Mikael Kalstad
Henrik Tronstad

Bacheloroppgave

Maskinsyn for hastighetsestimering av undervannsdrone

Hvor nøyaktig kan man estimere hastigheten til en undervannsdrone i en merd med feature detection, sammenliknet med optisk flyt?

Mai 2021

NTNU
Norges teknisk-naturvitenskapelige universitet.
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Mikael Kalstad
Henrik Tronstad

Maskinsyn for hastighetsestimering av undervannsdrone

Hvor nøyaktig kan man estimere hastigheten til en undervannsdrone i en merd med feature detection, sammenliknet med optisk flyt?

Bacheloroppgave
Mai 2021

NTNU
Norges teknisk-naturvitenskapelige universitet.
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Forord

Oppgaven som ble gitt av SINTEF Ocean AS gikk ut på å lage maskinsyns-algoritmer for en undervannsdrone. Vi jobbet med å utvikle og forske på løsninger for hastighetsestimering av undervannsdronen.

Vi valgte denne oppgaven gitt fra SINTEF Ocean AS, siden det virket som en interessant og lærerik utfordring. Vi har tidligere jobbet mye med prosjekter som involverer mye front-end og UI/UX, derfor ønsket vi en oppgave som kunne utvide horisontene våre.

Arbeidsprosessen har vært påvirket av korona-pandemien. Prosjektet har blitt gjennomført både digitalt, og fysisk på campus. Arbeidet har vært preget av eksperimentering og forskning. Vi brukte lean metode under utviklingen av løsningene.

Vi ønsker å takke veileder Jonathan Jørgensen for all hjelp og gode tips. Videre ønsker vi å takke oppdragsgiver og kontaktperson i SINTEF Ocean AS, Sveinung Johan Ohrem, for gode og hjelpsomme tilbakemeldinger underveis i prosjektet.

Dato: 19.05.2021

Sted: Trondheim

Kalstad, Mikael

Mikael Kalstad

Tronstad, Henrik

Henrik Tronstad

Oppgavetekst

Under ligger oppgaveteksten slik den ble publisert ved oppstart av bachelorprosjektet.

Arbeidstittel:	Maskinsyn for posisjonering og hastighetsestimering av undervannsrobot

Hensikten med oppgaven:
Utvikle maskinsynalgoritmer for posisjonering, hastighetsestimering, hulldeteksjon og groemengde for en undervanns notvaskerobot.

Kort beskrivelse av oppgaveforslag:
SINTEF Ocean AS jobber i prosjektet Netlean 24/7 med å utvikle autonome funksjoner for en undervanns, trådløs notvaskerobot som skal være permanent installert i merden. Denne roboten skal skånsomt børste noten med jevne mellomrom og dermed sørge for at vekster og organismer ikke får etablert seg. Roboten vil også utføre inspeksjonsoppgaver og måle relevante parametere med installerte sensorer. Ettersom GPS signaler ikke kan brukes under vann er posisjonering av roboten en utfordring. Roboten er utstyrt med et kamera og det er derfor et ønske om å bruke maskinsynsalgoritmer til å beregne robotens posisjon og hastighet ved å se på maskestrukturen i noten.

Følgende punkter foreslås som en del av oppgaven:

1) Utvikle maskinsynalgoritmer for sanntids estimering av posisjonen til notvaskeroboten i to dimensjoner.
2) Utvikle maskinsynalgoritmer for å detektere hull i noten under operasjoner.
3) Utvikle maskinsynalgoritmer for å detektere groe på noten under operasjoner.

Prosjektgruppen vil få tilgang til følgende:
1) Medveileder(e) fra SINTEF Ocean AS
2) Mulighet til å teste de utvikle algoritmene i feltforsøk.

Følgende leveranser er ønsket:
1) Algoritme for estimering av posisjon og hastighet
2) Algoritme for deteksjon av hull i not
3) Algoritme for deteksjon av groe på not
4) Brukerveiledning for SINTEF Ocean og sluttbrukere
5) Sluttrapport med beskrivelse av prosessen, programvaren mm.

For mer informasjon, kontakt Sveinung Ohrem (sveinung.ohrem@sintef.no / 93664407)

Underveis i prosjektet ble det avklart at gruppen kun skulle fokusere på det første punktet for oppgaven; utvikle maskinsyns-algoritmer for sanntids-estimering av posisjonen til en undervannsdrone i to dimensjoner. I løpet av prosjektet endret fokuset seg til estimering av hastigheten til undervannsdronen i én dimensjon. Algoritmene for deteksjon av hull eller groe i nettet ble sterkt nedprioritert, og det ble avtalt under møter tidlig i prosjektet at gruppen skulle se bort fra de andre algoritmene.

Det ble også utarbeidet et visjonsdokument i samarbeid mellom gruppen og oppdragsgiver, i tillegg til kravene gitt i oppgaveteksten. Visjonsdokumentet ligger i rapporten som vedlegg B. I visjonsdokumentet er funksjonelle egenskaper og brukerbehov angitt i nærmere detalj.

Sammendrag

I dette prosjektet har vi forsket på to ulike maskinsyns-algoritmer for hastighetsestimering av en undervannsdrone. Løsningene er produsert for SINTEF Ocean AS, hvor de skal brukes på en undervannsdrone som beveger seg langs et nettet i en merd. Vi har eksperimentert med flere ulike algoritmer, men vi valgte å forske på feature matching og optical flow. Fokuset i dette prosjektet var å sammenligne nøyaktigheten til feature matching og optical flow for hastighetsestimering av en undervannsdrone i en merd.

På grunnlag av dette kom vi fram til vår problemstilling: Hvor nøyaktig kan man estimere hastigheten til en undervannsdrone i en merd, ved bruk av feature matching sammenliknet med optical flow?

Det eneste av input til algoritmene er video-data fra undervannsdronen. Video-ressursene vi har brukt til testing, besto av både simulerte nett og opptak fra en reell merd. En utfordring er at nettet er visuelt uniformt. Dette gjør det vanskelig å behandle video-data, og beregne bevegelsen til undervannsdronen.

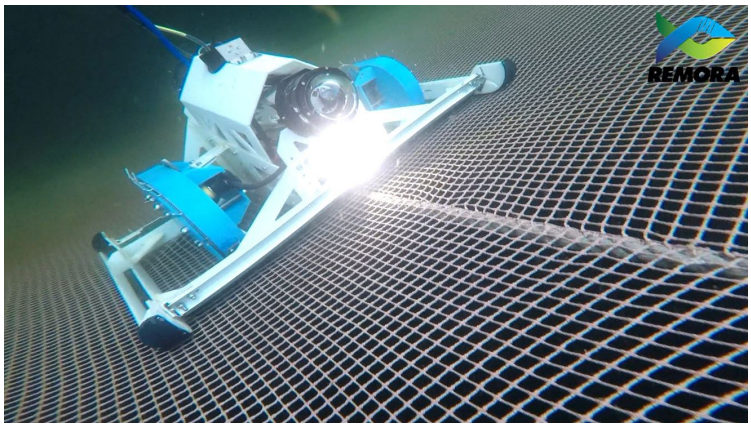
Resultatet etter testing og forskning viser at begge algoritmene er uegnet til praktisk bruk. Til tross for dette, er fortsatt feature matching mer nøyaktig enn optical flow ved høyere hastigheter i det simulerte miljøet. Optical flow er derimot mer nøyaktighet enn feature matching ved lavere hastigheter i simulert miljø. Feature matching er mer nøyaktig enn optical flow ved alle hastigheter for video-data fra en reell merd. Algoritmene gir forskjellige resultater, men generelt er ingen av de egnet til praktisk bruk.

Innholdsfortegnelse og ev. figur- og tabelliste

Kapittel 1: Introduksjon og relevans

SINTEF Ocean AS er en del av SINTEF-konsernet, som driver med innovasjon og forskning knyttet til havet. SINTEF Ocean AS driver med marinteknisk- og biomarin-forskning [1].

I prosjektet NetClean 24/7 [2] jobbet SINTEF Ocean AS med å utvikle autonome funksjoner for en undervannsdrone. Denne dronen er en notvaskerobot som skal forhindre vekst av groe på nettet. I tillegg skal dronen brukes for inspeksjon av nettet for å blant annet detektere hull i nettet. I forbindelse med dette prosjektet ønsket bedriften å utvikle ulike maskinsyn-algoritmer til denne dronen. Den første og mest aktuelle algoritmen gitt i oppgaven, går ut på å bruke kameraet installert på dronen til å estimere posisjon og hastighet.



Figur 1 - Remora prototype av en notvaskerobot utviklet av Mithal AS[1]

Det er et kjent problem at GPS er unøyaktig eller helt ufunksjonell under en viss dybde i vann [3]. Derfor har bedriften valgt å bruke kamera som eneste sensor for posisjonering. Bedriften har også planer for andre sensorer som skal brukes i tillegg til maskinsyns-algoritmene.

[1] "NetClean 24/7 - SINTEF." 10 sep.. 2019, https://www.sintef.no/prosjekter/2019/netclean-247/. Åpnet 27 apr.. 2021.

Denne rapporten er delt opp i seks ulike deler. Kapittel 2 går gjennom teorien som ligger til grunnlag for å forstå konseptene som brukes i prosjektet. I kapittel 3 finner man informasjon om valgene vi tok underveis i prosjektet, samt teknologier som ble brukt og hvordan problemer ble løst. Kapittel 4 inneholder resultater av testing, som blir diskutert i kapittel 5. I kapittel 6 konkluderer vi, og greier ut om eventuelt videre arbeid. Etterfulgt av disse kapitlene finner man kilder og alle vedlegg.

Vedleggene i denne rapporten er kategorisert med bokstaver. Vedlegg A er tester og B er visjonsdokumentet. Prosjekthåndboka ligger i en separat fil.

Forkortelser og terminologi

AS - Aksjeselskap

FAST - Features from Accelerated Segment Test

Git - Versjonskontrollsystem

GPS - Global Positioning System

NTNU - Norges Teknisk-Naturvitenskapelige Universitet

OpenCV - Open Source Computer Vision Library, et bibliotek for bildebehandling i python

ORB - Oriented FAST and Rotated BRIEF

Optical Flow - Optisk Flyt

PX - Piksel

RMSE - Root Mean Square Error

SIFT - Scale-invariant feature transform

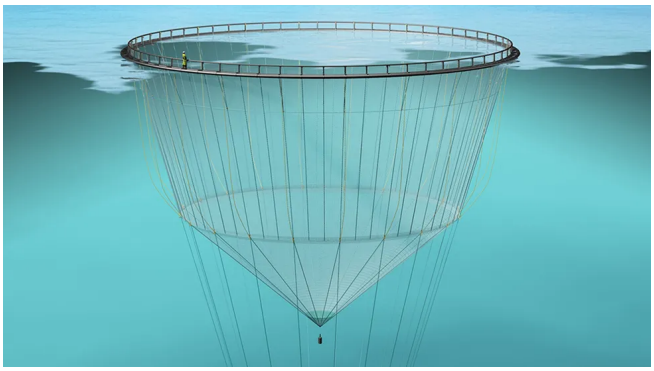
UI - User Interface

UX - User Experience

Kapittel 2: Teori

Not og merd

En merd er en notpose som holdes av et flytende rammeverk. Et not-nett er nettet inne i merden som holder fisken inne [4]. Videre i rapporten vil vi referere not-nettet som nettet.

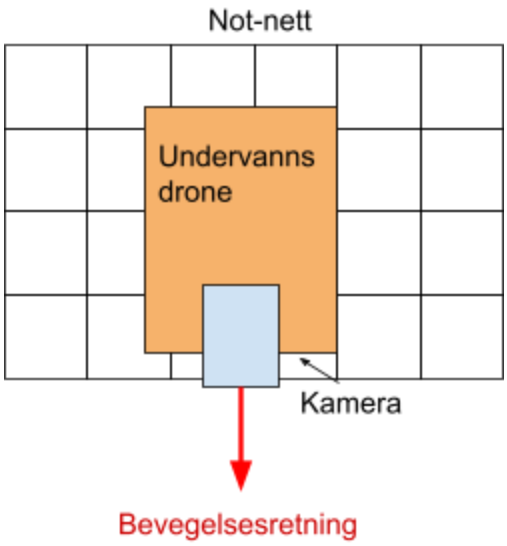


Figur 2 - Illustrasjon av en merd[2]

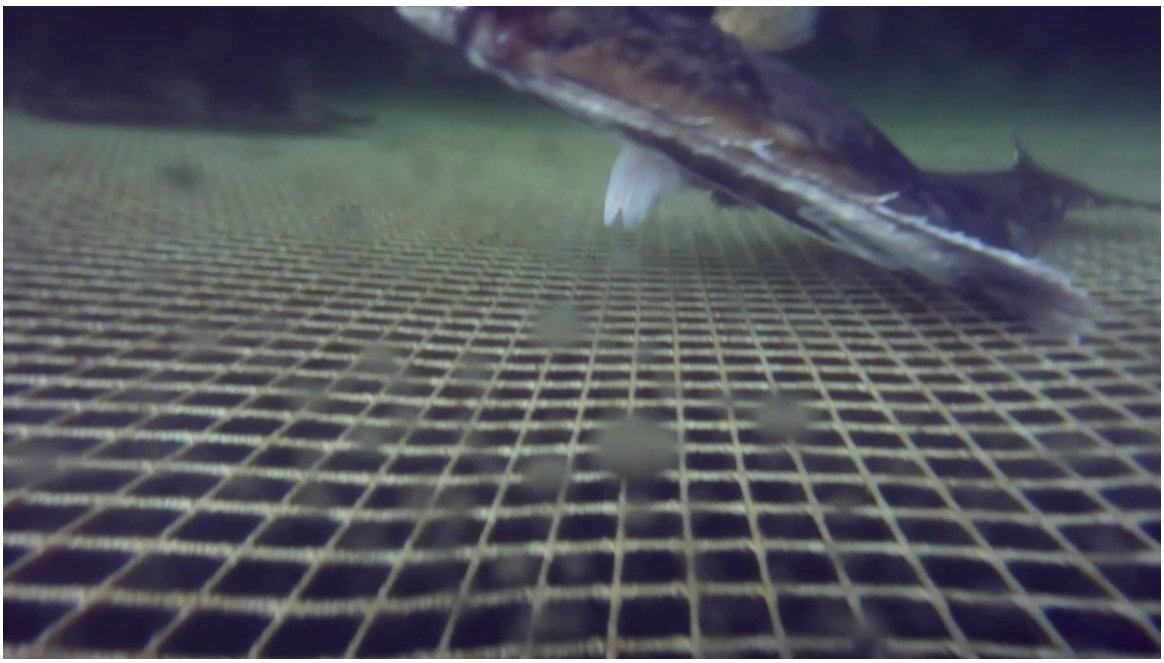
Undervannsdrone og kamera

Undervannsdronen beveger seg langs nettet og er utstyrt med et kamera, som illustrert i figur 3. Kameraet er plassert foran på undervannsdronen og ser fremover med bevegelsesretningen. Roboten har et kamera som er vinklet ned mot nettet. Figur 4 viser et skjermbilde fra kameraet i et ekte video-opptak med undervannsdronen.

[2] "Kjegleformet notpose - Egersund Net." https://www.egersundnet.no/produkter/notposer/kjegleformet-notpose. Åpnet 21 apr.. 2021.



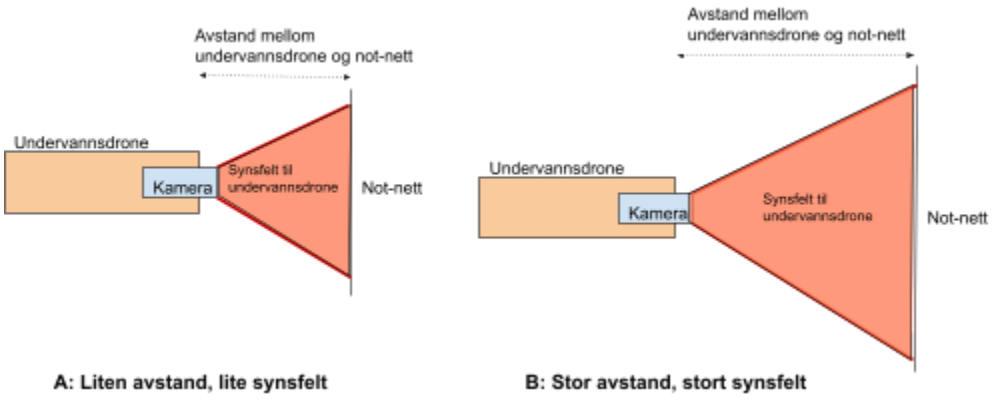
Figur 3 - Illustrasjon av undervannsdrone som beveger seg langs nettet.



