NTNU
Kunnskap for en bedre verden

Institutt for IKT og realfag

AIS2900 - Bacheloroppgave Ingeniørfag

Objektdeteksjon med YOLOv8 og LiDAR for maritim sikkerhet

Forfattere:

Dyrkolbotn, Joachim (10058)

Hestnes, Henning (10045)

21.05.2024

Forord

Denne bacheloroppgaven er gjennomført av en gruppe bestående av to studenter, hvor begge studerer automasjon og intelligente systemer ved NTNU-Ålesund. Oppgaven utføres på vårsemesteret 2024 med NTNU-Ålesund som oppdragsgiver. Prosjektet innebærer systemintegrasjon og omgivelsesforståelse for et fartøy.

Vi ønsker å takke veilederne våre Erlend Magnus Lervik Coates og Lars Ivar Hatledal for gode råd og tilbakemeldinger gjennom prosjektperioden. Vi vil også takke Sanjeev Kumar Ramkumar Sudha og Eirik Strøm Fagerhaug som har tatt seg tid til å assistere oss ved behov gjennom hele prosjektet. Til slutt vil vi takke Anders Sætermoen for å bestilling av deler og gode ideer for den praktiske utførelsen.

Vi ønsker også å takke NTNU-Ålesund for å ha gitt oss muligheten til å gjennomføre dette prosjektet. Tiden vår her har vært svært verdifull for vår utvikling. Vi vil også takke våre medstudenter for deres støtte og samarbeid gjennom studietiden.

Sammendrag

Denne avhandlingen handler om å integrere et Ouster Lidarsystem på en Otter USV. Ved å kombinere Otterens smidighet med Lidarens avanserte sensorteknologi, gir man Otteren muligheten til å detektere objekter. Vårt fokus er spesifikt rettet mot å identifisere og måle avstanden til båter nær Otteren for å forbedre operatørens situasjonsforståelse. Ouster lidar-systemet fungerer ved å sende ut laserpulser og måle disse pulsene når de reflekterer tilbake fra omgivelsene. Den kan måle avstand, signal, nær-IR og reflektivitet. Ved å bruke denne dataen kan vi generere et 2D-bilde som YOLO-algoritmen kan bruke til objektdeteksjon. Ved å bruke avstandsmålingene fra lidar og OpenCV er vi i stand til å generere et bilde der objekter på vannet er tydelige. Med en trent YOLOv8-modell er vi i stand til å oppdage båter i dette bildet. Ved å integrere Ouster lidar-systemet på Otter USV og navigere den til en båthavn, oppdaget vi flere båter vellykket. Ved hjelp av lidarens avstandsmålinger kan vi vise avstanden til båtene, og dermed gi nyttig informasjon til operatøren.

Summary

This thesis integrates an Ouster lidar system onto the Otter USV. By combining the Otter's agility with lidar's advanced sensing, we provide the Otter with object detection capabilities. Our focus is specifically on identifying and measuring the distance to boats near the Otter USV, enhancing operator awareness. The ouster lidar system operates by emitting laser pulses and measuring these laser pulses as they bounce back from the environment, it can measure range, signal, near-IR, and reflectivity. Utilizing this data, we can generate a 2D image that the YOLO algorithm can use for object detection. Using range values and OpenCV we are able to generate a picture where objects on the water are clearly visible. With a trained YOLOv8 model we are able to detect boats within this picture. By integrating the Ouster lidar system onto the Otter USV and navigating it to a boat dock, we successfully detected multiple boats. Using the lidar's range measurements we can display the distance to the boats, providing useful information to the operator.

Terminologi

- USV – Unmanned Surface Vehicle
- IMU – Inertial Measurement Unit
- LiDAR/lidar – Light Detection and Ranging
- Payloadboks – Payloadboksen som blir referert til, er en boks på Otteren som er tilgjengelig for brukeren.
- CVS – Vehicle Control System
- OBS-boks – Denne boksen, som er integrert på Otteren er ansvarlig for drift av Otteren og kommunikasjon.
- ToF – Time of Flight
- UDP – *User Datagram Protocol* er en kommunikasjonsprotokoll som i dette tilfellet blir brukt for kommunikasjon mellom lidar og PC.
- YOLO – *You Only Look Once* er en objektdeteksjonsalgoritme.
- Label – I denne oppgaven refererer ordet *label* til etikettene som YOLO-algoritmen plasserer over deteksjonene for å visualisere hvilket objekt som er detektert.
- Boks/bounding box – I denne oppgaven refererer dette til koordinatene til deteksjonene gjort med YOLO-algoritmen. Koordinatene bestående av (x_1, y_1) og (x_2, y_2) brukes til å tegne en boks rundt deteksjonen.
- MEMS – Micro-electromechanical system
- ROS – Robot Operating System

Innhold

Figurer

Tabeller

1 Innledning

Den maritime næringen er i stadig utvikling og Norge har et ønske om å fortsette som et ledende land innen feltet frem til 2030. For å oppnå dette vil Norge sette fokus på forskning, innovasjon og digitalisering og dermed kunne ta en ledende posisjon i det grønne skiftet innen sektoren [23]. En viktig brikke i denne satsingen er ubemannede fartøy. Dette er et fagområdet i stadig utvikling innen den maritime sektor og blir stadig mer aktuelle i en rekke forskning- og observasjonsarbeid. Disse fartøyene, kjent som Unmanned Surface Vehicles (USV), er automome eller fjernstyrte fartøy og tas i bruk som et hjelpemiddel for å redusere operasjonskostnader og redusere risiko knyttet til arbeidsoperasjoner. Med disse egenskapene er fartøyene ideelle til å utføre et stort spekter av oppgaver langs kysten som vannprøver eller kartlegging av havbunnen.

En relatert oppgaven for innholdet i denne rapporten er *Multi-sensor multi-target tracking using LIDAR and camera in a harbor environment*-masteroppgaven [4]. Her tas det i bruk en USV med diverse sensorer som lidar, GPS og kamera til å navigere fartøyet i områder med begrenset plass, ved hjelp av maskinlæringsalgoritmer. Dette er en dypere tilnærming til oppgaven som blir beskrevet videre i denne rapporten.

1.1 Bakgrunn

NTNU-Ålesund har i dag en tilgjengelig USV, også kjent som Otter USV. Universitetets formål er å ta fartøyet i bruk til å utføre diverse forskningsprosjekter som blant annet innsamling av vannprøver og kartlegging av havbunnen. Operasjonsområdet til Otteren vil med dette i hovedsak være langs kysten som til tider kan være høyt trafikkert. For å kunne ta i bruk Otteren på en sikker måte ønsker oppdragsgiveren å videreutvikle systemet på fartøyet ved å implementere og integrere ytterlige sensorer. Bakgrunnen for dette er å gjøre fartøyet i stand til å utføre et bredere spekter av oppgaver samt forsikre at dette kan utføres på en sikker måte.

1.2 Problemstilling

På bakgrunn av dette er problemstillingen man står ovenfor å øke situasjonsforståelsen rundt fartøyet for å minimere risikoen for sammenstøt med objekter, i dette tilfellet båter. For å oppnå dette vil det monteres og integreres en lidar på fartøyet med mål om å øke sikkerheten rundt utførelse av operasjoner. Dette vil gjøres ved å ta i bruk lidaren til å samle inn data fra omgivelsene. Med å ta i bruk denne innsamlede informasjonen vil det utvikles et system for å kunne varsle operatøren av fartøyet når et objekt kommer innenfor en angitt avstand. Denne løsningen vil dermed gjøre det mulig for fartøyet å utføre diverse stillestående oppgaver uten at operatøren aktivt må følge med for å unngå sammenstøt.

1.3 Motivasjon

Valget om å arbeide med oppgradering av Otter USV som vår bacheloroppgave i samarbeid med NTNU var motivert av muligheten til å jobbe med en oppgave med stor åpenhet. Måten oppgaven var representert på ga oss som gruppe, i samarbeid med veilederene, mulighet til å selv definere og forme oppgaven med bakgrunn av tidligere erfaringer og interesser. Gjennom oppgaven var det mulig til å videreutvikle det eksisterende systemet på Otteren ved å implementere nye løsninger og opprettholde den satte standarden, noe gruppen fant interessant. Gruppemedlemmene har gjennom studieløpet hatt delt interesse for en rekke emner som førte til en felles interesse for teknologien som var tilgjengelig gjennom oppgaven. Dette førte også til en enighet om hva vi ønsket å oppnå.

1.4 Omfang og begrensninger

På bakgrunn av at arbeidet med oppgaven strekker seg over et vårsemester ble det satt noen begrensninger på oppgaven for å skape en realistisk målsetting. Med tanke på prosjektets omfang og de tilgjengelige ressursene, ble det valgt to fokusområder som både er utfordrende og gjennomførbare innenfor rammen av oppgavens tidsfrist. Det ble på grunnlag av dette satt fokus på følgende punkter:

- **Systemintegrasjon av lidaren:** Dette området omfatter integrering av lidarsensoren i det eksisterende systemet på Otter USV'en. Målet med dette er å kunne ta i bruk lidaren til å samle inn data fra omgivelsene og visualisere dette i sanntid.

- **Objektdeteksjon av nærliggende objekter**: Fokuset her vil være på trening og implementering av algoritmer for å oppdage og klassifisere objekter i nærområdet ved bruk av lidarbilder. Dette inkluderer å identifisere objekter innenfor en viss avstand fra Otteren og dermed varsle operatøren for å kunne unngå kollisjoner. Ved denne implementasjonen vil Otteren kunne håndteres på en sikker måte.

Ved å fokusere på disse hovedmålene gjennom prosjektperioden ble det mulig å oppnå et fullstendig resultat som kan tas i bruk for å øke sikkerheten rundt bruken av Otteren.

1.5 Rapportstruktur

Rapporten er organisert i flere kappitler for å sikre en strukturert og oversiktlig fremstilling av oppgaven. I **Kapittel 2, Teori** presenteres det teoretiske grunnlaget som danner fundamentet for arbeidet med oppgaven. **Kapittel 3, Material** presenterer materialet og programvarer som er benyttet i utførelsen av prosjektet. En detaljert beskrivelse av bruken av material og teori, samt fremgangsmåten av prosjektet vil bli beskrevet i **Kapittel 4, Metode**. Videre i **Kapittel 5, Resultat** vil sluttproduktet av prosjektet bli presentert før det blir analysert og diskutert i **Kapittel 6, Diskusjon**. Som avslutning på rapporten vil oppgavens viktigste funn og observasjoner bli trekt frem i **Kapittel 7, konklusjon**.

2 Teori

I denne delen av rapporten vil det bli gitt en oversikt over de teoretiske konseptene, teknologien og standardene som ligger til grunn for arbeidet av oppgaven. Dette vil inkludere en detaljert gjennomgang av de grunnleggende prinsippene som definerer den relevante teknologien brukt gjennom prosjektet for å skape en forståelse av de tekniske aspektene.

2.1 Time of Flight (ToF)

Time of flight er et måleprinsipp mye brukt for kalkulering av avstander til et bestemt punkt og kan i tillegg brukes for å skape 3D-bilder. Måleprinsippet baserer seg på å måle tiden det tar for en laser puls å bevege seg fra sensoren, treffe et objekt og deretter returnere tilbake til sensoren. Ved å bruke denne beregnede tiden samt kjenne til verdien for lysets hastighet kan kan man ta i bruk formel 1 til å beregne avstanden til et bestemt objekt.

$$Avstand = \frac{c \cdot t}{2} \tag{1}$$

der:

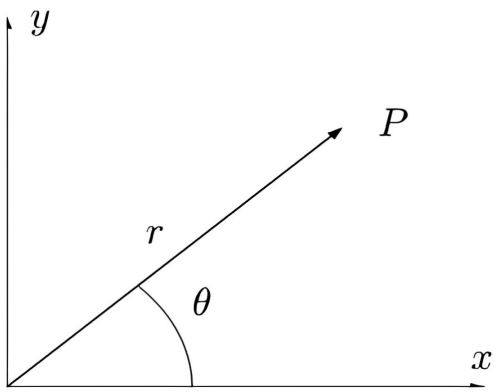
- *Avstand* er den målte avstanden mellom sensoren og målobjektet,
- t er tiden det tar for lyspulsen å reise frem og tilbake,
- c er hastigheten til lyset i vakuum ($\approx 3.00 \times 10^8$ meter/sekund).

Som ved alle måleprinsipper har ToF-prinsippet sine fordeler og ulemper. En av de største fordelene med dette prinsippet er at det trenger ikke eksterne lyskilder, noe som gjør den operativ selv ved lite lys. Måleprinsippet kan også baseres på enkel matematikk som gjør at det kan kjøres på mindre prosessorer og egner seg i applikasjoner hvor prosesshastigheten er viktig.

Med dette sagt kommer også måleprinsippet med noen ulemper som at selv om den baserer seg på å sende og motta refleksjon av sitt eget lys, kan den yte dårlig i naturlige omgivelser. Grunnen til dette er at bølger, fra for eksempel solen, kan forstyrre sensoren og gi falske målinger. Videre kan refleksjoner fra blanke overflater som reflekterer lyset i ulike retninger, eller fra hjørner som kan forårsake flere refleksjoner, representere potensielle feilkilder.

2.2 Kartesiske- og polarkoordinater

Posisjonen til et punkt i et plan kan beskrives ved hjelp av to forskjellige koordinatsystemer: kartesiske-koordinater og polarkoordinater. I kartesiske-koordinater er koordinataksene vinkelrette på hverandre, representert ved x-aksen og y-aksen i to dimensjoner. I polarkoordinater beskrives punktet P først og fremst ved avstanden r fra origo. Deretter defineres punktet ved polarvinkelen θ, som er vinkelen fra den positive x-aksen til linjen fra origo til punktet P, som vist i figur 1.



Figur 1: Polarkoordinater, hentet fra [22]

For å konvertere fra kartesiske-koordinater (x, y) til polarkoordinater (r, θ) kan formel 2 brukes:

$$x = r * cos(\theta), y = r * sin(\theta) \tag{2}$$

På samme måte kan formel 3 brukes for konvertering motsatt vei:

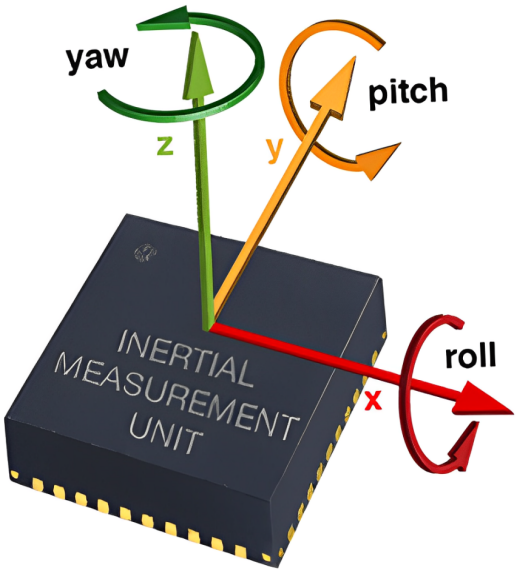
$$r^2 = x^2 + y^2, tan(\theta) = \frac{x}{y} \tag{3}$$

2.3 User Datagram Protocol (UDP)

User Datagram Protocol (UDP) er en av de grunnleggende transportlag-protokollene, kjent for sin effektive dataoverføring. Protokollen opererer i transportlaget av den fjerde av syv i OSI-modellen, et rammeverk av kommunikasjonsprotokoller som kobler sammen nettverksenheter [11]. UDP er en forbindelsesløs protokoll, noe som betyr at det ikke etableres toveiskommunikasjon mellom avsender og mottaker. Dette vil si at det ikke foregår noen fram-og-tilbake-kommunikasjon mellom partene. Dette bidrar til at protokollen kan opprettholde lite forsinkelse, selv om den ikke kan garantere pålitelig overføring av data. På grunn av sin enkelhet krever UDP bare minimalt med minne og prosesseringsressurser, noe som gjør den ideell for enkel implementering. Disse egenskapene gjør UDP spesielt egnet i situasjoner der tap av data er akseptabelt, men forsinkelser er uakseptable. På bakgrunn av dette benyttes UDP ofte i sanntidsapplikasjoner som video- og taleoverføring, der hastighet er avgjørende [47].

2.4 Inertial Measurement Unit (IMU)

Inertial Measurement Unit (IMU) er en elektronisk enhet brukt for å måle akselerasjon, vinkelhastighet, og i noen tilfeller orienteringen til et objekt. Enheten inneholder ofte akselerometre, gyroskoper, og eventuelt magnetometre, som opererer langs de tre aksene roll, pitch og yaw vist i figur 2. Denne kombinasjonen av komponenter gjør det mulig for enheten å beregne bevegelsen til objektet den er montert på. I vårt tilfelle er det et MEMS-akselerometer og et MEMS-gyroskop som tas i bruk [13].



Figur 2: IMU Akser, hentet fra [12]

Akselerometeret brukes for å måle endring i hastighet og består i hovedsak av to deler, en bevegelig masse og en fastmontert ramme. Når hastigheten til objektet endres, vil den bevegelige massen, på grunn av treghet, være forsinket i forhold til den fastmonterte rammen. Denne bevegelsen blir motstått av en fjær eller en lignende mekanisme, som trekker eller trykker på massen. Forskyvningen av massen i forhold til dens startposisjon fører til endringer i elektriske egenskaper, som kapasitans eller motstand, avhengig av sensortype. Disse endringene blir omgjort til elektriske signaler som gir nøyaktig informasjon om hastigheten og retningen objektet beveger seg i [8].

Gyroskopet i IMU'en måler vinkelhastighet som gir informasjon om hvordan enheten roterer rundt sin egen akse. Det fungerer basert på Coriolis-effekten, det vil si når enheten roterer, oppstår en forskyvning i en vibrerende masse innenfor gyroskopet. Denne forskyvningen detekteres av sensorer som måler endringen i bevegelsesretningen til massen. Disse målingene konverteres til elektriske signaler som representerer rotasjonshastigheten rundt enhetens akser. Gyroskopets data er essensielle for nøyaktig orientering og navigasjon i mange teknologiske applikasjoner, spesielt hvor presis styring er nødvendig [2].

2.5 Overfitting av maskinlærings-algoritmer

Overfitting, eller overtilpasning på norsk, refererer til en situasjon innenfor maskinlæring hvor en modell lærer treningsdataene for godt, til det punktet hvor den fanger opp støy eller tilfeldige svingninger i dataene som om de var meningsfulle mønstre. Dette resulterer i en modell som gjør det veldig bra på treningsdata, men som presterer dårlig til ny og ukjent data. Det vil si at på et visst punkt vil modellen slutte å bli bedre, eller til og med blir dårligere.

For å forhindre overfitting er det viktig å redusere støy i datasettet, dette gjør at modellen ikke trener og lærer av støyet. Det er også viktig å ha et tilstrekkelig stort og variert datasett som viser ulike scenarier og variasjoner. I tillegg kan man bruke *early-stopping* metoden for å få kontroll over treningen av modellen. Da er det viktig å stoppe på riktig tidspunkt, stopper man for tidlig oppnår man *under-fitting* og stopper man for sent får man *overfitting* [50].

2.6 Standarder

2.6.1 Kapslingsgrad

International Protection (IP) -klassifisering spesifiserer i hvilken grad elektrisk utstyr er beskyttet mot berøring og andre ytre påvirkninger som støv eller fuktighet. Standarden er utarbeidet av den International Electrotechnical Commission (IEC) som utvikler lover innen elektrofaget [18]. I Norge representeres denne komiteen gjennom Norsk Elektroteknisk Komité (NEK)[17].

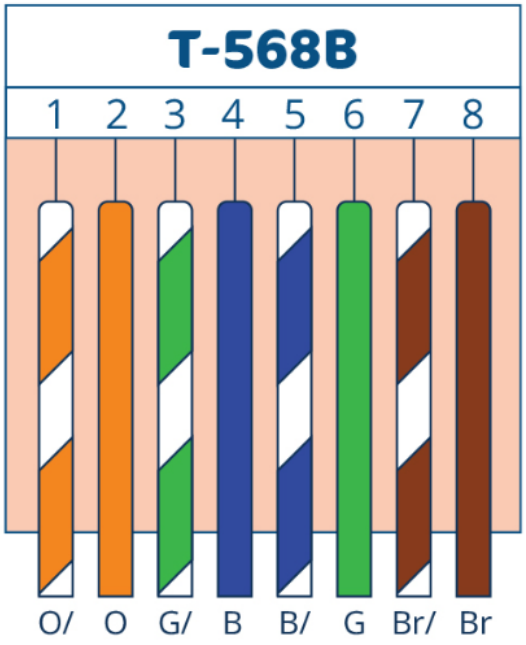
IP-klassifiseringen til komponenter blir representert med to sifre. Det første sifferet representerer beskyttelse mot faste gjenstander, mens det andre representerer mulighet for inntrenging av vann. I begge tilfeller vil høyere tall bety bedre beskyttelse. På figur 3 kan man se betydningen av de tre første sifferene i standarden. I tilfeller der det ikke stilles krav til en av inntrengnings metodene vil dette tallet byttes med en X [16].

Første siffer	Beskyttelse mot inntrengning av faste gjenstander/partikler	Andre siffer	Beskyttelse mot skadelig inntrengning av vann
0	Ubeskyttet	0	Ubeskyttet
1	Diameter ≥ 50 mm	1	Vertikale drypp
2	Diameter ≥ 12,5 mm	2	Vertikale drypp og utstyret med helning på 15º

Figur 3: IP Klassifisering, hentet fra [16]

2.6.2 T568-B

T568-B er en del av TIA/EIA-568 spesifikasjonene som er utarbeidet av Telecommunications Industry Association (TIA) og Electronic Industries Alliance (EIA) [1]. Den første utgaven av denne standarden ble utgitt i 1998 og har siden dette gått gjennom en rekke revisjoner. T568-B er, sammen med T568-A, de mest utbredte standarden for kategori 6a kabler, og definerer rekkefølgen på lederene i en 8 pins RJ-45 kontakt som vist i figur 4. Det er ingen forskjeller i hastighet og kvalitet mellom de to standardene, så valget vil derfor avhenge av andre faktorer som kundepreferanser eller eksisterende systemer [6].



Figur 4: T-568B-standard, hentet fra [1]

3 Material

I denne delen av rapporten vil det bli sett nærmere på det fysiske utstyret 3.1 og programvarene 3.2 som er benyttet. Dette omfatter en detaljert gjennomgang av både funksjonaliteten til de ulike komponentene og deres tekniske spesifikasjoner.

Materiallister

I tabell 1 listes alt av materiell som ble benyttet i konstruksjonen av lidarstativet. En skisse av dette stativet vises i figur 9, mens dens funksjonalitet blir beskrevet i seksjon 4.2.2.

Lidarstativ		
Navn og beskrivelse	**Varenummer**	**Mengde**
Aluminiumsprofil, 40x40	761-3319	103cm
Aluminiumsprofil vinkelfeste, 40x40x40	767-5695	4stk
Hexskrue, A4 M8x70mm	660-4658	2stk
Hexskrue, A4 M8x45	660-4649	4stk
Hexskrue, A4 M8x80mm	660-4651	4stk
Hexmutter, A4 M8	761-3319	6stk
Locking ring washer, A4 M8 6,1	H-HLRS8S	6stk

Tabell 1: Lidarstativ materialliste

Tabell 2 viser alt material som ble brukt i fabrikasjonen av ethernetkabelen. Kabelen kan sees i figur 16b og fremgangsmåten for hvordan kabelen ble laget er beskrevet i seksjon 4.2.4.

Ethernetkabel		
Navn og beskrivelse	**Varenummer**	**Mengde**
Ethernetkabel, SF/UTP RJ45 Cat6a	VLCP85320R50	5m
Amphenol conec plugg, RJ45 8P8C IP67 shielded	17-150234	2 stk
Krympestrømpe, 9.0-3.0mm	1835672	5.5cm

Tabell 2: Ethernetkabel materialliste

Materialet som ble brukt for å konstruere strømkabelen til interfaceboksen er alle opplistet i tabell 3. Fremgangsmåten for utførelsen med kobling av strømkabel til interfaceboks er videre beskrevet i 4.2.4.

Strømkabel til Interfaceboks		
Navn og beskrivelse	**Varenummer**	**Mengde**
Molex connector plug, nano-fit	105307-1202	1stk
Molex crimp terminal, 20-22AWG	105300-2200	2stk
Molex retainer, nano-fit	105325-1002	1stk
Barrel-jack cable mount, locking mekanismen må files ned	884-0932	2m

Tabell 3: Strømkabel til interfaceboks materialliste

I tabell 4 vises en liste med alt øvrig materiell som ble brukt for å gjennomføre prosjektet.

Øvrig materiell		
Navn og beskrivelse	**Varenummer**	**Mengde**
Ruter, TP-LINK	TL-WR841N V14	1stk
Universaltau, bruddstyrke 960kg 8mmx30m	25-1010	30m
Karabinkrok, bruddstyrke 800kg 8x80mm	25-0162	1stk
Overgangskabel, CEE til schuko	44-571	1stk
Raspberry Pi, 4B	3051887	1stk

Tabell 4: Øvrig materiell matrialliste

3.1 Fysisk utstyr

3.1.1 Ouster lidar

Ouster OS-1 er en lidar for bruk på medium avstand med høy oppløsning, produsert av Ouster. Dette amerikanske selskapet, som ble grunnlagt i 2015, spesialiserer seg på produksjon av lidar-teknologi. Deres hovedmål er å anvende denne teknologien i autonome kjøretøy for å forbedre kjøretøyenes situasjonsforståelse, noe som vil bidra til økt trafikksikkerhet.

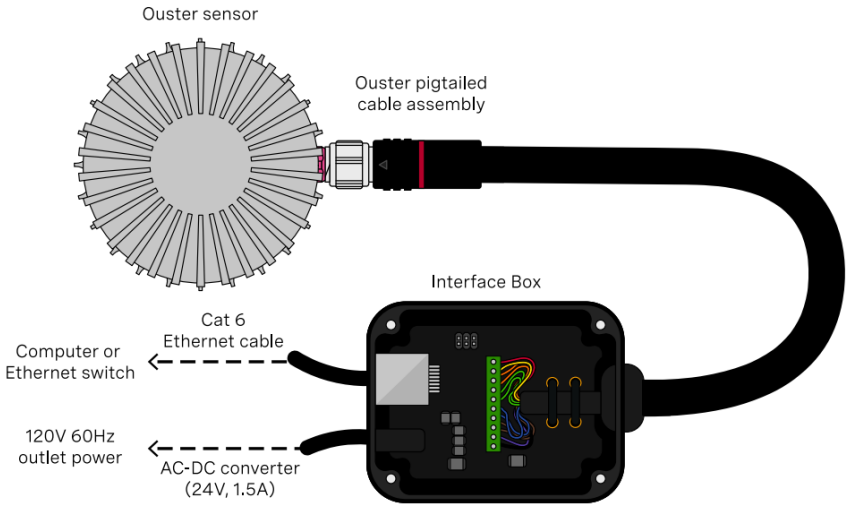
Lidaren som er tatt i bruk i denne oppgaven er en Ouster OS-1 som vist på figur 5. Dette er en middels rekkevidde-lidar med en maksimal måleavstand på 120 meter. Med en oppløsning på opptil 1024x64 punkter, et horisontalt synsfelt på 360° og et vertikalt synsfelt på 45° er den i stand til å levere med høy presisjon og en vinkelnøyaktighet på +/- 0,01°. Lidaren leverer datapunkter som inkluderer avstand, signalintensitet, reflektivitet, omgivelseslys, vinkel og tidsstempel, med en oppdateringsfrekvens på opptil 20 Hz. Det er en robust enhet med en kapslingsgrad på IP68 og kan operere i temperaturer fra -40°C til +60°C. Med alle disse kvalifikasjonene egner lidaren seg godt til bruk på autonome fartøy [37].

Figur 5: Lidar OS-1, hentet fra [9]

Lidaren er også utstyrt med en innebygd IMU, som inkluderer et MEMS-akselerometer og et MEMS-gyroskop [13]. Dataen sendes over UDP, og vi får tilgang til en rekke informasjon, blant annet IMU diagnostic time, akselerometer run time, gyroscope run time, vinkelhastighet i x-, y- og z-aksene, samt akselerasjon i x-, y- og z-aksene [39]. Strukturen til disse dataene vil bli beskrevet senere i rapporten under 4.4.1.

