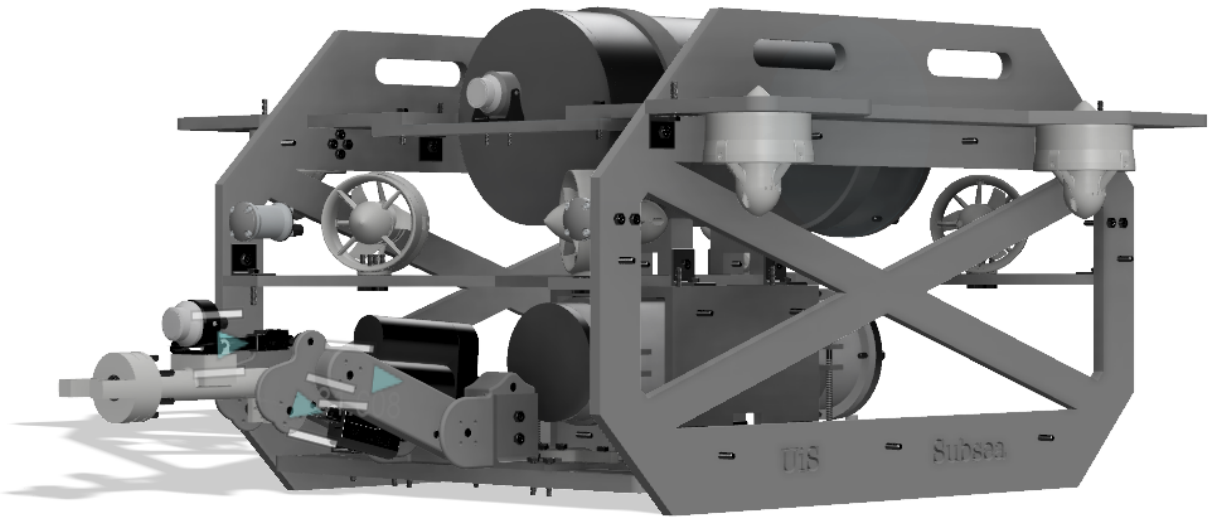
ALEKSANDER MØLLERDU OG BJØRN MAGNUS MYRHAUG
INSTITUTT FOR DATA- OG ELEKTROTEKNOLOGI

Autonom kjøring av ROV i samarbeid med UiS Subsea

Bacheloroppgave - Elektroteknologi - Mai 2024



Innhold

Ordliste

- **ROV:**
  Remotely Operated Vehicle
- **AUV:**
  Autonomous Underwater Vehicle
- **Kamera-regulering:** PID-regulering med posisjonsdata fra bildebehandling som referanse.
- **DVL-regulering:** PID-regulering med posisjonsdata fra DVL-sensor som referanse.
- **PMP:** Planar Move Plugin, programvareutvidelse for gravitasjonsløs bevegelse av ROV i simulator.

Sammendrag

Denne bacheloroppgaven går ut på å utvikle autonom kjøring av et fjernstyrt undervannsfartøy. Farttøyet utvikles av diverse arbeidsgrupper i studentorganisasjonen UiS Subsea. Målet med ROV-en er å bruke denne i den lokale konkurransen *TAC Challenge 2024*, for å samle så mange poeng som mulig i henhold til konkurransemanualen. I manualen finnes fire forskjellige oppdrag. I denne oppgaven utvikles det autonom oppførsel for følgende oppdrag: Rørledningsinspeksjon og Visuell inspeksjon av arbeidsbenk.

Testing av programvare krever en ferdigstilt ROV, da dette ikke var tilgjengelig i løpet av oppgaven, brukes heller simulering som løsning for testing av programvare. Simulering, og utviklingen av simuleringsmiljøet, har blitt en stor del av denne bacheloroppgaven. Simulatoren har vist seg å være svært nyttig som testplattform da ROV-en ikke var klar for testing til planlagt frist.

Valg av fremgangsmåte har i stor grad basert seg på hvilke sensorer som er tilgjengelige på ROV-en. For utføre inspeksjon av arbeidsbenk og rørledning brukes sensordata fra bunnkamera, frontkamera og fra en DVL-sensor.

DVL-sensoren gir ut posisjons- og rotasjonsdata fra et startpunkt. Ved bruk av trigonometriske funksjoner kan sensoren brukes til å sette målposisjoner relativt til ROV-en selv, og deretter nå disse målposisjonene. Ved å utføre bildebehandling på bildestrømmen fra kameraene, lokaliseres objekter slik at ROV-en kan regulere sin posisjon i forhold til de.

Posisjonsdataen fra sensorene anvendes med PID-regulatorer for å regulere posisjonen til ROV-en. Ved inspeksjon av rørledning reguleres frihetsgradene svai og gir, og for inspeksjon av arbeidsbenk reguleres gir, svai og jag. Det har blitt implementert overføringsfunksjoner for ROV-ens bevegelser i simulatoren. Disse overføringsfunksjonene legger til tidsforsinkelser, som skaper et mer realistisk grunnlag for justering av PID-parametere i simuleringen.

ROV-en i simulatoren er optimalisert og yter gode testresultater for regulering og bildebehandling. Bildebehandlingen gir også gode resultater under testing med bildestrømmen fra *TAC Challenge 2023*.