Figur 4 - Skjermbilde fra video-opptak av undervannsdrone i merd

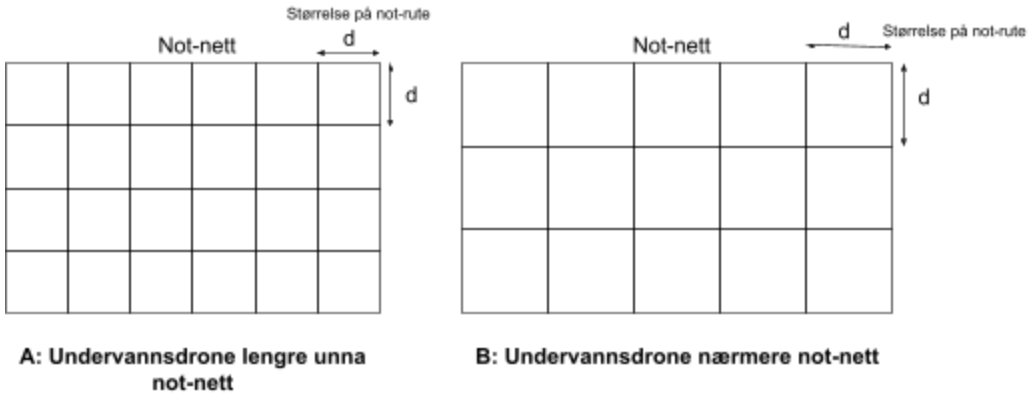
Plassering av undervannsdrone og tilhørende kamera

Hvordan undervannsdronen er plassert i forhold til nettet har mye å si for ytelsen til algoritmene. Hvis undervannsdronen er plassert nært nettet, vil man få færre not-ruter per bilde som vil gjøre det vanskeligere å estimere en hastighet nøyaktig. Grunnen til dette er at jo nærmere undervannsdronen er nettet, jo fortere beveger nettet seg relativt til kameraet. Dette fenomenet kalles bevegelses-parallakse [5]. I figur 5 ser man hvordan synsfeltet endrer størrelse ved endring av avstand mellom undervannsdrone og nettet.



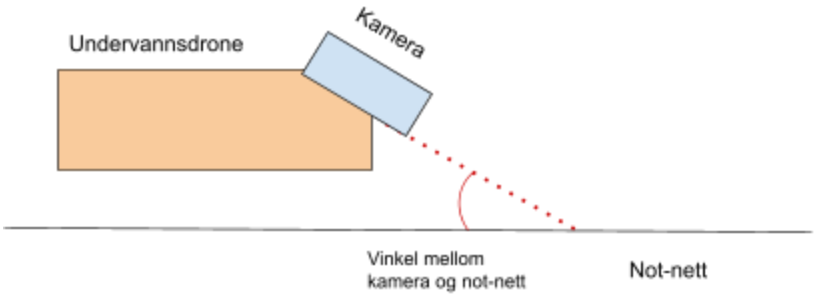
Figur 5 - Synsfeltet til undervannsdronen med ulik avstand til nettet

Størrelsen på not-rutene og hastigheten til undervannsdronen er konstant, uavhengig av avstand mellom undervannsdrone og nettet. Dette er illustrert i figur 6. Hvis undervannsdronen beveger seg nærmere nettet, vil den se mindre fysisk avstand i synsfeltet over samme tid siden hastigheten er konstant. nettet vil derfor tilsynelatende se ut som det beveger seg raskere når undervannsdronen er nærmere.



Figur 6 - nettet sett fra perspektivet til undervannsdronen med ulik avstand

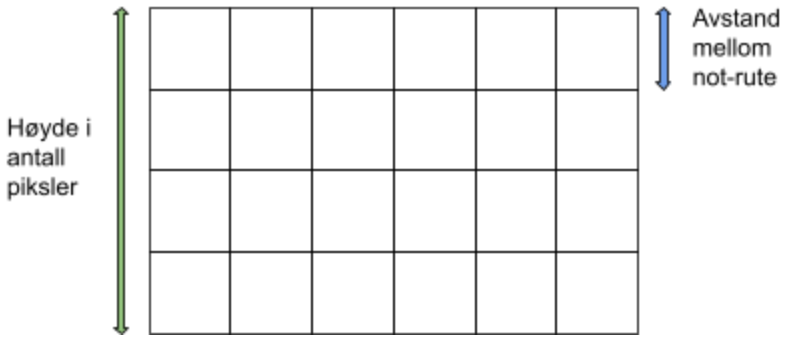
Vinkelen til kameraet på nettet har også mye å si for ytelsen til algoritmene. Hvis kameraet er vinklet slik at det ligger nærmest parallelt med nettet, kan det være vanskeligere å gjøre en god perspektiv transformasjonen. Dessuten vil man få færre not-ruter i bildet etter perspektiv transformasjonen hvis kameraet står nærmest parallelt på nettet. Kameraet bør heller ikke stå normalt på nettet hvis avstanden mellom undervannsdronen og nettet er liten. Grunnen til dette er at man får mindre ruter i bildet. Vinkelen mellom kamera og nettet er illustrert i figur 7.



Figur 7 - Illustrasjon av undervannsdrone med vinklet kamera på nettet

For å kunne estimere en hastighet trenger algoritmene en konstant for avstand i meter per piksel for video-ressursen. Dette er illustrert i figur 8. Denne konstanten finner man ved:

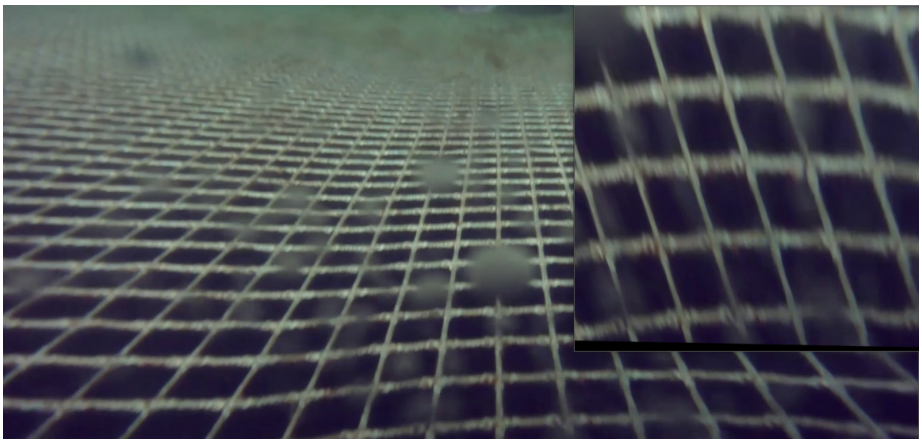
$$d_{px} = \frac{Avstand\ mellom\ not{-}rute * Antall\ not{-}ruter}{Høyde\ til\ video{-}ressurs} \quad (6.0)$$



Figur 8 - Illustrasjon av nettet med beskrivelse av parametre for beregning av d_x

Perspektiv-transformasjon

Perspektiv-transformasjon er en metode for å skifte kameraets perspektiv i bildet. En perspektiv-transformasjon [6] utføres ved å velge ut egnede punkter i bildet, som skal utgjøre det nye flate bildet. Deretter utføres forskjellige geometriske transformasjoner. Transformasjonen vil beholde kolineariteten til de valgte punktene. Dette vil si at punktene er justert til å være på samme linje med hverandre som før transformasjonen. Figur 9 viser hvordan perspektiv-transformasjon ser ut på et skjermbilde fra ett opptak fra undervannsdrone i merd.

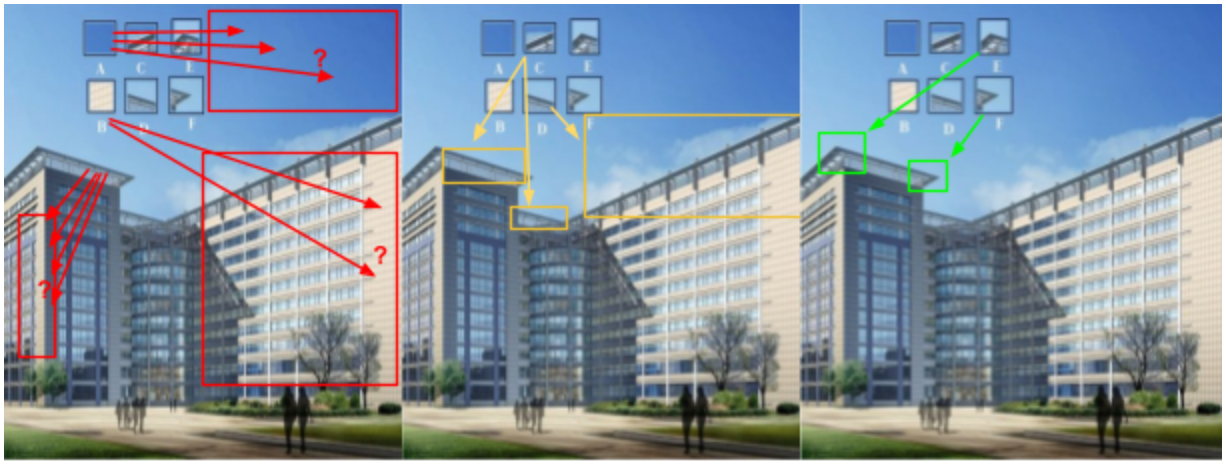


Figur 9 - Skjermbilde av video-opptak fra merd med perspektiv-transformasjon i høyre hjørne.

Feature detection

Feature detection [14] handler om å finne områder i et bilde som er lett å identifisere, og dermed differensierer bildet fra andre bilder. Disse områdene kaller man keypoints eller features, videre i rapporten kalles disse områdene for interessepunkter. En algoritme vil finne et interessepunkt med en X- og Y-posisjon til en spesifikk piksel ved å se på et gitt område rundt denne pikselen . Et interessepunkt vil derfor ha et tilhørende område rundt seg, siden man ikke kun kan se på egenskapene til en enkelt piksel.

Man kan bruke analogien med puslespill for å lettere forstå konseptet interessepunkter. Når man skal finne posisjonen til en brikke i et puslespill, vil man lete etter brikker som man kan identifisere eller plassere i bildet på esken. En puslespillbrikke kan ses på som et interessepunkt. Noen brikker vil være lette å plassere, mens posisjonen til andre brikker er vanskelig å identifisere. Kvaliteten til et interessepunkt vil derfor variere ettersom hvor vanskelig det er å identifisere interessepunktet i bildet.



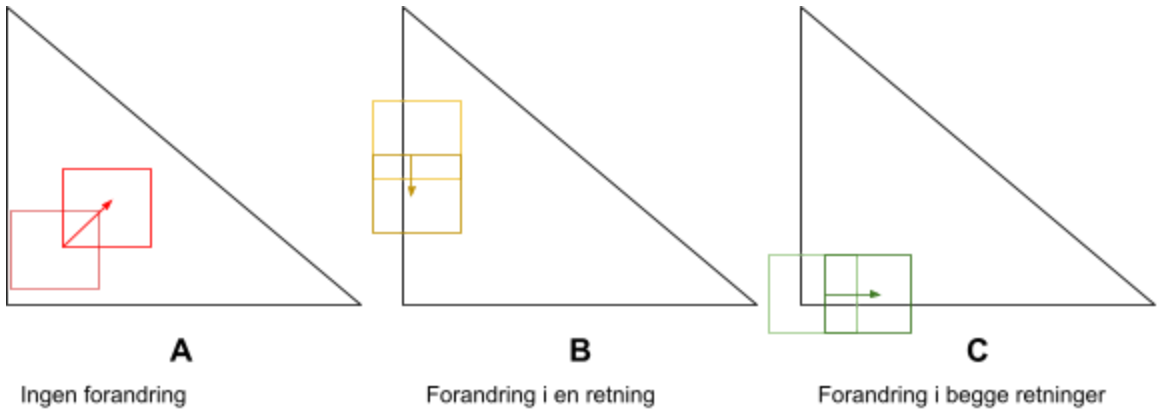
Figur 10 - Kvaliteten til et interessepunkt[3]

[3] "Understanding Features - OpenCV." https://www.docs.opencv.org/master/df/d54/tutorial_py_features_meaning.html. Åpnet 21 apr.. 2021.

I figur 10 er det illustrert hvordan noen interessepunkter er vanskeligere å identifisere enn andre. Interessepunkt A og B i figur 10 er angitt til å være av lav kvalitet, C og D av medium kvalitet og E og F av høy kvalitet. Områdene som de ulike interessepunktene kan ligge i, er markert i sine respektive farger. Legg merke til at området er større jo lavere kvalitet interessepunktet er angitt som. I tillegg er det verdt å merke seg at interessepunktene av høy kvalitet er hjørner.

Harris Corner Detection

Harris Corner Detection [7] er en algoritme som kan identifisere hjørner i et bilde. Algoritmen fungerer ved å bevege et vindu rundt i bildet, og deretter analysere intensiteten til pikslene i vinduet, både før og etter bevegelsen.



Figur 11 - Harris Corner Detection illustrasjon

I figur 11 er det illustrert hvordan Harris Corner Detection bruker forandringen i intensiteten til pikslene i vinduet til å finne hjørner. Når det ikke er noen forandring (A i figur 11), kan man være helt sikker på at man ikke har funnet noen hjørner i området. Når det er forandring i en retning (B i figur 11), vil man ha funnet en kant eller linje, men ikke et hjørne. Derimot når det er forandring i både X- og Y-retning (C i figur 11) vil man ha funnet et hjørne i området man søkte. Dette kan beskrives matematisk:

$$E(u,v) \;=\; \sum_{x,y} w(x,y)\,[I(x+u,\;y+v) \;-\; I(x,y)]^2 \quad (1.0)$$

I ligning 1.0 er $w(x,y)$ vindusfunksjonen, $I(x+u,\;y+v)$ er intensiteten i det skiftende vinduet, og I(x,y) er intensiteten i det originale vinduet. Problemet med denne ligningen er at den er veldig ineffektiv og krever dermed mye prosesseringskraft eller tid for å utregnes. Man kan derfor bruke en Taylor-rekke for å finne en estimering til ligning 1.0:

$$E(u,v) \;=\; [u\;v]\,M\,[u\;v]^T \quad (1.1.0)$$

$$M \;=\; \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix} \quad (1.1.1)$$

Ligning 1.1.0 er estimeringen av ligning 1.0. Ligning 1.1.1 er en utvidelse av ligning 1.1.0 hvor M beskrives. Ligning 1.1.0 gir et resultat som inneholder to ulike egenverdier. Hvis en av egenverdiene er betraktelig større enn den andre, har man funnet en kant. Hvis begge egenverdiene er små, har man funnet et flatt område. Derimot hvis begge egenverdiene er store, har man funnet et hjørne. Dette kan gjøres lettere ved å bruke en score-ligning:

$$R = det(M) - K(trace(m))^2 \quad (1.2)$$

M har to egenverdier $\lambda_1\lambda_2$, ligning (1.2) kan dermed skrives slik:

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \quad (1.3)$$

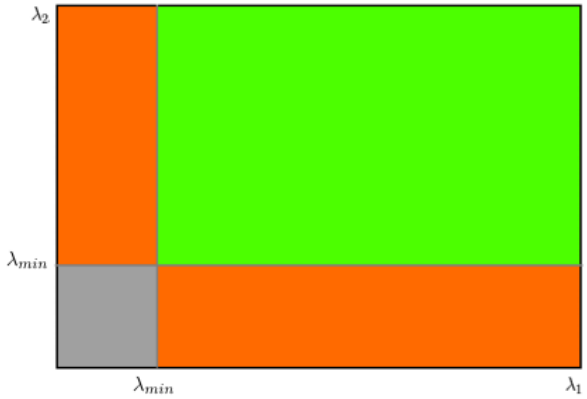
Når absoluttverdien til R er liten, vil det være et flatt område. Når R < 0 vil det være en kant i området. Når R er stor vil det være et hjørne i området. Ved å se på verdien til R (ligning 1.2) med en satt grenseverdi kan en datamaskin finne hjørner i et bilde.