3.1.2 Ouster interfaceboks

Lidaren beskrevet i seksjon 3.1.1 leveres også med tilhørende interfaceboks for strømforsyning og nettverkstilkobling. For å sikre at den fungerer med ulike systemer, finnes det 3 typer interfacebokser, type 1 og 2 støtter kun 24V spenning, mens type 3 støtter både 12V og 24V. I dette prosjektet benyttes type 3 interfaceboks og 12V som blir forsynet via 5.5x2.5 VIN barrel jack plug. Som vist på figur 6 gjør interfaceboksen det også mulig for datakommunikasjon via ethernet og CAT 6 kabel. Dette blir etablert med bruk av en 8 pin modular jack og en standard RJ45-kontakt. Interfaceboksen gjør det dermed mulig med strømforsyning og nettverkskommunikasjon som integreres til en kabel for å forsyne lidaren. Videre er interfaceboksen ment for innendørsbruk i rene og beskyttede miljøer ved temperaturer mellom -20 og 50 grader [30].

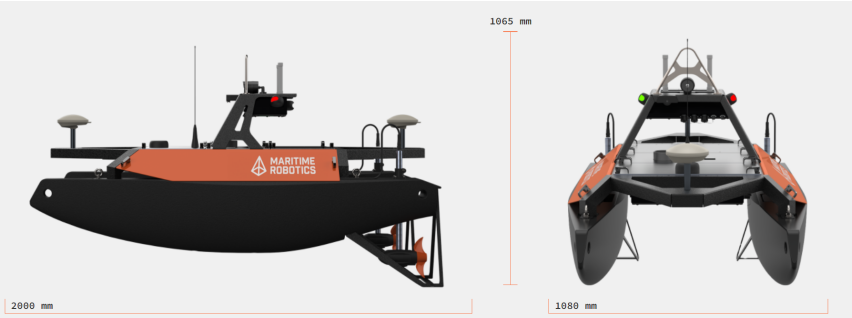


Figur 6: Tilhørende interfaceboks, hentet fra [32]

3.1.3 Otter USV

Det ubemannede fartøyet, også kjent som Otter USV, er produsert av firmaet Maritime Robotic som er lokalisert i Trondheim. Firmaet ble etablert i 2005 og er i dag en etablert leverandør av autonome teknologiløsninger innen marin kartlegging, miljøovervåking og inspeksjoner.

Otter USV'en som er tatt i bruk gjennom oppgaven er som vist på figur 7 en 2000mm x 1080mm x 1065mm katamaran. Fremdriftssystemet til fartøyet består av to elektriske thrustere drevet av 2 til 4 lithium-ion batterier. Dette gir den en toppfart på 6 knop, og muligheten til å operere fartøyet sammenhengende opp til 20 timer. Operasjonstiden vil naturligvis variere utifra forholdene og vekten på fartøyet. Otteren kan ha en nyttelast opp til 30kg i tillegg til en egenvekt på 62kg .



Figur 7: Otter USV, hentet fra [20]

Fartøyet er utstyrt med diverse kommunikasjonsmetoder som 5GHz, MIMO-radio og WiFi som muliggjør pålitelig dataoverføring tilpasset situasjonens behov. Videre er fartøyet utstyrt med et kamera som er designet for å gi situasjonsforståelse av omgivelsene. I tillegg til dette standardutstyret kan Otter USV'en utstyres med GPS for nøyaktig posisjon og heading eller lidar for å øke situasjonsforståelsen av omgivelsene [21].

3.2 Programvarer

Tabell 5 viser alle biblioteker og programvare som er brukt i prosjektet. Siste kolonne i tabellen refererer til seksjonene som gir en mer detaljert beskrivelser av programvarene.

Programvare		
Navn	**Versjon**	**Seksjon**
PyCharm	v.2023.3.3	3.2.1
Ouster studio	v.2.0.3	3.2.2
Autodesk Fusion 360	v.2.0.16976	3.2.3
Prusa slicer	v.2.7.4	3.2.4
RoboFlow	Web applikasjon	3.2.5
Ouster-SDK	v.0.10	3.2.6
OpenCV	v.4.9.0.8	3.2.7
Ultralytics YOLO	v.8.1.47	3.2.8

Tabell 5: Liste over programvarer

3.2.1 PyCharm

PyCharm er et integrert utviklingsmiljø (IDE) utviklet av det Tsjekkiske selskapet JetBrains og tas i bruk for koding i Python. Utviklingsmiljøet tilbyr flere funksjoner som forbedrer prosessen med effektiv koding og feilsøking, som blant annet en kodeeditor med syntaksfremheving, kodefullføring og feilmarkeringer. I tillegg er det tilgjengelig diverse verktøy for feilsøking, testing, profilering og integrering av versjonskontroll. Programvaren er også tilgjengelig i to utgaver: Community-utgaven og Professional-utgaven. Med disse nevnte funksjonene og utgavene av programvaren er PyCharm brukervennlig samt tilbyr en nyttig brukerveiledning [14].

3.2.2 Ouster Studio

Ouster Studio er en programvare utviklet av det Amerikanske firmaet Ouster og brukes for å kunne visualisere, registrere og analysere data fra deres egne lidarer. Programvaren er kompatibel med flere plattformer som gjør at den kan tas i bruk på Windows, MacOS og Ubuntu. Med diverse funksjoner for visualisering, opptak og analysering av data er programvaren brukervennlig og godt egnet til de fleste oppgaver.

3.2.3 Autodesk Fusion 360

Autodesk Fusion 360 er en skybasert programvare utviklet av Autodesk og ble utgitt for første gang i 2013. Programvaren er utvikler for 3D-modellering, simulering og produksjon. Fusion 360 blir i dag brukt av både ingeniører og privatpersoner for design og utvikling av deler til en rekke industrier. Programvaren er brukervennlig og tilbyr samarbeidsfunksjoner for team, samt et bredt utvalg av verktøy som gjør den egnet til enhver bruk [7].

3.2.4 Prusa slicer

Prusa Slicer er et åpent kildekode program som brukes til å *slice* 3D-modeller. Programvaren inneholder en rekke hjelpefunksjoner og parametre som valg av fyll, print kvalitet og tilpassbare støtter på 3D-modellene, noe som gjør programvaren lett å ta i bruk. Prusa Research som er utvikleren av programvaren oppdaterer den jevnlig og tilser at den inneholder alt som trengs for å eksportere filer til Prusa sine 3D-printere [43].

3.2.5 Roboflow

RoboFlow ble lansert i 2020 og er en maskinsyn-platform med fokus på å forbedre og automatisere prosessen med å lage datasett for maskinlæring. Platformen er åpen kildekode og inneholder en rekke ferdigtrente datasett tilgjengelig for alle brukere av Roboflow. Den inneholder funksjoner som sanntidssamarbeid og automatisk markering av objekter i bildene i datasettet, noe som gjør platformen lett å ta i bruk for utviklere. Datasettene som kan lastes ned gjennom platformen kommer i er rekke formater som yolo, coco og tensorflow [46].

Pythonbibliotek:

3.2.6 Ouster-SDK

Ouster-SDK er et åpent kildekode bibliotek designet for å integrere, administrere og manipulere data innsamlet gjennom Ouster sine lidarer. Biblioteket har vært i utvikling siden 2019 mens den nyeste tilgjengelige versjonen er Ouster-SDK 0.10.0 ble utgitt i 2023. For å ta i bruk denne versjonen kreves Python 3.7 eller nyere. Biblioteket gir brukere enkel tilgang til sensorinnstillinger, konfigurere parametre og ta opptak av sensordata. I tillegg til dette gjør biblioteket det letter å konvertere rå lidardata til mer håndterbar data som avstand, signal, nær-IR eller refleksjon som kan brukes til å visualisere omgivelsene. Med disse funksjonene er Ouster-SDK et brukervennlig bibliotek som gjør det lettere å håndtere data fra Ouster lidarer [42].

3.2.7 OpenCV

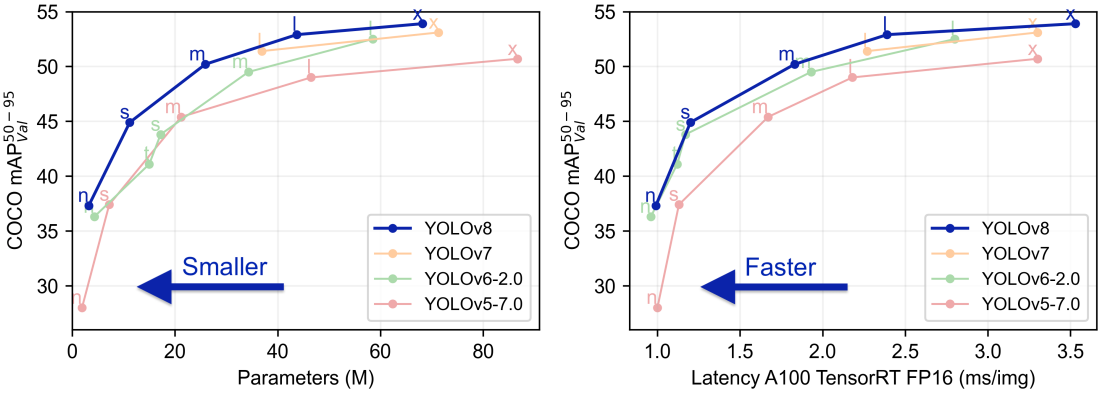
OpenCV, kort for *Open Computer Vision*, er et åpent kildekode data- og maskinlæringsbibliotek designet for å hjelpe utviklere med å lage applikasjoner i sanntid for bilde og videobehandling. Biblioteket ble utviklet av Corporation og vedlikeholdes i dag av OpenCV Foundation. Biblioteket støtter Windows, Linux, Android og Mac samt er tilgjengelig i programmeringsspråk som C++, Java og Python 3.6 eller nyere.

Biblioteket inneholder et bredt spekter av algoritmer for både bildebehandling og datamaskinsyn, som inkluderer funksjoner for deteksjon og sporing av objekter, segmentering av bilder, deteksjon av særtrekk og analyse av optisk strømning. Med dette har OpenCV Fundation mål om å fremme forskning og utvikling innen datamaskinsyn og gjøre det lettere for utviklere å lage applikasjoner i sanntid for bilde- og videobehandling [29] [45].

3.2.8 Ultralytics YOLO

Ultralytics YOLO er et åpent kildekode bibliotek utviklet med løsninger for AI-oppgaver, inkludert deteksjon, segmentering, klassifisering, sporing og posisjonsestimering. YOLOv8 er en av de nyeste versjonene av algoritmen og skal være raskere og mere optimalisert for objektdeteksjon i sanntid [48].

YOLO eller *you only look once* algoritmen er kjent for sin effektivitet innen sanntids objektdeteksjon. Den fungerer ved at den deler inn input bildet inn i et nett og designerer hver celle av nettet ett ansvar for å forutsi sannsynligheten for at cellen inneholder et objekt av en spesiell klasse. Ved bruk av convolutional neural network (CNN) kan YOLO hente ut relevant informasjon slik som siluetter og teksturer. YOLOv8 leverer fem forskjellige ferdige modeller, YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large) og YOLOv8x (extra large)[15]. Det er per dags dato 9 ulike versjoner av YOLO algoritmen, YOLOv9 er den nyeste og kom ut i februar 2024. Versjonene og deres effektivitet kan sees i figur 8.



Figur 8: YOLOv8 modeller, hentet fra [48]

4 Metode

I denne delen av rapporten vil det bli gitt en detaljert beskrivelse av fremgangsmåten som ble fulgt i arbeidet med oppgaven. Det vil bli beskrevet hvordan materialet nevnt i seksjon 3 er brukt, samt hvordan kommunikasjonen mellom enhetene ble opprettet. For å dokumentere dette på best mulig måte vil det tas i bruk en rekke bilder av arbeidsprosessen, innsamlet data og koblingsskjema av oppsettet.

4.1 Planlegging

Startfasen av prosjektperioden ble i hovedsak brukt til å finne ut hva man ønsket å oppnå med prosjektet og hvordan prosjektperioden skulle struktureres. Dette arbeidet gikk ut på å fastsette faste møter med veiledere, arbeidsrutiner i gruppen, framdriftsplan for prosjektet, hvordan arbeidet skal dokumenteres og risikoen knyttet til prosjektet. Resultatet av dette arbeidet kan sees i forprosjektrapporten under vedlegg. Ved å legge ned dette arbeidet var vi i stand til å følge den oppsatte planen, noe som har muliggjort en effektiv og sikker gjennomføring.

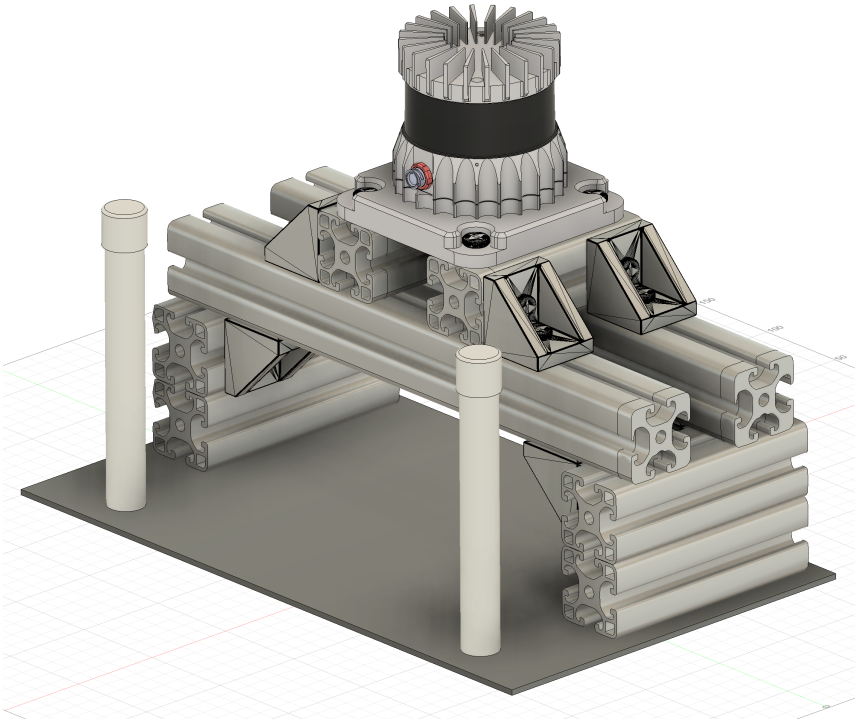
4.2 Montering og oppkobling

4.2.1 Modifisering av koblingsboks

Når payloadboksen på Otteren allerede er tatt i bruk, ble det gjort noen modifikasjoner for å få plass til utstyret vårt. Når den tilhørende interfaceboksen monteres inne i payloadboksen, var det nødvendig å lage et nytt hull for gjennomføringen av tilførselskabelen til lidaren. Dette ble gjort med å maskere de utsatte delene inne i boksen for å beskytte elektronikken mot metallspon. Deretter ble det tatt i bruk en skruehullstanser med en diameter på 20.4 mm. For å opprettholde kapslingsgraden til systemet ble det brukt en nippel med IP67 for gjennomføringen. Det var i utgangspunktet planlagt å lage en gjennomføring for ethernettkabelen til interfaceboksen, men dette er ikke ønskelig å gjøre før problemet med kommunikasjonen, beskrevet i seksjon 6.1.3, er løst.

4.2.2 Montering av lidar

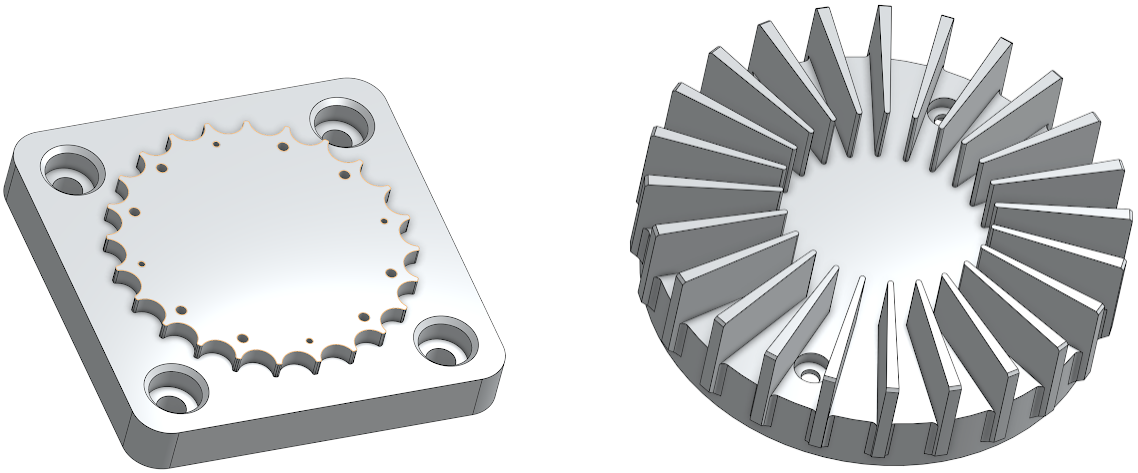
Når lidaren skulle monteres på Otteren var det ønskelig å montere sensoren på det høyeste punktet på båten. Grunnen til dette er hovedsakelig for å sikre at den får bra rekkevidde og et godt overblikk over omgivelsene rundt fartøyet. Samtidig vil en høy plassering forhindre at komponenter kommer i konflikt og blokkerer sikten til lidaren. For å oppnå dette ble stativet vist i figur 9 designet.



Figur 9: 3D-modell av lidar stativet

Ved installasjon av lidaren på Otteren ble det derfor brukt aluminiumsprofiler med en dimensjon på 40x40 mm. Dette sikrer at lidaren har et stabilt fundament å stå på, samtidig som vi får hevet den til et høyt punkt. Bakgrunnen for valget av aluminiumsprofiler er at dette er et metall med lav tetthet noe som gjør det forholdsvis lett. Dette er fornuftig å ta i bruk når Otteren har en maksimal lasteevne på 30kg som er ønskelig å utnytte mest mulig. Videre har aluminium god styrke og stivhet i forhold til sin egen vekt samtidig som det er et korrosjonsbestandig material. Det er også et material med høy varmeledningsevne noe som er anbefalt å ta i bruk ved montering ifølge Ouster [35]. Alle disse egenskapene gjør det godt egnet til vårt bruk på sjøen.

En annen faktor som ble tatt i betraktning under designet er varmgang i lidaren ved kjøring over lengre perioder. Grunnet dette er lidaren utstyrt med en kjøleribbebase og en radial kjøleribbe for termisk styring, som er avbildet på bilde 10 under. Dette bidrar til å holde sensoren på ønsket operasjonstemperatur. Ouster har selv utført tester for å finne ut hvor varmen konverterer ut til omgivelsene. Basert på disse testene forsvinner rundt 38% av varmen ut gjennom kjøleribbebasen som vist i figur 10a og de resterende 62% gjennom radial kjøleribbe som vist i figur 10b [40]. Grunnet dette er stativet designet slik at det er mulig med kontinuerlig luftsirkulasjon også under sensoren. I tillegg vil stativet, som er konstruert av aluminiumsprofiler, ha varmeledende egenskaper og fungere som et varmespredende skjold. Dette i tillegg til kjøling fra omgivelsene vil bidra til å holde operasjonstemperaturen på lidaren nede.

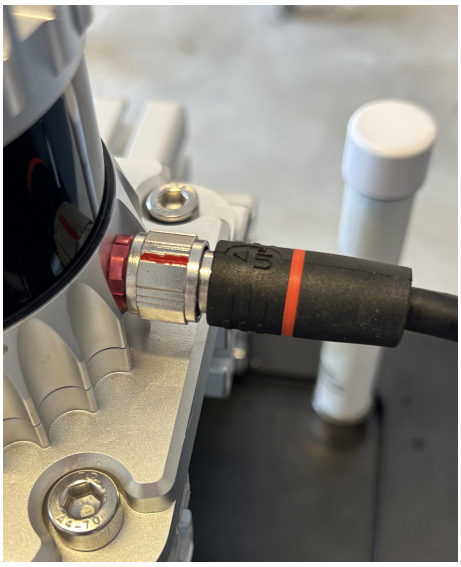


(a) Heatsink base

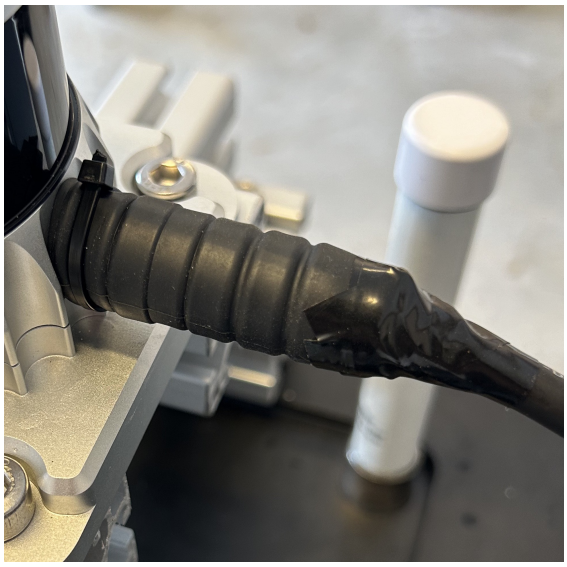
(b) Heatsink radial

Figur 10: Varmeskjold, hentet fra [41]

Ouster leverer lidaren med en gummimansjett som skal strekkes over koblingspunktet mellom lidaren og kabelen, som illustrert i figur 11. Mansjetten skal festes ordentlig over koblingspunktet, etterfulgt av stramming av en liten strips over selve tilkoblingen. Til slutt kan man forsegle mansjetten ved å teipe den med elektrikerteip, noe som hindrer vann og støvinntrengning. Når disse trinnene er utført, vil lidaren være klar til bruk under krevende forhold. En ferdig montert kabel med mansjett er vist i figur 11b. Denne prosessen er grundig beskrevet iOusters dokumentasjon [31].



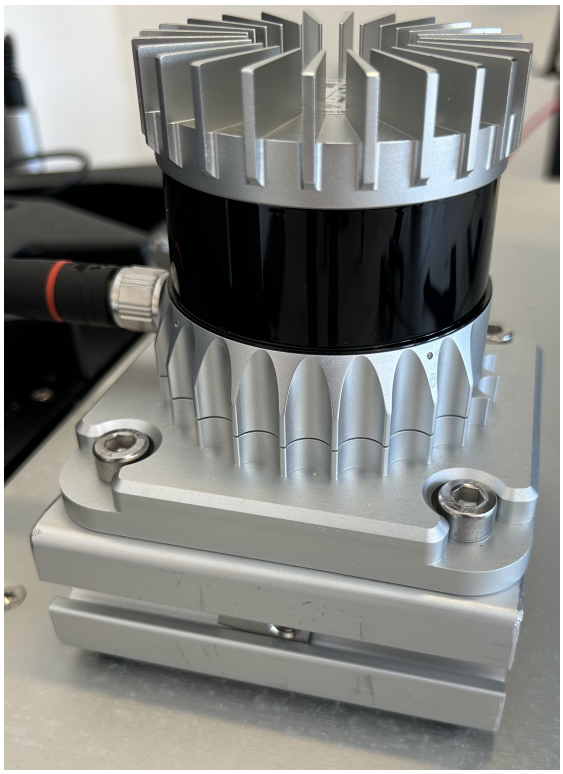
(a) Lidarkabel



(b) Lidarkabel med mansjett

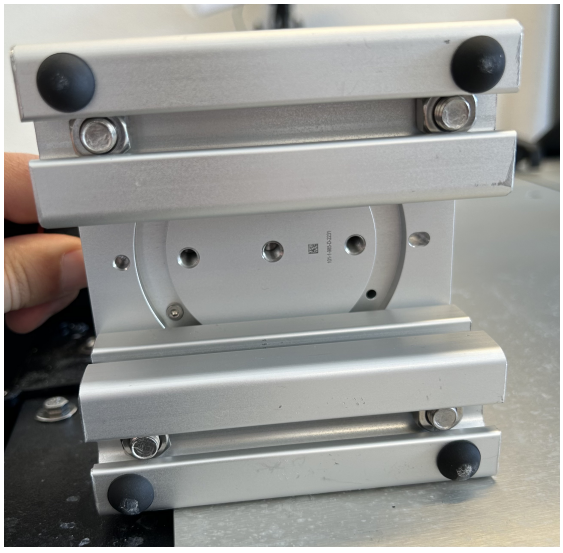
Figur 11: Lidarkabel og mansjett

Ved bruk av lidaren alene, uten resten av systemet, kan det være hensiktsmessig å demontere lidaren fra stativet. Lidaren er avhengig av tilstrekkelig luftgjennomstrømning for å unngå overoppheting. En vesentlig del av luftingen skjer gjennom kjøleribbene på toppen og bunnen av lidaren, som beskrevet i seksjon 4.2.2. Det er derfor viktig å la de to aluminiumsprofilene under lidaren være montert sammen med lidaren, som vist i figur 12. Dette løfter lidaren fra underlaget og bidrar til luftkjøling fra undersiden.



Figur 12: Lidar stående på aluminiumsprofiler

Lidaren kan generere betydelig vibrasjon, spesielt når den er i kontakt med et hardt underlag, noe som resulterer i økt støy. For å motvirke dette, har fire gummiknotter blitt limt på de to aluminiumsprofilene, som vist i figur 13. Disse knottene fungerer som vibrasjonsdempere, reduserer støy og sikrer dermed en mer behagelig og optimal drift av lidaren.

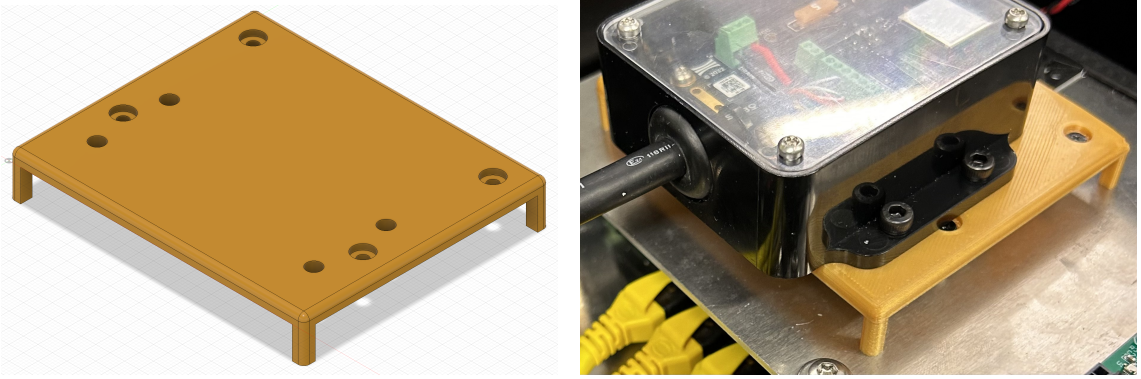


Figur 13: Gummiknotter under lidar

4.2.3 Montering av interfaceboks

For å unngå å lage unødvendige hull i payloadboksen, ble det laget en spesialdesignet monteringsplate til interfaceboksen ved hjelp av 3D-printing. Denne monteringsplaten har blitt utformet for å løfte interfaceboksen over eksisterende ledninger, samtidig som den sikrer at ledningene tilknyttet interfaceboksen opplever minimal bøyning. Dette bidrar til å opprettholde integriteten til både ledningene og selve interfaceboksen, samtidig som det forenkler installasjonsprosessen ved å unngå behovet for å bore nye hull i payloadboksen.

I figur 14a vises en 3D-modell av monteringsplaten. Denne modellen ble utformet ved hjelp av Autodesk Fusion 360. Monteringsplaten ble designet med hull til skruehodene, slik at interfaceboksen ligger i plan med monteringsplaten. Benene til platen ble laget i akkurat samme høyde som en 11 mm spacer, noe som sikrer jevn fordeling av vekten mellom spacerne og benene. Kantene rundt platen og benene er avrundet for å redusere slitasje på ledninger som kan gni mot kantene. I figur 14b vises interfaceboksen montert i payloadboksen.



(a) Ouster interfaceboks 3D-modell av monteringsplate

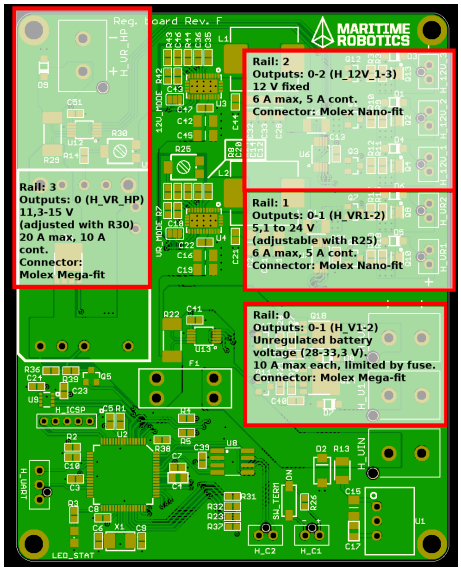
(b) Ouster interfaceboks montert i payloadboks

Figur 14: Ouster interfaceboks monteringsplate

4.2.4 Spenningstilførsel og kommunikasjon til Lidar

Otteren har hovedsaklig to koblingsbokser installert, der den ene inneholder utstyr fra produsenten Maritime Robotics, mens den andre benyttes av NTNU. Koblingsboksen benyttet av NTNU var allerede utstyrt med noen komponenter som blant annet en switch med ethernet porter, et PCB kort og en PC. I tillegg er det diverse gjennomføringer som i hovedsak er for spenning og kommunikasjon ut og inn av boksen.