Mot slutten av rapporten diskuteres forbedringer, alternative løsninger og fremtidige planer for ROV-en.

Forord

Først og fremst vil vi takke årets medlemmer av UiS Subsea for samarbeidet igjennom hele bacheloroppgaven. Prosjektarbeidet har vært en meget lærerik prosess, både på godt og vondt.
En spesiell takk går ut til Ingvild Bortheim som har hatt rollen som teknisk leder for elektro. Hun har bidratt som støttespiller, og med påfyll av kaffekannen igjennom hele bacheloren.

En takk går ut til Universitetet i Stavanger, som har bidratt med tilgang til maskinverkstedet, labben og ekstra dataskjermer. En stor takk sendes også ut til alle bedriftene som har bistått med økonomisk støtte, materialer og utstyr.

Sist men ikke minst vil vi takke Damiano Rotondo og Didrik Efjestad Fjereide, som har vært veileder og biveileder for bacheloren vår. De har bidratt med fantastisk veiledning og vist stort engasjement for oppaven vår. Vi setter stor pris på all tiden dere har satt av for å hjelpe med prosjektet vårt.

Kapittel 1

Introduksjon

Innhold

Dette kapittelet introduserer prosjektet og organisasjonen UiS Subsea. Det blir presenter en funksjonsbeskrivelse av oppgaven og en beskrivelse av oppgaven gitt av konkurransen som UiS Subsea skal delta i. Delkapittel 1.1 til 1.3 er en fellesintro skrevet av studentorganisasjonen UiS Subsea.

1.1 Innledning

Denne bacheloroppgaven er en del av et større prosjekt i samarbeid med UiS Subsea. Introkapittelet er skrevet i fellesskap av prosjektteamet og vil dermed være likt for alle gruppene. UiS Subsea designer og utvikler hvert år en *Remotely Operated Vehicle* (ROV), en fjernstyrt undervannsfarkost, med mål om å delta i konkurranser. De tidligere årene er det MATE ROV Competition i USA som har vært hovedfokus, og TAC Challenge noe som var i siste liten, og som var kjekt å være med på. I år er fokuset på TAC Challenge og ambisjonene er å ikke gjøre det dårligere i år enn i fjor, da UiS Subsea vant konkurransen med dronen *Yme*. Lista ligger dermed høyt for årets team.

Dette kapittelet skal gi en introduksjon av studentorganisasjonen UiS Subsea, innsikt i TAC-konkurransen, og en presentasjon av årets ROV med de ulike systemene og gruppene innenfor prosjektet.

1.2 Om UiS Subsea

UiS Subsea (logo i Figur 1.1) er en studentorganisasjon som ble opprettet i 2013/14 og har hatt som mål siden å motivere studenter til å delta i større prosjekt og å knytte nærmere bånd med næringslivet. Stavanger har et stort næringsliv innenfor olje og gass og her er det mye som foregår både on- og offshore.

Siden UiS Subsea først og fremst er et bachelorprosjekt utvikles det hvert år en ny ROV av årets team, eller en allerede eksisterende modell forbedres. I slutten av prosjektene deltar årets prosjekt i en konkurranse for å teste hvor bra ROV-en som er produsert er, i tillegg til å knytte bekjentskap til andre universitet og organisasjoner både nasjonalt og internasjonalt. Årets team som skal utvikle ROV-en består av 27 studenter fordelt på 10 bacheloroppgaver og grupper: 7 elektro-, 2 data- og 1 maskiningeniørgruppe.

Figur 1.1: UiS Subseas logo.

1.2.1 Litt om TAC Challenge

Tau Autonomy Center (TAC), med beliggenhet litt utenfor Stavanger, arrangerer en årlig AUV/ROV konkurranse for både norske og utenlandske studenter. Formålet er å fremme nyskaping og innovasjon relatert til undervannsteknologi for ubemannede fartøy. Ved å utfordre tverrfaglige grupper av studenter til å utføre både selvstyrte og autonome oppgaver i et industrielt undervannsmiljø får studenter muligheten til å ikke bare vise frem egen lærdom, men også tilegne seg et prosjektarbeid som er relatert til arbeidslivet. Det arrangeres også sosiale tilstelninger for alle konkurransetakere som gjør det mulig å bli kjent med folk fra hele verden. Dette er også med på å fremme sosiale og faglige bånd mellom fremtidige ingeniører og organisasjoner relatert til autonome og fjernstyrte undervannsteknologier [**rulesregulationsTAC2024**].

Utfordringene til studentgruppene som blir gitt av TAC Challenge er dermed å planlegge, designe, konstruere og til slutt utvikle fullt funksjonelle ROV-er som deretter skal konkurre med hverandre i utføringen av spesifikke oppgaver. Fokuset på oppgavene er gjenskaping eller tilnærming av realistiske problemstillinger fra næringslivet, og er utviklet av TAC i samarbeid med sponsorer eller andre bidragsytere. Selve konkurransen blir avholdt ved TAC og varer i 5 dager, fra 10. til 14. juni. Over denne perioden foregår alt fra sosiale aktiviteter og testing til selve konkurranseinnholdet [**rulesregulationsTAC2024**]. Grunnet den nære beliggenheten og det at UiS Subsea i tillegg vant fjorårets konkurranse, så har det vært god erfaring med TAC Challenge. På bakgrunn av dette ble det valgt å bare fokusere på TAC Challenge dette året som eneste konkurranse for ROV-en.