Shi-Tomasi

Shi-Tomasi Corner Detector [8] er en utvidelse av Harris Corner Detection. Denne algoritmen forbedrer Harris Corner Detection ved å endre på score-ligningen (1.3) til:

$$R = min(\ \lambda_1 \lambda_2\ ) \quad (1.4)$$

Når λ_1 og λ_2 er over en satt grenseverdi λ_{min}, vil det regnes som et hjørne. Figur 12 illustrerer denne nye score ligningen.

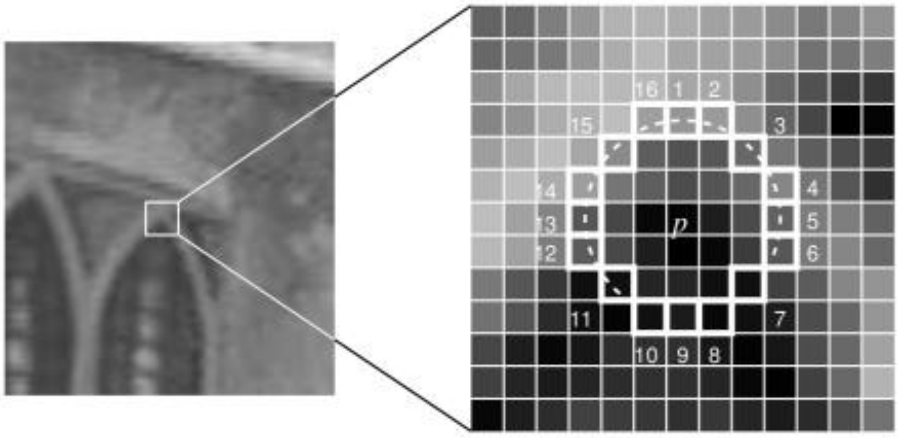


Figur 12 - Illustrasjon av Shi-Tomasi Corner Detector score ligning[4]

FAST

FAST (Features from Accelerated Segment Test) [9], [10] er også en algoritme som skal finne hjørner i et bilde. Formålet med algoritmen er at den skal være såpass rask at man skal kunne bruke den i applikasjoner som kjører i sanntid.

[4] "Shi-Tomasi Corner Detector & Good Features to Track - OpenCV." https://docs.opencv.org/master/d4/d8c/tutorial_py_shi_tomasi.html. Åpnet 22 apr.. 2021.



Figur 13 - FAST analyse av en enkel piksel[5]

FAST-algoritmen fungerer slik:

1. En piksel i bildet velges. Intensiteten til pikselen kan kalles I.
2. En grenseverdi velges. Denne verdien kan kalles t.
3. En sirkel på 16 piksler rundt pikselen velges.
4. En test kjøres for å analysere intensiteten til pikslene i sirkelen i forhold til I. Hvis N-antall piksler er mer eller mindre intens enn I, vil man anse pikslene som et hjørne.
    a. En piksel er lysere ved at intensiteten er høyere enn I+t.
    b. En piksel er mørkere ved at intensiteten er lavere enn I-t.
5. Algoritmen starter ved å se på piksel 1 og 9, og deretter 5 og 13 hvis de første passerer testen beskrevet i forrige punkt. Analysen fortsetter helt til man har sjekket N-antall piksler. For best mulig resultat anbefales det å velge N større eller lik 12.

Figur 6 viser en illustrasjon av hvordan FAST analyser hvert enkelt piksel.

[5] "FAST Algorithm for Corner Detection - OpenCV." https://docs.opencv.org/master/df/d0c/tutorial_py_fast.html. Åpnet 21 apr.. 2021.

Feature-deskriptorer

En deskriptor er en beskrivelse av et interessepunkt som kan brukes for å differensiere interessepunkter fra hverandre. Man kan se på en deskriptor som et digitalt fingeravtrykk for hvert interessepunkt som muliggjør effektiv matching av interessepunkter [10].

Disse deskriptorene blir kalkulert ved å konvertere relevant informasjon om eller i nærheten av et interessepunkt, til et digitalt avtrykk i form av en vektor. Man skiller mellom deskriptor-vektorer inneholder binære tall og deskriptor-vektorer som inneholder flyttall. En deskriptor skal være uavhengig av plasseringen til interessepunktet og skal dermed også være robust mot transformasjon.

BRIEF

BRIEF (Binary Robust Independent Elementary Features) [11] er en algoritme som finner deskriptor til et interessepunkt.

Figur 14 - Illustrasjon av en BRIEF analyse i et interessepunkt[6]

[6] "Introduction to ORB (Oriented FAST and Rotated BRIEF) | Data Breach." 1 jan.. 2019, https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf. Åpnet 21 apr.. 2021.

BRIEF fungerer slik:

1. Bildet behandles først med en Gaussian kernel, som er en konvulsjon for å fjerne støy og jevne ut bildet. BRIEF algoritmen bruker informasjon fra enkle piksler og er derfor sensitiv til støy.
2. To piksler, P1 og P2, velges tilfeldig i et gitt område rundt punktet. Området er et firkant med en gitt høyde og bredde. Høyden og bredden til området er et parameter som kan endres i algoritmen for å oppnå ulike resultater.
3. P1 velges fra en gauss distribusjon med senter i punktet, og et standardavvik på sigma σ.
4. P2 velges fra en gauss distribusjon med senter i den første pikslen, og et standardavvik på $\frac{\sigma}{2}$.
5. Testen sammenligner deretter intensiteten til P1 og P2 for å velge en bit-verdi som svar på testen.
    a. Hvis P1 > P2 vil bit-verdien være 1
    b. Hvis P2 > P1 vil bit-verdien være 0
    c. Dette kan skrives matematisk ved ligningen:

    (Latex: τ(I;P1,P2)=\begin{cases}1 & :I(P1)>I(P2)
    0 & :I(P2)<I(P2)\end{cases})

    $$(I; P1, P2) = \begin{cases} 1 & : I(P1) > I(P2) \\ 0 & : I(P2) < I(P2) \end{cases} \quad (2.0)$$

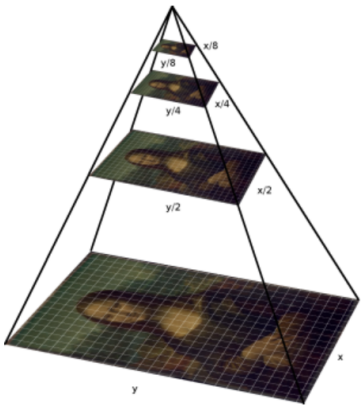
    Hvor $I(P1)$ er intensiteten til piksel $P1$
6. Algoritmen har dermed funnet en bit-verdi for testen, og vil fortsette N-antall ganger og lagre bit-verdiene i en vektor. For å oppnå tilfredsstillende resultater bør det velges en N-verdi på 128 eller 256.

Vektoren til en deskriptor til et interessepunkt kan beskrives matematisk:

$$v(n) = \sum_{1<i<n} 2^{i-1} \tau(p; x_i, y_i) \, (2.0)$$

ORB

ORB (Oriented FAST and Rotated BRIEF) [12] er et gratis alternativ til SURF- og SIFT-algoritmen som er patentert i OpenCV og kan dermed ikke brukes til kommersielt bruk uten kostnader [13]. ORB er en hybrid-algoritme som bruker både FAST og BRIEF, med ulike modifikasjoner for effektivisering og økt presisjon. ORB tar inn flere parametre, deriblant antall interessepunkter den skal finne, la denne verdien være N. ORB bruker FAST til å finne interessepunkter, og BRIEF for å finne deskriptorer.



Figur 15 - Illustrasjon av en bildepyramide[7]

FAST er en effektiv metode for å finne interessepunkter raskt, men den tar ikke hensyn til rotasjon og skalering. ORB løser dette ved å bruke en flerskala bildepyramide, som er en samling av bildet i ulike oppløsninger. Det kalles en bildepyramide siden bildene skaleres i flere omganger og kan visualiseres som en pyramide (se figur 15). ORB vil bruke FAST til å detektere

[7] "Introduction to ORB (Oriented FAST and Rotated BRIEF) | Data Breach." 1 jan.. 2019, https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf. Åpnet 21 apr.. 2021.

interessepunkt for alle bildene i bildepyramiden. Man har dermed gjort ORB delvis invariant til skalering. Dette kalles oFAST (FAST Keypoint Orientation).

Et annet problem med FAST er at den ofte finner interessepunkter langs kanter og ikke i hjørner. ORB løser dette ved å ha en lav nok terskel for å nok antall interessepunkter, og deretter filtrere interessepunktene ved bruk av Harrison Corner Detection. Man får da en sortert liste med de beste interessepunktene først i listen. Etter dette velger man N antall interessepunkt fra den sorteste listen.

Et av problemene med BRIEF er at hvis bildet roteres mer enn et par grader vil man se stor forandring i interessepunktene som algoritmen finner. Det man ønsker er å finne de samme interessepunktene uansett rotasjon. ORB løser dette ved å bruke en metode for å styre BRIEF i henhold til rotasjonen til interessepunktene. Dette kalles rBRIEF (Rotation-Aware Brief). For hvert interessepunkt i lokasjon (x_i, y_i) har man en 2xn matrise S:

$$S = \begin{pmatrix} x_1, \dots x_n \\ y_1, \dots y_n \end{pmatrix} (3.0)$$

Man kan lage en rotert versjon av S ved å bruke rotasjon til interessepunktet θ og den tilhørende rotasjonsmatrisen R_θ.

$$S_\theta = R_\theta S \, (3.1)$$

Den styrte BRIEF blir da:

$$g_n(p, \theta) := f_n(p) | (x_i, y_i) \in S_\theta \, (3.2)$$

Feature matching

Feature matching [14] handler om å sammenligne interessepunkter med tilhørende deskriptorer i to bilder, og finne likheter mellom bildene. Kvaliteten på interessepunktene og deskriptorene vil ha stor sammenheng med kvaliteten på matchene. Valg av algoritme for interessepunkt og deskriptor er derfor viktig i sammenheng med feature matching. Det finnes hovedsakelig to typer match-algoritmer i OpenCV; Brute-Force- og FLANN-matcher.

Brute-Force matcher

Denne algoritmen fungerer ved at den sammenligner en deskriptor til et interessepunkt i et datasett, med alle deskriptorer i et annet datasett. Et datasett i denne sammenhengen er deskriptorer og interessepunkt for et spesifikt bilde. Sammenligningen av to deskriptorer foregår ved å se på en distanse verdi mellom deskriptorene.

For binære deskriptorer vil man sammenligne ved å se på hamming avstanden [15]. Hamming avstanden kan kalkuleres ved å gjennomføre en XOR-operasjon mellom to bit-strenger. Antall bits som er 1 etter XOR-operasjonen vil være hamming avstanden. Under vises et eksempel på hvordan man kalkulerer hamming avstanden d_h for to deskriptorer p og q.

$$p = 011,\ q = 110,\ p \oplus q = 101,\ d_h(p,q) = 2\text{, siden 101 inneholder to bits med verdi 1}$$

For flyttall deskriptorer vil man sammenligne ved å se på den euklidske avstanden [16]. Den euklidske avstanden mellom to deskriptorer p og q av lengde n er gitt ved:

$$d_e(p,q) = \sqrt{((p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} \quad (4.0)$$

En Brute-Force matcher vil sammenligne alle mulige deskriptorer, og vil derfor alltid finne den beste matchen, men på bekostning av ytelse.

FLANN matcher

Fast Library for Approximate Nearest Neighbors [17], forkortet FLANN, er en samling av flere algoritmer som er optimalisert for å raskt finne nærmeste nabo i store datasett. Den inneholder blant annet algoritmer som LSH (locality-sensitive hashing) og k-d tree (space-partitioning datastruktur algoritme).

Fordelen med FLANN matcher er at den er raskere enn Brute-Force matcher. FLANN matcher kan derimot gi dårligere resultater siden nearest-neighbor algoritmen gjetter på hva som er nærmeste nabo. Ved å endre på parametrene til algoritmen kan man oppnå gode resultater med FLANN matcher uten bekostning på ytelsen.

Optical flow

Optical flow [18] kan måle bevegelse basert på intensitet-nivået i områder av bildet over tid. Enkelte bevegelser vil ha konstant intensitet over kort tid. Dette gir bevegelsesmønstre i bildene, forårsaket enten av kamerabevegelser eller bevegelser av objekter. Intensiteten kan finnes ved

$$I(x,t) = I(x + dx,\ t + dt) \quad (5.0)$$

Hvor dx og dy er forflytningen i lokale regioner, over tid dt. Man kan så ta Taylorrekke-tilnærmingen på venstre side, for å få følgende ligning:

$$I(x,t) = I(x,t) + \nabla I \cdot dx + dtI_t + O^2 \quad (5.1)$$

Hvor: $\nabla I = (I_x, I_y)$ *og* I_t er første-ordens partiell-deriverte av $I(x,t)$, *og* O^2, de høyere ordens partiell-deriverte som kan utelates. Ved å subtrahere $I(x, t)$ på begge sider og dividere med dt gir det:

$$\nabla I \cdot v + I_t = 0 \quad (5.2)$$

Her er $\nabla I = (I_x, I_y)$ den romlige intensitets-gradienten, $v = (u, v)$ er bilde-hastigheten i form av en vektor. Likning (5.2) kalles optical flow-likningen. Beregningen av optical flow kan gjøres ved Lucas-Kanade-Metoden [19].

Figur 16 - Optical flow visualisert[8]

[8] "Optical Flow - OpenCV." https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html. Åpnet 29 apr.. 2021.

Kapittel 3: Valg av teknologi og metode

Kravene spesifisert i visjonsdokumentet ligger som grunnlag for valg av teknologi og metode. I tillegg til dette tok vi med i betraktning tilbakemelding fra oppdragsgiver og veileder, samt tidligere erfaring med prosjektarbeid. Dessuten tilegnet vi oss ny kunnskap og lærdom underveis i prosjektet.

Arbeids- og rollefordeling

I dette prosjektet så vi ikke et behov for å ha en gruppeleder siden vi kun var to i gruppen. Dette prosjektet involverte mye eksperimentering av ulike metoder og teorier. I starten av prosjektet satt vi sammen og jobbet med ulike metoder og teorier for maskinsyn. Etterhvert som vi snevret oss inn i ulike løsninger, jobbet vi sammen om disse.

Vi hadde noen ansvarsområder underveis i prosjektet:

- Ansvarlig for git og repositoriet: Mikael Kalstad
- Ansvarlig for alt som omhandler møter: Henrik Tronstad
- Ansvarlig for feature matching-algoritmen: Mikael Kalstad
- Ansvarlig for optical flow-algoritmen: Henrik Tronstad

Arbeidsprosess

Dette prosjektet startet som et utviklingsprosjekt, men ble preget av forskning mot slutten. I starten valgte vi å bruke lean utviklingsmetode. Vi var en liten gruppe og det passet derfor for eksempel ikke med scrum, som krever en scrum-master. I tillegg hadde vi ingen daglige møter for å gjennomgå hva som har blitt gjort, og hva som skal gjøres videre. Vi jobbet tett gjennom hele prosjektet og møttes hovedsakelig fysisk på campus, men også digitalt i perioder på grunn av korona-pandemien. Vi jobbet vanligvis på mandager, tirsdager og onsdager.

I starten av prosjektet brukte vi en del tid på å sette oss inn i problemstillingen. Det var også behov for å lære oss grunnleggende bildebehandlings-algoritmer, samt OpenCV for å eksperimentere med forskjellige maskinsyns-algoritmer. Mot slutten av prosjektet gikk arbeidet over fra å finne algoritmer som kan fungere, til å finne ut hvorfor noen utvalgte algoritmer ikke fungerer optimalt. Arbeidet ble altså mer forsknings-preget utover i prosjektet.