Koblingsskjema for PCB kortet ble tilsendt av Maritime Robotic og kan sees på figur 15 under. Otteren er utstyrt med to litiumbatterier på 29V, disse er i tillegg til å drive Otteren, brukt som spenningsforsyning inn på koblingspunktet H_VIN på PCB kortet. Videre har PCB kortet en rekke utganger med justerbar spenning mellom 11.3-15V og 5.3-24V i tillegg til noen utganger satt til 12V, som alle er markert på figuren. Lidaren har en operasjonspenning på 12-24V og trekker med dette en strøm på 1 - 3.3 A, oppgitt på enheten. Grunnet disse spesifikasjonene ble H_VR1, etter å blitt aktivert som beskrevet i seksjon 6.1.2, justert til 12V ved å bruke potmeteret R25. Som tilførselskabel ble det brukt en DC-plugg som forsyner lidaren via tilhørende interfaceboks. Denne ble montert med molex-kontakt for å kunne tas i bruk på PCB kortet.



Figur 15: Koblingsskjema PCB kort

For å forsikre at tilførselen til lidaren var tilstrekkelig ble det tatt utgangspunkt i informasjonen i databladet [37] der det er oppgitt at lidaren bruker en spenning på 12V og et maksimalt strømforbruk på 22W ved oppstart. Med utgangspunkt i denne informasjonen ble følgende beregninger gjort ved bruk av formel 4:

$$I = \frac{P}{V} \tag{4}$$

hvor:

- I: Strømmen i ampere (A)
- P: Effekt i watt (W)
- V: Spenning i volt (V)

Med utgangspunk i:

$$P = 22\,\mathrm{W}$$

$$V = 12\,\mathrm{V}$$

Blir strømforbruket:

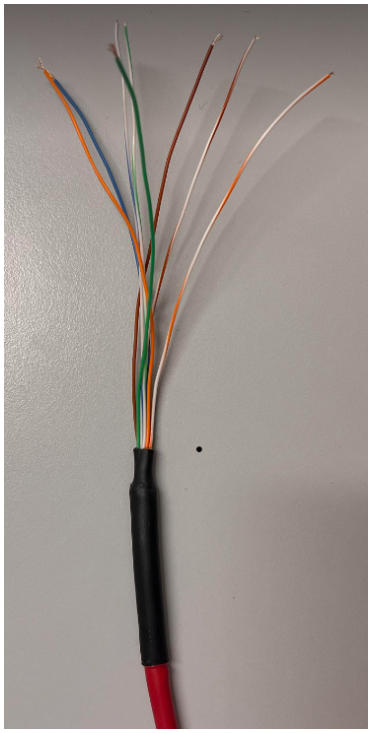
$$I = \frac{22\,\mathrm{W}}{12\,\mathrm{V}} = 1.83\,\mathrm{A} \tag{5}$$

Porten som tas i bruk for å forsyne lidaren er H_VR1 som vist på koblingsskjemaet 15. Som vist på bildet kan porten levere et strømforbruk opp til 5A og en spenning på 24V. Basert på beregningene i formel 5 vil lidaren ha et strømforbruk på 1.833A ved en spennnig på 12V noe som vil tilsi at at den tiltenkte porter H_VR1 vil være mer en tilstrekkelig for å forsyne lidaren.

For å klare å forsyne lidaren og interfaceboksen med strøm trenger man en tilpasset strømkabel. PCB kortet bruker Molex nano-fit kontakter og interfaceboksen bruker barrel-jack, dermed trenger vi en kabel med barrel-jack til molex nano-fit kontakt.

For å kommunisere mellom interfaceboksen og ruteren tas det i bruk en CAT6 ethernet kabel. Denne blir lagt fra interfaceboksen i koblingsboksen og opp til ruteren montert på toppen av fartøyet. Når utstyret på Otteren kan bli utsatt for vannsprut var det viktig å opprettholde IP-klassifiseringen på fartøyet. Vi hadde vanskeligheter med å få tak i en ethernet kabel levert med RJ45 connector med en IP på 67, og ble derfor nødt til å lage disse selv.

Dette ble gjort med å klippe ene enden av kabelen for å så tre på en Conec RJ45 plugg, med IP 67. Deretter avmantlet vi kabelen og trakk på krympestrømpe, som vist i figur 16a. Videre ble en 8P8C ethernetkontakt montert etter T568-B standarden. Når kontakten var ferdig montert ble kablelen testet med bruk av en nettverkstester, som vi ser i figur 16c for å verifisere at alt fungerte som tiltenkt. Den endelige kabelene vises i figur 16b.



(a) Avmantlet ethernetkabel

(b) Ethernet med RJ45 plugg montert

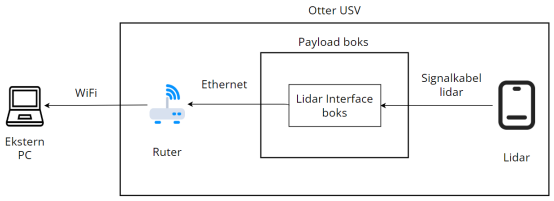
(c) Test av kabel

Figur 16: Tilpasset ethernetkabel

4.2.5 Kommunikasjon

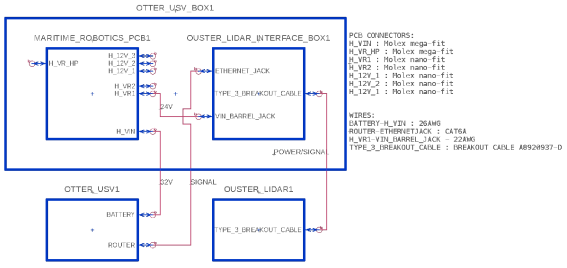
Det opprinnelige kommunikasjonssystemet som var planlagt på Otteren var å integrere lidaren inn i det eksisterende nettverket. Dette for å kunne sende data over WiFi til en ekstern PC for behandling av dataen. Dette ble gjort med å koble lidaren via ethernet til switchen som ligger i payload boksen, som derifra går videre til OBS boksen der den integrerte ruteren ligger. Under arbeidet med integreringen klarte vi ikke å finne lidaren på nettverket. Den videre prosessen med feilsøking er beskrevet i diskusjon 6.1.3.

Grunnet denne feilkilden ble det utviklet en alternativ metode for å opprette kommunikasjon med lidaren, og kommunikasjonsflyten kan sees på bildet 17. Dette oppsettet inkluderer en ekstern ruter som tildeler IP-adresser dynamisk via DHCP til enhetene i nettverket. Ruteren kommuniserer direkte med lidaren via ethernet til lidarens interfaceboks som dermed vil få en IP adresse. Dette skaper et lokalnett på Otteren, hvor brukere kan koble seg til ved å autentisere med riktig brukernavn og passord. En viktig faktor er at ruteren fungerer uavhengig av eksterne nettverk og kun opererer som et lokalt kommunikasjonspunkt på Otteren.



Figur 17: Kommunikasjonsflyt

All kommunikasjon mellom Otteren og ekstern PC skjer via ruteren over WiFi, der det mottas datapakker fra lidaren kontinuerlig via UDP. På denne måten kan vi koble oss på den eksterne ruteren med PC og motta lidardata trådløst over WiFi. Når vi jobber med et fjernstyrt fartøy som skal bevege seg i offentlig farvann er det viktig å utføre operasjonen sikkert. Dataen blir som nevnt i dag sendt over et lokalt nettverk på Otteren. Dette kan begrense rekkevidden noe men vil også gjøre det vanskelig for utestående å koble seg på og ta kontroll over fartøyet. Ved å sende dataen over UDP kan det føre til noe pakketap. I vårt tilfelle vil data bli sendt kontinuerlig så selv med noe pakketap vil man kontinuerlig motta data og klare å visualisere omgivelsene til lidaren. I denne tilnærmingen vil ikke noe data bli behandlet lokalt på Otteren, dette er en videre forbedring beskrevet i seksjon 6.2.1.

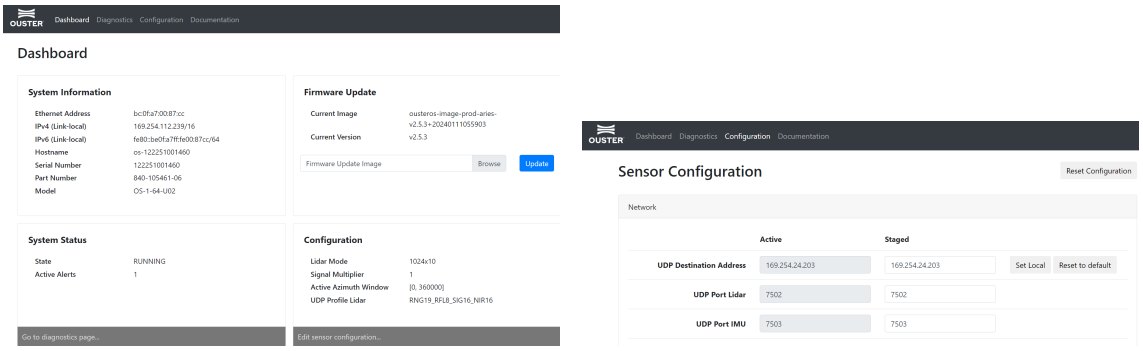


Figur 18: Koblingsskjema for strømforsyning og kommunikasjon

4.3 Lidar

4.3.1 Kommunikasjon og behandlig av lidardata

For å opprette kommunikasjon med lidaren bruker man dashboardet som Ouster har designet. Dette nås via *hostname*, som i vårt tilfelle er *os-122251001460.local*, og vi føres til Ouster-dashboardet som vist i figur 19.



(a) Skjermbilde av dashboard (b) Skjermbilde av konfigurasjonsdashboard

Figur 19: Skjermbilde av lidar-dashboard

Her har man mulighet til å lese av lidarens IP-adresse, samt konfigurere IP-adressen til mottakeren av dataene, som vist i figur 19b. Dette gjøres enkelt ved å velge *set local* for å finne mottakerens IP-adresse når enhetene er koblet sammen via ethernet, eller manuelt skrive inn IP-adressen til UDP-mottakeren. I tillegg har man informasjon om en rekke andre parametre som systemstatus, oppdateringer og feilmeldinger som eventuelt oppstår.

På mottaker siden opprettes kommunikasjon ved bruk av IP-adresse eller hostnavnet til lidaren. I tillegg må man velge UDP-porten som dataen blir sendt over, denne er oppgitt i Ouster dashboardet. I vårt tilfelle bruker vi hostnavnet til sensoren og port 7502 som vist i kodesnutt 1.

```
#lidar-konfigurasjon
hostname = "os-122251001460.local" #IP-adresse eller hostname
lidar_port = 7502 #standard port
```

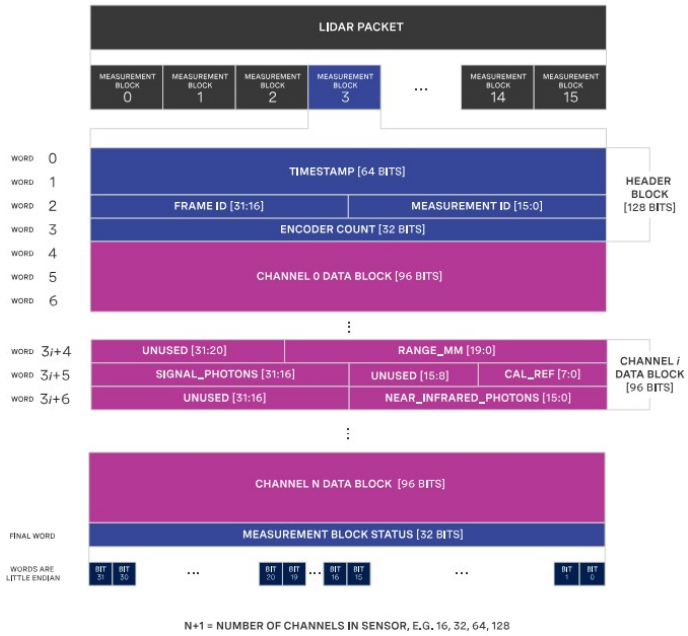
Kodesnutt 1: lidar konfigurasjon

For å opprette kontinuerlig tilkobling til lidaren og starte dataoverføringen brukes koden i kodesnutt 2. Her brukes `with closing()` for å sikre at overføringen av data avsluttes skikkelig uansett hvordan koden avsluttes. Videre brukes funksjonen `client.Scans` fra Ouster-SDK til å opprette kontinuerlig datastrøm fra lidaren. Sammen med parametre som `hostname` og `lidarport` som vist i kodesnutt 1. I tilleg blir argumentet `complete=False` definert for å kunne motta ufullstendige pakker. Til slutt er `for scan in stream` brukt for å iterere over hver `scan` som mottas og behandle denne videre.

```
with closing(client.Scans.Stream(hostname, lidar_port, complete=
    False)) as stream:
     for scan in stream:
```

Kodesnutt 2: Oppretting av datasrøm fra lidar

Som vist i figur 20 blir datapakkene som mottas fra lidaren overført som pakker bestående av 16 måleblokker. Her er hver blokk tilpasset antall kanaler tilgjengelig på lidaren. Måleblokkene inneholder en header blokk som inkluderer data om tidsstempling, ID for målingene, ramme ID for fullstendig rotasjon i skanningsprosessen og encoder count som reflekterer vinkelposisjonen til lidaren under skanning. Videre mottas datablokker som inneholder målt avstand, kalibrert reflektivitet, signalstyrke og nær-IR informasjon. Det er disse blokkene som vil være forskjellig basert på antall kanaler sensoren har tilgjengelig. Til slutt indikerer måleblokken status gyldigheten av dataen som mottas hvorav noe av dataen kan være fylldata eller padding på grunn av manglende kanaler eller data utenfor det målte området.



Figur 20: Lidar pakker, hentet fra [33]

Dataen som mottas fra lidaren lagres i to hovedfilformater der ene filen er en JSON-fil som er tekstbasert og andre er en PCAP-fil som er for lagring av pakker. JSON-filen inneholder informasjon om konfigurasjon og kalibreringsdataen til sensoren som kanaloppsett, IP-adresser og kalibrerings parametre. Det er informasjonen i denne filen som gjør det mulig å tolke rådataene som mottas i PCAP-filen på korrekt måte. Videre inneholder PCAP-filen rådata fra lidaren som forklart og vist i figur 20 over. Ved å ta i bruk både JSON og PCAP-filen kan man rekonstruere nøyaktige 3D-representasjoner av omgivelsene for eksempel i Ouster Studio.

4.3.2 Visualiseringsmetoder

Som vist i figur 20 mottar lidaren fire datablokker som inneholder nyttig informasjon for visualisering av lidarens omgivelser. Disse fire metodene blir listet opp i tabell 6 og er Range, Near-IR, Signal og Reflectivity. Dette gjør det mulig for brukere å ta i bruk den metoden som passer best i hvert enkelt tilfelle.

Output	Description
Range	The distance of the point from the sensor origin, calculated by using the time of flight of the laser pulse
Signal	The strength of the light returned to the sensor for a given point. Signal for Ouster digital lidar sensors is expressed in the number of photons of light detected
Near-IR	The strength of sunlight collected for a given point, also expressed in the quantity of photons detected that was not produced by the sensor's laser pulse
Reflectivity	The reflectivity of the surface (or object) that was detected by the sensor

Tabell 6: Ouster lidar outputs, basert på [5]

Ved å ta i bruk disse forskjellige metodene for å visualisere omgivelsene kan man tilpasse visualiseringen til ønsket bruk. Range visualisering bruker *time-of-flight*-prinsippet som beskrevet i seksjon 2.1 under teori. Her blir omgivelsene visualisert i et gråskalabilde der fargene vil indikere avstander. De nærmeste objektene vises i varme farger mens fjernere objekter i kalde farger som vil gi en god forståelse av dybden i bildet. Signal metoden tar i bruk styrken av laserreturen og visualiserer dette gjennom intensitet. Her vil sterkere lys gi mer distinkte farger noe som vil gjøre visualiseringsmetoden godt egnet for identifisering av materialtyper og overflater. Visualisering med Near-IR benytter nær-infrarøde kameraer som fanger opp lys som er usynlig for det menneskelige øye. Dette gjør denne metoden godt egnet til lavlysforhold. Til slutt har vi reflectivity som baserer seg på hvor mye lys som reflekteres tilbake fra ulike overflater i omgivelsene.

4.3.3 Kontraster i bildet

Algoritmer som brukes til trening av modeller har evnen til å oppdage og skille forskjeller i gråskalabilder på et mye mer detaljert nivå enn mennesker. Dette skyldes algoritmenes evne til å gjenkjenne subtile forskjeller i intensiteten av de ulike gråtonene i bildene. For å gjøre bildene lettere å jobbe med ble det derfor eksperimentert med de forskjellige visualiseringsmetodene vist i figur 21 for å skape et bilde som ga tydelige kontraster.

Figur 21: Visualiseringsmetoder, hentet fra [38]

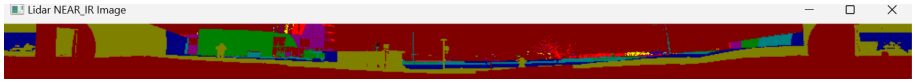
En tilnærming som ble testet for å oppnå dette var å legge på fargekart over bildene ved `create_high_contrast_colormap()` funksjonen vist i kodesnutt 3. Her opprettes det i `colors` en liste med en rekke farger i BGR-format som vil gi en tydelig kontrast mellom områdene i bildet. Videre beregnes det hvor mange piksler hvert fargeområde skal dekke ved å dele maksimal pikselverdi med antall farger som er definert i `colors`. Deretter opprettes et tomt fargekart med dimensjonene (256, 1, 3) som vist i `np.zeros` funksjonen. Her setter vi fargekartet til å ha 256 fargeområder med en høyde på en piksel og tre kanaler for å lese BGR-fargene. Til slutt går man gjennom hver farge i listen for å fylle fargekartet.

```
def create_high_contrast_colormap():
colors = [
    (0, 0, 128), (0, 128, 128), (128, 0, 0), (128, 128, 0), (0,
128, 0),
    (128, 0, 128), (0, 0, 255), (0, 255, 255), (255, 0, 0),
(255, 255, 0),
    (0, 255, 0), (255, 0, 255), (0, 0, 64), (0, 64, 64), (64,
0, 0),
    (64, 64, 0), (0, 64, 0), (64, 0, 64), (128, 128, 128), (64,
 64, 64),
    (192, 192, 192), (192, 0, 192), (192, 192, 0), (0, 192,
192), (192, 0, 0)
]
# calculate the bin size for each color
colormap = np.zeros((256, 1, 3), dtype=np.uint8)
n_bins = 256 // len(colors)  # Number of bins for each color
```

```
for i, color in enumerate(colors):
    start = int(i * n_bins)
    if i == len(colors) - 1:  # Last color should go to the end
        end = 256
    else:
        end = int((i + 1) * n_bins)
    colormap[start:end, 0, :] = color
return colormap
```

Kodesnutt 3: Oppretting av tilpasset fargekart

Resultatet fra denne funksjonen er vist i figur 22 under. Dette bildet er tatt på en åpen plass og vi ser her at det er tydelige kontraster mellom de forskjellige avstandene i bildet. På denne måten er det lettere å skille objekter fra bakgrunnen, noe som gjør detekteringen enklere. Her burde avstandene som er utenfor rekkevidde og dermed tilsvarer 0, som himmelen på bildet under, blitt tildelt en unik farge for å gjøre kontrastene enda tydeligere.



Figur 22: Bilde med pålagt fargekart

En annen tilnærming som ble brukt var å prøve å legge to videostrømmer over hverandre med bruk av `addWeighted` funksjonen fra OpenCV. Funksjonen tar inn de to ønskede videostrømmene i parametrene `src1` og `src2`. Videre settes `alpha` og `beta` som bestemmer vekten av hvert sitt bilde. Det siste parameteret er `gamma` og er en skalar som legges på begge bildene og dermed setter lysstyrken på bildet. Ved å bruke denne funksjonen ble de forskjellige bildene lagt oppå hverandre for å prøve å skape et bilde egnet for å se tydelige kontraster.

Siste tilnærming for å skape et bilde som visualiserer omgivelsene på en god måte var å bruke canny edge detection. Dette ble gjort med å bruke `GaussianBlur` funksjonen i OpenCV biblioteket på gråskalerte bilder. Funksjonen tar inn parametrene som vist i kodesnutt 4 der `src` er gråtonebildet som blir filtrert. Videre brukes `ksize` til å angi størrelsen på det gaussiske kernelen som vil bestemme hvor mye nabopikselen vil påvirke beregningen av en piksel i bildet.

Til slutt settes graden av blurring på bildet med `sigmaX`. Dette vil glatte ut bilde som vil føre til mindre skarpe overganger, funksjonen kan sees i kodesnutt 4. Videre i koden brukes `canny` funksjonen, også fra OpenCV som tar inn det tidligere filtrerte bildet og setter øvre og nedre grense for gradientverdiene i bildet.

```
cv2.GaussianBlur(scr, ksize, sigmaX)
```

Kodesnutt 4: Gaussian blur

Resultatet fra dette er vist i figur 23 under som er tatt ved arbeidspulten. Her kan man se tydelige konturer av objektene i bildet.

Figur 23: Canny Edge Detection

4.3.4 Visualisering med Ouster-SDK

Ouster har et eget Python-SDK som skal hjelpe utviklere å ta i bruk deres lidar. Dette SDK'et kan installeres ved hjelp av en package manager i terminalen, for eksempel PIP som er standard for Python. I kodesnutt 5 vises hvordan man kan laste ned Ouster-SDK ved hjelp av PIP.

```
pip install ouster-sdk
```

Kodesnutt 5: PIP install ouster-sdk

For å visualisere lidardata i Python trenger man data fra enten en live sensor eller et opptak av lidardata. Når man har dette kan man generere et `LidarScan` objekt ved hjelp av funksjonen i kodesnutt 6 fra Ouster lidar Python SDK.

```
scan = LidarScan(h, w, info.format.udp_profile_lidar)
```

Kodesnutt 6: Oppretting av LidarScan

Funksjonen vist i kodesnutt 6 organiserer lidarpakker til punkter i polarkoordinater. Deretter bør man konvertere dataene fra polarkoordinater til kartesiske koordinater, noe som vil forenkle visualiseringsprosessen. Hvis dataene er staggered, noe som betyr at hver kolonne representerer målinger tatt ved forskjellige tidsstempel, kan du måtte destaggere dem for å ordne kolonnene etter asimutvinkel for et mer naturlig 2D-bilde. Dette kan gjøres ved hjelp av funksjonene i kodesnutt 7.

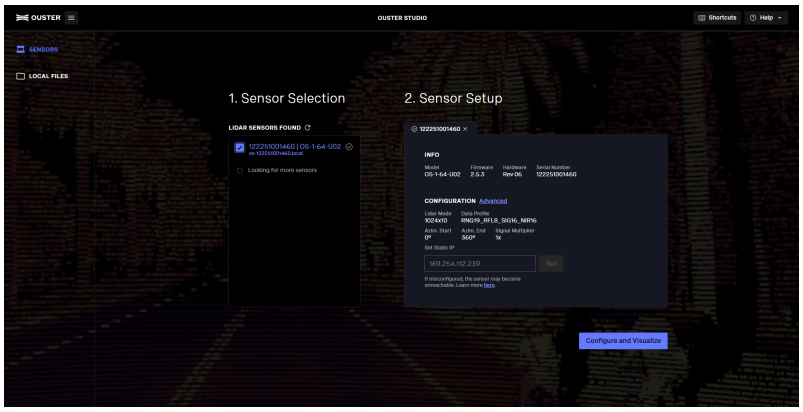
```
ref = scan.field(client.ChanField.REFLECTIVITY)
ref_destaggered = client.destagger(source.metadata, ref)
```

Kodesnutt 7: Data destaggering

`REFLECTIVITY` argumentet i funksjonen over vil vise refleksjonen av overflaten som ble detektert [5]. I tabell 6 kan man se dataen man kan hente fra lidaren, disse kan man også bruke som argumenter for å vise annen data enn refleksjon i funksjonen i kodesnutt 7. Etter behandling av dataene, kan man bruke et bibliotek som Matplotlib eller OpenCV for å plotte lidardataen som et 2D-bilde.

4.3.5 Visualisering med Ouster Studio

I startfasen av prosjektet ble Ouster Studio konfigurert for å skape en 3D-visualisering av sanntids-Lidardata. Programvaren identifiserer automatisk tilkoblede sensorer, som deretter vises i en liste, referert til i figur 24. Gjennom programvarens dashboard, som er tilkoblet Ouster Dashboard beskrevet tidligere i seksjon 4.3.1, konfigureres sensoroppsettet enkelt. Kommunikasjonen med sensoren muliggjør sanntidsvisualisering av omgivelsene i både 2D-bildeformat og som en 3D-punktsky. Videre er en rekke funksjoner tilgjengelige for å forbedre visualiseringen, inkludert justering av punkttetthet og fargekoding basert på variabler som avstand og intensitet. Disse funksjonene ble utforsket aktivt under prosjektet for å optimalisere dataframstillingen. Ouster Studio tilbyr også funksjonalitet for å visualisere tidligere opptak ved å laste opp PCAP- og JSON-filer, med muligheter for å justere avspillingshastigheten.



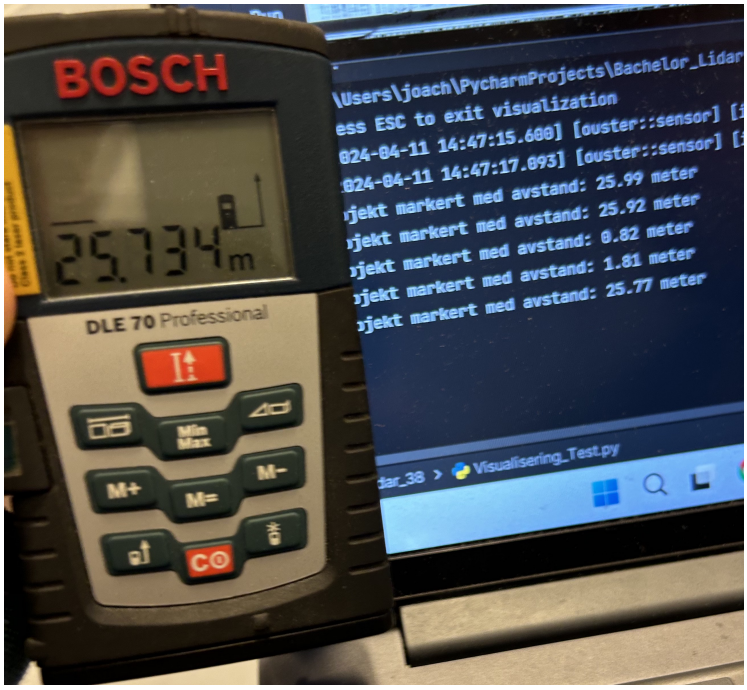
Figur 24: Ouster Studio dashboard

Med å ta i bruk programvaren fikk vi på en enkel måte mulighet til å bli kjent med dataen som skal jobbes med. Ouster studio ble også brukt til å bli kjent med de tekniske detaljene om lidaren. Alle visualiseringsmetodene klarte å visualisere på en minimumavstand rundt 33 cm noe som stemmer godt overens med databladet. På bakgrunn av dette ble det testet hvordan lidaren håndterer avstander med se på hvordan objekter som var utenfor den oppgitte avstandsgrensen på 0.3 til 120 meter ble observert. Her så vi at objekter under denne grensen vil skape et svart felt på bildet. Hvis objektet er over maks avstand for lidaren vil ikke objektet bli truffet og dermed ikke bli observert. Avstandsberegningen gjort med lidaren ble også verifisert ved å bruke en ekstern lasermåler. Resultatene fra begge målingene var så like som man kan forvente med en slik test, resultatet vises i figur 25 og beskrives i seksjon 4.3.6. Til slutt ble Field of view (FOV) til lidaren verifisert med å sette et objekt på en avstand slik at øverste delen av objektet ikke vises i bildet. Deretter ble pytagoras brukt til å beregne vinkelen til øverste delen av objektet. Resultatet vi fikk var 21.8 grader opp og ned, noe som vil til si en totalt field of view (FOV) på 43.6. I følge databladet er FOV 45 grader, feilmarginen skyldes sannsynligvis unøyaktigheter i våre målinger.. Feilmarginene på både FOV og minimumsavstanden er så små at vi regner dem som ubetydelig.

