] Python Software Foundation, «Python interface to Tcl/Tk,» 2001-2022. [Internett]. Available: https://docs.python.org/3/library/tkinter.html#. [Funnet 12 Mai 2022].

[24] T. Fiorenzani, «how_do_drones_work/opencv/,» 6 juni 2018. [Internett]. Available: https://github.com/tizianofiorenzani/how_do_drones_work/tree/master/opencv. [Funnet 23 mai 2022].

Figurliste

Appendiks A Fremdriftsplan



Appendiks B Programinstallasjon

For å gjøre det enkelt å ta i bruk Python scriptet uten å måtte installere nødvendige tillegg, er det omgjort til en ferdig programfil som inneholder alle bibliotek som trengs for å kjøre programmet. Programfilen er pakket sammen med nødvendige mapper og filer som kameramatrise og forvrengningskoeffisienter slik at programmet er klart for å testes, uten å måtte kalibrere kameraet.

Pakk ut den komprimerte mappen som ligger som et vedlegg; BO22EH_01_Program.zip

Start programmet ved å kjøre Main_Bach_CV.exe

Kilde for webkamera er satt til 0 som standard.

Vedlagt ligger det bildefil av Aruco-markør og sjakkbrett for testing av Aruco-metode og kalibrering.

Kildekode kan også finnes på Github:

https://github.com/bj8rnar/HVL-bachelor-computer-vision-2022.git

Appendiks C Vedlegg

Vedlegg til bachloroppgaven er listet opp nedenfor:

Kildekode for program:	Main_bach_CV.py
Klargjort program for test:	BO22EH_01_Program.zip
Timeliste:	Timeliste.xlsx
Fremdriftsplan:	Fremdriftsplan Gant.xlsx