På grunn av innholdet i oppgaven var det ingen behov for å utvikle wireframes, use-case eller user-stories. Selve kravene til algoritmene står beskrevet i visjonsdokumentet som vi utviklet i samarbeid med oppdragsgiver. Vi benyttet ikke issue board i GitLab siden prosjektet var eksperimentelt, og vi hadde dermed ikke noen spesifikke arbeidsoppgaver.

Vi hadde møte med oppdragsgiver omtrent annenhver uke. Dette gjorde vi for å holde oppdragsgiver oppdatert på fremdrift, samt stille spørsmål og gi muligheten for tilbakemeldinger. Vi hadde også møter med veileder mot slutten av prosjektet, for å komme med spørsmål til den administrative delen av prosjektet som for eksempel dokumentasjon og forskerspørsmål. Møtene med veileder ble gjennomført sjeldnere på grunn mengden timer veileder har allokert til denne type arbeid.

Vi skrev hver vår timeliste med en beskrivelse av arbeidet som ble utført denne dagen. I slutten av hver uke skrev vi en statusrapport som beskriver arbeidet som ble gjort den uken.

Valg av teknologier

Maskinsyns-bibliotek

Tidlig i prosjektet så vi et behov for et rammeverk eller bibliotek for bildebehandling. Målet med dette biblioteket var at det skulle gjøre det lettere for oss å eksperimentere med ulike metoder og teorier for maskinsyn. Etter undersøkelser og eksperimentering med flere maskinsyns-biblioteketer, valgte vi OpenCV. Grunnen til at vi valgte dette biblioteket var at det er godt dokumentert, åpen kildekode og inneholder mange moduler for maskinsyn og objekt-detektering og -sporing. I tillegg var det en fordel at OpenCV finnes for python, siden vi har jobbet med dette språket tidligere.

Algoritmer

Problemstillingen i dette prosjektet er et bildebehandlings-problem. Valg av algoritme for bildebehandling var derfor viktig. Vi eksperimenterte med mange ulike algoritmer underveis i prosjektet. Etterhvert var det to algoritmer som skilte seg ut og fungerte delvis; feature matching og optical flow.

Feature matching ble valgt på grunnlag av at man kunne finne unike interessepunkter i et bilde og deretter finne det igjen i et lignende bilde. OpenCV inneholder flere algoritmer for både feature detection og matching, slik at vi kunne finjustere og teste flere algoritmer innenfor denne generelle algoritmen.

For feature detection valgte vi å bruke ORB for å finne interessepunkter og deskriptorer på grunnlag av ytelsestester; som beskrevet i [20, p. 6], "ORB is the most efficient feature-detector-descriptor with least computational cost". For feature matching valgte vi å bruke FLANN matcher siden denne er mer effektiv enn Brute force-matcher som beskrevet i kapittel 2.

Etter eksperimentering med feature matching, så vi et behov for en annen algoritme som baserer seg på bevegelse av hele bildet fremfor bevegelse av enkeltpunkter. Dermed falt valget på optical flow.

Vi undersøkte muligheten til å bruke maskinlæring for å løse problemstillingen, men mengden video-data ville ikke vært tilstrekkelig. Tidlig i prosjektet fikk vi beskjed om at oppdragsgiver hadde lite video-data ettersom dronen fortsatt er under utvikling. Dermed la vi fra oss ideen om å bruke maskinlæring.

Metode

Feature matching

Denne algoritmen fungerer ved at man finner interessepunkter og deskriptorer for et bilde ved å bruke ORB. Deretter finner man interessepunkter og deskriptorer for det neste valgte bildet i videoen. Legg merke til at hvor mange bilder det skal være mellom hver estimering er et parameter som kan endres. Videre vil man finne matcher mellom de to bildene, og bruke en

FLANN matcher for å finne gode matcher mellom bildene. Etter dette beregnes det en estimert gjennomsnittsavstand i Y-retning basert på matchene. Denne avstanden i piksler konverteres deretter til avstand i meter. Ved å bruke denne avstanden og antall bilder siden siste estimering, kan man finne tid og dermed hastighet.

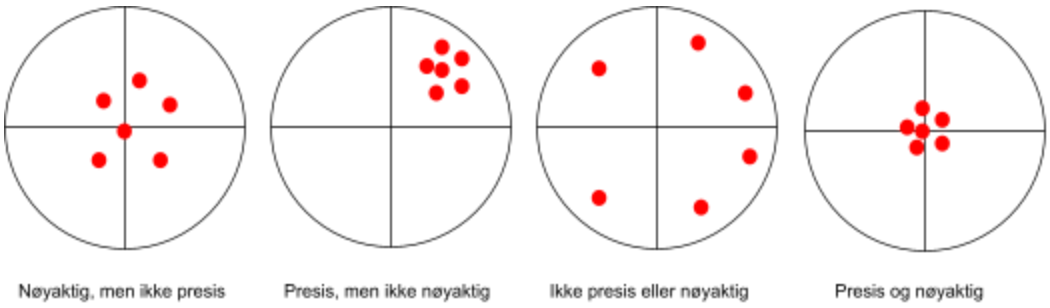
Denne algoritmen bruker altså først feature detection, og deretter feature matching. Vi har valgt å kalle denne algoritmen for feature matching, siden feature detection er grunnlaget for feature matching.

Optical flow

Algoritmen finner først egnede punkter til sporing ved å bruke Shi-Tomasi Corner Detection. Deretter brukes optical flow-algoritmen til å spore de best egnede punktene, samt lagre bevegelsen over tid til hvert punkt i en liste. Disse verdiene vil være bevegelse over antall piksler. Dermed må man regne om fra antall piksler til antall meter. Ved å ta gjennomsnittet av bevegelsen over tid for de egnede punktene, vil man kunne regne ut hastigheten.

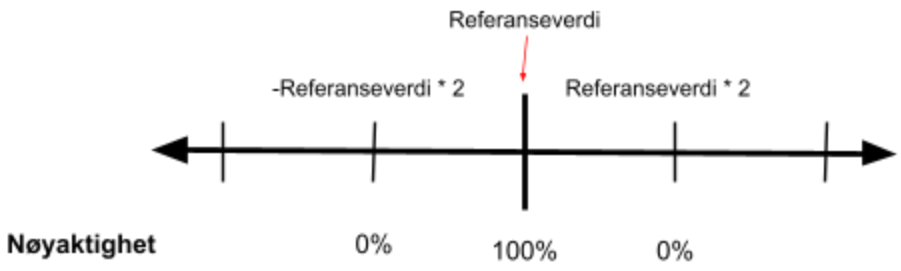
Presisjon og nøyaktighet

Presisjon har en sammenheng med standardavviket og viser hvor mye estimeringene avviker fra hverandre. Nøyaktighet har en sammenheng med referanseverdien eller fasiten, og viser hvor langt unna estimeringene er fra denne verdien. Dette er illustrert i figur 17.



Figur 17 - Forskjellen mellom nøyaktighet og presisjon

Hvis resultatet er 100% nøyaktig ville gjennomsnittet vært lik referanseverdien. Nøyaktighet er en relativ verdi og vil derfor overstige 100% hvis absoluttverdien til gjennomsnittet er større enn to ganger referanseverdien. For å få en nøyaktighet verdi mellom 0-100% blir nøyaktigheten satt til 0% hvis den ligger utenfor grenseverdien på to ganger referanseverdien. Dette er illustrert i figur 18. Det beregnes også en relativ feil som kan gi verdier over 100%, og kan dermed brukes i sammenheng med nøyaktighet hvis nøyaktigheten er 0%.



Figur 18 - Nøyaktighet i forhold til referanseverdi

Testing

Det var nødvendig å teste algoritmene for å finne ut om de kan brukes i praksis. Vi har fått tilsendt flere videoer fra oppdragsgiver SINTEF Ocean AS. Flere av disse er opptak fra en undervannsdrone i en merd. Problemet med disse videoene er at vi ikke har noen tilhørende data som beskriver den faktiske hastigheten til undervannsdronen.

Vi fikk derimot en video hvor hastigheten så nærmest konstant ut i et tidsintervall, og vi kunne dermed telle antall ruter i dette tidsintervallet og beregne en referansehastighet. Størrelsen på hver rute var 3.0cm for nettet i denne videoen. Videre i rapporten vil vi referere til denne videoen som remora. De andre tilsendte videoene fra undervannsdrone i merd hadde for varierende hastighet og det var mye støy i bildet. Vi kunne dermed ikke bruke disse videoene for testing.

En annen video vi fikk tilsendt simulerer en undervannsdrone som beveger seg normalt på nettet med en konstant hastighet. Videoen har en tilhørende referansehastighet, og kan dermed brukes for testing. For å teste ved ulike hastigheter er både remora- og simulerings-videoen redigert til

flere videofiler med forskjellige hastigheter. På denne måten kunne algoritmene testes i et kontrollert miljø med ulike hastigheter.

Testen fungerer ved at hver algoritme ble kjørt på alle de ulike simulerings-videoene. Deretter ble det samlet inn hastighetsdata for hver estimering som algoritmen gjennomfører. Estimering av hastighet utføres hver tiende bilde i alle algoritmene. Det utføres flere statistiske beregninger basert på hastighetsdata: maksimum, minimum, gjennomsnitt, standardavvik, variasjon, presisjon, relativ presisjon, nøyaktighet og RMSE. Den totale tiden algoritmen bruker blir også målt, og det beregnes en verdi på antall bilder per estimering.

Presisjon

I denne testen er presisjon en måling som sier noe om hvor nært hastighetene er hverandre i et datasett. Denne verdien har ingen nytte i seg selv, men kan brukes for å sammenligne presisjon med andre datasett. Hvis estimeringene er relativt samlet kan denne målingen være en god indikator på presisjon. Derimot hvis det finnes noen estimeringer som ligger langt utenfor gjennomsnittet, vil disse skyve maksimum og minimum langt utenfor gjennomsnittet og dermed ikke være en egnet indikator for presisjon.

$$Presisjon = (max - min) \quad (7.0)$$

Det måles også relativ presisjon i denne testen. Denne presisjonen er relativt til gjennomsnittshastigheten, og sier noe om hvor nært estimeringene ligger hverandre relativt til gjennomsnittet. I motsetning til presisjon kan man bruke relativ presisjon uten å sammenligne med andre datasett.

$$Relativ\ presisjon = \frac{Gjennomsnittshastighet}{(max\ hastighet - min\ hastighet)} \quad (7.1)$$

Nøyaktighet

Nøyaktighet er en måling som sier noe om hvor nært gjennomsnittshastigheten ligger relativt til referansehastigheten. Det finnes flere måter å beregne nøyaktighet. Først og fremst beregnes det relativ feil, som sier hvor mye feil gjennomsnittshastigheten er i forhold til referansehastigheten. Relativ feil er gitt ved:

$$Relativ\,feil\ = \frac{|\ Gjennomsnittshastighet - Referansehastighet\ |}{|\ Referansehastighet\ |} \quad (8.0)$$

Ved å bruke relativ feil kan man finne en nøyaktighet med verdi fra 0 til 100% hvis den relative feilen er lik eller mindre enn 100%. Den relative feilen vil bli større enn 100% hvis gjennomsnittshastigheten er større enn to ganger referansehastigheten. Nøyaktighet er gitt ved funksjonen $f(a)$ hvor a er den relative feilen i %:

$$f(a) = \begin{cases} 100 - a & : a \leq 100 \\ 0 & : a > 100 \end{cases} \quad (8.1)$$

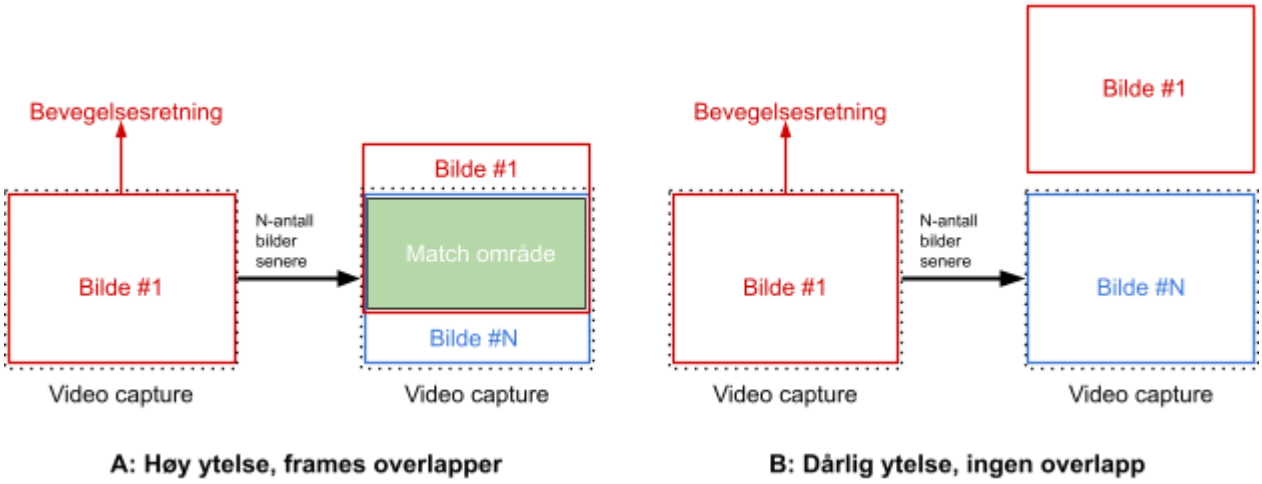
RMSE

Root mean square error, forkortet RMSE, er en måling som sier noe om feilen i et datasett. Verdien sier ikke noe i seg selv, men kan sammenlignes med andre datasett. RMSE er gitt ved:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(x_i - y_i)^2}{n}} \quad (9.0)$$

Total tid og antall bilder per estimering

Et av kravene i visjonsdokumentet er at algoritmene skal kjøre i sanntid. Det måles derfor hvor lang tid hver algoritme bruker per video. Ut fra denne tiden kan man også kalkulere hvor mange bilder i gjennomsnitt i videoen det er mellom hver estimering. Dette er interessant siden kan ha stor effekt på ytelse til algoritmen. Det er hovedsakelig et problem som kan oppstå hvis en algoritme kjører tregt ved bruk i sanntid; hvis en algoritme algoritme bruker for lang tid på en estimering, vil ikke bildene som sammenlignes overlappe.



Figur 19 - Ytelse av algoritme ved bruk i sanntid

Begge algoritmene baserer seg på en sammenligning av ulike bilder, hvis denne sammenligningen skjer etter for lang tid vil man ikke ha noe overlapp mellom bildene. Dette er illustrert i figur 19. I figur 19 (A) ser man at bildene overlapper og man får et match område markert i grønt. I figur 19 (B) ser man at Frame #1 er utenfor Video Capture og den får dermed ingen overlapp med Frame #N. Man kan oppleve å få matcher uten overlapp mellom bildene, men dette er dårlige matcher som bør forkastes.