All den tekniske dataen som ble verifisert er som nevnt oppgitt i databladet og ikke nødvendig å beregne selv. Bakgrunnen for forsøkene var derfor mer rettet mot å skape en forståelse for hvordan disse tekniske begrensningene til lidaren vil påvirke faktisk bruk av lidaren enn å bekrefte at den oppgitte dataen faktisk stemmer.

4.3.6 Avstandsberegning til objekter

For å teste nøyaktigheten til lidaren, ble målingene gjort med lidaren sammenlignet med en Bosch lasermåler på en avstand på rundt 25 meter. Som man ser på bilde 25 er det minimale forskjeller i avstanden mellom de to enhetene, Bosch lasermåleren viser 25.734 meter og lidaren beregnet en avstand på 25.77 meter. På bakgrunn av dette og dokumentasjonen fra Ouster [37], tilsier at det er lite avvik i avstandsmålingene.



Figur 25: Sammenlignet med lasermåler

Ved bruk av den trente modellen vil de observerte objektene i bildet bli markert med bokser. Boksene som blir lagt på bildet vil bli representert med x og y koordinater som kan hentes ut som en liste som vist i kodesnutten under. Denne listen vil bestå av fire tall `[x1, y1, x2, y2]` som blir mappet som vist i kodesnutt 8. På denne måten vil punktene `(x1, y1)` representerer koordinatene for det ene hjørnet og `(x2,y2)` representerer koordinatene for det diagonale hjørnet. Når disse koordinatene er kjent kan avstanden innenfor markeringen beregnes.

```
for det in detections.xyxy:
    x1, y1, x2, y2 = map(int, det)
```

Kodesnutt 8: Kalkulering av størrelse på deteksjonsbokser

I løpet av arbeidet med å beregne avstanden til de observerte objektene, ble det testet flere tilnærminger for å finne metoden som ga mest nøyaktige resultat. En metode var å beregne gjennomsnittet av alle punktene innenfor markeringen. For at denne metoden skal fungere ble punktene som er utenfor sensorens rekkevidde og dermed vil gi 0 sett bort fra. Dette ga et godt estimat av avstanden men er unødvendig komplisert når de fleste punktene innenfor markeringen vil være nullpunkter.

En annen tilnærming var å finne midtpunktet av markeringen ved å dele koordinatene som representerer hjørnene på 2 for å finne midtpunktet, og dermed beregne avstanden til dette punktet. Markeringen av objekter skjer ved at objektet plasseres i sentrum av firkanten, noe som betyr at det midterste punktet av markeringen, med høy grad av sikkerhet, vil være det objektet man ønsker å beregne avstanden til. Ulempen med denne tilnærmingen er at den baserer seg bare på et punkt noe som gjør tilnærmingen sårbar for feilmålinger. For å unngå dette kan det tas gjennomsnittet av punktene innenfor en viss avstand fra midtpunktet. Også her må eventuelle nullpunkt ses bort fra, men dette ga en mer nøyaktig beregning av avstanden til det observerte punktet.

For å være i stand til å vise den beregnede avstanden til objektet i sanntidsbildet brukes `putText()` funksjonen fra OpenCV-biblioteket, som vist i kodesnutt 9. På denne måten kan sanntidsavstand til objektene bli lagt direkte på bildet sammen med alarm tekst for å varsle når objektene krysser alarmgrensen.

```
cv2.putText(Image, Text, Org, Font, FontScal, Color, Thickness)
```

Kodesnutt 9: putText funksjonen

Funksjonen `putText()` i OpenCV er ansvarlig for å legge til tekst på et bilde, og tar inn flere parametere som styrer hvordan teksten vises på bildet. Disse parametrene inkluderer `Image`, som representerer bildestrømmen, og `Text`, som er teksten som legges på bildet. Koordinatene til teksten på bildet, angitt med x- og y-verdier, bestemmes av parametre `Org`. Deretter kan skrifttypen spesifiseres med `Font` parametre før størrelsen på teksten justeres med `FontScale`. Til slutt kan fargen og tykkelsen på teksten endres ved hjelp av `Color` og `Thickness`-parametrene, henholdsvis. Fullstendig oversikt over alle argumentene finnes her [25].

4.4 IMU

IMU'en innebygget i lidaren består av et 3-akse gyroskop og akselerometer som vil gjøre den i stand til å beregne *pitch* og *roll* til Otteren.

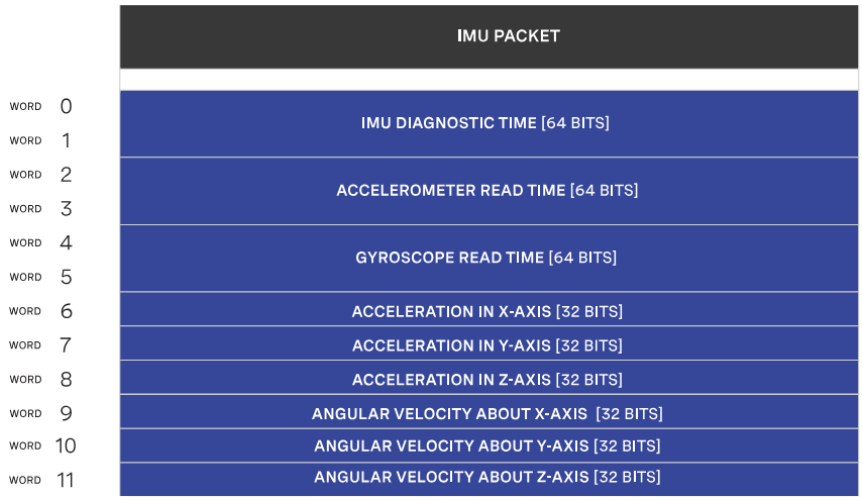
4.4.1 Kommunikasjon og behandling av IMU-data

For å samle inn data fra IMU'en benyttes UDP som kommunikasjonsprotokoll med UDP-port 7503 som er standard-port for overføring av IMU-data. Som vist i kodesnutt 10 blir denne informasjonen brukt til å opprette en UDP-socket for å sikre kontinuerlig mottak av IMU-data.

```
ip = ""  # IP address of the PC that will receive the IMU data
port = 7503  # Port number for the IMU data, default is 7503

udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  #
    Create a UDP socket
udp_socket.bind((ip, port))  # Bind the socket to the IP address
    and port
```

Kodesnutt 10: Oppretting av UDP-socket

UDP-pakkene med IMU-dataen mottas med formatet som vist i figur 26. Her representerer *word* dataenheten i pakken sammen med bits som angir størrelsen på hvert dataelement. *word* 0 og 1 blir brukt til diagnosering av IMU'en og blir dermed ikke brukt under vanlig drift. Videre blir *word* 2-3 og 4-5 representert med 64 bits og gir informasjon om tidsstemplingen for når akselerometer- og gyroskopdata blir lest. Denne tidsstemplingen er den samme som blir brukt for lidardata og er representert med 64 bits for å sikre nøyaktighet. De resterende *word* enhetene blir representert med 32 bits og gir informasjon om akselerasjon og vinkelhastighet i x,y og z-aksen [39].



Figur 26: IMU-datapakker, hentet fra [39]

Dataen som mottas fra IMU'en er rå sensoravlesninger og blir derfor presentert i little endian binær-format, som vist i figur 27 [39]. For å kunne tolke dataene ble den derfor konvertert ved bruk av formatstrengen *QQQfffff* i struct-biblioteket. Denne formatstrengen indikerer at man forventer tre 64-bits unsigned integer etterfulgt av seks 32-bits flyttall [44]. Ved å ta i bruk denne konverteringen er man i stand til å kunne bruke IMU verdiene videre.

```
Client IP Address:('169.254.112.239', 38034)
Message from Client:b'\xe4\x91\xc8\xddQ\x00\x00\x00v\xd4\xcb\xddQ\x00\x00\x00R6\xcf\xddQ\x00\x00\x00\x00\x00\x94<\x00\x00\x8e\xbc\x00\xc0\x80?\x00\x1a\xcc?\x00@\x
9c\xbe\x00p\x14\xbe'
```

Figur 27: Little endian IMU-data

Det ble testet å filtrere IMU-dataen ved å implementere et kalmanfilter som kombinerer de mottatte dataene fra IMU'en. Under implementeringen av filteret merket vi ikke store endringer i variansen på dataene. På bakgrunn av at vi ikke implementerte IMU'en videre i prosjektet samtidig som vi ikke ønsket et for komplekst system valgte vi å ikke bruke tid på videre filtrering av IMU-data.

4.4.2 Beregning av roll og pitch

Når dataene fra IMU'en er dekodet riktig tas den i bruk for å beregne roll og pitch av Otteren. Dette gjøres ved funksjonene `calculate_dt()` og `calculate_pitch_roll()` forklart under.

I funksjonen `calculate_dt()` blir tidsintervallet `dt` mellom to etterfølgende datainnsamlinger fra IMU'en beregnet. Dette er essensielt for å vite tidsstemplingen på dataene fra akselerometeret og gyroskopet. Når to tidsstemplinger er mottatt regnes differansen på tidsstemplingene ut. Dette definerer verdien til variabelen `dt` slik at den kan bli brukt i integrasjon, dette vises i kodesnutt 11.

```
def calculate_dt(current_timestamp_ns):
    global prevTime
    current_timestamp_s = current_timestamp_ns / 1e9
    dt = current_timestamp_s - prevTime
    prevTime = current_timestamp_s
    return dt
```

Kodesnutt 11: Kalkulering av tidsintervall

Videre i funksjonen `calculate_pitch_roll()` brukes tidsstemplingen sammen med data fra akselerometeret og gyroskopet for å beregne pitch og roll. Dette blir gjort ved å konvertere akselerometermålingene fra m/s til gravitasjonsenheten g ved å dele på tyngdeakselerasjonen 9.81 m/s^2. Deretter beregnes pitch-vinkelen ved å ta arctangenten av forholdet mellom akselerasjonen på y-aksen og kvadratroten av summen av kvadratene til akselerasjonene på x- og z-aksene. På samme måte beregnes roll-vinkelen ved å ta arctangenten av forholdet mellom akselerasjonen på x-aksen og kvadratroten av summen av kvadratene til akselerasjonene på y- og z-aksene. Begge disse vinklene konverteres fra radianer til grader. Til slutt integreres gyroskopets målinger av vinkelhastighet med disse initialvinklene over det beregnede tidsintervallet `dt` for å oppdatere og finjustere vinklene. Denne funksjonen kan man se i kodesnutt 12.

```
def calculate_pitch_roll(accel_x, accel_y, accel_z, gyro_x, gyro_y,
    gyro_z, dt):
    # Convert acceleration from m/s^2 to g
    accel_x_g = accel_x / 9.81
    accel_y_g = accel_y / 9.81
    accel_z_g = accel_z / 9.81

    # Calculate pitch and roll from accelerometer
    pitch_acc = math.atan2(accel_y_g, math.sqrt(accel_x_g ** 2 +
   accel_z_g ** 2)) * 180 / math.pi
    roll_acc = math.atan2(accel_x_g, math.sqrt(accel_y_g ** 2 +
   accel_z_g ** 2)) * 180 / math.pi

    # Integrate the gyroscope data to get the pitch and roll
    pitch = pitch_acc + gyro_x * dt
    roll = roll_acc + gyro_y * dt

    return pitch, roll
```

Kodesnutt 12: Kalkulering av pitch og roll

Den beregnede IMU-dataen blir skrevet ut som vist på bildet 28 under. Her starter enheten i en vannrett posisjon før den forflyttes i en jevn bevegelse opp til 90°, og tilbake til startposisjonen. Utklippet fra dataen under ser man i første kolonne tidsstemplingen og deretter pitch-vinkelen fra 18 - 23 grader. Roll dataen ligger rundt null når det ikke var noen endringer i denne vinkelen.

```
757.82804594,-23.393647032985324,0.3943928296922848
757.838046095,-22.922544234562075,-0.380977329116322
757.84804608,-22.27740636911637,-0.33726207208549697
757.858046125,-20.71781427717731,-0.17988868921389245
757.86804594,-20.319182548954043,-0.06882123390175385
757.878046105,-21.275744429822705,-0.06154648826060705
757.88804597,-18.985076984047215,0.005798261566193619
```

Figur 28: Timestamp, pitch og roll verdier fra IMU

4.5 Objektdeteksjon

4.5.1 Oppretting av treningssett

For å kunne få YOLOv8-algoritmen til å detektere objekter i et lidarbilde må den trenes på nettopp dette. I vårt tilfelle ønsker vi å gjenkjenne båter, så lidaren ble derfor tatt med ut for å filme båter. Under dette arbeidet merket vi at det ikke var optimalt å filme i båthavner der båter sto tett, siden dette ikke ga en tydelig struktur for gjenkjenning av objektene. Det ble derfor fokusert mer på å filme frittgående båter for å kunne trene modellen best mulig. Opptakene som ble tatt ble alle filmet ved bruk av range metoden men med forskjellige fremgangsmåter. De tre metodene var vanlig decoding av range, logaritmisk skalering og fargekart. Dette ble gjort for å finne ut hvilke bilde som er lettest å jobbe med samt trene lidaren i forskjellige miljøer.

I kodesnutt 13 vises det hvordan det klargjøres til opptak av lidarbilder. Først hentes tidsstempelet i formatet *YYYY-MM-DD_HH:MM*, som deretter lagres i `timestamp`-variabelen. Videre blir filnavnene og lagringssted definert, filene blir lagret i *savedVideosAndPictures*-mappen og får de respektive navnene sine i tillegg til tidsstempelet når opptaket begynte. `fourcc`-variabelen definerer video-codec til filene, dette brukes for å komprimere og dekomprimere data, noe som reduserer størrelsen på videofilene. Til slutt opprettes det `VideoWriter`-objekter til begge filene, dette inneholder filnavn, video-codec, bildefrekvens og bildeformat. I seksjon 4.5.5 og figur 34 vises det at vi klarer å kjøre objektdeteksjon på mellom 9 til 12 bilder i sekundet. Derfor har vi satt bildefrekvensen i denne koden til 20 bilder i sekundet. Dette gir oss en tilstrekkelig høy frekvens, samtidig som at filstørrelsene ikke blir unødvendig store.

```
# Get the current date and time, format is YYYY-MM-DD_HH:MM
timestamp = datetime.datetime.now().strftime("%Y-%m-%d_%H:%M")

# Filenames, saved in the savedVideosAndPictures folder
logVideo_filename = 'savedVideosAndPictures/logVideo_' + timestamp
    + '.mp4'
rangeVideo_filename = 'savedVideosAndPictures/rangeVideo_' +
    timestamp + '.mp4'

# Set up video recording
fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Codec
logVideo = cv2.VideoWriter(logVideo_filename, fourcc, 20.0, (1024,
    64))  # logarithmic scaling video
rangeVideo = cv2.VideoWriter(rangeVideo_filename, fourcc, 20.0,
    (1024, 64))  # range values video
```

Kodesnutt 13: Klargjøring for opptak av lidarbilde

Videre tas `write`-funksjonen, også fra OpenCV, i bruk for å skrive videoen til den opprettede videostrømmen. Dette gjøres med å ta inn bilde som et parameter i funksjonen og skrive dette til videofilen. Et eksempel på dette vises i kodesnutt 14.

```
# Write the frame to video
rangeVideo.write(range_img_)
logVideo.write(range_img_bw_)
```

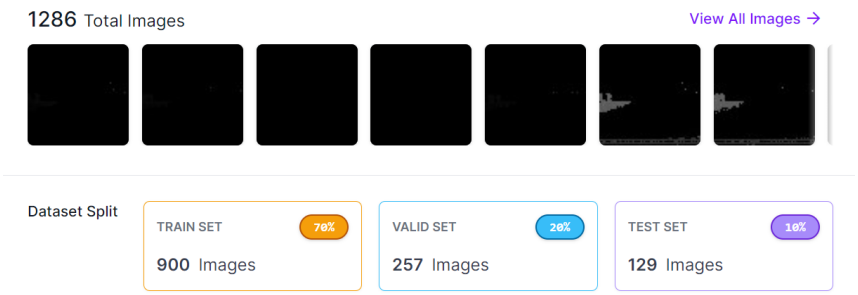
Kodesnutt 14: Skriv bilde til videofil

Etter man har fått filmet det man ønsker tas `release()`-funksjonen i bruk for å avslutte opptaker og frigjøre ressursene. Til slutt for å forsikre at ingen OpenCV vinduer forblir åpen etter programmet er avsluttet benyttes `cv2.destroyAllWindows()`-funksjonen. På denne måten blir alle vinduene lukket og man sikrer at programmet avsluttes på riktig måte, dette ser man i kodesnutt 15.

```
# Release the video writer and close all windows
rangeVideo.release()
logVideo.release()
cv2.destroyAllWindows()
```

Kodesnutt 15: Avslutt opptak

Når opptakene er fullført lastes det opp til Roboflow, og angis frekvensen for sampling av videoene, det vil si intervallet for uttak av enkeltbilder fra opptaket. Opptakene som ble tatt var fra et fast ståsted der omgivelsene ikke endres stort og det ble derfor i vårt tilfelle ikke nødvendig å sample fra videoene så ofte. Etter sampelfrekvensen er satt sitter man igjen med en samling av bilder som må markeres. Roboflow har funksjonen *autolabeling* som skal markere objektene i bilde automatisk men denne var ikke optimal å bruke på lidarbilder. Grunnet dette ble denne prosessen gjort manuelt der man gikk gjennom bildene og markerte objektene man ønsket å gjenkjenne. Når dette er gjort deles datasettet inn i tre deler: trening, validering og test. Som vist på bildet 29 under består datasettet vårt av rundt 1300 bilder hvor 700 av disse er lidarbilder med en fordeling på 70% til trening, 20% til validering og 10% til test.



Figur 29: Trening, validering og testsett

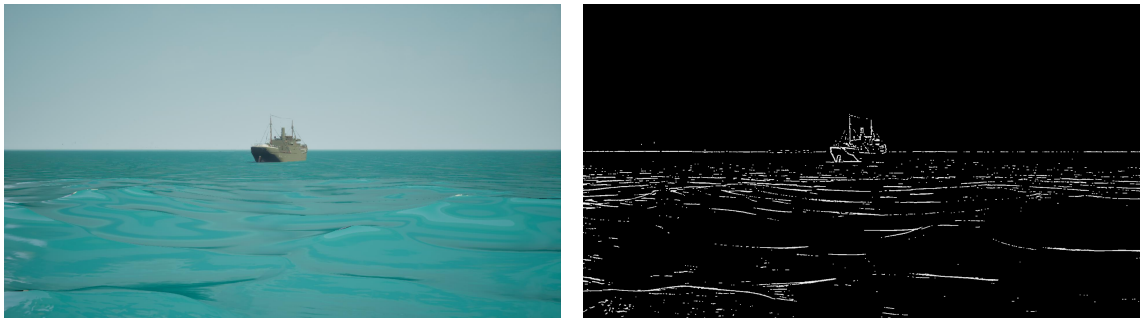
Før vi hadde anledning til å filme selv ble det forsøkt å gjøre om fargebildet til bilder som skulle etterligne lidarbilder. Dette ble gjort med et Pythonscript og funksjonen som vises i kodesnutt 16. Først konverteres det innsendte bildet til gråskala ved hjelp av `cv2.cvtColor` funksjonen for å forenkle behandlingen. Deretter blir bildet glattet ved å bruke gaussisk glatting for å redusere støy ved hjelp av `GaussianBlur` funksjonen. Et adaptivt terskelbilde blir deretter opprettet for å skille objekter fra bakgrunnen. Et sett med morfologiske operasjoner, inkludert erosjon og dilatasjon, brukes til å forbedre formen på objektene. Konturene av objektene blir deretter funnet, og et nytt bilde blir opprettet hvor disse konturene er tegnet og fylt ut. Avstanden fra hver piksel til nærmeste kontur blir beregnet, resulterende i et dybdebilde.

Dette bildet normaliseres for å få verdier mellom 0 og 1 og deretter undergår gamma-korreksjon for å forbedre kontrasten, denne gamma korreksjonen måtte finjusteres manuelt. Til slutt blir det korrigerte bildet konvertert tilbake til 8-bits gråskalabilde og returnert som en lidaretterligning.

```
def process_image(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    thresholded = cv2.adaptiveThreshold(blurred, 255, cv2.
    ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)

    kernel = np.ones((3,3),np.uint8)
    eroded = cv2.erode(thresholded, kernel, iterations=1)
    dilated = cv2.dilate(eroded, kernel, iterations=1)

    contours, _ = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)

    range_image = np.zeros_like(gray)

    cv2.drawContours(range_image, contours, -1, (255), thickness=
    cv2.FILLED)

    depth_map = cv2.distanceTransform(range_image, cv2.DIST_L2, cv2
    .DIST_MASK_PRECISE)

    depth_map_normalized = cv2.normalize(depth_map, None, 0, 1, cv2
    .NORM_MINMAX)

    gamma = 0.2  #Contrast
    gamma_corrected = np.power(depth_map_normalized, gamma)

    intensity_image = (gamma_corrected * 255).astype(np.uint8)

    return intensity_image
```

Kodesnutt 16: RGB til lidarbilde

I figur 30 kan man se to bilder, bilde 30a er det opprinnelige fargebildet og bilde 30b er lidaretterligningen. Koden i kodesnutt 16 ble kjørt på rundt 600 bilder som ble lagt inn i datasettet. Etter å ha testet modellen kun trent på disse bildene var ikke objektdeteksjonen stabil og presis nok.



(a) Fargebilde av båt (b) Lidaretterligning av båt

Figur 30: Sammenligning av fargebilde og lidaretterligning

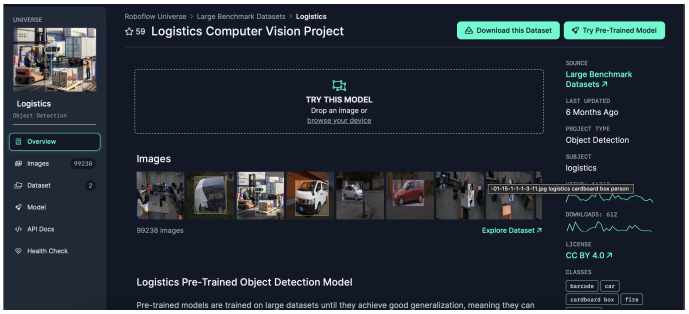
Denne prosessen kunne gjennomføres på mange bilder og legges inn i datasettet. Dette er fordi vi kan hente ferdige bilder fra Roboflow-datasett, disse bildene vil ha ferdige labels. Vi kan dermed laste ned bildene og label, gjøre bildene om til lidaretterligninger, bruke koordinatene fra de opprinnelige labelene og putte de eksisterende labelene over lidaretterligningene. Dette gjorde at vi ikke trengte å markere bildene på nytt, som hadde vært en veldig tidskrevende prosess.

4.5.2 Trening av YOLO-modell med Roboflow-datasett

Ultralytics YOLOv8-algoritmen ble benyttet til objektdeteksjon, denne algoritmen er beskrevet i seksjon 3.2.8 og metoden for objektdeteksjon er beskrevet i seksjon 4.5.5. For å gjøre algoritmen presis og nøyaktig i kombinasjon med lidar ble det benyttet et tilpasset datasett laget i Roboflow, metoden for opprettelsen av dette datasettet er beskrevet i seksjon 4.5.1. Datasettet blir hentet inn i et Pythonprosjekt, deretter blir det trent en ny YOLO-modell på det nye datasettet. Dette resulterte i en algoritme spesielt effektiv på å identifisere objekter i lidarbilder.

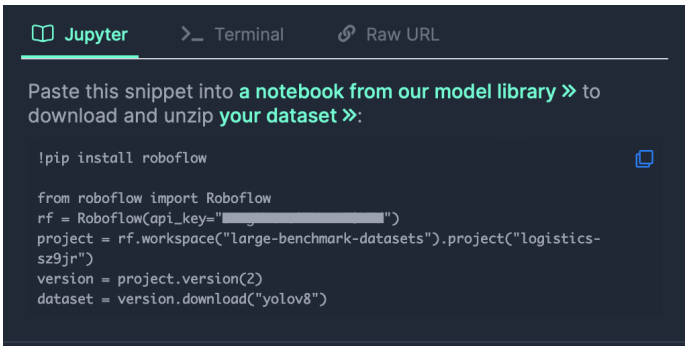
Roboflow har muligheten til å generere egne tilpassede datasett. Dette gjøres ved å laste opp valgte bilder eller videoer og deretter sette bokser rundt objektene man vil gjenkjenne. Til slutt må man si hva som er i den markerte boksen, dette blir kalt en label. Datasettet kan eksporteres tilpasset den YOLO-versjonen man selv vil bruke, datasettet fordeles da inn i training, test og validation sett, som beskrevet i seksjon 4.5.1. Disse settene brukes av YOLO-algoritmen til å trene en ny tilpasset YOLO-modell.

For å importere Roboflow datsettet må man inn på deres nettside, finne et datasett man vil laste ned, eller eventuelt lage sitt eget. Dermed må man trykke på *download this dataset* knappen, som man ser oppe i høyre hjørne i figur 31.



Figur 31: Roboflow-datasett

Etter dette må man velge hvilken YOLO-versjon man skal bruke, og trykke *continue*. Da blir man møtt med vinduet som man ser i figur 32. Ved å kopiere denne kodesnutten og lime det inn i en Python-celle kan man laste ned datasettet med konfigurasjonsfilen direkte inn i prosjektet man jobber i.



Figur 32: Roboflow download code

Python-cellen skal se ut som i kodesnutt 17. Datasettet blir lagret i prosjektet med navet `NAVN PÅ DATASETT - VERSJON`, så ved å bruke placeholderene vi har brukt i kodesnutten under vil navnet være `PROJECT-1`.

```
from roboflow import Roboflow

rf = Roboflow(api_key='YOUR_API_KEY')
project = rf.workspace('WORKSPACE').project('PROJECT')
version = project.version(1)
dataset = version.download('YOLO_VERSION')
```

Kodesnutt 17: Importer datasett fra Roboflow

For å trene en YOLO-modell på datasettet kan man bruke `train`-funksjonen fra ultralytics-biblioteket. I `YOLO()`-funksjonen i kodesnutt 18 laster man inn en ferdigtrent YOLO-modell, det anbefales av Ultralytics å bruke en av deres ferdigtrente modeller. I kodesnutten bruker man `train`-funksjonen, her må `data`-argumentet være banenavnet til konfigurasjonsfilen til datasettet man lastet ned, dette skal være en `.yaml` fil.

```
from ultralytics import YOLO

model = YOLO("PATH_TO_PRETRAINED_MODEL")

model.train(data="PATH_TO_CUSTOM_YAML_FILE",
            epochs="NUMBER_OF_EPOCHS",
            imgsz="IMAGE_SIZE",
            project="PATH_TO_SAVE_DIRECTORY")
```

Kodesnutt 18: Trening av YOLO-modell

I tabell 7 vises alle argumentene brukt i `train`-funksjonen ovenfor. Fullstendig oversikt over alle argumentene finnes her [3].

Argument	**Default**	**Description**
model	None	Specifies the model file for training. Accepts a path to either a .pt pretrained model or a .yaml configuration file. Essential for defining the model structure or initializing weights.
data	None	Path to the dataset configuration file (e.g., coco8.yaml). This file contains dataset-specific parameters, including paths to training and validation data, class names, and number of classes.
epochs	100	Total number of training epochs. Each epoch represents a full pass over the entire dataset. Adjusting this value can affect training duration and model performance.
imgsz	640	Target image size for training. All images are resized to this dimension before being fed into the model. Affects model accuracy and computational complexity.
project	None	Name of the project directory where training outputs are saved. Allows for organized storage of different experiments.