1.2.2 Årets konkurranse

Detaljer om konkurranseoppgavene

Konkurransen består av ulike oppgaver som er fordelt inn i kategoriene *statiske* og *dynamiske* oppgaver. For de statiske oppgavene må alle lagene avholde en *gruppepresentasjon* og levere en *teknisk designrapport*, som blir evaluert av en jury innen fagfeltet. Den dynamiske delen av oppgavene består av 4 forskjellige oppdrag, hvor fokuset på evalueringen ligger på resultatet og utførelsesmetoden [**rulesregulationsTAC2024**].

1. **Subsea docking og dataoverføring**
   (Oversatt fra TAC Rules and Regulations 2024 rev 1, §7.1.1.)
   En dockingstasjon er posisjonert i et innendørs treningsbasseng i TAC-bygningen. Fartøyet skal lokalisere dockingstasjonen og forsøke å docke. Dockingstasjonen er utstyrt med en induktiv 250W Subsea Power Puck, som kan overføre data og kraft til fartøyet. ROV-en skal demonstrere kraftoverføring ved å ha lys som skrus på som er koblet til power pucken montert på ROV-en og samtidig etablere kommunikasjon til docking-

stasjonen. Dockingstasjonen er også utstyrt med forhåndsdefinerte visuelle indikatorer (ArUco merker). Spesifikk autonom oppførsel kan også gi poeng.

2. **Rørledningsinspeksjon**
   (Oversatt fra TAC Rules and Regulations 2024 rev 1, §7.1.2.)
   En rørledning er posisjonert på havbunnen innenfor operasjonsområdet og behøver inspeksjon. Rørledningen har en ukjent bane og lengde, og det er også et ukjent antall ArUco merker langs rørledningen som skal identifiseres. Levering av merkekoden i rett rekkefølge trengs for å oppnå makspoeng, samt spesifikk autonom oppførsel kan også gi poeng.

3. **Visuell inspeksjon av undervannsstruktur**
   (Oversatt fra TAC Rules and Regulations 2024 rev 1, §7.1.3.)
   En undervannsstruktur er posisjonert på havbunnen innenfor operasjonsområdet og behøver inspeksjon. Forskjellige ArUco merker er posisjonert på selve strukturen og må bli gjenkjent for å oppnå poeng. Spesifikk autonom oppførsel kan også gi poeng.

4. **Ventilintervensjon**
   (Oversatt fra TAC Rules and Regulations 2024 rev 1, §7.1.4.)
   En undervannsstruktur er posisjonert på havbunnen innenfor operasjonsområdet og behøver inspeksjon. Forskjellige ArUco merker er posisjonert på selve strukturen og må bli gjenkjent for å oppnå poeng. Spesifikk autonom oppførsel kan også gi poeng.

Poengfordeling

1. **Tekniske krav**
   ROV-en må også være kompatibel med TAC Challenge sitt Launch And Recovery System (LARS), hvor dronen kan bli festet til en karabinkrok eller fatle. Dronen kan heller ikke veie mer enn 100 kg, og lavere vektklasser tildeles ekstrapoeng i den statiske delen. I tillegg er det en maksbegrensning på 60V fra eventuelle batterisystemer om bord samt 50V begrensning til kraftkabelen fra land. Det er også et krav til positiv oppdrift. Disse kravene blir sjekket via den tekniske rapporten og gruppepresentasjonen i tillegg til at TAC har muligheten til å foreta inspeksjoner før konkurransen.

2. **Statiske oppgaver**
   Teknisk rapport (opp til 100 poeng)
   Hver gruppe i konkurransen må produsere en egen teknisk rapport som utgreier konkurransestrategi, sikkerhetstiltak, systemarkitektur, testing og validering. Disse punktene blir dermed evaluert basert på teknisk dybde og innovative løsninger på de tekniske utfordringene i konkurransen. I tillegg blir organiseringen og den litterære presentasjonen av rapporten også tatt til etterretning i den endelige poenggivningen.
   Gruppepresentasjon (opp til 100 poeng)
   Gruppene må også presentere ROV designet, utviklingsprosessen og konkurransestrategien til dommerne. Tilsvarende så er det en utdyping av disse punktene som dermed blir evaluert av dommerne basert på strategi og ingeniørfaglige hensyn.

3. **Dynamiske oppgaver (oppdragene)**
   De tidligere nevnte oppdragene blir evaluert basert på gjennomføring og strategi. Det kan som nevnt også gis poeng for både full eller delvis autonom gjennomføring.

Hele poengfordeling for både de statiske og dynamiske oppgavene er oppsummert i Tabell 1.1 nedenfor [**rulesregulationsTAC2024**]. Disse poengene fordeles av dommerpanelet ved utføringen av de statiske og dynamiske oppgavene i løpet av konkurransedagene.