Antall bilder per estimering er gitt ved likningene 10.0 og 10.1:

$$Total\ antall\ estimeringer = \frac{Video\ total\ tid * video\ bildefrekvens}{Estimeringsfrekvens} \quad (10.0)$$

$$Antall\ bilder\ per\ estimering = \frac{Total\ tid\ algoritme * video\ bildefrekvens}{1000 * total\ antall\ estimeringer} \quad (10.1)$$

Kapittel 4: Resultater

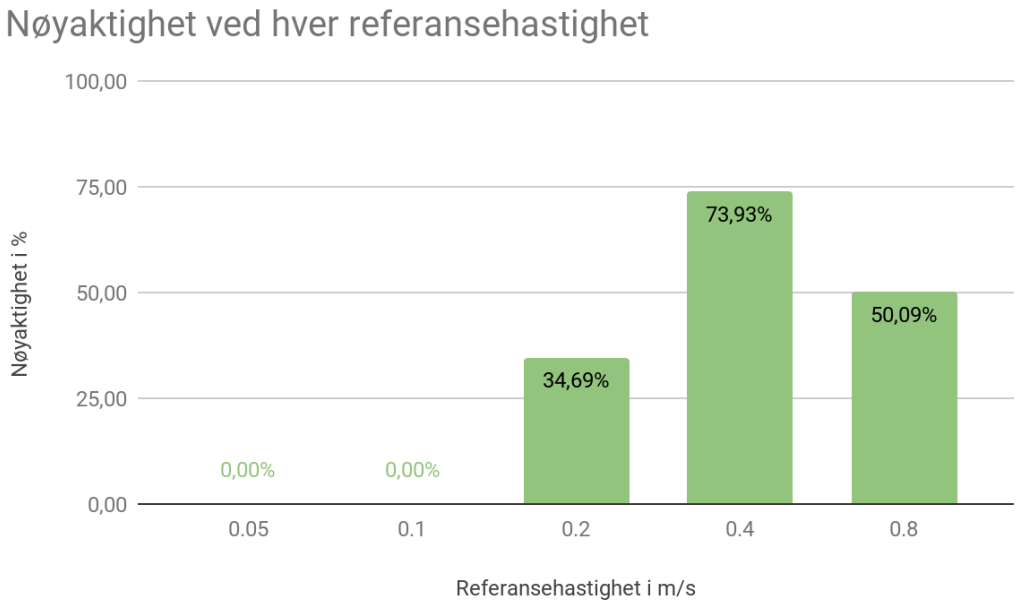
Vitenskapelige Resultater

Resultatene under viser data som er samlet inn etter tester på algoritmene med flere ulike videoer. I resultatene skilles det mellom simulerings video og remora videoen fra en merd. Videre skilles det også mellom algoritmene for hver type video. Hver tabell i resultatene inneholder resultater for de ulike referansehastighetene.

Simulerings-video

Feature matching

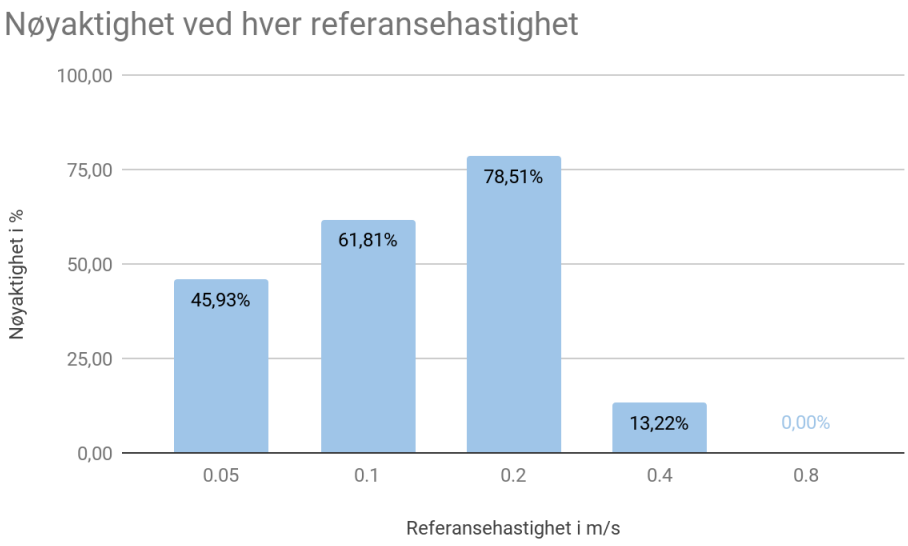
Referansehastighet (m/s)	0.05	0.1	0.2	0.4	0.8
Tid total (ms)	17 802,17	8 437,13	2 101,70	2 243,69	1 219,23
Hastighet gjennomsnitt (m/s)	0,28	0,29	0,33	0,30	1,20
Hastighet min (m/s)	−1,04	−0,92	−0,68	−0,54	−0,63
Hastighet max (m/s)	0,95	1,17	2,98	0,83	5,60
Hastighet varianse (m/s)^2	0,13	0,14	0,41	0,14	4,33
Hastighet standardavvik (m/s)	0,36	0,37	0,63	0,36	1,93
Relativ feil (%)	451,05	189,58	65,31	26,07	49,91
Nøyaktighet (%)	0,00	0,00	34,69	73,93	50,09
Antall bilder per estimering	5,09	4,96	2,63	5,61	6,10
Presisjon	1,99	2,09	3,66	1,36	6,23
Relativ resisjon (%)	13,84	13,87	9,04	21,72	19,24
RMSE	0,36	0,39	0,65	0,60	1,94



Figur 20 - Nøyaktighet ved hver referansehastighet for feature matching med simulerings-video

Optical flow

Referansehastighet (m/s)	0.05	0.1	0.2	0.4	0.8
Tid total (ms)	46 648,60	23 569,33	6 245,44	5 525,34	2 466,71
Hastighet gjennomsnitt (m/s)	0,08	0,14	0,16	0,05	0,00
Hastighet min (m/s)	0,00	0,00	0,00	0,00	0,00
Hastighet max (m/s)	0,16	0,22	0,24	0,22	0,00
Hastighet varianse (m/s)^2	0,00	0,00	0,00	0,01	0,00
Hastighet standardavvik (m/s)	0,03	0,04	0,05	0,08	0,00
Relativ feil (%)	54,07	38,19	21,49	86,78	100,00
Nøyaktighet (%)	45,93	61,81	78,51	13,22	0,00
Antall bilder per estimering	13,33	13,86	7,81	13,81	12,33
Presisjon (m)	0,16	0,22	0,24	0,22	0,00
Relativ presisjon (%)	49,51	63,71	64,81	24,27	0,00
RMSE	0,04	0,06	0,07	0,36	0,80

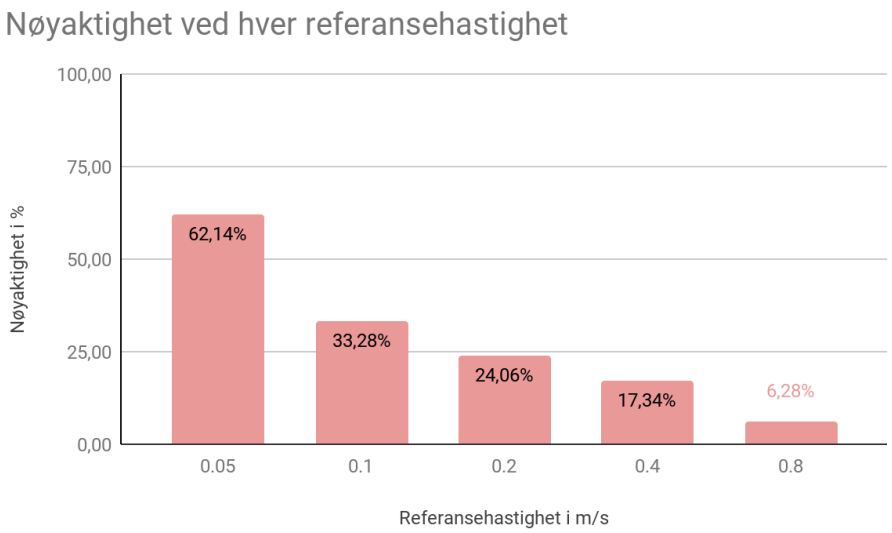


Figur 21 - Nøyaktighet ved hver referansehastighet for optical flow med simulerings-video

Remora-video fra merd

Feature matching

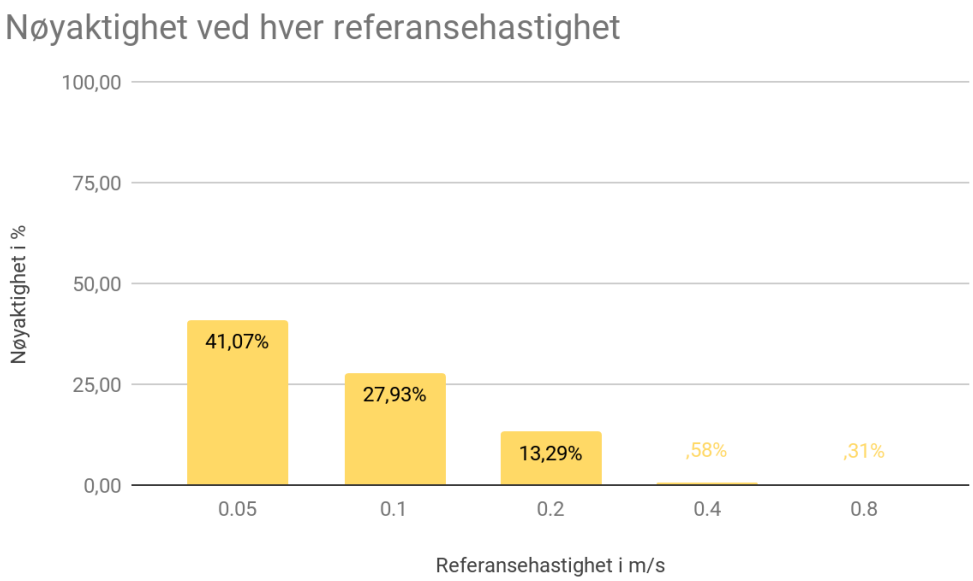
Referansehastighet (m/s)	0.05	0.1	0.2	0.4	0.8
Tid total (ms)	6 176,46	3 078,11	1 219,69	889,33	456,76
Hastighet gjennomsnitt (m/s)	0,03	0,03	0,05	0,07	0,05
Hastighet min (m/s)	−0,11	−0,10	−0,18	−0,12	−0,09
Hastighet max (m/s)	0,14	0,46	0,46	0,64	0,40
Hastighet varianse (m/s)^2	0,00	0,00	0,01	0,02	0,01
Hastighet standardavvik (m/s)	0,04	0,06	0,09	0,13	0,11
Relativ feil (%)	37,86	66,72	75,94	82,66	93,72
Nøyaktighet (%)	62,14	33,28	24,06	17,34	6,28
Antall bilder per estimering	0,77	0,72	0,76	0,83	0,85
Presisjon	0,25	0,56	0,65	0,75	0,49
Relativ resisjon (%)	12,36	5,94	7,42	9,20	10,29
RMSE	0,06	0,10	0,21	0,41	0,79



Figur 22 - Nøyaktighet ved hver referansehastighet for feature matching med remora video

Optical flow

Referansehastighet (m/s)	0.05	0.1	0.2	0.4	0.8
Tid total (ms)	17 619,82	8 361,04	3 130,68	1 843,65	948,74
Hastighet gjennomsnitt (m/s)	0,02	0,03	0,03	0,00	0,00
Hastighet min (m/s)	−0,03	−0,02	−0,01	0,00	−0,03
Hastighet max (m/s)	0,04	0,06	0,06	0,02	0,01
Hastighet varianse (m/s)^2	0,00	0,00	0,00	0,00	0,00
Hastighet standardavvik (m/s)	0,01	0,02	0,02	0,00	0,01
Relativ feil (%)	58,93	72,07	86,71	99,42	99,69
Nøyaktighet (%)	41,07	27,93	13,29	0,58	0,31
Antall bilder per estimering	2,20	1,95	1,95	1,72	1,77
Presisjon (m)	0,07	0,09	0,07	0,02	0,03
Relativ presisjon (%)	28,99	31,99	35,49	12,64	7,51
RMSE	0,03	0,07	0,17	0,40	0,80



Figur 23 - Nøyaktighet ved hver referansehastighet for optical flow med remora video

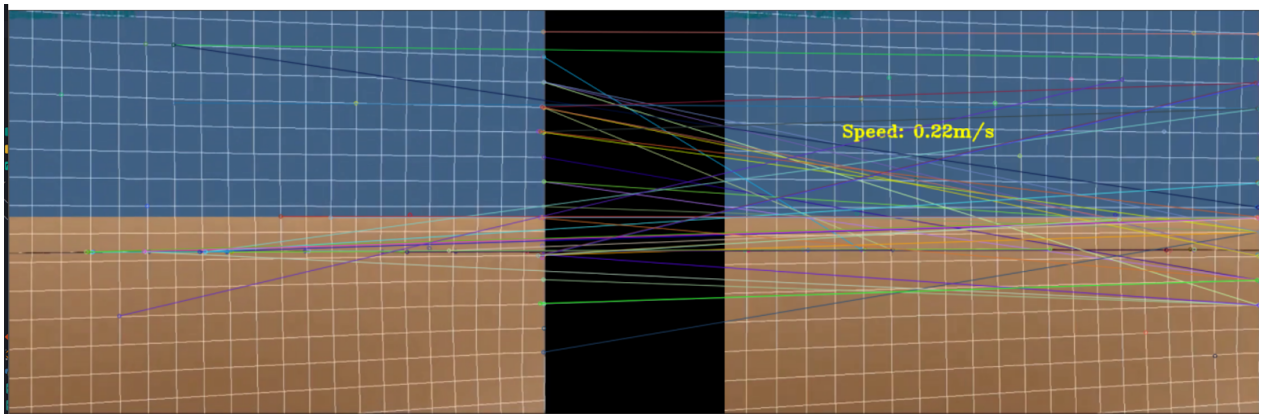
Ingeniørfaglige Resultater

Produktet

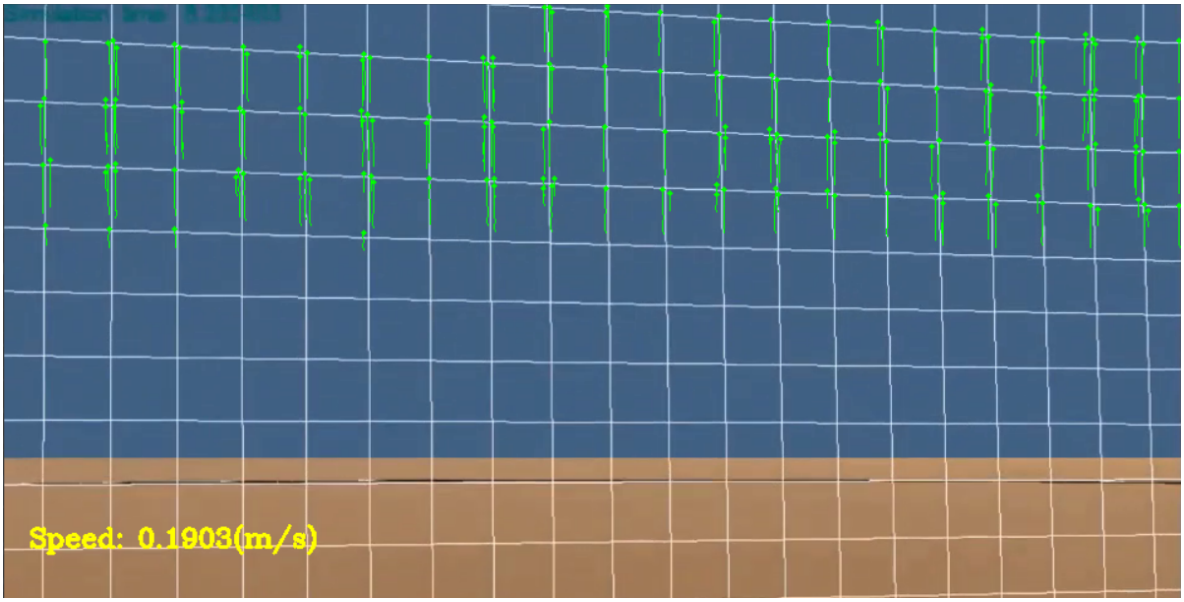
Sluttresultatet er to algoritmer; feature matching og optical flow. Disse algoritmene estimerer hastigheten til undervannsdronen mens den beveger seg langs nettet i en merd. Begge algoritmene kan visualisere estimatet, samt punktene i bildet som brukes i beregningene.

Under ligger skjermbilder av begge algoritmene for både simulering- og remora-video.