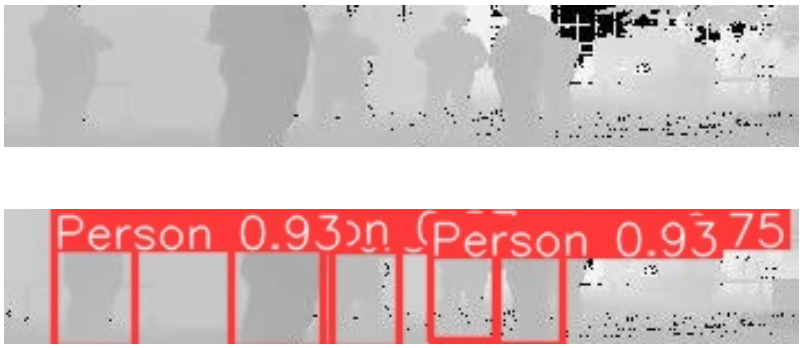
Tabell 7: Argumenter brukt i `train`-funksjon, basert på [3]

I vårt tilpassede datasett tok en epoch rundt 10 minutter å kjøre på vår lokale PC. For å få kontroll over algoritmen kjørte vi 10 epochs av gangen. Etter 10 epochs testet vi modellen på ulike videoer og bilder for å se om resultatet var bra. Hvis vi ikke var helt fornøyde, ble den siste modellen vi trente brukt som utgangspunkt for å trene den neste modellen. Dette ga kontroll under trening og motvirket sjansen for overfitting, som er beskrevet i seksjon 2.5. Når modellen presterte dårlig på ny data, men perfekt på treningsdata kunne vi gå tilbake til en tidligere modell som presterte bedre. En slik tilnærming blir ofte kalt *early-stopping*, som også er beskrevet i seksjon 2.5.

4.5.3 Testdatasett og test-YOLO-modell

For enkelhetsskyld ble det opprettet et datasett og en YOLO-modell for objektdeteksjon av mennesker. Det ble fulgt samme metode som beskrevet i seksjon 4.5.1 og 4.5.2. Det var betraktelig enklere å samle inn lidarbilder av mennesker enn av båter, derfor ble dette forsøkt før det ble brukt tid på å opprette den endelige modellen.

I figur 33 ser vi øverst det originale bildet som ble tatt med lidaren, og nederst ser vi bildet etter objektdeteksjon. Dette gir et tydelig resultat på at YOLO-modellen ble en suksess, og det ble klart for oss at det var mulig å trene YOLO-modeller på lidarbilder.



Figur 33: Objektdeteksjon av mennesker

4.5.4 OpenCV

OpenCV står for Open Source Computer Vision. Det er et åpen kildekodebibliotek som er spesialisert på bildebehandling og maskinsyn, biblioteket er beskrevet i seksjon 3.2.7.

For objektdeteksjon spiller OpenCV en avgjørende rolle som et kraftig verktøy. Biblioteket gir omfattende muligheter for bildebehandling. Funksjonen `findContours` er viktig for å kunne identifisere objekter i et bilde ved å finne og representere konturer. Denne funksjonen tar et bilde som input og konverterer det til et binærbilde, hvor hver piksel blir enten 1 eller 0 basert på en terskelverdi. Deretter finner funksjonen konturene i det binære bildet og lagrer dem som en vektor av punkter [28]. Dette gjør det mulig å isolere og analysere objekter i bildet, og danner grunnlaget for videre behandling og identifikasjon i objektdeteksjonsalgoritmer.

OpenCV tilbyr også funksjoner for å vise bilder i et GUI-vindu, noe som er nyttig for å visualisere resultatene av objektdeteksjonen. En slik funksjon er `imshow`, som gir muligheten til å åpne et vindu og vise et bilde i det.

```
cv.imshow(winname, mat)
```

Kodesnutt 19: OpenCV visualisering av bilder

Funksjonen er vist i kodesnutt 19 og tar to argumenter, `winname` er navnet på vinduet som skal åpnes, og `mat`, som er bildet som skal vises i vinduet [26]. Denne funksjonen gjør det enkelt å inspisere resultatene av objektdeteksjonen i sanntid eller i etterkant ved å vise detekterte objekter eller annen relevant informasjon visuelt.

Når man tester en objektdeteksjonsalgoritme, er det viktig å kunne håndtere både statiske bilder og dynamiske videoer. For å arbeide med videoer, brukes funksjonen `VideoCapture`. Denne funksjonen kan åpne videofiler, bilde-sekvenser, URL-lenker til videostrømmer, eller til og med ta imot live videostrømmer. Dette gjør det enkelt å teste objektdeteksjonen på en rekke forskjellige videokilder og scenarioer [24]. Når det gjelder statiske bilder, kommer `imread` funksjonen til nytte. Denne funksjonen brukes til å åpne en enkelt bildefil [27]. Ved å bruke `imread` kan du enkelt laste inn enkeltbilder for testing av algoritmen. Ved å kunne bruke både `VideoCapture` og `imread` i OpenCV, har man muligheten til å teste og validere algoritmen på både statiske og dynamiske scener, og sikre at den fungerer som forventet i ulike situasjoner.

I dette prosjektet ble OpenCV brukt til å justere bilder og åpne bildevinduer for visualisering, samt å teste ulike objektdeteksjonsalgoritmer og modeller. Under testing av avstandsberegninger ble `findContours` funksjonen fra OpenCV brukt før implementeringen av en pålitelig YOLO-modell var på plass. Når YOLO-modellen var ferdig ble den testet med videoer og bilder som ble åpnet med OpenCV, noe som muliggjorde enkel testing av modellen uten behov for live-data.

4.5.5 Objektdeteksjon med YOLOv8

For å detektere objekter ved hjelp av YOLOv8 må man først laste inn en ferdigtrent YOLO-modell. Dette gjøres ved hjelp av kodesnutt 20. Hvis man ikke har en ferdigtrent modell og ønsker å trene sin egen, må man følge stegene beskrevet i seksjon 4.5.2.

```
# Load the model, chose one of the models from the models folder
model = YOLO("models/")
```

Kodesnutt 20: Last inn YOLO-model

Objektdeteksjon med YOLOv8 kan hovedsakelig utføres på to måter, ved hjelp av `track` og `model`-funksjonene. Forskjellen ligger i at `track`-funksjonen automatisk tegner etiketter og bokser rundt deteksjonene. I tillegg kan man bruke argumentet `persist=True` for å opprettholde deteksjoner over flere bilder, med andre ord *tracking* av objekter. Med `model`-funksjonen må man hente koordinater ved hjelp av `supervision`-biblioteket og deretter tegne label og bokser. I kodesnutt 21 vises `model`-funksjonen, mens i kodesnutt 22 ser vi `track`-funksjonen. I tabell 8 vises alle relevante argumenter som kan brukes i `track`- og `model`-funksjonene.

```
# Perform object detection on the image
result = model(source=frame, conf=0.25, iou=0.5)[0]
```

Kodesnutt 21: Objektdeteksjon med `model` funksjon

```
# Perform object detection on the image
result = model.track(source=frame, persist=True, conf=0.25, augment
    =True, iou=0.5, show=True)
```

Kodesnutt 22: Objektdeteksjon med `track` funksjon

Argument	Default	Description
source	'ultralytics/assets'	Specifies the data source for inference. Can be an image path, video file, directory, URL, or device ID for live feeds. Supports a wide range of formats and sources, enabling flexible application across different types of input.
conf	0.25	Sets the minimum confidence threshold for detections. Objects detected with confidence below this threshold will be disregarded. Adjusting this value can help reduce false positives.
iou	0.7	Intersection Over Union (IoU) threshold for Non-Maximum Suppression (NMS). Lower values result in fewer detections by eliminating overlapping boxes, useful for reducing duplicates.
imgsz	640	Defines the image size for inference. Can be a single integer 640 for square resizing or a (height, width) tuple. Proper sizing can improve detection accuracy and processing speed.
stream_buffer	False	Determines if all frames should be buffered when processing video streams (True), or if the model should return the most recent frame (False). Useful for real-time applications.
augment	False	Enables test-time augmentation (TTA) for predictions, potentially improving detection robustness at the cost of inference speed.
show	False	If True, displays the annotated images or videos in a window. Useful for immediate visual feedback during development or testing.
persist	False	Enables persistent object tracking across frames. Objects are tracked over time, and their identities are maintained.

Tabell 8: Argumenter brukt i `tack` og `model`-funksjonene, basert på [10]

For å tegne label og bokser ved hjelp av `modell`, benytter vi `supervision`-biblioteket. I kodesnutt 23, ser vi konfigurasjonene for boksene som er gitt til `supervision`.

```
box_annotator = sv.BoxAnnotator(  # Create a box annotator
    thickness=1,  # Set the thickness of the box
    text_thickness=1,  # Set the thickness of the text
    text_scale=1  # Set the scale of the text
)
```

Kodesnutt 23: supervision box annotator

I kodesnutt 24, observerer vi bruken av `supervision` for å hente koordinater og konfigurasjon av bokser.

```
detections = sv.Detections.from_ultralytics(result)  # Convert the
    detections to Supervision format
xyxy_array = detections.xyxy  # Get the coordinates of the
    detections
bounding_box_annotator = sv.BoundingBoxAnnotator()  # Create a
    bounding box annotator
label_annotator = sv.LabelAnnotator()  # Create a label annotator
```

Kodesnutt 24: Supervision henter deteksjoner og boks configurasjon

Til slutt kan vi bruke disse koordinatene til å tegne boksene og legge til label. Dette gjøres over det opprinnelige bildet som ble brukt av YOLO-algoritmen til objektdeteksjon. Implementasjonen av dette vises i kodesnutt 25.

```
# Annotate the frame with bounding boxes and labels
bounding_box_annotator.annotate(scene=frame, detections=detections)
label_annotator.annotate(scene=frame, detections=detections, labels
    =labels)
```

Kodesnutt 25: Supervision tegner label og boks

Når alt dette settes sammen, får vi en modell som gjør det mulig å detektere båter i lidarbilder og videoer. For å visualisere bildet kan man bruke `frame`, for eksempel ved hjelp av `openCV`. Programmet vil gå gjennom videoen bilde for bilde, noe som kan føre til at noen bilder må hoppes over for at videoen skal kunne kjøre i normal hastighet. I figur 34 ser vi hastigheten for fire bilder, der algoritmens tid for å detektere objekter varierer fra 80,1 ms til 104,4 ms. Med andre ord kan vi si at algoritmen klarer å prosessere mellom 9 og 12 bilder i sekundet.

```
0: 64x640 2 boats, 86.0ms
Speed: 1.1ms preprocess, 86.0ms inference, 0.6ms postprocess per image at shape (1, 3, 64, 640)

0: 64x640 3 boats, 81.8ms
Speed: 1.1ms preprocess, 81.8ms inference, 0.5ms postprocess per image at shape (1, 3, 64, 640)

0: 64x640 3 boats, 104.4ms
Speed: 1.2ms preprocess, 104.4ms inference, 0.6ms postprocess per image at shape (1, 3, 64, 640)

0: 64x640 2 boats, 80.1ms
Speed: 1.1ms preprocess, 80.1ms inference, 0.5ms postprocess per image at shape (1, 3, 64, 640)
```

Figur 34: Objektdeteksjon hastighet

I figur 35 kan vi se hvordan det ser ut når lidaren har produsert et bilde. I tillegg kan vi se hvordan det ser ut når algoritmen har gjot deteksjoner og tegnet label og bokser rundt deteksjonene.

Figur 35: Lidarbildet med 3 båtdeteksjoner

4.5.6 Pythonprosjekt

Til dette prosjektet har alt blitt kodet i Python. I den forbindelse har det blitt lagd et Pythonprosjekt som inneholder alt man trenger for å gjennomføre objektdeteksjon med og uten lidar. I tillegg er det vedlagt de ferdige modellene vi har trent på vår egen data. Videre er det vedlagt Python-script som gjør at man kan trene egne modeller eller videreutvikle våre eksisterende modeller.

Prosjektet ligger som vedlegg A til denne rapporten. Gjennom denne seksjonen skal vi gjennomgå alle funksjoner Pythonprosjektet vårt tilbyr, samt hvordan man tar disse i bruk.

Pythonprosjektet består av en `README` fil som dokumetasjon, 10 Python-script, 1 mappe med YOLOv8 modeller, 1 mappe med videoer og bilder, 1 mappe hvor alle opptak lagres, og i tillegg en `requirements.txt` fil som inneholder alle biblioteker som trengs. Mappen med bilder og videoer heter `lidarExamples` og inneholder en mappe med lidarbilder og en mappe med lidarvideoer. Mappen med modeller heter `models` og inneholder `Ouster_Lidar_Boat_Weights_V4` og `V5` som er våre egne modeller. I tillegg inneholder mappen alle standardmodeller som YOLOv8 inkluderer. Prosjektet har blitt testet i et virituelt Python 3.8 miljø, dette er anbefalt for senere bruk.

Python-scriptene i prosjektet heter:

- main.py
- pictureDetection.py
- videoDetection.py
- videoDetectionSave.py
- videoTrack.py
- videoTrackSave.py
- recordWithLidar.py
- IMUdataRead.py
- IMUdataReadWrite.py
- modelTraining.py

I avsnittene under ser man hva de forskjellige scriptene gjør, og hvordan man tar de i bruk.

main.py

Dette scriptet inneholder kode som henter live lidardata og bruker YOLOv8 til å detektere objekter. Scriptet har en `ALARM_DISTANCE`-variabel som man kan bruke til å sette avstanden til alarmen, om noen objekter er innenfor denne grensen vil det gå av en alarm.

I dette scriptet må man laste inn en YOLO-modell fra `models`-mappen. I tillegg må man konfigurere `hostname` og `lidar_port`-variablene. Samtidig som man detekterer objekter blir det lagret 2 videoer. Den ene videoen heter `output_video` å inneholder det samme som det som blir visualisert, den andre videoen heter `videostream` å inneholder en 2D-lidarvideo uten deteksjoner. Disse blir lagret i `savedVideosAndPictures` med dato og tidspunktet opptaket begynte.

pictureDetection.py

For objektdeteksjon i bilder kan man bruke dette scriptet. Alt man trenger for dette er å laste inn en modell fra `models`-mappen og laste inn et bilde.

videoDetection.py og videoDetectionSave.py

Objektdeteksjon i videoer har 2 forskjellige script `videoDetection.py` er for kun objektdeteksjon og `videoDetectionSave.py` tar i tillegg opptak av deteksjonene. I disse scriptene må man laste inn en modell fra `models`-mappen, i tillegg må man laste inn en video eller koble seg til et videokamera. Etter objektdeteksjonen er påbegynt kan man trykke `ESC` for å avslutte. Når det er avsluttet blir videoene lagret i `savedVideosAndPictures`-mappen.

videoTrack.py og videoTrackSave.py

`videoTrack.py` og `videoTrackSave.py`-scriptene fungerer på samme måte utenom at `videoTrackSave.py` tar videoopptak i tillegg. Her må man også laste inn en modell fra `models`-mappen, i tillegg til at man må laste inn en video eller koble seg til et videokamera. Samme som tideligere funksjoner kan man trykke `ESC` for å avslutte, og opptak lagres i `savedVideosAndPictures`-mappen. Disse to scriptene skiller seg ut ved å bruke en `track`-funksjon til å detektere objekter, mens andre script bruker `model`-funksjonen.

recordWithLidar.py

Vil man ha opptak av lidarbilder kan man bruke dette scriptet. Likt som i `main.py` må man fortsatt koble til lidaren, dette gjøres ved å endre `hostname` og `lidar_port` variablene. I dette scriptet foregår det ingen objektdeteksjon, så det eneste andre man må endre er `duration`-variabelen, denne bestemmer hvor lenge scriptet skal filme. Likt som før lagres alle videoene i `savedVideosAndPictures`-mappen.

IMUdataRead.py og IMUdataReadWrite.py

`IMUdataRead.py`-scriptet gjør det mulig å lese IMU-data fra lidarens innebygde IMU. For å gjøre dette må man opprette en UDP-socket som IMU'en kan sende til. Dette gjør man med å endre `ip` og `port`-variablene, `ip` skal være IP-adressen til enheten som skal motta IMU-data, `port` skal være UDP-porten som dataen blir sendt over. `IMUdataReadWrite.py` gjør alt det samme som `IMUdataRead.py` den eneste forskjellen er alt all dataen blir skrevet til en cvs-fil som lagres med filnavnet `IMU_data_YYYY-MM-DD_HH:MM.csv`, med dato og tidspunkt når skrivingen begynte.

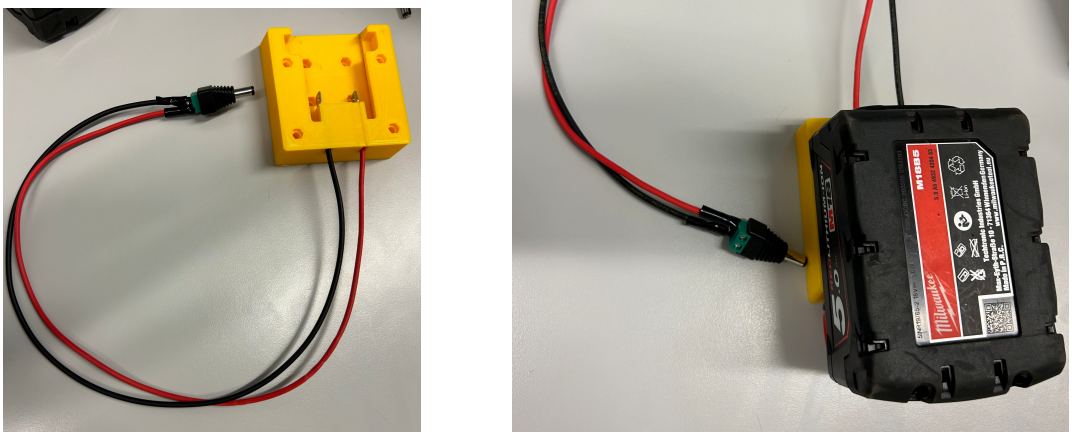
modelTraining.py

Dette scriptet brukes hvis man vil trene en YOLOv8-modell på egen data. Scriptet kan brukes for trening, og metoden beskrives i nærmere i seksjon 4.5.2.

4.6 Testing

4.6.1 Testing av lidar

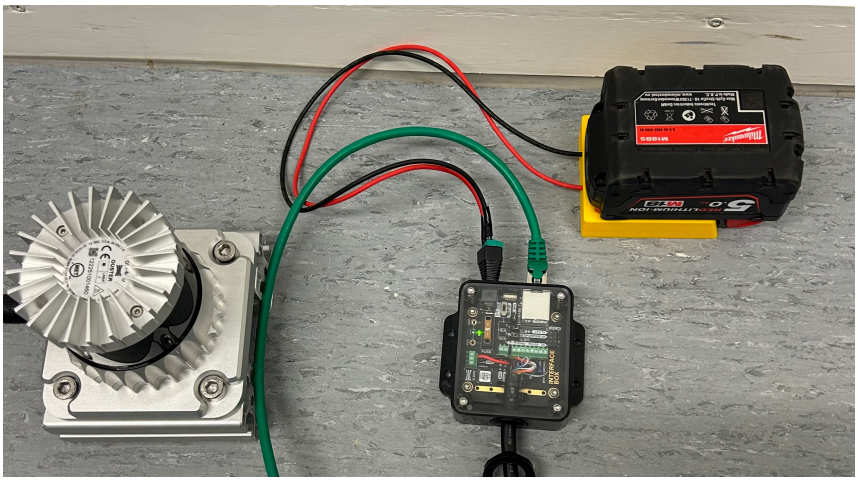
For testing av lidaren ble det brukt et Milwaukee drillbatteri som strømforsyning til lidaren. I flere tilfeller var det nødvendig å teste lidaren uten å opprette kommunikasjon eller montere den på Otteren. Det eneste problemet da er at lidaren ikke har et batteri og må alltid være tilkoblet en strømkilde. Dette problemet ble løst ved å bruke en drillbatteriadapter og en barreljack plugg, dette vises i figur 36. Interfaceboksen trenger minimum 9V før den gir en varsel, og maks spenning er 34V [36]. Drillbatteriet ble målt med et multimeter når det var fullladet å ga ut en spenning på 20V. Det viser at det er trygt å bruke drillbatteriet som strømforsyning til lidaren.



(a) Drillbatteriadapter (b) Drillbatteriadapter med drillbatteri

Figur 36: Drillbatteriadapter og drillbatteri

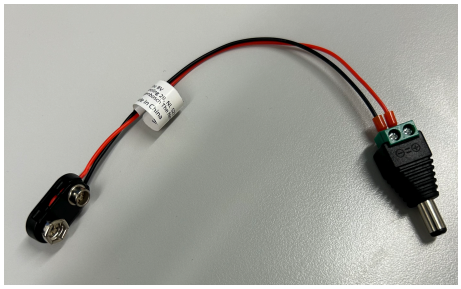
I figur 37 ser man oppsettet som ble brukt for testing av lidaren. Drillbatteriet gir strøm til interfaceboksen, som igjen gir strøm til lidaren. Lidaren vil strømme data tilbake til interfaceboksen som videresender dataen til en ekstern PC gjennom en ethernetkabel. Dette gjorde at vi kunne teste lidaren uten å være avhengig av en stikkontakt.



Figur 37: System for testing av lidar

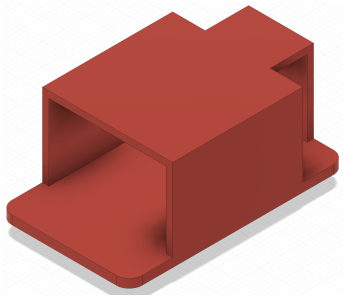
4.6.2 Testoppsett

På grunn av utfordringer med å integrere lidaren inn i nettverket på båten, ble det bestemt å bruke en ekstern ruter for kommunikasjon mellom lidaren og VCS-PC'en under testingen. Denne ruteren har ikke tilgang til strøm gjennom payload-boksen på grunn av manglende kabelgjennomføring som er tilpasset denne ruteren, og siden dette var en midlertidig løsning ble det ikke boret en ny kabelgjennomføring. I stedet får ruteren strøm fra et 9V-batteri gjennom ledningen som vist i figur 38.

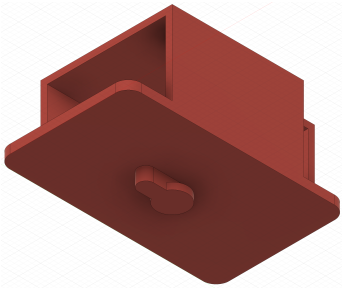


Figur 38: 9V batteri til barreljack kabel

For å sikre at 9V-batteriet holdt seg på plass og ikke dro ut ledningen, ble det designet en batteriholder. Denne batteriholderen ble designet slik at den kunne festes bak på ruteren ved hjelp av det eksisterende veggfestet. Modellen ble designet i Autodesk Fusion 360 og deretter 3D-printet med en Prusa MINI. På figur 39 kan man se fram og baksiden av batteriholderen, og i figur 39b kan man se festepunktet for batteriholderen.



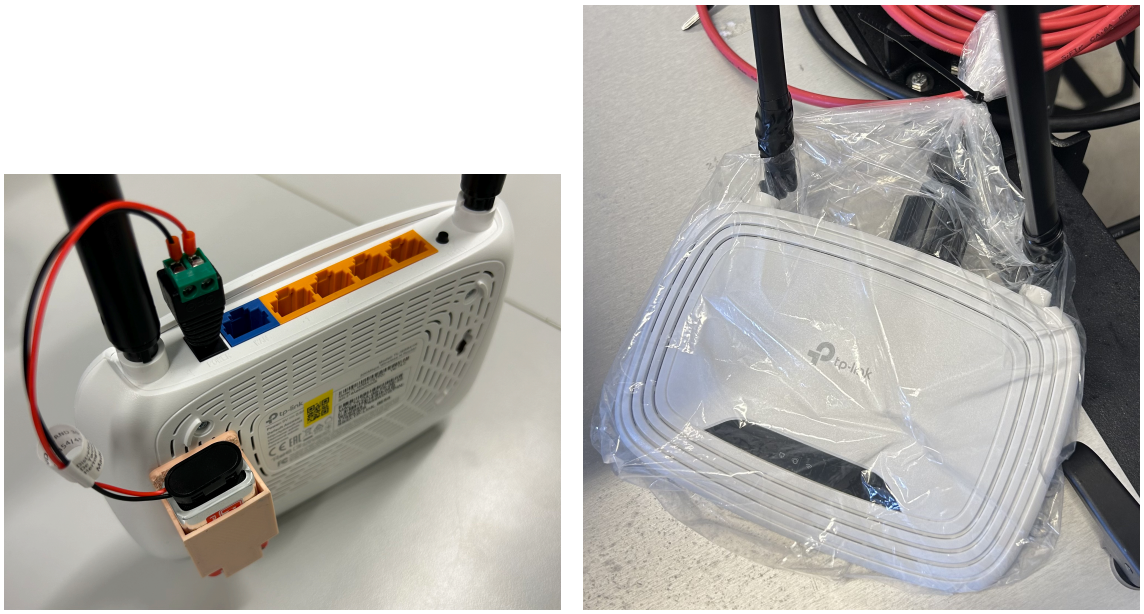
(a) Overside av batteriholder med feste



(b) Underside av batteriholder med feste

Figur 39: Batteriholder med feste

I figur 40 vises ruteren utstyrt med et 9V batteri som er festet på baksiden. Videre ser man en ledning som forbinder batteriet og gir strøm til ruteren. Dette integrerte systemet muliggjør full bærbarhet for ruteren, noe som gir den evnen til å fungere selvstendig. Denne egenskapen gjør det mulig for oss å benytte ruteren til testing ombord på båten uten behov for å lage unødvendige hull i payloadboksen. I tillegg kan vi enkelt beskytte ruteren og batteriet mot vann og fukt ved å plassere dem i en plastpose, noe som øker robustheten under testingen.



(a) Ruter og 9V-batteri (b) Vannbeskyttet ruter montert på otter

Figur 40: Ruter med og uten vannbeskyttelse

De tekniske spesifikasjonene til ruteren[19], tilsier at den forsynes av 9V og vil med dette trekke 0.6A. Batteriet som er tatt i bruk er et RS 9V-batteri med 1Ah kapasitet, ref datablad [37]. Basert på dette ble det beregnet hvor lenge batteriet er i stand til å drive ruteren, med formel 6 for kapasitetstid T:

$$T = \frac{C}{I} \tag{6}$$

hvor:

- C er batteriets kapasitet i ampere-timer (Ah),
- I er strømmen som trekkes av enheten i ampere (A).

Gitt at ruteren trekker 0.6 A og batteriet har en kapasitet på 1 Ah, setter vi inn verdiene i formel 7:

$$T = \frac{1\,\text{Ah}}{0.6\,\text{A}} \approx 1.67\,\text{timer} \tag{7}$$

Dette tilsvarer som vis i utregningen 8:

$$1.67\,\text{timer} \times 60\,\text{minutter} \approx 100\,\text{minutter} \tag{8}$$

Basert på disse beregningene kan ruteren drives av et 9V batteri med 1Ah kapasitet i omtrent 100 minutter under ideelle forhold. Selv om ideele forhold ikke blir realistisk i kretsen vår vil i underkant av 100 minutter være tilstrekkelig for tidsperioden vi skal utføre eksperimentet.

4.6.3 RPI

Det ble også sett må muligheten for å montere en raspberry Pi 4 på båten med tilhørende SD-kort. Tanken var å ta i bruk raspberry PI for å samle inn data fra lidaren og lagre dette lokalt på fartøyet under eksperimentet. På denne måten vil man selv med brudd i kommunikasjonen være i stand til å samle inn den nødvendige dataen.

For å kunne ta i bruk RPI'en ble SD-kortet flashet med bruk av Raspberry Pi Imager. Her bestemmer man hvilke enhet man har og hvilke operativsystem man ønsker å installere. Videre settes hostname på enheten og man gir den et brukernavn og passord for å forsikre at ikke uvedkommende kan koble seg på. Man har også mulighet til å koble seg på WiFi hvis ønskelig. For å enkelt kunne jobbe med raspberry Pi ble den koblet til ekstern skjerm og datamus. RPI-en ble koblet til et internett for å kunne installere de nødvendige bibliotekene med PIP. Dette ble gjort med å dele nettverk fra mobilen når det ikke er ønskelig å koble seg til skolenettet uten tillatelse. Deretter ble det opprettete et virtuelt miljø på RPI'en og de nødvendige programvarene ble lastet ned. For å samle inn den ønskede dataen ble det tatt i bruk Ouster's github-repo og `record_pcap` funksjonen [34]. Dette oppretter en JSON og en PCAP-fil som blir lagret på den angitte banen. Disse filene kan videre behandles med å laste de opp i Ouster Studio.

4.6.4 Eksperiment

I forbindelse med å teste det sammesatte systemet ble Otteren sjøsatt. Sjøsettingen har i hovedsak to formål der det første er å teste om det ferdig integrerte systemet fungerer som tiltenkt under realistiske forhold. Det andre målet var å få en realistisk visualisering av objektene rundt fartøyet. Ved å sjøsette fartøyet får vi lidarbilder fra en realistisk vinkel samtidig som det bli letter å innhente bilder av forbigående fartøy. Dette gjør at vi får samlet inn data som vil representere den faktiske synsvinkelen til lidaren på Otteren. Dataen som ble samlet inn ble markert og brukt til videre trening av systemet, som beskrevet i seksjon 4.5.2.