Konkurransescoring			
Statiske oppgaver	Standardpoeng	Bonuspoeng	Totalt
Teknisk dokumentasjon	100	Ingen bonuspoeng	100
Gruppepresentasjon	100	Ingen bonuspoeng	100
Dynamiske oppgaver (oppdrag)	Standardpoeng	Bonuspoeng	Totalt
Docking	120	100	220
Rørledningsinspeksjon	100[1]	300[1]	400[1]
Visuell inspeksjon	120[1]	120[1]	240[1]
Ventilintervensjon	100	200	300

Tabell 1.1: Oversikt over poengfordelingen ved TAC. Tabellen er oversatt fra *TAC Rules and Regulations 2024 rev 1, §2.4.*

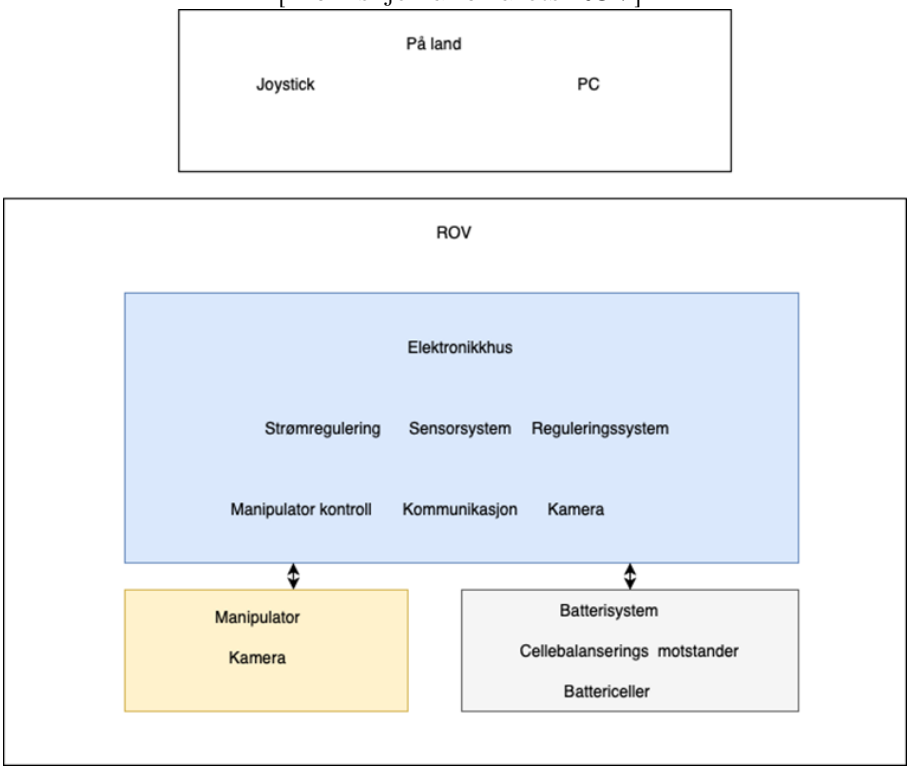
1.3 Årets ROV: Draugen

ROV-en som utvikles av årets team har fått navnet *Draugen*. Blokkskjema og design for ROV-en er vist i Figur 1.2.

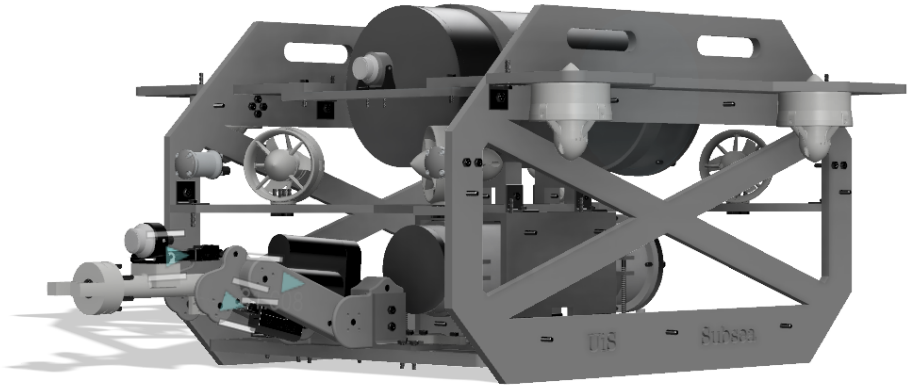
Fokuset for årets ROV design var stabilitet, hydrodynamiske egenskaper og miljøvennlige produksjons- og maskinerings metoder. Ettersom Draugen blir den første ROVen med montert batteri, krevde dette ekstra plass og en noe større ROV en fjorårets. Dette førte igjen til en tyngre ROV, som var en bekymring i begynnelsen av prosjektet. Men dette viste seg å ikke være et problem i prosjektets sluttfase, da ROV ble utrolig stabil, både på land og i vann.

[1] Eksakte verdier er ikke oppgitt grunnet et ukjent antall markører som skal identifiseres i oppgaven. Se vedlagt konkurransemanual og regelheftet for mer detaljer [**missionbookletTAC2024**] [**rulesregulationsTAC2024**].

[Blokkskjema for årets ROV]



[Årets ROV Draugen]



Figur 1.2: Oversikt over gruppestruktur og ROV 2024

Prosjektledelsen for prosjektet 2024 er :

Prosjektleder: Sveinung Laupland Høyvik
Konkurranseansvarlig: Bjørge Zagros Rysstad
Teknisk leder, Elektro: Ingvild Bortheim
Teknisk leder, Data: Christofer Juul
Teknisk leder, Maskin: Magnus Dolmen

Prosjektleder
Prosjektlederens ansvar er i hovedsak å se til at prosjektet i sin helhet kommer i havn, innen gitte frister. Dette innebærer en generell oversikt over hva de ulike gruppenes ansvarsområder er på ROVen, samt oppfølging av disse. Som følger er kommunikasjon mellom prosjektledelsen og de ulike gruppene en svært viktig faktor. Prosjektlederen er også ansvarlig for å arrangere de ukentlige møtene gjennom prosjektet. Møtene har som regel bestått av en kort gjennomgang fra hver gruppe over hva som har blitt gjort, og hva som skal gjøres frem til neste møte. Det er da opptil hver enkelt gruppe å se til at disse punktene blir gjennomført, men her må også prosjektleder ha en viss oversikt.