Simulering-video

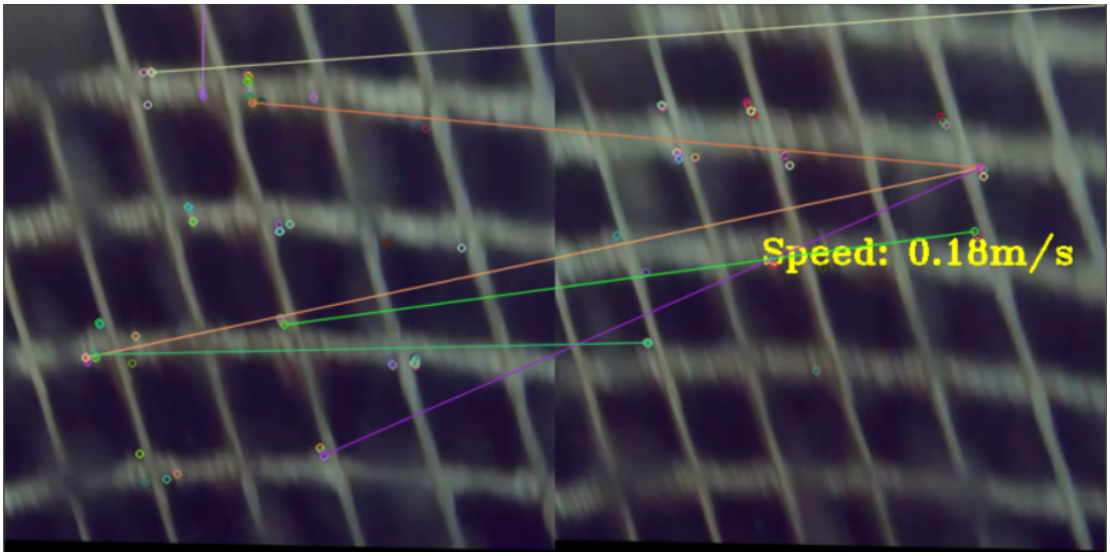


Figur 24 - Skjermbilde av feature matching på simulering-video med visualiserte matcher mellom to bilder.

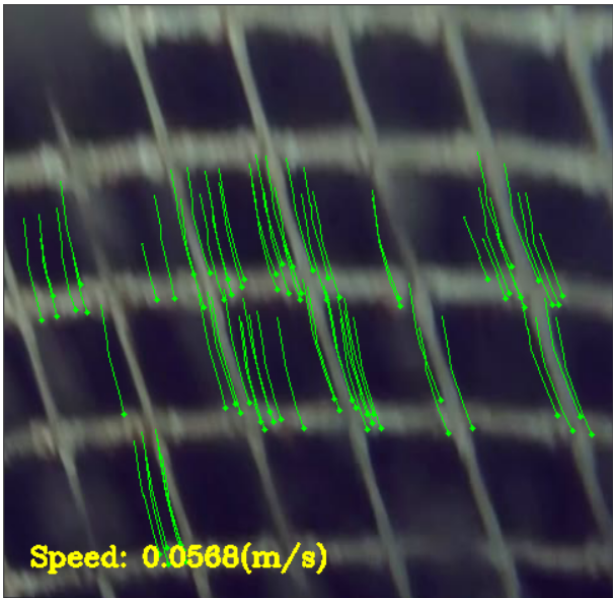


Figur 25 - Optical flow i bruk på simulert nettet

Remora-video



Figur 26 - Skjermbilde av feature matching på remora-video, etter perspektiv-transformasjon, med visualiserte matcher mellom to bilder.



Figur 27 - Skjermbilde av optical flow i bruk på remora-videoen, etter perspektiv-transformasjon, med visualiserte spor.

Mål og Status

I visjonsdokumentet er det utarbeidet noen mål for prosjektet. Under ligger det en tabell med disse målene hvor det også er oppført status for hvert mål ved levering.

Behov	Prioritet	Påvirker	Dagens løsning	Foreslått løsning	Fullført	Status
Sanntids-estimering av notvaskerobot i to dimensjoner	Høy	Alle	Ingen eksisterende løsning	Algoritme som bruker maskinsyn for å estimere posisjonen og hastigheten i sanntid	Ja	Fungerende algoritmer, men lav nøyaktig og presisjon.Forskning en fokuserte kun på en dimensjon.
Deteksjon av hull i not	Medium	Alle	Ingen eksisterende løsning	Algoritme som bruker maskinsyn for å detektere hull i not	Nei	Ingen løsning
Deteksjon av groe i not	Lav	Alle	Ingen eksisterende løsning	Algoritme som bruker maskinsyn for å detektere groe i not	Nei	Ingen løsning

Administrative Resultater

Tidsforbruk

Tabellen under viser tidsforbruk for gruppen basert på aktivitet.

Aktivitet	Antall timer
Planlegging	146.5
Utvikling	339.5
Testing	101.5
Dokumentasjon	429.5
Total	**1007**

Kapittel 5: Diskusjon

Vitenskapelige Resultater

Simulerings-video

Feature matching

Resultatene viser til varierende nøyaktighet og lav presisjon. Algoritmen har høy nøyaktighet, omtrent 74%, ved referansehastighet 0.4m/s, men 50% eller mindre nøyaktighet ved 0.2m/s og 0.8m/s. Ved de laveste hastighetene er nøyaktigheten 0%. Det vil si at den relative feilen er generelt høy i de fleste videoene. Man ser også at standardavviket er høyt i forhold til referanshastigheten, noe som bekrefter den lave presisjonen. RMSE stiger ved høyere referansehastighet. Dette betyr at algoritmen gir mer feil jo høyere hastighet.

Den høye nøyaktigheten ved referansehastighet 0.4m/s kan i dette tilfellet være tilfeldigheter. Man ser at presisjon er svært lav på 22%, og et standardavvik på nesten 100% av referansehastigheten. Resultatene tilsier dermed at feature matching har både lav nøyaktighet og presisjon for simulerings-video med ulike hastigheter. Dette er også testet i test 1 og 2 i Vedlegg A, som viser til hvorfor feature matching fungerer dårlig i nettet. nettet kan dessuten overlappe ved noen intervaller, som vist i test 2 i Vedlegg A.

Optical flow

Resultatet av optical flow er høy nøyaktighet ved referansehastighet 0.2m/s. Nøyaktigheten synker ved de lavere referansehastighetene. Ved de høyere referansehastighetene er det svært lav nøyaktighet. Med den reelle farten 0.4m/s, tror algoritmen at hastigheten er 0.05m/s. Algoritmen tror den reelle hastigheten er 0m/s, når den egentlig skal være 0.8m/s. Høy hastighet gir lav nøyaktighet. Grunnen er trolig at nettet beveger seg for mye hvert bilde, slik at algoritmen ikke klarer å spore punktene som beveger seg.

Svært lav hastighet gjør algoritmen mindre nøyaktig. Dette kan være på grunn av mengden målinger den gjør, og at den ofte estimerer hastigheter på 0m/s. Dette vil trekke ned

gjennomsnittshastighetene. Ved å se på RMSE er den lav for hastigheter opp til 0.2m/s. Hastigheter over har mye større feil. Det er dermed mindre feil ved lavere hastighet.

Den relative presisjonen er høy ved 0.1m/s og 0.2m/s, henholdsvis 64% og 65%. Dette indikerer at nøyaktigheten ved disse hastighetene ikke er tilfeldig. Hadde nøyaktigheten vært høy, men presisjonen lav, ville det nok ha vært tilfeldig at nøyaktigheten var høy.

I den simulerte videoen tilsier resultatene at nøyaktigheten øker med hastigheten til et visst punkt, hvor den ikke lengre klarer å spore bevegelsen i bildet.

Sammenligning

Resultatene fra simulering-video viser at feature matching har middelmådig nøyaktighet ved høyere hastigheter (0.4-0.8m/s), mens optical flow har middelmådig nøyaktighet ved lavere hastigheter (0.05-0.2m/s). Ingen av algoritmene har altså høy nøyaktighet for flere ulike hastigheter. Begge algoritmene er dermed uegnet til estimering av hastighet til en undervannsdrone i merd. Det viser derimot at begge algoritmene er effektive og har god ytelse, videre er feature matching mer effektiv enn optical flow for simulerings-video.

Remora-video

Feature matching

Testing av remora-video med feature matching ga resultater som tilsier lav nøyaktighet og presisjon for alle referansehastighetene. Ved den laveste hastigheten er nøyaktigheten på 62%, men minsker til 33% ved 0.1m/s. Nøyaktigheten fortsetter å synke ved høyere referansehastighet, og er nede på 6% ved den høyeste hastigheten. Man ser altså at nøyaktigheten går nedover etterhvert som referansehastigheten øker.

Den relative presisjonen er lav for alle referansehastighetene. Ved 0.1m/s er den relative presisjonen 12%, og ligger mellom 6-10% for de andre referansehastighetene. Standardavvikene er forholdsvis lavt sammenlignet med referansehastighetene, utenom 0.05m/s hvor

standardavviket er nesten 100% av referansehastigheten. Man ser derfor også at variansen er nærmest lik 0, siden denne har sammenheng med standardavviket som er lavt.

Resultatene fra remora-video med feature matching, viser at algoritmen har både lav nøyaktighet og presisjon for alle referansehastighetene. feature matching er derfor uegnet til estimering av hastighet og posisjon i remora-video. Videoen i denne testen er et eksempel på video-data fra undervannsdronen i en merd, og er derfor et realistisk eksempel for testing av algoritme. Siden feature matching er uegnet til estimering av hastighet og posisjon i denne videoen, vil algoritmen dermed være uegnet å bruke for posisjonering og estimering av hastighet til undervannsdronen.

Optical flow

Nøyaktigheten i remora-videoen synker ved økende hastighet. Den er også bare 41% ved den laveste hastigheten. Den estimerte hastigheten øker bare litt fra målingene med referansehastighet 0.05m/s til 0.1m/s og til 0.2m/s, henholdsvis 0.02m/s, 0.03m/s og 0.03m/s. Ved de to høyeste hastighetene er estimatet 0m/s for begge. Den beste nøyaktigheten på 41% kan ha vært tilfeldig, siden den relative presisjonen ligger på rundt 30% hos de tre laveste referansehastighetene.

Resultatet tilsier at optical flow ikke gir nøyaktige målinger på reelle videoer, i motsetning til simulerte videoer. En forskjell på simulator-videoen og remora-videoen er distansen mellom nettet og kameraet. I simulator-videoen filmer kameraet normalt på nettet, og det er et godt stykke unna. Dette gjør at nettet beveger seg sakte i forhold til kameraet, som gjør det lettere å spore.

Sammenligning

Resultatene viser at begge algoritmene har lav nøyaktighet som henholdsvis synker ved økende hastighet. Den estimerte hastigheten og referansehastigheten til de ulike videoene har dermed negativ korrelasjon. Feature matching viser seg derimot å være mer nøyaktig enn optical flow ved alle hastigheter. Det er mulig at det er kun er tilfeldigheter som gjør feature matching bedre enn

optical flow i remora-video. Videre viser det seg at feature matching er mer effektiv enn optical flow på remora-video.

Feilkilder

Tester

Test 2 i vedlegg A viser at omtrent 70% av matchene til feature matching peker i feil retning av bevegelsen til undervannsdronen. Resultatene er derfor verken presise eller pålitelige til sammenligning av de to algoritmene

Estimasjonsintervallet

En mulig feilkilde i testene er hvor ofte algoritmene estimerer hastigheten. I testene estimeres hastigheten omtrent hvert tiende bilde. Det vil si at avstanden nettet har beveget seg vil være forskjellig i de ulike videoene, siden de har ulik referansehastighet. Ved høyere referansehastigheter vil nettet bevege seg lengre enn ved lavere referansehastigheter, siden estimeringen utføres hvert tiende bilde for alle videoene. Ved referansehastighet 0.8m/s og estimering hvert tiende bilde vil nettet ha beveget seg:

$$s_1 = vt = 0.8m/s * \frac{10}{30} = \frac{4}{15} \approx 0.27m$$

Nettet i remora videoen består av kun fem ruter på 3.0cm etter perspektiv-transformasjon. Lengden på nettet er dermed kun 0.15m. Estimeringen ved høyere hastigheter vil derfor gi dårligere nøyaktighet siden nettet beveger seg helt ut av bildet og skal egentlig ikke gi noen matcher mellom bildene med feature matching.

Optical flow estimerer hastighet hvert tiende bilde, men sjekker bevegelsen hvert femte bilde. Optical flow bør derfor teoretisk sett fungere bedre ved høyere hastigheter enn feature matching. nettet vil derimot fremdeles bevege seg langt ved referansehastighet 0.8m/s:

$$s_2 = vt = 0.8m/s * \frac{5}{30} = \frac{2}{15} \approx 0.13m$$

Siden nettet i remora-videoen er kun 0.15m, vil nettet ha beveget seg nesten helt ut av bildet når optical flow sjekker bevegelsen ved referansehastighet 0.8m/s. Dette vil gi lav presisjon og nøyaktighet, som vist i resultatene.

En løsning på dette kan være å beregne en estimert avstand for hvert bilde, og deretter sette en terskelverdi på beveget avstand for at hastigheten skal estimeres og lagres. Problemet med denne løsningen er at algoritmen skal være robust mot varierende hastighet innenfor et satt intervall. Ved lavere hastigheter vil det ta mye lengre tid å finne estimert hastighet siden det tar lengre tid før nettet har beveget seg lengre enn terskelverdien. Hvis undervannsdronen endrer hastighet mellom estimeringene, vil avstanden den har beveget seg bli feil over lengre tid. Man ønsker derfor å oppdatere estimert avstand og hastighet ofte for å unngå feilberegninger over lengre tid.

Nettet størrelse og avstand

En annen mulig feilkilde er størrelsen på rute i nettet, samt avstanden fra undervannsdronen og nettet. Hvis en av disse parametrene endres, vil beregningene bli inkorrekte.

Referansehastigheten

Hvis referansehastigheten i noen av test-videoene er upresise i forhold til reell hastighet, vil nøyaktigheten, samt andre beregninger, til algoritmene bli inkorrekte.

Krav og Mangler

I oppgaveteksten ble tre algoritmer beskrevet. Det ble bestemt i et møte, tidlig i prosjektet, at fokuset skulle være kun på den første algoritmen; estimering av posisjon og hastighet. De to andre, detektering av hull og detektering av groe, ble satt til side. Disse algoritmene ville krevd bruk av maskinlæring.

Vi kunne også tatt i bruk maskinlæring på algoritmen for estimering av posisjon og hastighet. Problemet er at det har vært vanskelig å få video-data fra dronen ettersom prosjektet er under utvikling. Med den lille mengden data vi har fått jobbet med ville ikke maskinlæring gitt et godt resultat.

Algoritmene beregner ikke posisjonen til undervannsdronen. Grunnen til dette er at vi ønsket å forenkle koden for testing, forskning og eksperimentering. Posisjonen kan enkelt beregnes når man vet hastigheten og tiden.

Svakheter

Algoritmene viser seg å ha både lav presisjon og nøyaktighet. Dessuten er de lite robust mot varierende hastighet og miljø. Flere av parametrene til algoritmen må endres basert på avstanden til nettet, avstanden mellom rutene i nettet, og synsvinkelen og synsfeltet til kameraet.

Et av disse parameterne er en konstant for avstand i meter per piksel for video-ressursen. Hvis undervannsdronen beveger seg nærmere eller lengre unna nettet under bevegelse, vil estimeringene bli enda mer upresis og unøyaktig. Det samme gjelder hvis man bruker undervannsdronen i ulike nettet med forskjellige størrelse på rutene. Endringer i størrelse på rute eller avstand til nettet medfører at konstanten ikke stemmer med den reelle verdien.

Hvis kameraet er vinklet skarpt slik at det står nesten parallelt med nettet, vil det bli mer vanskelig for algoritmene å estimere hastighet. Når man utfører en perspektiv transformasjon slik at video-data står normalt på nettet, vil man miste flere ruter i nettet i forhold til hvis kameraet står normalt på nettet. Færre ruter vil gi mindre total avstand i bildet, og dermed gi mindre rom

for sammenligning. Algoritmene vil derfor kunne få problemer med å henge med den kunstige høye farten i den transformerte videoen.

En svakhet ved forskningen er mengden videoer som ble brukt til testing. Det var svært lite video-ressurser tilgjengelig. Dessuten var få av disse passende for testing. De fleste manglet referansehastigheter som er essensielt for testing. Testresultatene ville vært mer pålitelig hvis vi hadde hatt flere video-ressurser til testing.