Otteren ble fraktet med varebil til plassen for sjøsettingen som vist på bildet 41 og markert med rød ring. Området sjøsettingen blir utført på er et området med lite sjøtrafikk som gjør at det er liten risiko for å komme i konflikt med andre trafikanter. Bildet 42 viser dronebilde av plasseringen for eksperimentet. Her ser man de fortøyde båtene som ble brukt til innsamling av data.

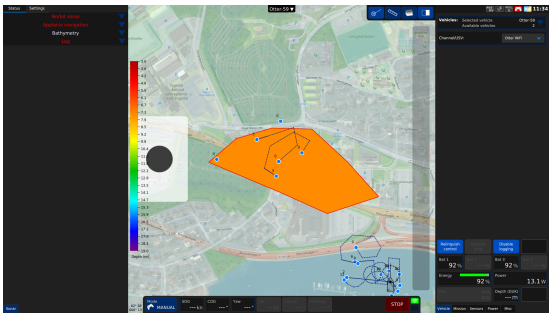
Figur 41: Plassering for sjøsetting

Figur 42: Dronebilde av Otter under eksperimentet

For å sikre at alt utstyr fungerte som tiltenkt før sjøsettingen ble det komplette systemet på Otteren funksjonstestet. Under dette arbeidet ble det også verifisert at kommunikasjonene og styringen av båten fungerte, noe det gjorde. Når Otterene også kan kommunisere over radio ble det satt opp en tilhørende antenne for å ha muligheten til å kommunisere over radio. På bilde 43 var den klar til sjøsetting og ble løftet ut i vannet ved bruk av påler som er tredd gjennom båten forran og bak. For at denne operasjonen skal bli så lett og risikofri som mulig var det viktig at sjøsettingen ble utført under flo.

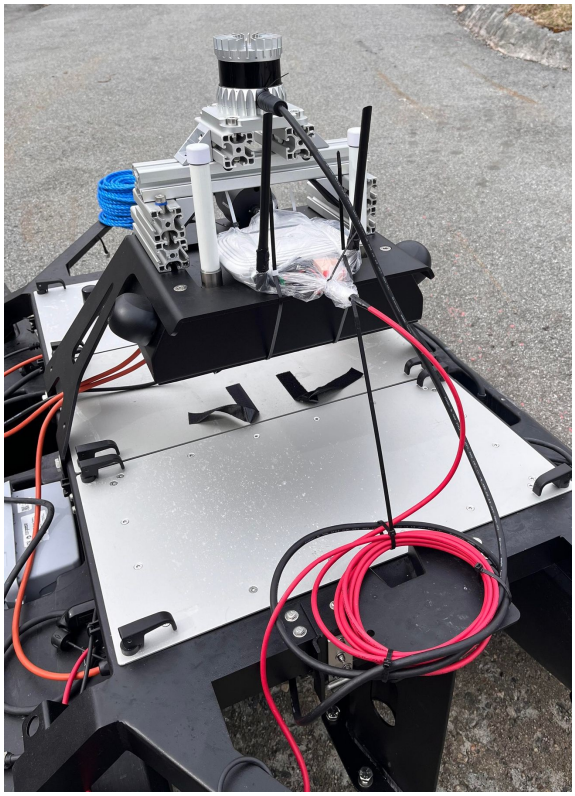
Otteren ble videre dradd ut med tau til en nærliggende flytebrygge for å unngå å få tang i propellene. Dette var også et tilltak for å sikre kontroll over Otteren selv ved eventuelle kommunikasjonsfeil. Tauet som var brukt er et 30 meter flytetau som er festet til Otteren med en karabinkrok. Ved å ta i bruk et flytetau unngår man at tauet kommer i kontakt med propellene. Disse sikkerhetstiltakene er tatt i bruk for å opprettholde full kontroll på fartøyet under hele eksperimentet. Selv om det var opprettet kommunikasjon til Otteren og ruteren før sjøsetting oppsto det likevell problemer med at vi mistet kommunikasjonen for styringen av Otteren. Etter en del feilsøking fra land ble båten tatt oppigjen av vannet for å få en mer effektiv feilsøkingsprosess. Alle tilkoblingspunktene ble sjekket samt batteriene når dette er det vi har enklest tilgang på. Eneste som kunne tyde på noe feil var tilkoblingen til ene batteriet som ikke var helt tilskrudd. Det ble opprettet kommunikasjon til Otteren over en gitt periode for å forsikre oss at denne var stabil. Når vi anslo at kommunikasjonen var i orden ble det opprettet kommunikasjon til ruteren og startet mottak av data. Dette var for å sjekke om signalene fra de to systemene vil forstyrre hverandre. Når all kommunikasjon var opprettet ble båten sjøsatt i samme prosedyre og fraktet ut til flytebryggen. Denne gangen oppsto det ikke noe kommunikasjonsproblemer og Otteren ble dermed kjørt ut i sundet ved bruk av en tilhørende VCS-PC. Måten for manuell styring av Otteren kan sees på bildet 44 under.

Figur 43: Sjøsetting av Otter



Figur 44: Manuell styring av Otter

Eksperimentet ble utført i sundet ved Sunnmøre Museum etter tillatelser fra dem. Ved dette sundet ligger det en rekke båter i tillegg til at det til tider passerer noen båter som gjør det ideelt for innsamling av dataen vi trenger. Som nevnt i seksjon 4.6.2 ble det tatt i bruk en ruter under testing av båten på grunn av utfordringene med å integrere lidaren på Otter nettverket, beskrevet i seksjon 6.2.2. Oppsettes som ble brukt under testing kan sees på bildet 45 under. Ved bruk av dette oppsettet fikk vi kjørt båten med VCS-PC. Reserveløsningen med bruk av raspberry Pi som beskrevet i seksjon 4.6.3 ble ikke aktuelt å ta i bruk når kommunikasjonen mellom PC og lidar fungerte som tiltenkt og vi fikk samlet inn den dataen vi ønsket.



Figur 45: Testoppsett

Under tidsperioden eksperimentet ble utført var det desverre ikke noen passerende båter. Dette hadde vært ønskelig for å få lidarbilder fra en realistisk situasjon av fartøy i bevegelse. Det ble det derfor fokusert på å filme de fortøyde båtene fra diverse vinkler for å skape et fullstendig bilde av dem. Grunnen til at dette ikke er helt ideelt er at båtene vil ha relativ lik avstand som til land noe som vil føre til at de vil forsvinne inn i bakgrunnen og dermed vanskelig å observere.

Ved frittgående båter vil det vises som et hvit objekt som skiller seg tydelig fra den mørke bakgrunnen og er derfor lett å observere. På bildet 46 ser man tydelig kajaken som passerer mens de fortøyde båtene forsvinner inn i bakgrunnen. Med dette sagt vil dataen som er innsamlet være tilstrekkelig med tanke på videre trening av modellen og for å skape en forståelse for hvorden det integrerte systemet fungerer.

Figur 46: Kontraster i lidarbildet

Det ble også gjort et forsøk på å teste den integrerte alarmgrensen i systemet som viste seg å være noe vanskelig når det ikke var noen frittgående båter i området. Det ble gjort et forsøk på å teste alarmgrensen på de fortøyde båtene, men det var vanskelig å få det ønskede resultatet. Dette grunnet faktorene at vi ikke ønsket å komme for nære samtidig som båtene lett ble en del av bakgrunnen som tidligere nevnt. På grunn av dette fikk vi ikke noen gode testresultat på alarmgrensen under eksperimentet. Med dette sakt er alarmgrensen godt testet tidligere og baserer seg på målingene fra lidaren som er verifisert opp mot eksterne målinger og virker ganske nøyaktig.

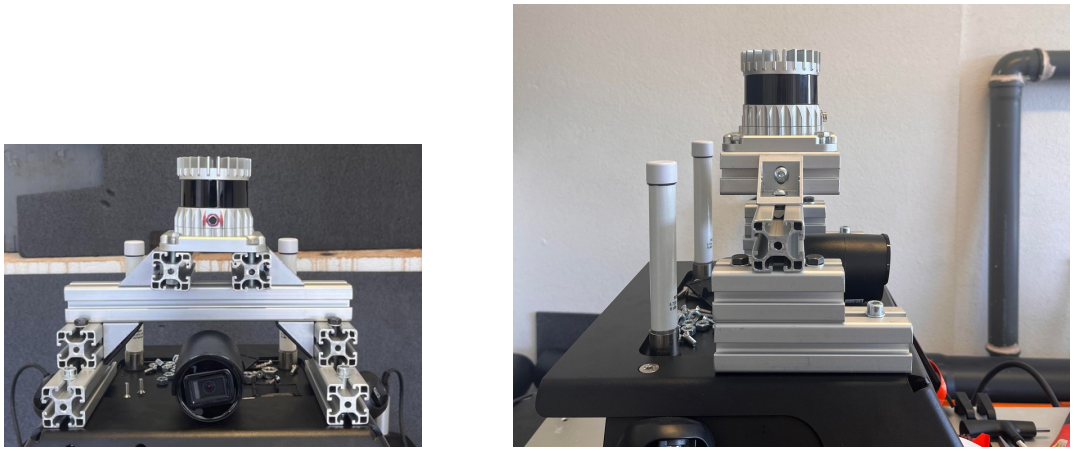
Når dette var første gang det er samlet inn data i en realistisk situasjon er det viktig å kunne ta i bruk denne dataen til videre utvikling av systemet og finne eventuelle forbedringer. For innsamling av data gjennom eksperimentet ble det derfor tatt i bruk et Python-script der vi fikk sanntidsbilder fra lidaren og ble lagret på en angitt fil. Den innsamlede dataen vil gi et innblikk i hvordan systemet fungerer i en realistisk situasjon og vil bli tatt i bruk for videre trening av modellen for å en mer nøyaktig objektdeteksjon. Under eksperimentet var det relativ stille sjø noe som førte til at det ikke var store endringer på IMU-dataen. Denne dataen vil si noe om hvor mye ujevn sjø vil påvirke lidarbildet og i hvor stor grad stabiliseringen av bildet er nødvendig. Alt i alt var det gjennomførte eksperimentet velykket og den nødvendige dataen ble innsamlet.

5 Resultat

I denne delen av rapporten vil de endelige løsningene som er implementert for å oppnå resultatet bli presentert. Det utførte arbeidet resulterte i følgende:

5.1 Montering og oppkobling

Det endelige resultatet for stativet til lidaren ble laget av aluminiumsprofiler som vist på bildet 47 under. Dette designet gir lidaren en god høyde og gjør den i stand til å levere gode bildet av omgivelsene. Høyden sikrer også at lidaren ikke kommer i konflikt med andre komponenter, noe som hindrer forstyrrelser i lidarbilder.



(a) Montert lidar

(b) Sideprofil av lidar

Figur 47: Montering av lidar

Tilhørende interfaceboks er montert inne i payload boksen på en egen designet brakett som vist på bildet 48. Her ser man at interfaceboksen blir forsynt av en barrel jack plug som henter en spenning på 12V fra PCB kortet. Ethernettilkoblingen går fra interfacboksen og videre til en gjennomføring ut av boksen. En del av lidarkabelen er kveilet opp inne i boksen for å minimere bevegelige deler på utsiden. Alt utstyr som ble brukt for å montere lidaren og interfaceboksen kan sees i tabell 1 og 3.

Figur 48: Resultat av oppkobling i Payloadboksen

Når integreringen av lidaren i det eksisterende nettverket ikke fungerte som tiltenkt, ble en ekstern ruter tatt i bruk for å sende data. Ruteren forsynes av en batteripakke og kobles videre inn i payloadboksen og sammen med interfaceboksen for å etablere kommunikasjon mellom enhetene. Ved bruk av ruteren opprettes et nettverk som gjør det mulig å koble seg på og dermed motta lidardata over WiFi ved bruk av UDP.

5.2 Lidar

Lidarbildet brukt i det endelige resultatet ble visualisert ved bruk av rangemetoden. Bilde 49 viser visualiseringen fra eksperimentet. Dette bildet ga det beste resultatet av de tilgjengelige visualiseringsmetodene. Visualiseringen gir et tydelig bilde av omgivelsene med gode kontraster som gjør det enklere å skille objekter fra bakgrunnen i et åpent landskap. I midten av bildet ser man tydelig en person på bryggen. Det er vanskeligere å skille objekter fra hverandre når de står samlet, da de får liknende farger. Et eksempel på dette sees på venstre side i bildet der man ser konturene av objektene men det er noe vanskeligere å se detaljer. En datamaskin vil kunne skille mellom ulike nyanser av grått og vil derfor ikke ha utfordringer med å gjenkjenne kontrastene mellom objektene. Når Otteren i hovedsak skal operere i et åpent landskap ute på sjøen vil dette problemet ikke ha stor betydning under vanlig drift.

Figur 49: Endelig visualiseringsresultat

Når rangemetoden bruker avstander til å skape en visualisering av omgivelsene, ble disse dataene videre brukt til å finne avstanden til objektene i omgivelsene. Dette ble gjort med å hente ut en liste over x- og y-koordinater, som representerer posisjonene til boksene YOLO-modellen bruker for å markere de identifiserte objektene i bildet. Ved å ta i bruk denne informasjonen regnes midtpunktet av hver enkelt boks ut. Dette punktet brukes deretter til å finne avstanden til objektet. Denne metoden er uttestet gjennom eksperimentet og ser ut til å fungere som tiltenkt på bakgrunn av at YOLO-modellen markerer objektene med en boks som vil ha objektet den markerer i sentrum, som vist på bildet i figur 50.

Figur 50: Objektmarkering

Det endelige systemet inneholder logikk for et alarmsystem for å varsle operatørene mot nærliggende objekter. I lidarbildet vil det til enhver tid vise sanntidsavstand til objektene den detekterer. Det settes derfor en minimum avstand til objektene som må krysses før man vil bli varslet. Når et objekt krysser denne grensen vil operatøren bli varslet med en beskjed som vil legges på lidarbildet. Bildet vist i figur 51 er tatt i garasjen der Otteren er lagret, siden det viste seg å være vanskelig å teste alarmgrensen under eksperimentet. Hvis objektet beveger seg utenfor minimumsavstanden vil alarmen igjen forsvinne. Dette ble gjennomført ved å bruke en YOLOv8-modell vi har trent på gjenkjenning av mennesker, som forklart i seksjon 4.5.3. Denne modellen, som gir oss muligheten til objektdeteksjon av mennesker, gjorde testingen av funksjoner som alarmgrensen veldig enkel.

Figur 51: Aktivert alarm

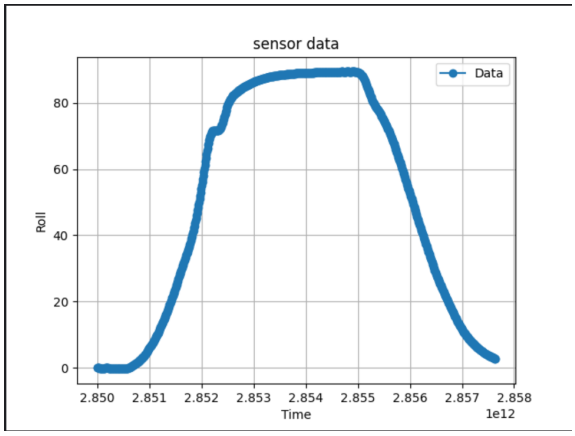
5.3 IMU

IMU-dataen som mottas regnes om til *pitch* og *roll* som vist i bilde 52, denne prosessen er beskrevet i seksjon 4.4.2. Første kolonne viser tidsstempler i millisekunder, mens de to påfølgende kolonnene representerer beregnede vinklene for pitch og roll, i den rekkefølgen. Positive verdier indikerer tipping fremover og rulling mot høyre mens negative verdier vil indikere det motsatte. Dataene under er representert i grader, dette vil si at verdiene er en direkte representasjon av vinkelen til objektet.

```
757.82804594,-23.393647032985324,0.3943928296922848
757.838046095,-22.922544234562075,-0.380977329116322
757.84804608,-22.27740636911637,-0.3372620720854969 7
757.858046125,-20.71781427717731,-0.17988868921389245
757.86804594,-20.319182548954043,-0.06882123390175385
757.878046105,-21.275744429822705,-0.06154648826060705
757.88804597,-18.985076984047215,0.005798261566193619
```

Figur 52: Beregnet IMU data

På figur 53 ser man endringen i roll til IMU'en i lidaren over en gitt periode vist i millisekunder. Lidaren begynner i en horisontal stilling og beveger seg deretter gradvis til en vertikal posisjon, før den igjen returneres til sin opprinnelige posisjon. De to uventede støyenelementene rett før og rett etter toppen av grafen kan trolig forklares ved at lidaren har skarpe kanter som den støtter seg på når den blir løftet til en vertikal stilling. Basert på resultatene fra denne testen fungerer IMU'en som tiltenkt og vil gi den nødvendige informasjonen om bevegelsene til Otteren for å kunne stabilisere lidarbildet. Videre bruk av dataene for å stabilisere lidarbildet ble det ikke tid til i løpet av prosjektperioden.

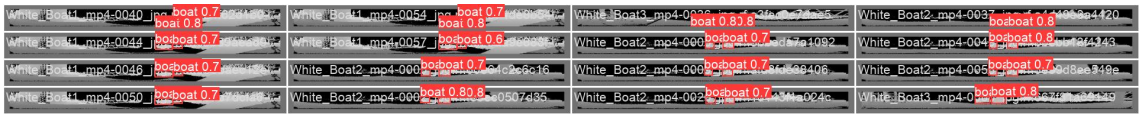


Figur 53: IMU Roll Vinkel

5.4 Objektdeteksjon med tilpasset YOLOv8-modell

Det ble utformet en YOLO-modell tilpasset å detektere båter i et lidar 2D-bilde. Modellen ble trent på et datasett bestående av lidar-etterligninger og faktiske lidar-bilder. Vår tilpassede modell ble trent på dette datasettet, som beskrevet i seksjon 4.5.2.

I figur 54 observerer vi modellens deteksjoner på vår valideringsdata. Som nevnt i seksjon 4.5.1, er datasettet oppdelt i trenings-, valideringss- og testsett. Modellen blir først trent på treningssettet og deretter brukt til å detektere objekter på valideringssettet. Testsettet kan deretter benyttes til å evaluere ytelsen til modellen vår. Ifølge figur 54 viser modellen vår god evne til å detektere båter. Den oppdager av og til avvik som ikke er båter, men den er nesten alltid i stand til å identifisere faktiske båter.

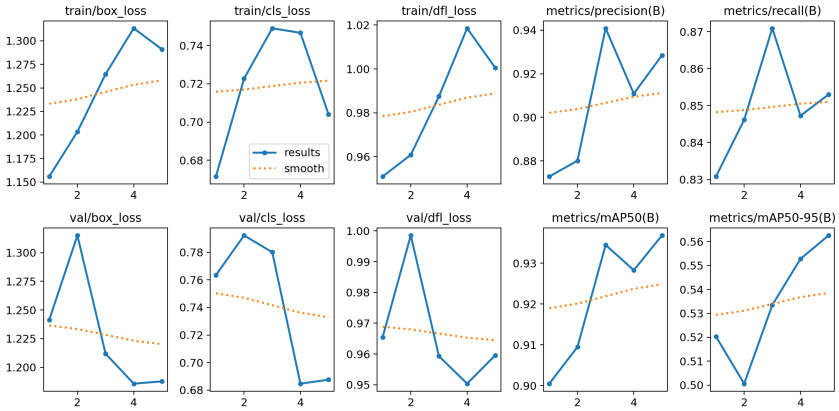


Figur 54: Validation batch prediction

I figuren 55 under ser man det endelige trenings- og valideringss-resultatene av den trente YOLO-modellen over flere epochs. Dette gir et innblikk i modellens læring og ytelsesforbedringer basert på analyser for forskjellige typer tap som box loss, classification loss og distribution focal loss (DFL), samt ytelsesmålinger som presisjon, recall og mean Average Precision (mAP).

Box og classification loss måler nøyaktigheten av boksplasseringen og objektklassifisering og har som man ser en nedadgående trend i valideringsdataen. Dette indikerer at modellen blir mer nøyaktig og er en positiv indikator på at modellen lærer de riktige egenskapene fra treningsdataen og er i stand til å bruke kunnskapen på ny usett data. *DFL loss* viser også en forbedring over tid noe som tilsier at modellen blir bedre på å håndtere klasseubalanse. Dette er en viktig egenskap i systemer som skal kjenne igjen mange forskjellige objekter, hvor noen er mer uvanlige enn andre.

Presisjon og recall gir en indikator på hvor nøyaktig modellen klarer å identifisere objekter. Som man ser blir tallene bedre men har noen svingninger som er normalt og kommer ofte av små endringer i hvordan modellen lærer eller i dataene den trenes på. Til slutt viser mAP-verdiene, som gir et gjennomsnitt av presisjon ved forskjellige nivåer av nøyaktighet, en tydelig forbedring. Dette indikerer at modellen er i stand til å identifiserer objekter med høy nøyaktighet [51][49].



Figur 55: Resultatene fra YOLOv8

Resultatene tyder på en vellykket trening av YOLO-modellen, der tapet minsker over tid samtidig som deteksjonsnøyaktigheten øker. Dette vil gi oss en nøyaktig og pålitelig modell som er i stand til å detektere objekter i realistiske scenarier.

5.5 Eksperiment

Det utførte eksperimentet ble gjort for å se resultatet av det fullstendige systemet samt få et innblikk i hvordan den trente YOLO-modellen fungerer under realistiske forhold. Kommunikasjonen til Otteren fungerte som tiltenkt og ruteren ga en rekkevidde som var tilstrekkelig under utførelsen av eksperimentet. På denne måten var vi i stand til å motta lidardata over WiFi via UDP under utførelsen. På bildet 56 under ser man resultatet av den trente YOLO-modellen. Her ser vi inn mot land og ser at modellen er i stand til å detektere to båter i lidarbildet samt markere disse med tilsvarende bokser. Som man kan se, vises sanntidsavstand til de detekterte båtene, henholdsvis 26 og 48 meter, noe som virker realistisk. Avstanden er oppgitt i millimeter, men enheten er notert som meter, noe som er en feil vi er klar over. Den viser også hvilken klasse de detekterte objektene er i. I vårt tilfelle er lidaren trent på å gjenkjenne båter og objektene den detekterer er dermed markert med klassen *Boat*.

Figur 56: Lidar sanntidsdeteksjon

5.6 Python-script

Som nevnt i seksjon 4.5.6 har det blitt lagd et fullstendig Python-prosjekt med all nødvendig kode for å kjøre eksperimenter eller jobbe videre med prosjektet. Dette scriptet inneholder en `main.py` fil som er den endelige koden vår der alle funksjoner er satt sammen. Denne koden detekterer objekter, kalkulerer avstand til objekter, varsler når objekter er innenfor en avstandsgrensegrense, og tar opptak av deteksjoner og videostrøm. Denne koden ligger vedlagt som vedlegg A.

6 Diskusjon

I denne delen av rapporten vil det diskuteres hvordan utførelsen av oppgaven har fungert gjennom prosjektperioden. Videre vil det bli sett på løsningen vi endte opp med og hvordan det eventuelt kunne ha vært forbedret ved bruk av andre metoder. Utfordringene som vi møtte på gjennom prosjektperioden vil bli beskrevet og det blir diskutert rundt hvordan vi kunne ha unngått dem, og i tillegg blir det forklart hvordan de ble løst. Til slutt vil det bli sett på eventuelle videre forbedringer av prosjektet.

6.1 Feil og problemer

I denne delen vil vi gå gjennom feil og problemer vi hadde da vi løste vår oppgave.

6.1.1 Feil Pythonversjon

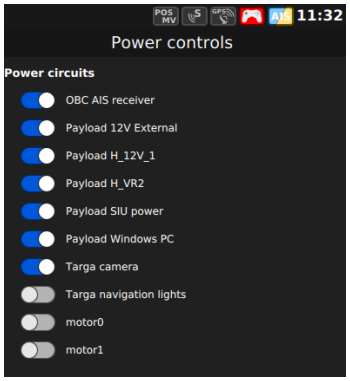
Vi hadde et problem med å bruke Ouster sitt Python-SDK. Vi brukte Python 13, men fant ut at dette SDK'et støttet Python 3.7 og nyere, men tydeligvis ikke den nyeste Python 13. Ved hjelp av PyCharms virtuelle miljø fikk vi kjørt SDK'et med riktig versjon, og dette løste problemene våre.

6.1.2 Aktivering av PCB-utgang i samarbeid med Maritime Robotics

Under arbeidet på PCB-kortet oppdaget vi at det ikke var spenning på utgang H_VR2 og H_12V_1. Det var disse utgangene vi ønsket å ta i bruk for å forsyne lidaren med spenning, som beskrevet i seksjon 4.2.4. Det ble derfor sett på koblingsskjema tilsendt av Maritime Robotics, som vist i figur 15, som tilsier at denne kan justeres ved bruk av potensiometer R25. Dette ble prøvd uten hell, men det ble verifisert at spenningen på H_VR1 ble justert som ønsket.

På bakgrunn av dette ble Maritime Robotics kontaktet når vi lurte på om utgangene på PCB-kortet måtte aktiveres. Her ble det bekreftet at utgangene måtte aktiveres av dem i en konfigurasjonsfil gjennom OBS PC'en i Otteren.

Det ble derfor koblet opp ethernet kommunikasjon fra PCB-kortet gjennom OBS-PC'en og til VCS-PC'en slik at de gjennom TeamViewer møte kunne aktivere de nødvendige utgangene. Etter utgangene hadde blitt aktivert ble de synlig på listen over utgangene på VCS-PC'en som vist i figur 57. Utgangene representeres her med skyveknapper som gir oss muligheten til å aktivere og deaktivere utgangene på PCB-kortet etter eget behov.



Figur 57: Aktiverte PCB utganger

6.1.3 Kommunikasjon til Otter-nettverket

Under arbeid med montering av lidaren på Otteren var det ønskelig å integrere den i det eksisterende nettverket. Dette vil gjøre systemet i stand til å bruke den installerte OBS-PC'en til å sende lidardata over WiFi til en ekstern PC som vil gjøre at man slipper å ta i bruk testoppsettet som beskrevet i seksjon 4.6.2.

Under arbeidet med å integrere lidaren inn på det nåværende nettverket på Otteren oppsto det noen utfordringer. Når det ble prøvd å integrere lidaren ble det oppdaget at vi ikke klarte å finne den på nettverket. Det ble derfor gjort noen tiltak som å skifte ut eventuelle defekte ethernetkabler, samt koble lidaren direkte til OBS-boksen. Deretter ble det forsøkt å pinge lidaren uten hell. Neste som ble forsøkt var å sette en statisk IP på lidaren. På denne måten vet vi IP-adressen til sensoren og dermed kunne prøve å finne den i nettverket, fortsatt uten hell. Det ser dermed ut at payloadswitchen og OBS-boksen er på to forskjellige nettverk. Når vi hadde lite informasjon om hvordan systemet er bygget opp i tillegg til at vi ikke har tilgang til OBS-boksen ble Maritime Robotics kontaktet for support. Her fikk vi tilbakemelding på at nettverket på Otteren primært benytter 53-subnettet og blir deretter rutet gjennom OBS-boksen til andre nettverksgrensesnit som WiFi.

6.2 Videre forbedringer

I denne delen av diskusjon seksjonen vil det diskuteres hvilke videre forbedringer som kan implementeres i systemet for å eventuelt øke kvaliteten og effektiviteten.

6.2.1 Lokal prosessering av data

I det endelige systemet som blir presentert under resultater i seksjon 5.1 blir lidardata sendt over WiFi uten noen form for lokal behandling på Otteren. Denne fremgangsmåten gjør at man har tilgang til all lidardataen i VCS-PC'en.

En videre implementering i systemet kan være videre arbeid med integreringen av Raspberry Pi på Otteren, som beskrevet i seksjon 4.6.3. Dette vil muliggjøre lokal behandling av data på Otteren, slik at man kan filtrere ut og sende kun den relevante informasjonen over WiFi til den eksterne PC'en. Denne implementasjonen vil redusere forsinkelsen i systemet, noe som bidrar til forbedret ytelse og responstid. Datamengden som trengs å sendes over nettverket vil også være betydelig mindre som vil være gunstig i områder med begrenset eller upålitelig nettverkstilkobling.