Konkurranseansvarlig
Som konkurranseansvarlig er hovedfokuset på at ROVen møter de tekniske kravene som er satt for å kunne delta i årets konkurranse og i tillegg anskaffe mest mulig poeng i oppgavene som skal utføres. Dette tilsier først og fremst at det helhetlige systemet holder seg innenfor totalbegrensningene som er satt for deltakelse. Siden oppgavene som skal utføres i konkurransen også har problemstillinger innenfor forskjellige arbeidsområder, blir disse derfor kommunisert til de relevante gruppene med samme arbeidsområde, som dermed også får ansvaret for å implementere en løsning. Det er også konkurranseansvarlig som påtar seg kommunikasjon og påmelding til konkurransen, og opptrer som kontaktpersonen mellom TAC og UiS Subsea.

Teknisk leder, Elektro
Hovedoppgaven for teknisk elektroleder er å få alle elektrogruppene til å samarbeide med hverandre og tverrfaglig med de andre faggreinene slik at sluttproduktet blir realisert. Siden elektro avdelingen i UiS Subsea er den største, så følger det og med mer personalledelse. Elektroleder håndterer eventuelle utfordringer, enten det er av teknisk art internt i prosjektet eller relatert til samarbeidet mellom elektroteamet og de forskjellige undergruppene.

Teknisk leder, Data
Som teknisk leder for data er den viktigste oppgaven å sørge for god kommunikasjon internt mellom data-gruppene og med resten Subsea-teamet. Teknisk leder for data er også ansvarlig for at data-gruppene leverer det som er forventet av de for at prosjektet skal komme i mål. Ved data relaterte tekniske valg er det data-lederen som er ansvarlig for beslutningen, og for at det blir valgt gode løsninger.

Teknisk leder, Maskin
Teknisk leder for maskingruppen er ansvarlig for å ha kontroll på at arbeidsoppgaver knyttet til maskinstudentene blir overholdt og utført. En av hovedoppgavene er å lage konkrete

delmål, knyttet til det overordnede målet som, for maskingruppen, er å produsere en funksjonell ROV og manipulator. På grunn av at dette er et tverrfaglig prosjekt på tvers av tre ingeniør disipliner, er det viktig for den tekniske maskin ansvarlige å opprettholde kontakt med grupper som kan bli berørte ved eventuelle design endringer på både ROVen og manipulatoren.

1.3.1 Design av ROV

- **Data, GUI operatørsystem:** *-Christoffer Juul, Jan Phillip Aartun*

  Hovedoppgaven til GUI-gruppen er å utvikle et overvåkning og kontrollsystem for ROV-en. Det skal lages et grafisk operatørgrensesnitt (GUI) som viser all relevant informasjon fra ROV-en, som sensordata og videostrøm, i sanntid til operatøren(e) på en oversiktlig måte.

  GUI-gruppen står også for å implementere styring av ROV-en og manipulatoren. Dette gjøres ved å ta imot data fra en kontroller og sende dataen ned til ROV-en hvor den vil bli prossesert av de respektive bachelor gruppene.

  For at alt dette skal fungere er det nødvendig med et system som kan sende data mellom ROV-en og Top Side. Dette systemet skal implementeres i sammarbeid med kommunikasjons-gruppen.

- **Data, Autonom docking:** *-Karina Delagdo, Anne Isabel Rivero Jotun*

  Formålet med oppgaven er å utvikle et program som får ROV-en til å lokalisere en dockingstasjon posisjonert under vann i et basseng, og forsøke å docke stasjonen autonomt. ROV-en utstyres med kamera og dvl-sensor som kan brukes til å navigere omgivelsene. Dockingstasjonen er festet på en palle som er utstyrt med ArUco-merker i hvert hjørne som brukes til posisjonering og gjenkjenning.

- **Ele, Autonom kjøring:** *-Bjørn Magnus Myrhaug, Aleksander Møllerud*

  Formålet med oppgaven er å utvikle programmer som gjør at ROV-en autonomt kan inspisere en rørledning og en arbeidsbenk under vann. ROV-en skal lese av AruCo-koder plassert langs rørledningen og på benkens overflater. Autonom kjøring oppnås ved å estimere ROV-ens posisjon med bruk av videostrøm fra kameraer og posisjonsdata fra DVL-sensor.

- **Ele, Regulering:** *-Anne Brit Berge, Erlend Sandve Solvang, Halvard Erlandsen*

  Oppgaven går ut på å utvikle styre- og reguleringssystemet til ROV-en. For å kunne utføre oppgavene på best mulig måte i TAC-challenge er ROV-en avhengig av gode manøvreringsegen- skaper, som er et resultat av et funksjonelt styre- og reguleringssystem.