Det er gjennomført uavhengige tester (Test 1 og 2 i vedlegg A) for feature matching, for å prøve å finne ut hvorfor algoritmen har lav nøyaktighet. Det er derimot ikke utført noen lignende tester for optical flow. Grunnen til dette er tidsbegrensning.

Styrker

Gjennom prosjektet har gruppen hatt fokus på eksperimentering og forskning. Det var derfor viktig å teste og legge frem grundige undersøkelser for begge algoritmene. Resultatene etter testing er omfattende og fremlegger hvorfor algoritmene ikke fungerer som forventet.

Testene er gjennomført i forskjellige miljøer, både simulert og ekte, med ulike hastigheter. Man har dermed testet algoritmene for forskjellige miljø og ulike hastigheter uavhengig av hverandre, slik at man kan se hvilke situasjoner algoritmene eventuelt gjør gode eller dårlige estimeringer. På grunnlag av dette ønsker gruppen å trekke frem grundig testing og analysering som en styrke.

På bakgrunn av disse testene kan vi trekke frem noen styrker ved algoritmene. Generelt er begge algoritmene effektive og gir gode estimeringer ved noen referansehastigheter.

Vi ønsker også å trekke frem prosessen vår i dette prosjektet som en styrke. Lean utviklingsmetode viste seg å være et godt valg for gruppen, med tanke på at prosjektet ble mer forsknings-preget mot slutten. I tillegg har vi testet algoritmene ofte og sammenlignet resultatene for å finne ut hva som fungerer best, og deretter utviklet algoritmene videre. Både vi og oppdragsgiver er fornøyd med prosjektet i sin helhet.

Tidsforbruk

Vi har nådd målet med å jobbe 1000 timer med 10% margin. Det viste seg derimot at Gantt-diagrammet var lite representativt for arbeidsfordeling basert på aktiviteter i prosjektet. Vi undervurderte tiden det tok å skrive dokumentasjon. Grunnen til denne feilvurderingen er fordi oppgaven ble mer forsknings-preget utover prosjektet. Det var derfor viktig for oss å ha grundig og god dokumentasjon.

Planleggingsfasen tok lengre tid enn vi forutså. En av grunnene til dette var at prosjektet krevde mye eksperimentering. Dermed var det behov for å planlegge ofte underveis. I starten av prosjektet planla vi et utviklingsprosjekt, og satte derfor av mye tid til utvikling. Vi brukte mindre tid på utvikling enn planlagt, grunnet endringen av type prosjekt, som krevde mer fokus på testing, eksperimentering og dokumentering. Vi tok heller ikke høyde for utdypende testing og eksperimentering.

Gruppearbeidet

Gruppen har har hatt god flyt i gjennom hele prosjektet. Medlemmene i gruppen har jobbet sammen tidligere, og jobber dermed godt sammen. Arbeidet har bestått av mye forskning og eksperimentering, derfor har gruppen jobbet sammen fysisk på campus store deler av prosjektet. Pandemien har derimot medført til at arbeidet har foregått digitalt i noen perioder.

Kapittel 6: Konklusjon og videre arbeid

Konklusjon

Resultatene viser at feature matching og optical flow ikke kan nøyaktig estimere hastigheten til en undervannsdrone i en merd. Videre viser resultatene at feature matching er mer nøyaktighet enn optical flow, spesielt ved høyere hastigheter. Optical flow er mer nøyaktig enn feature matching ved lavere hastigheter i et simulert miljø. Begge algoritmene viser seg derimot å være uegnet til å estimere hastighet til en undervannsdrone i en merd, vi kan derfor ikke konkludere med at den ene er mer nøyaktig enn den andre. Vi konkluderer likevel med at både feature matching og optical flow er uegnet for estimering av hastighet til en undervannsdrone i en merd.

Algoritmene ga lave nøyaktigheter på videoene de ble testet på. Både simulerte og ekte video-ressurser ga samme resultater. Dermed kan man konkludere med at bruk av algoritmene i både simulerte og reelle situasjoner er uegnet. Tilgangen på videofiler har vært liten. Ved å kjøre algoritmene på flere videoer, vil man få et tydeligere svar på problemstillingen.

Et av målene med prosjektet var å lage en algoritme som bruker maskinsyn for å estimere posisjon og hastighet i sanntid. Dette målet har gruppen oppnådd, men algoritmene er lite nøyaktige og presise. Algoritmene har fremdeles en verdi for oppdragsgiver, siden grunnlag for hvorfor algoritmene ikke fungerer er fremlagt i denne rapporten. Oppdragsgiver har dessuten muligheten til å ta arbeidet med algoritmene videre, og forbedre nøyaktigheten.

Valget av arbeidsmetoden har vist seg å være gunstig. Prosjektet omhandlet hovedsakelig eksperimentering og forskning. Utvikling var derfor ikke i fokus. Vi kan dermed konkludere med at valg av arbeidsmetode var fornuftig. Både vi og oppdragsgiver er fornøyd med gjennomføringen og resultatet av prosjektet.

Videre arbeid

Videre arbeid vil omhandle seg om å fikse svakhetene som er fremlagt i kapittel 5. Mer spesifikt involverer dette blant annet å gjøre algoritmene mer robuste mot endringer i nettet. Det er muligheter for å eksperimentere med algoritmene videre for å forbedre nøyaktigheten og presisjonen. Bruk av maskinlæring i kombinasjon med maskinsyns-algoritmer kan være en løsning på problemstillingen. På grunn av lite data og ressurser, samt manglende kunnskaper, hadde ikke gruppen mulighet til å eksperimentere med dette. Mer eksperimentering og finjustering av parametre ville vært fordelaktig for å forbedre algoritmene ytterligere.

Referanser

[1] "Om oss - SINTEF." https://www.sintef.no/ocean/om-oss/ (accessed Apr. 29, 2021).

[2] "NetClean 24/7 - SINTEF." https://www.sintef.no/prosjekter/2019/netclean-247/ (accessed Apr. 29, 2021).

[3] G. Taraldsen, T. A. Reinen, and T. Berg, "The underwater GPS problem," *OCEANS 2011 IEEE - Spain*, pp. 1–8, 2011, Accessed: May 13, 2021. [Online].

[4] B. Misund, "merd." https://snl.no/merd (accessed Apr. 21, 2021).

[5] T. Ringnes, "parallakse," *Store Norske Leksikon*. https://snl.no/parallakse (accessed May 13, 2021).

[6] R. N. Devich and F. M. Weinhaus, "Image Perspective Transformations," in *Image Processing for Missile Guidance*, Dec. 1980, vol. 0238, pp. 322–333, Accessed: Apr. 30, 2021. [Online].

[7] C. Harris and M. Stephens, "A combined corner and edge detector," *Procedings of the Alvey Vision Conference 1988*, pp. 231–236, 1988, Accessed: Apr. 21, 2021. [Online].

[8] "OpenCV: Shi-Tomasi Corner Detector & Good Features to Track." https://docs.opencv.org/master/d4/d8c/tutorial_py_shi_tomasi.html (accessed Apr. 29, 2021).

[9] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," *Computer Vision – ECCV 2006*, pp. 430–443, 2006.

[10] T. Drummond, R. Porter, and E. Rosten, "Faster and Better: A Machine Learning Approach to Corner Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105–119, 2010, Accessed: Apr. 21, 2021. [Online].

[11] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features," in *Computer Vision – ECCV 2010*, Sep. 2010, pp. 778–792, Accessed: Apr. 21, 2021. [Online].

[12] G. Bradski, K. Konolige, V. Rabaud, and E. Rublee, "ORB: An efficient alternative to SIFT or SURF," *2011 International Conference on Computer Vision*, pp. 2564–2571, 2011, Accessed: Apr. 21, 2021. [Online].

[13] "OpenCV: ORB (Oriented FAST and Rotated BRIEF)." https://docs.opencv.org/master/d1/d89/tutorial_py_orb.html (accessed Apr. 29, 2021).

[14] A. Jakubovic and J. Velagic, "Image Feature Matching and Object Detection Using Brute-Force Matchers," *2018 International Symposium ELMAR*, pp. 83–86, Accessed: Apr. 21, 2021. [Online].

[15] A. Bookstein, V. A. Kulyukin, and T. Raita, "Generalized Hamming Distance," *Inf. Retr. Boston.*, vol. 5, no. 4, pp. 353–375, Oct. 2002, Accessed: Apr. 22, 2021. [Online].

[16] M. D. Agostino and V. Dardanoni, "What's so special about Euclidean distance? A characterization with applications to mobility and spatial voting," *Soc. Choice Welfare*, vol. 33, pp. 211–233, Aug. 2009, Accessed: Apr. 22, 2021. [Online].

[17] D. G. Lowe and M. Muja, "Fast Matching of Binary Features," *2012 Ninth Conference on Computer and Robot Vision*, 2012, doi: 10.1109/crv.2012.60.

[18] S. S. Beauchemin and J. L. Barron, "The computation of optical flow," *ACM Computing Surveys*, vol. 27, no. 3, pp. 433–466, 1995, Accessed: Apr. 29, 2021. [Online].

[19] F. W. S. P., U. Lanka, and P. Pubudu, "Identification of moving obstacles with Pyramidal Lucas Kanade optical flow and k means clustering," *2007 Third International Conference on Information and Automation for Sustainability*, pp. 111–117, 2007, Accessed: Apr. 29, 2021. [Online].

[20] S. A. K. Tareen and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK," *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pp. 1–10, 2018, Accessed: Apr. 22, 2021. [Online].

Vedlegg

Prosjekthåndboken ligger i en separat fil.

A - Tester

Vedlegg A

Test 1

Hypotese:

Interessepunktene for bilder av et not-nett er for lik hverandre slik at feature matching ikke klarer å finne gode matcher. Dette medfører at den estimerte avstanden not vaskeroboten har beveget seg blir svært lite nøyaktig og presis. Dette reflekteres også i den estimerte hastigheten som beregnes på grunnlag av den estimerte avstanden.

Oppklaring:

Vi definerer algoritmen som nøyaktig hvis absoluttverdien til gjennomsnittet er innenfor 75% av referanseverdien. I tillegg definerer vi algoritmen som presis hvis det relative standardavviket er større eller lik 75%.

Bildene som brukes i denne testen er skjermbilder fra en video ressurs som vi har fått tildelt av oppdragsgiver i SINTEF OCEAN AS. Videoen er et opptak fra et program som simulerer en not vaskerobot som beveger nedover i en merd.

Vi bruker ORB for å finne interessepunkter og deskriptorer for bildene. En FLANN matcher brukes for å finne matcher mellom bildene.

Metode:

For å teste denne hypotesen vil vi gjennomføre en test hvor vi bruker to bilder fra ulike videoressurser av not-nett. La bilde 1 være bildet før bevegelsen i videoen, og bilde 2 som bildet etter bevegelsen. Vi finner først interessepunkt og deskriptorer for de to bildene, og bruker FLANN-matcher for å finne matcher mellom de to bildene.

Deretter kalkulerer vi avstanden mellom to interessepunkter i en match. Dette kan man gjøre ved å finne avstanden i meter per piksel for den aktuelle videoen. Vi vet at det er omtrent 14 ruter i not-nettet for hvert bilde, hvor hver rute er 3.0cm. I tillegg er høyden på videoen 720 piksler.

$$d_{px} = (0.03m * 14)/720px = \frac{7}{12000} m/px \approx 5.84 * 10^{-4} m/px$$

Den reelle avstanden i meter mellom to interessepunkt, p og q, i en match er gitt som:

$$d(p,q) = d_{px} * avstand\ mellom\ p\ og\ q\ i\ px$$

Vi vet at referansehastigheten i videoen er 0.2m/s, og at bildefrekvensen til videoen er 30. Vi kan dermed finne avstanden not vaskeroboten har beveget seg i løpet av 5 bilder:

$$t = \frac{5}{30fps} \approx 0.17s$$

$$s = vt = 0.2m/s * \frac{1}{6}s = \frac{1}{30}m \approx 0.03m$$

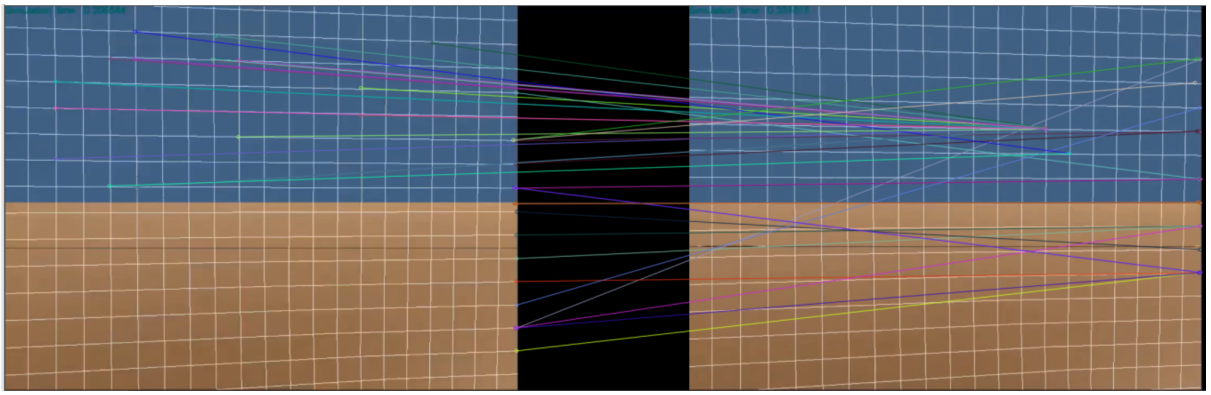
Når vi har funnet den estimerte distansen kan vi sjekke hvilken retning matchen peker. Hvis notvaskeroboten beveger seg nedover, vil matchen peke oppoveri forhold til bilde 1, og vice versa.

Resultat:

Under ligger en tabell med resultatene fra testen.

Måling	Verdi
Avstand referanse (m)	0,03
Avstand gjennomsnitt (m)	0,09
Avstand min (m)	−0,27
Avstand max (m)	0,16
Avstand varianse (m)	0,01
Avstand standardavvik (m)	0,09
Presisjon (%)	21,56
Nøyaktighet (%)	0,00
Relativ feil (%)	176,05
Antall matcher nedover	28
Antall matcher oppover	12

Under ligger et skjermbilde som viser matchene mellom bilde 1 og bilde 2.



Figur 1 - Skjermbilde av matchene mellom bilde 1 (venstre) og bilde 2 (høyre).

Diskusjon:

Resultatene viser at matchene mellom bilde 1 og 2 har meget lav nøyaktighet i forhold til referanseverdien på 0.03m. I gjennomsnitt er den estimerte avstanden 0.09m, omtrent tre ganger høyere enn referanseverdien, som er grunnen til at nøyaktigheten er 0%.I tillegg er standardavviket like høyt som gjennomsnittet, noe som tilsier at matchene er upresise. Presisjon er ca 22%, som vil si at matchene varierer veldig i forhold til gjennomsnittet.

Resultatene viser også at det er stor forskjell i retningen til matchene. 12 matcher peker oppover, og 28 peker nedover. Siden bevegelsen mellom bilde 1 og bilde 2 kun er i en retning, vil det si at flere av matchene peker i feil retning i forhold til den simulerte bevegelsen, og dermed er dårlige matcher. I denne testen beveger not vaskeroboten seg nedover, som vil si at matchene skal peke oppover fra bilde 1 til bilde 2. Det vil si at 28/40 eller 70% av matchene peker i feil retning og dermed er dårlige matcher. Man ser i tillegg at det er stor forskjell på minimum- og maksimum verdien, som indikerer lav presisjon.

En mulig løsning på dette er å fjerne matcher som peker i feil retning i forhold til bevegelsesretningen. Dette forutsetter derimot at man vet hvilken retning not vaskeroboten beveger seg, noe som ikke er inkludert i ressursene for dette prosjektet.

En mulig feilkilde i denne testen er at videoe ressursen bruk til skjermbildene er fra et simuleringsprogram. En video fra en merd vil kunne gi andre resultater. Det er dessuten nødvendig å teste på flere bilder og videoer for å oppnå mer pålitelige resultater.