6.2.2 Integrering av Ouster lidar og Otter USV

En videre forbedring av systemet vil være å løse opp i problemet med integreringen av lidaren inn på Otter nettverket som beskrevet tidligere i seksjon 6.1.3. Ved å løse opp i dette problemet vil man være i stand til å sende lidardata uten behov for å ta i bruk testoppsettet som nevnt i 4.6.2, noe som vil føre til et mer fullstendig system.

6.2.3 Utvikling av YOLOv8-modell

Det endelige resultatet av YOLO-modellen vår er presentert i seksjon 5.4. Modellen som er utviklet fungerer som tiltenkt basert på de presenterte resultatene og er i stand til å detektere båter i lidarbildet. Ved videre innsamling av data fra Otteren kan modellen trenes videre for å være i enda bedre stand til å observere båter. Det er også mulig å implementere andre klasser i modellen for å være i stand til å skille mellom for eksempel motorbåt og seilbåt. Dette er noe vi ikke har tatt stilling til når vi har trent den på generelle båter når dette er tilstrekkelig for å kunne varsle operatøren.

6.2.4 Stabilisering av lidarbildet

IMU-dataen som blir hentet ut gir informasjon om bevegelsene til Otteren som beskrevet i seksjon 4.4.2. Denne dataen kan videre brukes til å stabilisere lidarbildet ved ujevn sjø noe som vil være en videre forbedring av systemet. En tilnærming for å løse denne problemstillingen kan være å ta i bruk en rotasjonsmatrise for å kompensere for bevegelsene til Otteren. En rotasjonsmatrise er en matematisk transformasjon som kan benyttes til å rotere koordinatene i 2 eller 3 dimensjoner. Ved å ta i bruk pitch- og roll- verdiene inn i rotasjonsmatrisen kan lidarbildet roteres i motsatt retning av Otterens bevegelser. Dette vil være en forbedring av det nåværende systemet for å få stabile lidarbilder selv med ujevn sjø.

6.2.5 Filtrering av IMU-data

Som nevnt gjennom rapporten ble et kalmanfilter implementert for å teste hvordan dette påvirket IMU-dataen. På bakgrunn av at vi ikke implementerte det videre i prosjektet samt at vi ikke ønsket det for komplekst valgte vi å ikke bruke tid på filtrering av IMU-dataen. Ved videre arbeid kan det være aktuelt med implementering av et egnet filter for å skape mer stabile og pålitelige verdier. Dette vil gjøre dataen mer egnet for stabilisering av lidarbildet selv ved ujevn sjø.

6.2.6 Robot Operating System (ROS)

I startfasen av prosjektet ble det sett på muligheten til å ta i bruk Robot Operating System (ROS), som er et åpent kildekoderammeverk. Programvaren er utviklet for å gjøre det lettere for brukere å integrere forskjellige maskinvarekomponenter, som Ouster lidar, i et prosjekt. Den inneholder en rekke funksjoner og pakker som er designet direkte mot Ouster sensorer. Dette skal gjøre det lettere å ta i bruk sensoren med tanke på kommunikasjon, datahåndtering og simulering. Ouster har også et Python-SDK knyttet til sine lidarer med funksjoner som beskrevet i seksjon 3.2.6. Dette er også laget direkte mot Ouster lidarer og er lett å ta i bruk.

Dette var noe gruppen tok stilling til tidlig i prosjektet ved å se på fordelene og ulempene med de forskjellige fremgangsmåtene. Med å få et innblikk i ROS fant vi ut at det er et bra verktøy men at bruken ville kreve en betydelig læringskurve når ingen av gruppemedlemmene har jobbet med programvaren tidligere.

Når gruppen har gode Pythonkunnskaper, falt avgjørelsen på å ta i bruk Ousters Python-SDK gjennom utførelsen av prosjektet. SDK'et gir en enklere og mer direkte tilnærming til å arbeide med Ouster-lidarer, noe som passer godt med vår eksisterende kompetanse og prosjektets krav. Bruken av SDK muliggjorde rask utvikling og testing av nødvendige funksjoner uten å måtte tilegne oss kunnskap om bruken av ROS. Dette førte til en mer effektiv arbeidsflyt og raskere implementering av løsninger, som var avgjørende for å holde prosjektet innenfor tidsrammene og målene vi hadde satt.

6.2.7 Python vs C++

Siden gruppen har større kjennskap til Python, ble det besluttet å gjennomføre prosjektet ved hjelp av dette programmeringsspråket. Selv om Ouster-SDK støtter både Python og C++, var dokumentasjonen bedre for Python. Det er også enklere å lære og raskere å utvikle i, men når det gjelder ytelse og ressursutnyttelse i krevende objekt-deteksjonsoppgaver, kan C++ være et bedre valg.

Det er sant at C++ kjører raskere enn Python, spesielt når det gjelder ressursintensive oppgaver som bildebehandling. I tillegg gir C++ utvikleren mer kontroll over minneadministrasjonen, noe som kan være viktig for optimal utnyttelse av systemressurser i bildebehandlings-applikasjoner. C++ gir også mulighet for enkel implementering av parallelle beregninger ved hjelp av threading, noe som kan være nyttig for å akselerere behandlingen av store datamengder i sanntid.

6.2.8 Avstandsberegning

Avstandsberegningen tar, som nevnt i seksjon 4.3.6, midtpunktet av markeringen for å beregne avstanden. Denne måten fungerer som tiltenkt med forutsetningen at markeringen tar objektet i senter. Fremgangsmåten fungerer som tiltenkt men her kan det legges på en buffer, eksempelvis å regne gjennomsnittet av et $10 \cdot 10$ område og se bort fra ugyldige verdier for å øke nøyaktigheten på målingene.

6.2.9 Bruk av aluminiumsprofiler

Lidarstativet er som beskrevet i seksjon 4.2.2 er laget av aluminiumsprofiler. Dette er et material som egner seg på grunn av lav tetthet, samtidig som det har god styrke og stivhet i forhold til sin egen vekt. Dette er også et fornuftig valg når Otter'en har en maks lasteevne på 30kg som det er ønskelig å utnytte best mulig. Under valget med å bruke aluminiumsprofiler ble det sett på anbefalingene fra Ouster [35], men også påvirkningene dette kan ha på lidarsignalene. Aluminium er et materiale med høy reflektivitet egenskaper, noe som kan føre til forstyrrelser i lidarbildet hvis aluminiumen kommer i lidarbildet. Dette ser vi ikke på som et problem i systemet vårt når det ikke er fysisk mulig for lidaren å se aluminiumet. I tillegg til dette er det satt en 1 meters grense på lidaren som gjør at den ikke ser objekter innenfor denne grensen.

Et alternativ for å unngå dette kunne vært å bruke et annet type materiale som for eksempel plast, som har lavere reflektivitet og dermed minimerer risikoen for signalforstyrrelser. Plast er også lettere enn aluminium, noe som vil være med å redusere totalvekten og dermed øke batterilevetiden eller lastekapasiteten til Otter'en. På den andre siden har plast vanligvis lavere styrke og stivhet, noe som kan kreve et mer robust design for å oppnå samme stabilitet som med aluminium.

6.2.10 Bildefrekvens

Som beskrevet i seksjon 4.5.5 har YOLOv8 en funksjon kalt `track`. Denne funksjonen har et argument kalt `persist`, som gjør det mulig å opprettholde deteksjoner over flere bilder. Siden bildefrekvensen vår var så lav som 9 bilder per sekund, kan det være at `track`-funksjonen ville vært enda mer effektiv med en vesentlig høyere bildefrekvens. Dette ville ført til mindre endringer mellom hvert bilde som skal prosesseres. For å øke bildefrekvensen, måtte koden trolig optimaliseres og det ville vært nødvendig med kraftigere hardware med mer prosesseringskraft. Som beskrevet i seksjon 6.2.7 kunne også en overgang til C++ gjort bildebehandlingen enda raskere.

7 Konklusjon

Denne bacheloroppgaven har demonstrert en vellykket integrasjon av lidar på en USV, noe som har økt fartøyets situasjonsforståelse. Gjennom testing og utvikling av en YOLOv8-modell har prosjektet oppnådd sitt mål ved å være i stand til å detektere nærgående fartøy. Ved bruk av lidarbilder og maskinlæringsalgoritmer er det mulig å detektere og klassifisere nærliggende fartøy med høy nøyaktighet. Sanntidsavstanden til fartøyene vises i lidarbilder, sammen med en alarm ved kryssing av den satte alarmgrensen. Dette bidrar til å øke operatørens situasjonsforståelse, noe som vil resultere i en betydelig reduksjon i faren for sammenstøt.

Implementeringen av lidar og utviklingen av detekteringssystemet har ikke bare økt sikkerhetsaspektet ved bruk av Otteren, men har også lagt grunnlaget for fremtidig utvikling av fartøyet og dets bruk i forskningsoppgaver. Videre arbeid med Otteren kan omfatte integrasjon av flere sensorer for økt funksjonalitet og forbedring av algoritmens evne til å fungere under varierende miljøforhold.

Gruppen har gjennom prosjektperioden tilegnet seg verdifull erfaring innen lidarteknologi, sensorintegrering og programmering, samt bruken av dette innen maritim sektor. Prosjektet har vist nytten av integreringen av lidaren, samt potensialet for videre utvikling og økt funksjonalitet til Otteren.

Kilder

[1] Coommsexpress. *T568A AND T568B Wiring Standards*. URL: https://www.comms-express.com/infozone/article/t568a-and-t568b/ (sjekket 2. mai 2024).

[2] D. Piyabongkarn and R. Rajamani and M. Greminger. *The development of a MEMS gyroscope for absolute angle measurement*. 2005. URL: https://ieeexplore.ieee.org/abstract/document/1397754.

[3] dependabot and glenn-jocher and fcakyon and Laughing-q and Burhan-Q. *Model Training with Ultralytics YOLO*. URL: https://docs.ultralytics.com/modes/train/ (sjekket 30. apr. 2024).

[4] Didrik Grove. *Multi-sensor multi-target tracking using LIDAR and camera in a harbor environment*. 2021. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2781101.

[5] Fisher Shi. *Object Detection and Tracking using Deep Learning and Ouster Python SDK*. URL: https://ouster.com/insights/blog/object-detection-and-tracking-using-deep-learning-and-ouster-python-sdk (sjekket 1. mai 2024).

[6] Fluke Networks. *Differences Between Wiring Codes T568A vs T568B*. URL: https://www.flukenetworks.com/knowledge-base/application-or-standards-articles-copper/differences-between-wiring-codes-t568a-vs (sjekket 16. mai 2024).

[7] Fusion. *Autodesk Fusion: det er mer enn CAD, det er fremtiden for design og produksjon*. URL: https://www.autodesk.no/products/fusion-360/overview?term=1-YEAR&tab=subscription (sjekket 18. mai 2024).

[8] Geir Andresen. *akselerometer*. URL: https://snl.no/akselerometer (sjekket 16. mai 2024).

[9] Génération Robots. *Ouster Mid-Range OS1*. URL: https://www.generationrobots.com/en/403994-lidar-ouster-os1-medium-range-high-performance.html (sjekket 19. mai 2024).

[10] glenn-jocher and UltralyticsAssistant and Burhan-Q and plashchynski and tensorturtle and AyushExel and Laughing-q. *Model Prediction with Ultralytics*

YOLO. URL: https://docs.ultralytics.com/modes/predict/ (sjekket 16. mai 2024).

[11] Harald Øverby. *OSI (datakommunikasjon)*. URL: https://snl.no/OSI_-_datakommunikasjon (sjekket 16. mai 2024).

[12] Hetarth Chopra. *PROGRAMMING OF REMOTE SURVEILLANCE ROBOT AND ODOMETRY SENSOR CALIBRATION PROTOCOL FOR CMR'S*. 2021. URL: https://www.researchgate.net/publication/352488675_Programming_Of_Remote_Surveillance_Robot_And_Odometry_Sensor_Calibration_Protocol_For_CMR's.

[13] InvenSense. *High Performance Automotive 6-Axis MotionTracking Device*. 2021. URL: https://invensense.tdk.com/wp-content/uploads/2022/11/DS-000481-IAM-20680HT-Rev.1.0-TYP.pdf.

[14] JetBrains. *Quick start guide*. URL: https://www.jetbrains.com/help/pycharm/quick-start-guide.html (sjekket 16. mai 2024).

[15] Juan R. Terven and Diana M. Cordova-Esparaza. *A COMPREHENSIVE REVIEW OF YOLO: FROM YOLOV1 TO YOLOV8 AND BEYOND*. 2023. URL: https://www.researchgate.net/publication/369760111_A_Comprehensive_Review_of_YOLO_From_YOLOv1_to_YOLOv8_and_Beyond.

[16] Knut A. Rosvold. *Kapslingsgrad*. URL: https://snl.no/kapslingsgrad (sjekket 26. apr. 2024).

[17] Knut A. Rosvold. *NEK 400*. URL: https://snl.no/NEK_400 (sjekket 16. mai 2024).

[18] Knut Hofstad. *International Electrotechnical Commission*. URL: https://snl.no/International_Electrotechnical_Commission (sjekket 26. apr. 2024).

[19] TP-link. *300Mbps Wireless N Router TL-WR841N*. 2020. URL: https://static.tp-link.com/res/down/doc/TL-WR841N_11.0.pdf.

[20] Maritime Robotic. *The Otter*. URL: https://www.maritimerobotics.com/otter (sjekket 19. mai 2024).

[21] Maritime Robotics. *The Otter*. URL: https://maritimerobotics.ams3.digitaloceanspaces.com/statamic/files/otter-brochure-2024-digital.pdf (sjekket 1. mai 2024).

[22] Marius Thaule. *Kurver i planet*. URL: https://wiki.math.ntnu.no/tma4105/tema/kurver (sjekket 16. mai 2024).

[23] Nærings- og fiskeridepartementet. *Maritim næring*. 2021. URL: https://www.regjeringen.no/no/tema/naringsliv/maritim-naring/ny-temaside/forste-kolonne/maritime-naringer/id2589227/.

[24] OpenCV. *cv::VideoCapture Class Reference*. URL: https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html#a57c0e81e83e60f36c83027dc2a188e80 (sjekket 2. mai 2024).

[25] OpenCV. *Drawing Functions*. URL: https://docs.opencv.org/4.x/d6/d6e/group__imgproc__draw.html (sjekket 1. mai 2024).

[26] OpenCV. *High-level GUI*. URL: https://docs.opencv.org/4.x/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563 (sjekket 2. mai 2024).

[27] OpenCV. *Image file reading and writing*. URL: https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56 (sjekket 2. mai 2024).

[28] OpenCV. *Structural Analysis and Shape Descriptors*. URL: https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga95f5b48d01abc7c2e0732db24689837b (sjekket 2. mai 2024).

[29] OpenCV Team. *opencv-python 4.9.0.80*. URL: https://pypi.org/project/opencv-python/ (sjekket 2. mai 2024).

[30] Ouster. *Accessories*. URL: https://static.ouster.dev/sensor-docs/hw_user_manual_OS1/hw_common_sections_OS1/Accessories.html (sjekket 18. mai 2024).

[31] Ouster. *Enhanced Ingress Protection Guide*. 2022. URL: https://data.ouster.io/downloads/application-guides/Enhanced-Ingress-Protection-Guide.pdf.

[32] Ouster. *Interface Box 24V Compatibility (Standard)*. URL: https://static.ouster.dev/sensor-docs/hw_user_manual_OS1/hw_common_sections_OS1/Accessories.html (sjekket 19. mai 2024).

[33] Ouster. *Lidar Packet Format Update*. 2020. URL: https://data.ouster.io/downloads/software-user-manual/lidar-packet-format-update.pdf.

[34] Ouster. *Module ouster.sdk.examples*. URL: https://static.ouster.dev/sdk-docs/python/api/examples.html#ouster.sdk.examples.client.record_pcap (sjekket 7. mai 2024).

[35] Ouster. *Mounting Guidelines*. URL: https://static.ouster.dev/sensor-docs/hw_user_manual_OS1/hw_common_sections_OS1/os1-mechanical-interface.html (sjekket 2. mai 2024).

[36] Ouster. *OS1 Hardware User Manual*. 2022. URL: https://data.ouster.io/downloads/hardware-user-manual/hardware-user-manual-rev05-os1.pdf.

[37] Ouster. *OS1 High Resolution Imaging LIDAR*. 2021. URL: https://www.datasheetinc.com/app/uploads/2019/10/Ouster-OS1-Datasheet.pdf.

[38] Ouster. *Ouster Data Layer*. URL: https://ouster.com/insights/blog/object-detection-and-tracking-using-deep-learning-and-ouster-python-sdk (sjekket 1. mai 2024).

[39] Ouster. *Software User Manual*. 2021. URL: https://data.ouster.io/downloads/software-user-manual/software-user-manual-v2.1.x.pdf.

[40] Ouster. *Thermal Integration Guide*. URL: https://static.ouster.dev/sensor-docs/image_route1/image_route2/thermal_int_guide/therm_int_guide.html (sjekket 2. mai 2024).

[41] Ouster. *Thermal Integration Guide*. URL: https://static.ouster.dev/sensor-docs/image_route1/image_route2/thermal_int_guide/therm_int_guide.html (sjekket 19. mai 2024).

[42] Ouster Sensor SDK Developers. *ouster-sdk 0.11.1*. URL: https://pypi.org/project/ouster-sdk/#description (sjekket 2. mai 2024).

[43] PrusaResearch. *Original Prusa MINI+ kit*. URL: https://www.prusa3d.com/en/page/prusaslicer_424/ (sjekket 2. mai 2024).

[44] Python. *struct — Interpret bytes as packed binary data¶*. URL: https://docs.python.org/3/library/struct.html (sjekket 18. mai 2024).

[45] ramswarup_kulhary. *What is OpenCV Library?* URL: https://www.geeksforgeeks.org/opencv-overview/ (sjekket 10. mai 2024).

[46] Roboflow. *Our Company.* URL: https://roboflow.com/about (sjekket 2. mai 2024).

[47] Tron Bårdgård. *TCP, UDP og porter.* URL: https://ndla.no/nb/subject:26f1cd12-4242-486d-be22-75c3750a52a2/topic:6e8a2eaf-4983-4d42-a9b0-911b5921b44a/resource:1aeca2b5-6233-401f-bd3f-7a6127afe9d5 (sjekket 16. mai 2024).

[48] Ultralytics. *Ultralytics YOLO.* URL: https://github.com/ultralytics/ultralytics (sjekket 28. apr. 2024).

[49] Xiang, Li and Wenhai, Wang and Lijun, Wu and Shuo, Chen and Xiaolin, Hu and Jun, Li and Jinhui, Tang and Jian, Yang. *Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection.* 2020. URL: https://arxiv.org/pdf/2006.04388.

[50] Xue Ying. «An Overview of Overfitting and its Solutions». I: *Journal of Physics: Conference Series* (2019). URL: https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022/pdf.

[51] Zhaohui, Zheng and Ping, Wang and Wei, Liu and Jinze, Li and Rongguang, Ye and Dongwei, Ren. *Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression.* 2019. URL: http://arxiv.org/pdf/1911.08287.

Vedlegg

A - Eksterne vedlegg

De følgende vedleggene finnes i den vedlagte vedleggmappen:

1. Python-prosjekt
2. Framdriftsrapporter
3. Møtedokumentasjon
4. Midtveispresentasjon

B - Plakat

NTNU

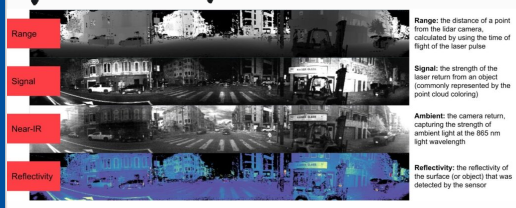
Institutt for IKT og realfag

Upgrading the Otter USV

Integrating a Ouster LiDAR with the Otter USV

This thesis integrates an Ouster LiDAR system onto the Otter USV. By combining the Otter's agility with LiDAR's advanced sensing, we provide the Otter with object detection capabilities. Our focus is specifically on identifying and measuring the distance to boats near the Otter USV, enhancing operator awareness.

The ouster LiDAR system operates by emitting laser pulses and measuring these laser pulses as they bounce back from the enviorment, it can measure range, signal, near-IR, and reflectivity. Utilizing this data, we can generate a 2D image that the YOLO algorithm can use for object detection.

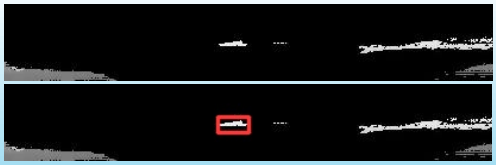


YOLOv8

OpenCV

Using range values and OpenCV we are able to generate a picture where objects on the water are clearly visible.

With the use of a custom trained YOLOv8 model we are able to detect boats within this picture.

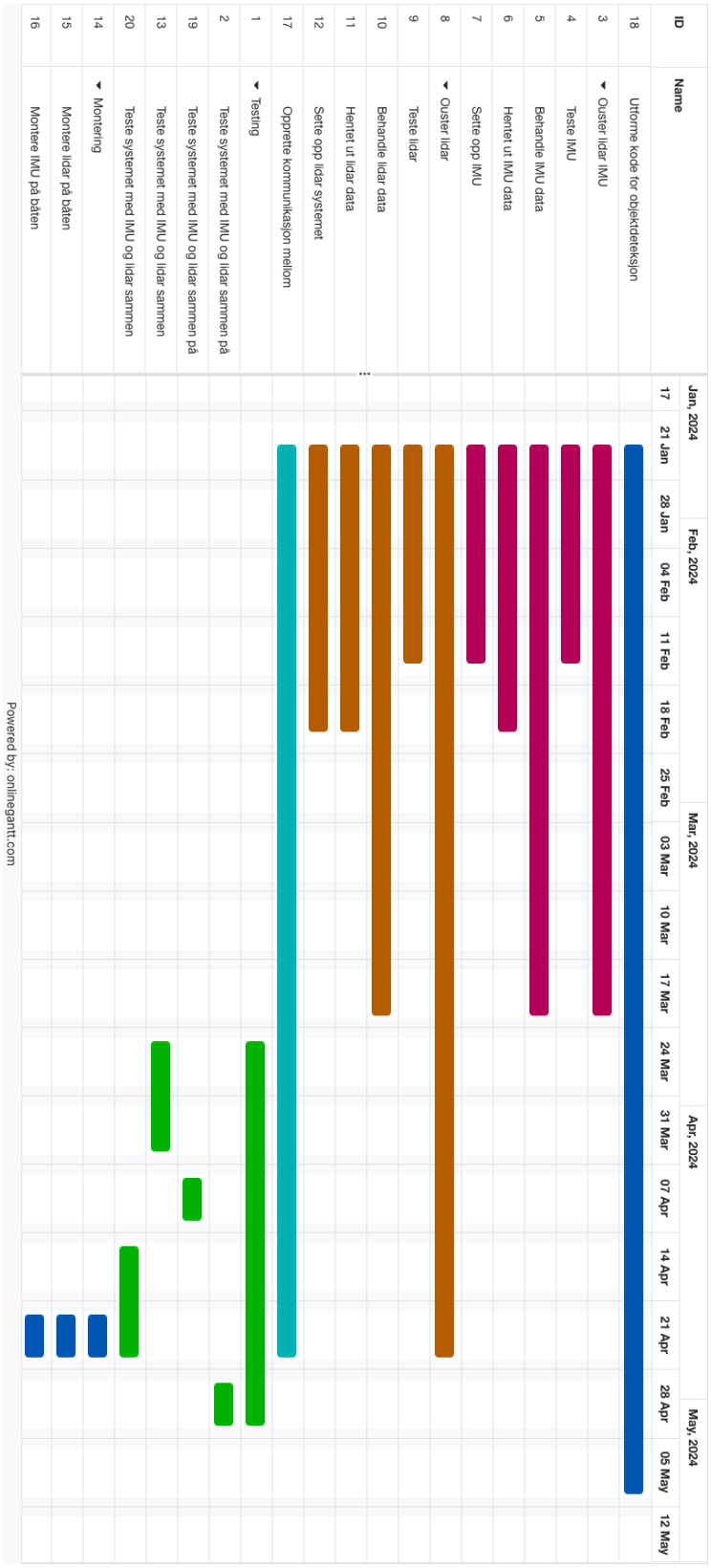


By integrating the Ouster LiDAR system onto the Otter USV and navigating it to a boat dock, we successfully detected multiple boats. Using the LiDAR's range measurements we can display the distance to the boats, providing useful information to the operator.

Ref: https://ouster.com/insights/blog/object-detection-and-tracking-using-deep-learning-and-ouster-python-sdk

Figur 58: Bachelorposter

C - Ganttdiagram



Figur 59: Ganttdiagram

D - Forprosjekt Rapport

NTNU
Kunnskap for en bedre verden

Institutt for IKT og realfag

AIS2900 - Bacheloroppgave ingeniørfag

Forprosjektrapport

Forfattere:
Joachim Dyrkolbotn
Henning Hestnes

25.01.2024

Table of Contents

List of Figures

List of Tables

1 Innledning

Bacheloroppgaven utføres av to studenter fra linjen automasjon og intelligente systemer. Formålet med oppgaven er å implementere og integrere diverse sensorer på NTNU sin uncrewed surface vessel (USV), også kjent som Otter USV. Dette skal gjennomføres for å forbedre funksjonaliteten til båten og gjøre det enklere å ta den i bruk til å samle og behandle relevant informasjon fra omgivelsene.

2 Begreper og forkortelser

2.1 Begreper

Otter Autonomous Navigation System (USV) - Dette er en ubemannet katamaran laget av Maritime Robotics. Katamaranen er utstyrt med diverse sensorer og navigasjonssystemer, noe som gjør den optimal for oppsamling av data ved kysten.

Light Detection and Ranging (LiDAR/lidar) - Dette er en type måleteknikk som bruker et reflektert lys til å måle avstanden til diverse objekter. Dette gjør det mulig å skape et 360 grader kart rund lidaren.

Inertial Measurement Unit (IMU) - Dette er en enhet som brukes til å måle akselerasjon og rotasjon av et objekt. Den gir informasjon om objektets bevegelse og orientering i rommet.

2.2 Forkortelser

Forkortelse	Beskrivelse
USV	Uncrewed surface vessel
LiDAR/lidar	Light Detection and Ranging
IMU	Inertial Measurement Unit

Table 1: Forkortelser

3 Prosjektorganisasjon

3.1 Prosjektgruppe

Gruppen består av 2 medlemmer fra Automatisering og intelligente systemer ved NTNU Ålesund. Stillingene til gruppemedlemmene kan sees i tabellen under:

Navn	Arbeidsoppgave
Henning Hestnes	Prosjektleder
Joachim Dyrkolbotn	Sekretær

Table 2: Forkortelser

Henning Hestnes vil være prosjektleder for dette prosjektet. Gruppemedlemmet har tidligere tilegnet seg ledererfaring fra Forsvaret. Denne erfaringen har gitt god selvdisiplin og struktur i arbeidet. Gruppemedlemmet har relevante erfaringer fra tidligere prosjekter fra studiet.

Joachim Dyrkolbotn fungerer som sekretær i gruppe og vil utfører de arbeidsoppgavene dette medfører. Tidligere har gruppemedlemmet fullført fagbrev som automatiker og jobbet som fagarbeider

1,5 år. Gruppemedlemmet har med dette praktisk erfaring innen fagfeltet som kan komme til nytte gjennom prosjektperioden.

Gruppens har en felles interesse for å arbeide med diverse sensorer og opprette kommunikasjon. Prosjektgruppen har samarbeidet i tidligere prosjekter gjennom studieløpet og vet med dette at vi er i stand til å jobbe bra i gruppe og flink til å fordele arbeidsoppgaver.

3.1.1 Oppgaver for prosjektgruppen

Prosjektgruppen skal sørge for at de utfører de arbeidsoppgavene som er nødvendig for å komme i mål med prosjektet. Prosjektgruppen vil også ha ansvar for å innkalle til møter, skrive statusrapporter og rappotere fremdriften i prosjektet.

3.1.2 Oppgaver for prosjektleder

Ansvaret for prosjektlederen er i hovedsak å holde generell oversikt over prosjektet og skaffe de resursene som trengs for gjennomføring av prosjektet. De vil fordele ansvarsoppgaver utover de forskjellige gruppemedlemmene og følge opp disse.

Arbeidsoppgaver prosjektlederen vil ha gjennom prosjektperioden er følgende:

- Påse at prosjektgruppen har de nødvendige resursene for å gjennomføre prosjektet samt å skaffe nye resurser ved behov.
- Fordele arbeidsoppgaver utover gruppemedlemmene får å forsikre god arbeidsflyt.
- Oppfølging av oppgavene i arbeidsgruppen.