- **Ele, Manipulator arm:** *-Ingvild Bortheim, Eirik lilledal Haarberg, Øyvind Hadland*

  Manipulatorarmen sitt design utarbeides som ett samarbeid med maskin gruppen. Målet til manipulator gruppen er å løse oppgavene som er beskrevet i TAC challenge 1.2.2 Ventilintervensjon. Hoved oppgaven til denne gruppen er å utforme styresystemet ved bruk av kinematikk og implemetere det ved programering i C.

- **Ele, Sensorsystem:** *-Rolf Svanheim, Svein-Thore Hammerseth Wighus, Tina Ravndal Askø*

  Oppgaven omhandler utviklingen av ROV-ens sensorsystem. Systemet skal samle inn rådata fra ulike sensorer og behandle disse for å kunne styre og overvåke ROV-en. Behandlede data sendes videre til det overordnede systemet over kommunikasjonsgrensesnittet CAN FD. For regulering og autonom kjøring (og docking) av ROV-en kreves data om orientering og posisjon, og samtidig skal det kommuniseres dersom det blir lekkasje eller for høye temperaturer i elektronikkhuset. Sensorsystemet realiseres ved utvikling av kretskort med mikrokontroller og aktuelle sensorer.

- **Ele, Kraftforsyning:** *-Bjørge Zagros Rysstad, Sigve Hamilton Aspelund, Saeed Mohammed Issa*

  Kraftforsyningsgruppen har det overordnete ansvaret for å passe på at alle andre moduler mottar spenningen de trenger og at de mottar nok kraft for å kunne driftes som ønsket. Systemet består av 2 kretskort og en kabel fra land. Det er overvåkingsfunksjoner for kretser med høyt strømtrekk i tillegg til generell overvåking via sikringskretser på hvert uttak. Disse målingene sendes da til kommunikasjonskortet fra en mikrokontroller på kretskortet, som dermed sender dataen videre til topside. ROV-en har også et batteri ombord som automatisk skal brukes som kraftkilde dersom det ikke blir levert strøm fra kraftkabelen.

- **Ele, Kommunikasjon:** *-Paolo Lumanlan Maglambayan, Tor Øverland Blikra, Shajan Naranderan*

  Kommunikasjons- og videooverføringsgruppen skal utvikle grensesnittet for intern kommunikasjon mellom kretskortmodulene og eksternt mellom topside og ROV-en. Grensesnittet for kommunikasjon er realisert ved hjelp av henholdsvis CAN FD- og Ethernetprotokollene. Samtidig skal gruppen bearbeide videostrømmene fra kameraene plassert foran, under og på manipulatorarmen på ROV-en, før de sendes opp til topsiden. I likhet med fjorårets gruppe har kommunikasjonsgruppen ansvar for å utvikle elektronikkhuset. Gruppen skal utvikle et mal på formen av kretskort som inkluderer et krets for et CANFD-transceiver, som skal bli brukt av gruppene som skal ha et kretskort inni ROV-en.

- **Ele, Batteri:** *-Asbjørn Hagum Bogstad, Ann Sissel Nilsen, Erik Sunde Hetland*

  Batterigruppen har ansvaret for å utvikle batteri- og lysstyring. Batterisystemet skal sørge for levering av kraft til ROV-en slik at den kan operere uten ekstern strømforsyning. Lyssystemet skal gi sikt under vann i varierende forhold og skal være dimmbare.

Batteriet bygges i egen kapsling med battericeller og styringssystem (BMS) mens Lyssystemet skal bygges inn sammen med kraftforsyning med lyskastere og tilhørende elektronikk.

Batteri er en ny satsing for UiS Subsea og det er første gang et slikt system er utviklet. Målet er at fremtidige ROV-er skal kunne kjøre helt uten ekstern kraftforsyning. ROV-en dette året er derfor laget slik at batterisystemet ikke er nødvendig under normal drift og kan kobles fra dersom problemer skulle oppstå.

- **Mask, Rov og manipulator design:** *-Sveinung L Høyvik, Magnus Dolmen, Flavio Padillo III*

  Maskinstudentene på årets prosjekt er samlet i en gruppe bestående av 3 studenter som har ansvaret for henholdsvis ROV design, elektronikk- og batteri huset, og manipulatoren. Hovedmålet for gruppen i år er å designe og produsere en funksjonell ROV og manipulator, som har de funksjonelle egenskaper for å kunne utføre utfordringene gitt på TAC Challenge.

  ROV designet legger selve grunnlaget for prosjektet. ROVen har i hovedsak blitt designet med hensyn til kravene for TAC Challenge konkurransen 2024. Designet er også påvirket av krav og ønsker fra de andre bachelor gruppene. Nytt for årets prosjekt, er at det også skal produseres et batterihus. Både batterihuset og elektronikkhuset skal beskytte elektronikk mot vanninntrenging. Som en følge av dette følger absolutt kravet om at både elektronikk- og batterihuset skal være vanntett.

  I år er arbeidet med manipulatoren et tverrfaglig samarbeid på tvers av to ingeniør disipliner, henholdsvis maskingruppen og elektrogruppen ansvarlige for manipulatoren. Dette samarbeidet åpner for muligheten om å utvikle en mer avansert manipulator enn tidligere år. Målet for årets manipulator prosjekt er å utvikle en robust manipulator, kapabel til å utføre utfordringene knyttet til TAC Challenge konkurransen.