Konklusjon:

På grunnlag av resultatene fra testen kan vi konkludere med at interessepunktene i et not-nett er for like hverandre for at feature matching klarer å finne gode matcher. Omtrent 70% av matchene peker i feil retning, og algoritmen har lav presisjon. Det er derimot nødvendig å teste på flere bilder for å kunne oppnå mer pålitelige resultater. En mulig løsning er å begrense søket etter interessepunkter til et mindre område.

Test 2

Hypotese:

Ved noen bestemte intervaller vil bilder fra en video av et not-nett være nærmest helt like, og dermed gjøre det vanskelig for maskinsyn algoritmer å estimere en avstand mellom bildene.

Oppklaring:

Bildene som brukes i denne testen er skjermbilder fra en video ressurs som vi har fått tildelt av oppdragsgiver i SINTEF Ocean AS. Videoen er et opptak fra et program som simulerer en not vaskerobot som beveger nedover i en merd.

Metode:

For å teste denne hypotesen vil vi gjennomføre en test hvor vi bruker to bilder fra ulike videoressurser av not-nett. La bilde 1 være bildet før bevegelsen i videoen, og bilde 2 som bildet etter bevegelsen.

Det er gitt at hver rute er i noten er 3.0cm. Bildefrekvensen til videoen er 30. Ved en hastighet $v$kan man kalkulere hvor mange bilder N det tar for not vaskeroboten å bevege seg samme avstand som en rute i not-nettet.

Man bruker først formelen for tiden basert på strekning og hastighet.

$$t = \frac{s}{v}$$

Tiden er gitt som et forhold av Nantall bilder og bildefrekvensen til videoen.

$$t = \frac{N}{30fps} = \frac{s}{v}$$

Løser for N, s er gitt som avstanden mellom hver rute i not-nettet.

$$N = \frac{0.03m * 30fps}{v}$$

Ved en hastighet på 0.2m/s som er referansehastigheten i test-videoen, vil videoen overlappe ved:

$$N = \frac{0.03m * 30fps}{0.2m/s} = 4.5$$

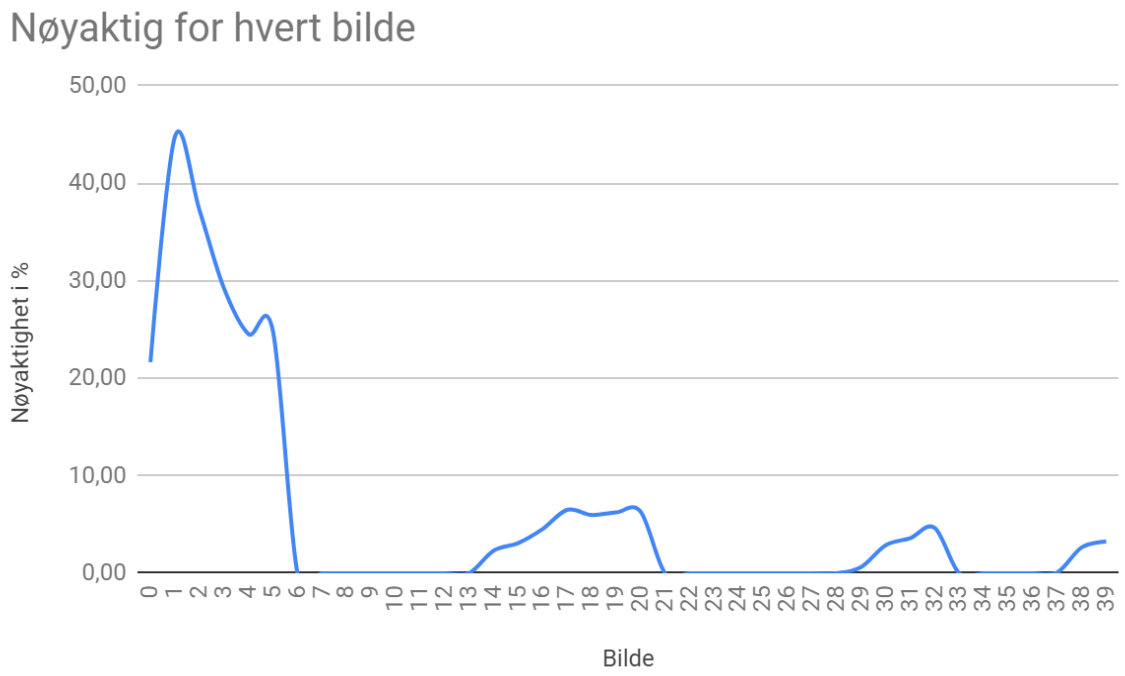
Siden indeksen til et bilde må være et heltall vil overlapp i dette tilfellet skje ved hvert niende bilde. For å teste dette, vil vi kjøre feature matching algoritmen ved forskjellige bilde-intervaller for å se om dette har en effekt på algoritmen.

Resultater:

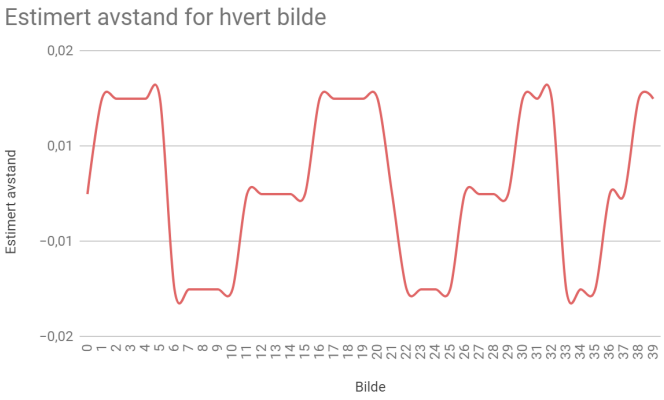
Under ligger en tabell som viser resultatene etter kjøring av testen.

Bilde indeks	Avstand estimert (m)	Avstand referanse (m)	Differanse fra referanse (m)	Relativ feil (%)	Nøyaktighet (%)
0	0,00	0,01	0,01	78,36	21,64
1	0,01	0,01	0,01	55,21	44,79
2	0,01	0,02	0,01	62,75	37,25
3	0,01	0,03	0,02	70,78	29,22
4	0,01	0,03	0,03	75,46	24,54
5	0,01	0,04	0,03	75,19	24,81
6	−0,01	0,05	0,05	117,37	0,00
7	−0,01	0,05	0,06	120,29	0,00
8	−0,01	0,06	0,07	116,27	0,00
9	−0,01	0,07	0,08	113,33	0,00
10	−0,01	0,07	0,08	109,42	0,00
11	0,00	0,08	0,08	105,80	0,00
12	0,00	0,09	0,09	102,67	0,00
13	0,00	0,09	0,09	101,13	0,00
14	0,00	0,10	0,10	97,69	2,31
15	0,00	0,11	0,10	96,92	3,08
16	0,01	0,11	0,11	95,48	4,52
17	0,01	0,12	0,11	93,50	6,50
18	0,01	0,13	0,12	94,02	5,98
19	0,01	0,13	0,13	93,76	6,24
20	0,01	0,14	0,13	93,69	6,31
21	0,00	0,15	0,15	100,00	0,00
22	−0,01	0,15	0,16	107,48	0,00
23	−0,01	0,16	0,17	106,42	0,00
24	−0,01	0,17	0,17	104,75	0,00

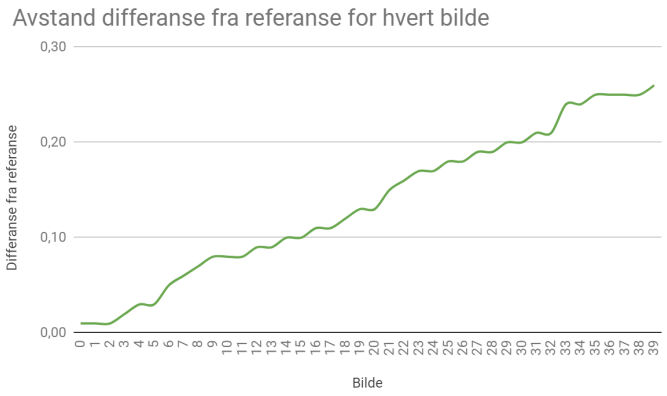
25	−0,01	0,17	0,18	103,69	0,00
26	0,00	0,18	0,18	102,70	0,00
27	0,00	0,19	0,19	100,37	0,00
28	0,00	0,19	0,19	100,77	0,00
29	0,00	0,20	0,20	99,37	0,63
30	0,01	0,21	0,20	97,13	2,87
31	0,01	0,21	0,21	96,40	3,60
32	0,01	0,22	0,21	95,34	4,66
33	−0,01	0,23	0,24	105,03	0,00
34	−0,01	0,23	0,24	103,87	0,00
35	−0,01	0,24	0,25	102,21	0,00
36	0,00	0,25	0,25	100,73	0,00
37	0,00	0,25	0,25	99,93	0,07
38	0,01	0,26	0,25	97,35	2,65
39	0,01	0,27	0,26	96,76	3,24



Figur 1 - Nøyaktighet for estimeringen for hver bilde-sammenligning i testen



Figur 2 - Estimert avstand for hver bilde-sammenligning i testen



Figur 3 - Avstand differanse fra referanse for hver bilde-sammenligning i testen

Diskusjon:

Resultatene viser at nøyaktigheten går betraktelig ned i intervaller hvor not-nettet i bildene er nærmest helt like. I metode fant vi ut at for denne videoen vil denne overlappen skje ved hver niende bilde. I resultatet ser vi at nøyaktigheten går ned til 0% fra bilde 6-13 i videoen. Nøyaktigheten går derimot ikke ned ved bilde 18, men fra og med bilde 21. Grunnen til dette kan være at referansehastigheten ikke er helt nøyaktig, eller at feature matching gir dårlige matcher i området bilde 13-21. Resultatene for nøyaktighet viser ingen sammenheng med hypotesen.

Den estimerte avstanden for hver bilde-sammenligning varierer i stor grad, se figur 2. Dessuten ser man differansen mellom estimert avstand og referansene avstand øker for hver bilde-sammenligning i testen, se figur 3.

Test 1 viser dessuten at feature matching har problemer med å finne gode matcher med videoressurser av not-nett. Dette kan være grunnen til at vi disse variable intervallene.

Konklusjon:

Resultatene viser at nøyaktigheten går ned i intervaller, men disse intervallene er ikke konstante. Derfor kan vi ikke konkludere med at nøyaktigheten vil variere i gitte intervaller, men heller at nøyaktigheten vil variere en viss grad i variable intervaller. Disse variable intervallene vil gjøre det vanskelig for algoritmen å estimere avstanden mellom bildene.

Teorien i metoden viser at not-nettene kan overlappe i noen bilder ved ulike hastigheter, men resultatene kan ikke bekrefte dette. Feilkilden ligger i feature matching algoritmen som brukes i denne testen. Resultatene fra denne testen forsterker konklusjonen i test 1, som viser hvorfor feature matching har problemer med å finne gode matcher med videoer i not-nett.

B - Visjonsdokument

048

Vedlegg B
Visjonsdokument

Versjon 2.0

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
12.01.2021	1.0	Første utkast	Mikael Kalstad, Henrik Tronstad
27.01.2021	1.1	Andre utkast	Mikael Kalstad, Henrik Tronstad
03.05.2021	2.0	Rettskriving og formatering	Mikael Kalstad, Henrik Tronstad

Innholdsfortegnelse

1. Innledning

Hensikten med dette dokumentet er å definere tankene rundt resultatet vi skal oppnå i dette prosjektet.

1.1 Referanser

2. Sammendrag problem og produkt

2.1 Problemsammendrag

Problem med	beregning av posisjonering og hastighet av notvaskerobot under vann. Samt behov og ønske om deteksjon av hull og groe på not i merd.
berører	Mithal, oppdretter
som resultatet av dette	økonomisk tap, fiskehelse, belastning på noten
en vellykket løsning vil	bidra til at vekster og organismer ikke får etablert seg i merden. Samt detektere hull i not. Bidra til å lage kontrollsystemer for notvaskeroboten.

2.2 Produktsammendrag

For	SINTEF Ocean AS og Netclean 24/7
som	har behov for maskinsynalgoritmer til notvaskerobot
produktet navngitt	maskinsyns-algoritmer
som	vil beregne posisjon og hastighet av notvaskerobot, samt detektering av hull og groe
I motsetning til	dagens løsning av Mithal
Har vårt produkt	løsninger tilgjengelig for SINTEF Ocean AS

3. Overordnet beskrivelse av interessenter og brukere

3.1 Oppsummering interessenter

Navn	Utdypende beskrivelse	Rolle under utviklingen
SINTEF Ocean AS	Oppdragsgiver	Veiledning, testing og tilgang til data
Mithal	Prosjekteier	Video-ressurser og relevant data
NTNU	Jonathan Jørgensen	Veiledning

3.2 Oppsummering brukere

Navn	Utdypende beskrivelse	Rolle under utviklingen	Representert av
Mithal	Bruker av notvaskerobot	Tilbakemelding ved testing	Seg selv

3.3 Brukermiljøet

Brukermiljøet vil være i merder hvor notvaskerobot er utplassert.

3.4 Sammendrag av brukernes behov

Behov	Prioritet	Påvirker	Dagens løsning	Foreslått løsning
Sanntidsestimering av notvaskerobot i to dimensjoner	Høy	Alle	Ingen eksisterende løsning	Algoritme som bruker maskinsyn for å estimere posisjonen og hastigheten i sanntid

Deteksjon av hull i not	Medium	Alle	Ingen eksisterende løsning	Algoritme som bruker maskinsyn for å detektere hull i not
Deteksjon av groe i not	Lav	Alle	Ingen eksisterende løsning	Algoritme som bruker maskinsyn for å detektere groe i not

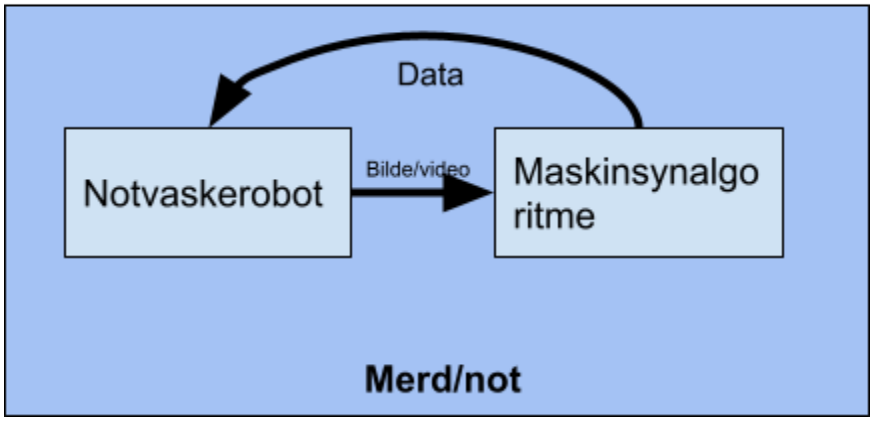
3.5 Alternativer til vårt produkt

Etter undersøkelser på nett har vi ikke funnet noen andre kommersielle alternativer til vårt produkt.

4. Produktoversikt

4.1 Produktets rolle i brukermiljøet

Figuren under viser en skisse av produktet med hensyn til omgivelsene.



4.2 Forutsetninger og avhengigheter

Andre sensorer er ikke tilgjengelig i dette prosjektet. Mithal står bak video-ressursene. Hastighet oppgis i meter per sekund.

5. Produktets funksjonelle egenskaper

- System
    - Algoritmene skal brukes i sanntid.
    - Installert på mikrokontrolleren.
    - Algoritmene bør fungere best mulig, så optimaliseres etterpå.
    - Programmeringsspråk: Python.

- Posisjonering
    - Posisjonering angis i meter.
    - Posisjonering angis i x- og y-akser.
    - Posisjonens nøyaktighet bør være så godt som mulig.

- Hastighet
    - Hastighet angis i meter per sekund.
    - Hastighetens nøyaktighet bør være så god som mulig.

6. Ikke-funksjonelle egenskaper og andre krav

Koden skal være godt dokumentert og skrives på engelsk. Dokumentasjonen skal derimot skrives på norsk.

7. Referanser

1. **Oppgavebeskrivelse**