3.1.3 Oppgaver for sekretær

Oppgaver som sekretæren vil ha gjennom prosjektoppgaven vil være:

- Planlegge og kordinere møter for prosjektet og inkludere alle de aktuelle personene på møtet. Dette innebærer også å finne et egnet møterom der alle de riktige resursene er tilgjengelig.
- Utarbeide møteagenda før de ukentlige møtene i samarbeid med resten av medlemmene i gruppen. På denne måten sikrer man en god struktur på møtene samt forsikrer seg at alle de nødvendige problemstillingene og spørsmålene blir tatt opp og diskuteres med veileder.
- Utarbeide møtereferat etter gjennomførte møter for å dokumentert hva som har kommet frem under møtet. Dette referatet vil inneholde hva som har blitt diskutert og hvilke løsninger/fremgangsmåter man har blitt enig om.
- Holde kontinuerlig kommunikasjon med veildere ved behov for møter utenom de allerede oppsatte møtene.

All dokumentasjon som blir utarbeidet av sekretæren gjennom prosjektperioden vil bli samlet i en egnet mappe. På denne måten er det enkelt å finne tilbake til og kan tas i bruk under prosjekt arbeid og rapportskriving.

3.2 Styringsgruppen

Styringsgruppen består av 3 ansatte ved NTNU. Disse skal fungere som veiledere og mentorer for oppgaven. Oppgaven deres gjennom prosjektperioden vil være å delta på jevnlige møter for diskusjon og bidra til problemstillinger som oppstår.

Styringsgruppens medlemmer består av:

Erlend Magnus Lervik Coates:

- Førsteamanuensis
- Institutt for IKT og realfag
- Fakultet for informasjonsteknologi og elektroteknikk

Lars Ivar Hatledal:

- Førsteamanuensis
- Institutt for IKT og realfag
- Fakultet for informasjonsteknologi og elektroteknikk

Eirik Strøm Fagerhaug:

- Stipendiat
- Institutt for IKT og realfag
- Institutt for teknisk kybernetikk

4 Avtaler

4.1 Avtaler med oppdragsgiver

Det er avtalt at det skal være møter med oppdragsgiver hver 14. dag. Disse møtene har primærtid onsdager 14:15 til 15:00, med forbehold om endringer på grunn av ulike årsaker. Møtene har blitt lagt inn i Microsoft Teams kalender, slik at alle skal ha oversikt over fremtidige møter.

I forkant av alle møter blir det sendt en framdriftsrapport. Framdriftrapporten består av all framdrift de siste 2 ukene, her blir det beskrevet framdrift fra sist møte. I denne framdriftrapporten blir det også skrevet foreslåtte fokusområder for neste periode. Framdriften i forrige periode og veien videre vil fungere som angenda for neste møte, alt utenom framdrift vil også bli beskrevet i møteagendaen.

I etterkant av alle møter blir det sendt ut et møtereferat. Dette referatet inneholder alle vesentlige elementer fra møtet. Dette dokumenterer hva som har blitt gjennomgått hvert møte, slik at alle har mulighet til å være oppdatert på prosjektet.

Det er også mulig med kontakt via e-post for ytteligere veiledning.

4.2 Arbeidssted og ressurser

4.2.1 Tilgang til arbeidsplass

Arbeidsplass vil i hovedsak være i labben og i garasjen der Otter USV står, itillegg vil det bli utført forsøk gjennom prosjektperioden som vil bli utført på en egnet plass . I labben vil det i hovedsak foregå testing og oppkobling av utstyret før det blir oppkoblet på Otter USV.

4.2.2 Tilgang til ressurser

Det er avtalt med veileder at utstyret som skal tas i bruk blir utdelt når vi har funnet en egnet plass og oppbevare dette. For tilgang til garasjen Otter USV er oppbevart i vil nøkklene bli utlevert ved å kontakte veileder når flere trenger tilgang til denne garasjen,

4.2.3 Tilgang til personer

Det er avtalt møte med veileder hver 14. dag for diskusjon av fremgangen i prosjektet og eventuelle problemer som har oppstått. Det er også avtalt at veiledere kan kontaktes på mail eller fysisk på kontoret ved behov utenom de oppsatte møtene.

4.2.4 Datasikkerhet/informasjon unndratt offentlighet

Prosjektet skal utføres for NTNU og det er med dette de som eier resurssene som tas i bruk gjennom prosjektperioden. Prosjektet inneholder ikke noe informasjon som skal unngås offentligheten.

4.2.5 Avtalt rapportering

Det vil på slutten av hver uke bli skreve en ukesrapport. For å klare å opprettholde dette vil det etter hver arbeidsdag noteres ned hva som er jobbet med og hvilke moteoder/teorier som er brukt. Disse notatene vil bli finskrevet om til en ukesrapport som igjen vil bli utgangspunktet når vi skal skrive rapporten til bachelor.

4.3 Gruppenormer - Sammerbeidsregler - Holdninger

Alle deltakere skal møte opp til avtalt tid, om dette ikke er mulig skal det gis beskjed i forkant. Gruppen ønsker å tilrettelegge timeplanen slik at gruppens medlemmer kan møtes og samarbeide.

Prosjektleder skal ta ansvar for at prosjektet har den progresjonen som trengs. Prosjektleder skal også sørge for at informasjonsflyten i gruppen er bra, og at alle er oppdatert på prosjektet.

Sekretæren skal dokumentere og organisere arbeidet som gjøres i prosjektet.

Alle i gruppens medlemmer har ansvar for at de selv arbeider tilstrekkelig med prosjektet og de her ansvarlige for å oppdatere resten av gruppen på progresjon.

Om en i gruppen ikke overholder disse punktene eller ikke fungerer i gruppen vil det bli gitt en muntlig beskjed. Om dette problemet fortsetter vil det bli gitt en muntlig advarsel. Om det fortsatt ikke blir bedre vil emneansvarlig bli kontaktet og gruppemedlemmet vil bli håndtert deretter.

5 Prosjektbeskrivelse

5.1 Problemstilling - Målsetting - Hensikt

NTNU Ålesund har i dag en Otter USV som blir brukt til å utføre diverse arbeidsoppgaver. Det er per dags dato bestilt inn diverse sensorer som ønskes å integreres på båten for å forbedre dens funksjonalitet og bruksområde.

Målsetningen til prosjektgruppen gjennom prosjektperioden vil derfor være systemintegrasjon av diverse sensorer på Otter USV. Bakgrunnen for dette arbeidet er å bruke disse sensorene til å øke funksjonaliteten og bruksområdet til båten.

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Resultatet av prosjektet som skal utføres er som nevnt tidligere å integrere sensorer på Otter USV for å forbedre dens funksjonalitet. Krav til en god løsning av prosjektet er at sensorene kompatibel med det nåværende systemet samt at de robust nok til båtens bruk.

Oppgaven regnes som ferdig når målene over er nådd og når rapporten er skrevet ferdig. Rapporten skal forklare hvordan alt er koblet sammen, hvordan den fungerer og hvorfor løsninger er valgt.

5.3 Planlagt framgangsmåte for utviklingsarbeidet – metoder

Fremgangsmåten for prosjektet blir bestemt av gruppens medlemmer i samarbeid med styringsgruppen. En av gruppens medlemmer vil sette seg inn i sensorene og lese seg opp på de. Parallelt med dette vil et annet gruppemedlem begynne å se på håndtering av kommunikasjon, det vil da bli undersøkt forskjellige biblioteker og sett på kommunikasjonsprotokoller for sensor data.

Etter denne fasen vil gruppen begynne med å sette opp sensorene inne på et bord, målet da er å få den til å fungere og forstå hvordan den fungerer. Deretter vil det bli forsøkt å hente og behandle dataen som sensorene produserer.

Når sensorene fungerer som tiltenkt vil det bli bygd et testoppsett. Dette består av en plate der sensorene er montert, en ruter, en raspberry Pi og en PC som håndterer data. Målet er da å simulere hvordan sensorene vil oppføre seg på båten og med dette finne ut hvilke tilltak som må gjøres for å få systemet til å virke som tiltenkt etter montering på otter USV. Det vil her bli eksperimentert med diverse sensorer for å stabilisere målingene før de sendes til en raspberry Pi som behandler dataen. Når dataen er behandlet vil den bli sendt til en PC over wifi. På denne PCen vil vi kunne se dataen som sensorene detekterer.

Deretter rettes fokuset med å sende denne dataen over wifi, dette er fordi Otter USV kommuniserer over wifi. Når kommunikasjonen er i orden vil det bli forsøkt å montere sensorene på Otter båten, og deretter teste systemet i sin helhet.

Et problem med denne framgangsmåten er hvis montering og behandling av sensorene og i hovedsak lidar blir alt for lett. Hvis dette viser seg å bli for lite arbeid vil fokuset rettes mot å bruke de integrerte sensorene til å forhindre at båten kolliderer med diverse hinder ute på sjøen.

Et annet problem er hvis det viser seg at vi ikke får de sensorene vi ønsker å montere til å fungere. Da vil fokuset bli rettet mot å finne ut hva som ikke fungerer og hvorfor den ikke fungerer og dokumentere dette på en bra måte. Fokuset vil videre bli rettet mot alternative sensorer som kan monteres på båten.

5.4 Informasjonsinnsamling – utført og planlagt

Vi har fått informasjon om ROS2 biblioteket som kan brukes for å behandle sensor data. Vi har også fått en introduksjon om Otter USV båten og dens komponenter.

Vi har sett på en masteroppgave som har brukt en tilsvarende lidar som vi har tilgjengelig. Vi har også tilgang til dokumentasjon fra Ouster som produserer lidaren, og Maritime Robotics som produserer Otter USV.

5.5 Vurdering – analyse av risiko

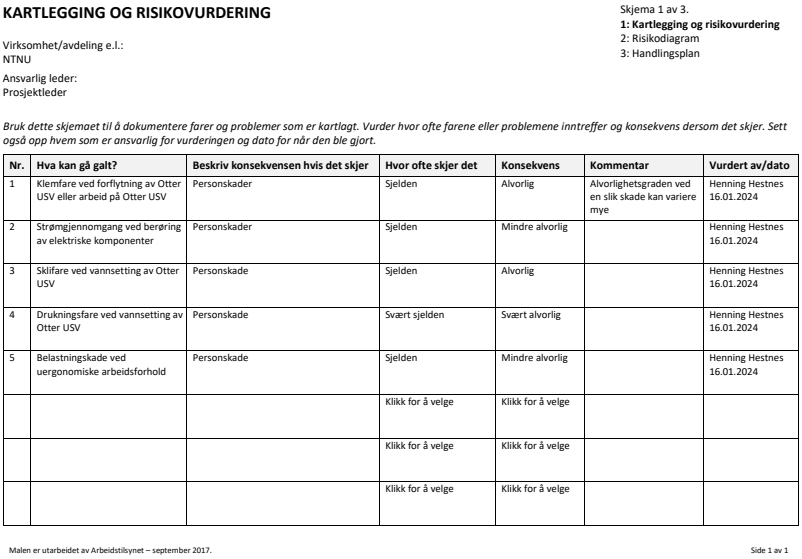
Det blir ansett som en risiko at prosjektet ikke blir fullført. Hvis prosjektets progresjon stagnerer vil det bli sett på alternative løsninger på problemene.

Det er ikke klart for gruppen om avgrensningen av oppgaven er tilstrekkelig. Det er uklart om arbeidsmengden i oppgaven er for mye eller for lite. Det skal lages prioriteringsliste over hva som

kan gjøres med Otter båten, der lidar er øverste prioritet. Hvis oppgaven blir ferdig med mye ekstra tid vil gruppen gå videre til neste punkt på prioriteringslisten, som vil gjøre løsningen og oppgaven enda bedre.

For at gruppen skal lykkes er det ekstremt viktig at alle i gruppen tar ansvar og arbeider effektivt med oppgaven. Hvis progresjonen går for sakte eller stopper opp er det avgjørende å ta et tidlig valg om man skal fortsette å prøve eller se på alternative løsninger.

kartlegging og risikovurdering kan bli sett i figure 1, risikodiagram i figure 2 og handlingsplan i 3.



KARTLEGGING OG RISIKOVURDERING

Skjema 1 av 3.
1: Kartlegging og risikovurdering
2: Risikodiagram
3: Handlingsplan

Virksomhet/avdeling e.l.:
NTNU

Ansvarlig leder:
Prosjektleder

Bruk dette skjemaet til å dokumentere farer og problemer som er kartlagt. Vurder hvor ofte farene eller problemene inntreffer og konsekvens dersom det skjer. Sett også opp hvem som er ansvarlig for vurderingen og dato for når den ble gjort.

Nr.	Hva kan gå galt?	Beskriv konsekvensen hvis det skjer	Hvor ofte skjer det	Konsekvens	Kommentar	Vurdert av/dato
1	Klemfare ved forflytning av Otter USV eller arbeid på Otter USV	Personskader	Sjelden	Alvorlig	Alvorlighetsgraden ved en slik skade kan variere mye	Henning Hestnes 16.01.2024
2	Strømgjennomgang ved berøring av elektriske komponenter	Personskader	Sjelden	Mindre alvorlig		Henning Hestnes 16.01.2024
3	Sklifare ved vannsetting av Otter USV	Personskade	Sjelden	Alvorlig		Henning Hestnes 16.01.2024
4	Drukningsfare ved vannsetting av Otter USV	Personskade	Svært sjelden	Svært alvorlig		Henning Hestnes 16.01.2024
5	Belastningskade ved uergonomiske arbeidsforhold	Personskade	Sjelden	Mindre alvorlig		Henning Hestnes 16.01.2024
			Klikk for å velge	Klikk for å velge		
			Klikk for å velge	Klikk for å velge		
			Klikk for å velge	Klikk for å velge		

Malen er utarbeidet av Arbeidstilsynet – september 2017.

Side 1 av 1

Figure 1: Kartlegging og risikovurdering

I risikodiagrammet 2 under er det vist hvor sannsynlig det er for at de forskjellige hendelsene oppstår og hvor alvorlig de er.

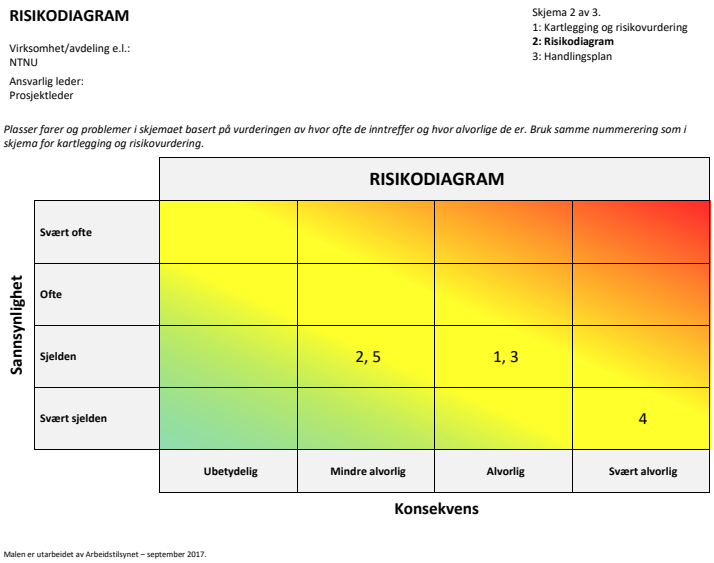


Figure 2: Risikodiagram

I figur 3 under er det listet opp hvilke tilltak som skal gjøres for å unngå de risikoene som er kartlagt i arbeidet. Tilltakene er prioritert utifra skaden som kan oppstå og hvor sansynlig det er at den oppstår.

HANDLINGSPLAN

Skjema 3 av 3.
1: Kartlegging og risikovurdering
2: Risikodiagram
3: Handlingsplan

Virksomhet/avdeling e.l.:
NTNU
Ansvarlig leder:
Prosjektleder

Dokumenter tiltak for å redusere risikoen. Bruk samme nummerering som i skjema for kartlegging og risikovurdering og risikodiagrammet. Farer eller problemer som inntreffer ofte/svært ofte med en alvorlig/svært alvolig konsekvens må prioriteres først.

Nr.	Kort beskrivelse av faren/problemet	Prioritering	Tiltak for å redusere risikoen	Ansvarlig(e)	Tidsfrist
1	Klemfare ved forflytning av Otter USV eller arbeid på Otter USV	2	God kommunikasjon ved forflytning av Otter USV og god kommunikasjon før forflytning. Påse at båten står stabilt på en flat overflate	Prosjektleder	
2	Strømgjennomgang ved berøring av elektriske komponenter	4	Benytte spenningsmåler for å teste komponenter, forsikre at anlegget er spenningsløst for arbeidet begynner	Prosjektleder	
3	Sklifare ved vannsetting av Otter USV	5	Vurdere forholdene før man vannsetter båten. Avslutte arbeidet hvis forholdene tilsier det.	Prosjektleder	
4	Drukningsfare ved vannsetting av Otter USV	1	Ha en person klar med redningsbøye som har oversikt over vannsettingen.	Prosjektleder	
5	Belastningskade ved uergonomiske arbeidsforhold	3	Organiser arbeidet slik at man unngår tidspress og monotont arbeid. Skaff gode hjelpemidler og arbeidsutstyr som passer til oppgaven.	Prosjektleder	

Malen er utarbeidet av Arbeidstilsynet – september 2017.

Side 1 av 1

Figure 3: Handlingsplan

5.6 Hovedaktiviteter i videre arbeid

Hovedaktivitetene gjennom prosjektperioden er listet opp i fremdriftsplanen i seksjon 5.7 under og vil bli fordelt med hovedansvar ut over prosjektgruppen. Dette er hovedoppgavene gjennom prosjektperioden slik prosjektgruppen ser det nå.

5.7 Framdriftsplan – styring av prosjektet

Framdriftsplan er laget i Microsoft teams ved hjelp av Microsoft planner.

Oppgavenavn	Navn på samling	Fremdrift	Prioritert	Tilordnet til	Opprettingsdato	Startdato	Forfallsdato	Er regelmessig	Forsinket	Fullføringsdato	Fullført av	Fullførte sjekkliste-elementer	Sjekkliste-elementer	Etiketter	Beskrivelse
Møte med styringsgruppen	Møter	Ikke startet	Medium	Joachim Halvorsen Dyrkolbotn;Henning Hestnes	16.01.2024	24.01.2024	24.01.2024	true	false			0/1	Skrive møte referat	Rosa	
Teste systemet med IMU og lidar sammen på båt	Testing på båt	Ikke startet	Medium		16.01.2024			false	false					Lys blå	
Montere IMU på båten	Montering på båt	Ikke startet	Medium		16.01.2024			false	false					Lilla	
Montere lidar på båten	Montering på båt	Ikke startet	Medium		16.01.2024			false	false					Lilla	
Teste systemet med IMU og lidar sammen	Teste med testoppsett	Ikke startet	Medium		16.01.2024			false	false			0/1	Bygge test oppsett	Rød	
Opprette kommunikasjon mellom komponenter	Kommunikasjon	Ikke startet	Medium		16.01.2024			false	false			0/1	Undersøke aktuelle kommunikasjonsprotokoller	Gul	
Teste systemet	IMU	Ikke startet	Medium		16.01.2024			false	false					Blå	
Behandle data	IMU	Ikke startet	Medium		16.01.2024			false	false					Blå	
Hentet ut data	IMU	Ikke startet	Medium		16.01.2024			false	false					Blå	
Sette opp IMU	IMU	Ikke startet	Medium	Joachim Halvorsen Dyrkolbotn	16.01.2024	01.02.2024		false	false					Blå	
Teste systemet	Lidar	Ikke startet	Medium		16.01.2024			false	false					Grønn	
Behandle data	Lidar	Ikke startet	Medium		16.01.2024			false	false			0/1	Bli kjent med ROS2 biblioteket	Grønn	
Hentet ut data	Lidar	Ikke startet	Medium		16.01.2024			false	false					Grønn	
Sette opp lidar systemet	Lidar	Ikke startet	Medium	Henning Hestnes	16.01.2024	01.02.2024		false	false					Grønn	

Figure 4: Framdriftsplan

I figur 4 ser vi framdriftsplanen per i dag, denne vil bli justert og oppdatert etterhvert som oppgavens omfang blir kartlagt.

5.7.1 Hovedplan

Hovedplanen for prosjektet blir beskrevet i framdriftsplanen i figur 4. Det består per dags dato av å bli kjent med lidar og IMU systemet. Deretter skal disse komponentene integreres med en Otter USV. Her er det forbehold om endringer der eventuelt andre sensorer kan bli valgt å blir integrert på Otter USV.

5.7.2 Styringshjelpemidler

Styringshjelpemidler som tas i bruk for å opprettholde god struktur på tekniske data og framdrift av prosjektet vil være.

- Teams/Wiki: Dette vil være en felles side for alle deltagerene i prosjektet. Her vil man strukturere siden etter behov og samle blant annet teknisk data, timeplaner og rolletildeling.
- Overleaf: Dette vil i utgangspunktet bli brukt av prosjektgruppen og er her rapporter vil bli skrevet. Styringsgruppen vil ha tilgang til de dokumentene de ønsker.
- Microsoft planner: Dette er verktøyet som er bruk til å lage framdriftsplan. Denne planen vil bli brukt til å vise framgang i de forskjellige deloppgavene prosjektet består av.

5.7.3 Intern kontroll – evaluering

Det er opprettet en framdriftsplan som inneholder sentrale oppgaver som skal gjennomgås i prosjektet. Denne planen vil bli brukt av alle medlemmene i prosjektplanen til å oppdatere framdriften på deres ansvarsområder. Dette vil gjøres ved å angi hvor nært utført oppgaven er i en prosentandel samt å legge ved kommentarer som gjør det lett for øvrige medlemmer å forstå hva som er blitt gjort. Det vil som nevnt tidligere også bli skreve ukesrapport der hvert medlem må forklare hva de har gjort på sitt ansvarsområdet. Dette er en lett måte for de andre medlemmene og styringsgruppen å få en forståelse for framdriften av prosjektet.

5.7.4 Beslutninger – beslutningsprosess

Det har i oppstartmøte med styringsgruppen blitt diskutert hvordan det skal settes sammen en realistisk oppgave. Her har det blitt satt sammen en oppgave som kan utføres innen tidsrammene for prosjektperioden. Oppgaven er også satt sammen av temar som innteriserer prosjektgruppen.

Avgjørelser som må tas under prosjektoppgaven og har betydning for videre arbeid av prosjektet vil bli diskutert innad i gruppen. Når gruppen bare består av to personer og en av disse er prosjektlede vil slike diskusjoner på en saklig måte. Hvis gruppen ikke kommer til enighet vil det bli tatt opp i møte med styregruppen for å få innspill om hvilke fremgangsmåte som de mener vil gi den beste løsningen. Hvis det ikke styregruppen har noe spesielle meninger om dette vil prosjektlede ha siste ordet.

6 Dokumentasjon

6.1 Rapporter og tekniske dokumenter

Det vil i utgangspunktet bli brukt overleaf for rapportskriving og møtereferat. Hvis ønskelig vil styringsgruppen få tilgang til disse dokumentene fra starten. Det vil også bli opprettet en team/wiki

side som vil bli brukt til å samle nødvendige data på en egnet plass.

7 Planlagte møter og rapporter

7.1 Møter

Gjennom prosjektperioden vil det være aktuelt faste møter med styringsgruppen for å forsikre god framdrift i prosjektet og for å diskutere problemstillinger som eventuelt oppstår. Det kan også bli aktuelt med møter med andre personer som sitter med kunnskap vi kan få bruk for i prosjektet, disse møtene vi i tilfelle bli avtalt ved behov.

Før alle møtene uavhengig hvem som er deltagere vil prosjektgruppen utarbeide en møteagenda for å for å forsikre effektivitet i møtene og at de nødvendige temaene blir tatt opp og diskutert.

7.1.1 Møter med styringsgruppen

Det er avtalt et fast møtetidspunkt med styringsgruppen annenhver uke. Disse møtene vil bli brukt på å snakke om hva som er jobbet med de to foregående ukene, hvilke metoder som er testet og eventuelle problemstillinger som har oppstått.

Styringsgruppen vil komme med innspill til til arbeidet som er lagt ned og komme med eventuelle ideer eller rettelser for å hjelpe prosjektgruppen på vei. De har også muligheter til å ta opp temaer de mener er aktuelt.

Det vil til slutt bli diskutert med styringsgruppen hva som blir jobbet med de neste ukene.

7.1.2 Prosjektmøter

Det er ikke avtalt noen faste møter innad i prosjektgruppen når denne gruppen bare består av to personer. For å likevell klare å holde god framdrift i prosjektet vil vi opprettholde god kommunikasjon mellom medlemmene. Vi har også blitt enig om å ha kjernetid fra 08:15 - 16:00, ved unntak fra denne tiden må det bli gitt beskjed til de aktuelle. Denne vil gjelde torsdag og fredag frem til mars når vi har andre fag de tre resterende dagene. Etter avlagt eksamen i andre fag vil denne tiden gjelde hele uken.

7.2 Periodisk rapporter

Det vil som nevnt tidligere bli skreve notater av fremdriften etter hver arbeidsdag som på slutten av uken vil bli finskrevet til en ukesrapport. Det vil under skrivingen av rapporten bli tatt utgangspunkt i disse ukesrapportene for å forsikre korrekt og bra informasjon i sluttrapporten. Innholdet i disse ukesrapportene vil bli utgangspunktet i møte med styregruppen for å legge fram hva som er jobbet med den foregående uken.

7.2.1 Framdriftsrapport

De ukentlige rapportene vil bli brukt til å kartlegge framdrift i prosjektet. I tillegg vil oppgave fanen i Microsoft planner gruppen bli brukt til å vise progresjon.

8 Planlagt avviksbehandling

Hvis det gjennom prosjektperioden oppstår at fremdriften av prosjektet ikke går som planlagt vil dette bli tatt tak i så raskt som mulig. Det vil i en slik situasjon i førsteomgang bli gjort tilltak for å rette opp i fremdriften, dette vil være tilltak som:

- Diskutere problemene som har oppstått innad i gruppen og finne ut grunnen til at de har oppstått. Utifra hva man kommer frem til her vil tilltak bli gjort.
- Ved behov vil styringsgruppen kontaktes for å få hjelp til å komme tilbake på riktig spor.
- Hvis der er at prosjektgruppen sitter fast med et spesifikk problem over lengre tid vil vi prøve å komme i kontakt med personer som sitter med mer informasjon om problemet.

Hvis tilltakene over ikke rekker til for å få god fremdrift i prosjektet vil mer drastiske tilltak gjøres.

Det er i oppstartmøtet med styringsgruppen diskutert en rekke forskjellige oppgaver som kunne vært aktuellt som bachelor oppgave. Hvis det oppstår en situasjonen der prosjektgruppen ikke kommer seg videre i arbeidet vil disse oppgavene bli diskutert. Det vil i et slikt tilfelle bli aktuelt å dokumentere alt arbeidet som er gjort med de integrerte sensorene for å så bytte til å forsette arbeidet med å montere alternative sensorer på Otter USV'en.

Prosedyren i et slik tilfelle vil i korte trekk være:

- Dokumentere og ferdigstille arbeidet med integrerte sensorer så godt det lar seg gjøre for å vise hva som er gjort og hvor problemene oppsto.
- Diskutere med styringsgruppen hva som er realistiske mål med tanke på tiden som er igjen av prosjektperioden.
- Begynne med arbeidet med de aktuelle sensorene.

Det vil i starten av prosjektarbeidet bli diskutert ansvarsfordelingen slik at hvert gruppemedlem har hovedansvar for en del av prosjektet. Hvis fremdriften av en del i prosjektet ikke går som planlagt må personen med hovedansvaret for denne delen ta ansvar for å informere om dette i de faste møtene med styringsgruppen. Dette må informeres om i god tid slik at det ikke oppstår en situasjon der det er for seint å gjøre noe med problemet. På denne måten får man diskutert problemet og ta en beslutning om man bår gjøre endringer i oppgaven for å komme i mål.

9 Utstyrsbehov/Forutsetninger for gjennomføring

Når detter er videreutvikling av et allerede eksisterende prosjekt som ønskes å videreutvikle er de fleste resursene allerede tilgjengelig. Programvare og utstyret er allerede tatt i bruk og prosjektgruppen vil få tilgang til dette og en kort innføring i dette av personer som tidligere har tatt dette i bruk.

Utstyr og programmvarer som skal tas i bruk er følgende, med forbehold om ytterlig behov om nødvendig.

- Diverse sensorer
- Otter USV
- Raspberry Pi
- ROS2
- Python
- PC
- IMU

10 Referanser

10.1 Vedlegg