1.4 Autonom kjøring - Fra simulering til virkelighet

Hovedmål for oppgaven er å utvikle autonom oppførsel for ROV-en, med formål om utføre rørinspeksjon og visuell inspeksjon. Dette større hovedmålet deles opp i fire underoppgaver:

1. **Simulert miljø og ROV:** For utvikling av programvare for autonom kjøring trengs et simulert miljø og en simulert ROV. Her kan programvaren for bildebehandling og autonom kjøring produseres, testes og optimaliseres. Simulert miljø og ROV utvikles i simulatorverktøyet *Gazebo*. Modellering av overføringsfunksjoner for bevegelse av ROV-en skal brukes for å gi et godt utgangspunkt for innstilling av PID-parametere.
2. **Behandling av sensordata:** ROV-en har tilgang til videostrømmen fra tre kameraer, samt posisjonsdata fra en DVL-sensor. ROV-en kan bruke kameraene til å kartlegge omgivelsene sine og DVL-sensoren til å sjekke hvor den selv har kjørt. Egne klasser for bildebehandling og behandling av posisjonsdata må produseres for å bruke disse sensorene til autonom kjøring.
3. **Regulering:** Diskrete PID-regulatorer brukes for regulering av fart, med det formål å navigere ROV-en til bestemte posisjoner. Forskjellige sensorer gir forskjellige datatyper som må tilpasses PID-regulatorene. Regulatorene må justeres slik at ROV-ens autonome kjøring blir robust og effektiv.
4. **Programvare, fra simulering til virkelighet:** Etter programvare for autonom kjøring er utviklet ved hjelp av simulering, skal dette integreres opp i mot den virkelige ROV-en.

1.4.1 Funksjonsbeskrivelse

Nedenfor listes nøkkelfunksjoner som skal utvikles for å nå målene for prosjektet.

1. **Simulering**
   - Modell: ROV
   - Modell: ArUco-koder
   - Modell: Rørledning
   - Modell: Arbeidsbenk
   - Simplifisert fysikk/undervannsmiljø
   - Overføringsfunksjoner for ROV-ens bevegelser
2. **Behandling av sensordata**
   - Tilpasning av fargespektrum
   - Avlesning av ArUco-koder
   - Lokalisering av rør

- Lokalisering av benk
- Beregning av vinkler og avstander i bilder
- Lese posisjon med DVL-sensor

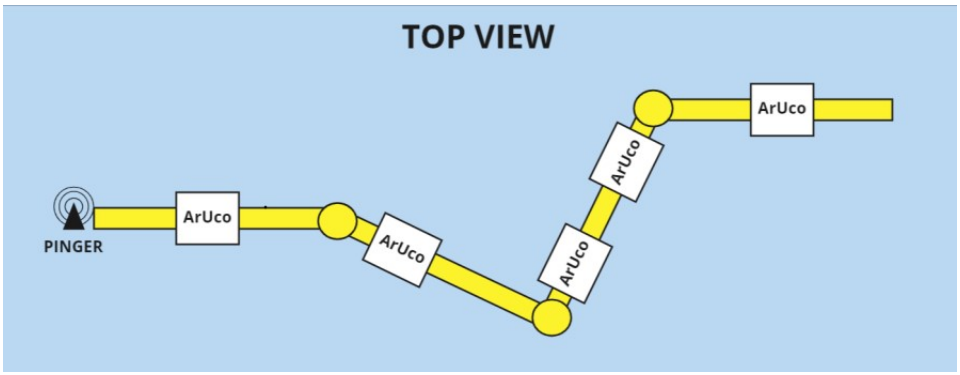
3. **Regulering**
   - PID-regulering av alle aktuelle bevegelser
   - Regulering med referanse fra bildebehandling
   - Regulering med referanse fra DVL-sensor
   - Hjem funksjon
   - Logging
4. **Programvare, fra simulering til virkelighet:**
   - Integrasjon mot signaler fra reelle sensorer
   - Justere regulatorparametere mot ROV-ens reelle bevegelser
   - Justere fargespektrum i bildebehandling til sikt under vann.

1.5 Oppgavene fra *TAC challenge* for Autonom kjøring

Oppgavene fra *TAC challenge* er beskrevet på engelsk i [15]. I dette kapittelet oversettes oppgavene som er relevante for autonom kjøring.

1.5.1 Rørinspeksjon

Det skal utføres en inspeksjon på en gul rørledning under vann. Røret er gult og har vinkler som kan varriere fra -90° til 90°. Langs rørledningen er det ukjent antall ArUco-koder som skal identifiseres. Poengsum for inspeksjon av rørledning i grenen autonom kjøring beregnes ut ifra tabell 1.2. Figur 1.3 viser et eksempel for en rørleding med plassering av ArUco-koder.



Figur 1.3: Eksempel på rør oppsett fra *TAC challenge* teamet

ArUco-kodene

- ArUco-kodene som er brukt er laget med generatoren fra https://chev.me/arucogen/
- Det er mellom 4 og 10 ArUco-koder langs rørledningen, det nøyaktige nummeret vil være ukjent før konkurransen.
- ArUco-kodene sin ID kan vaiere fra 1 til 99, det vil ikke være duplikater langs rørledningen.
- Alle ArUco-kodene er plassert horisontalt på rørledningen, med minst 200 mm mellomrom. Rotasjonen til ArUco-kodene er tilfeldige.
- Rammen rundt ArUco-koden er 50 mm tykk og hvit.
- Selve ArUco-koden er 150x150 mm, 200x200 mm med rammen.

Pinger

- En akustisk pinger er plassert ved den ene enden av rørledningen, den marker starten av rørinspeksjonen.
- Pingeren er MFP-1 fra JW Fisher
- Pingeren har en Frekvens på 30 kHz, repetisjonsrate på 2 sek, og pulslengde på 4 ms

Rørledningen

- Rørledningen består av et ukjent antall rette rør, rørene er koblet sammen av sammenkoblinger med uvisst vinkel fra -90° til 90°, antall sammenkoblinger er ukjent.
- Rørledningen har en total lengde på maks 10 meter.
- Røret har en konstant høyde(så godt det lar seg gjøre).
- Røret har en diameter på 200 mm og er gult.

Poeng for autonom kjøring		
Oppgave	Beskrivelse	Poeng
Autonom deteksjon av ArUco-kodene	Liste over avleste ArUco-koder i avlest rekkefølge, skal printes til GUI eller skrives til en fil når inspeksjonen er fullført. Oppgaven må løses innen et fornuftig tidsrom. Manuell endring av ArUco-listen vil diskvalifisere gruppen fra å få poeng på denne oppgaven.	+10p per detekterte ArUco-kode
Autonom lokalisering av rørledningen fra hjem-posisjon	Når oppdraget har begynt skal AUV-en autonomt lokalisere pingeren. Enhver form for manuell endring eller styring vil diskvalifisere gruppen fra å få poeng på denne oppgaven.	+100p
Autonom kjøring langs rørledningen	AUV-en kan manuelt styres for å finne rørledningen, men når rørledingen er funnet er det ikke lov med manuell styring. Enhver form for manuell endring vil diskvalifisere gruppen fra å få poeng på denne oppgaven.	+100p
Autonom kjøring til hjem-posisjon	Når rørinspeksjonen er fullført skal AUV-en autonomt returnere til hjem-posisjon. Enhver form for manuell endring vil diskvalifisere gruppen fra å få poeng på denne oppgaven.	+50p

Tabell 1.2: Poeng for autonom kjøring av rørinspeksjon

1.5.2 Inspeksjon av arbeidsbenk

For denne oppgaven skal arbeidsbenken vist i figur 1.4 inspiseres.

Figur 1.4: 3D modell av arbeidsbenk laget av *TAC challenge* teamet

Arbeidsbenken er lokalisert på havbunnen. Det er plassert flere ArUco-koder rundt omkring på arbeidbenken, målet for oppgaven er å identifisere ArUco-Kodene.
Pengesum for autonom kjøring i oppdraget beregnes ut ifra tabell 1.3. ArUco-kodene er plassert med varierende vanskelighetsgrad i forhold til synlighet og vil være av samme standard som i oppgaven rørinspeksjon beskrevet i delkapittel 1.5.1.

Poeng for autonom kjøring		
Oppgave	Beskrivelse	Poeng
Autonom deteksjon av ArUco-kodene	Listen med avleste ArUco-koder generert av ROV/AUV skal printes til GUI eller en fil når inspeksjonen er fullført. Oppgaven må løses innen et fornuftig tidsrom. Manuell endring av ArUco-listen vil diskvalifisere gruppen fra å få poeng på denne oppgaven.	+20p for hver riktig ArUco-kode

Tabell 1.3: Poeng for autonom kjøring ved inspeksjon av arbeidsbenk

1.6 Prosjektstyring

For at arbeidet på prosjektet skal være så effektivt som mulig er det viktig med god prosjektstyring. God prosjektstyring innebærer kontinuerlig planlegging og effektiv arbeidsstyring.

1.6.1 Arbeidsstrategi

For å opprettholde god kommunikasjon har mesteparten av arbeidet foregått på lab der gruppen sitter samlet. Felles arbeidsplass gir lav terskel for å spørre om hjelp eller meninger fra andre på gruppen.
For å holde styr på forefallende arbeidsoppgaver brukes en liste over gjøremål som oppdateres fortløpende. For planlegging av timer samt time registrering brukes et GANT-skjema[2] . GANT-skjemaet gir en generell pekepinn på tidspress. Figur 1.5 sammen ligner timene planlagt mot hvor timer igjen for å vise hvor mange timer som forventes å bruke for å nå målene.

Timer planlagt		Timer brukt		Timer igjen		Timer pr uke		Timer pr dag	
BM	AM	BM	AM	BM	AM	BM	AM	BM	AM
608,5	608,5	354	335	254,5	273,5	33,9	36,5	6,4	6,8

Figur 1.5: Time beregning for jobbing med prosjektet

[2]Figur G.1 viser GANT-skjemaet for prosjektet

1.6.2 Møter

For å legge til rette for god kommunikasjon holdes det flere forskjellige møter med forskjellige deltakere. De jevnlige møtene var:

1. Felles veiledningsmøter (ved invitasjon)
2. Subsea møter (Ukentlig)
3. Veiledningsmøter (ca månedlig)
4. Team møter (ved behov)

1. **Felles veiledningsmøter:** Møter med alle veilederne og medlemene som går data og elektro som skriver bachelor for UiS Subsea
2. **Subsea møter:** Møter for alle som skriver bachelor for UiS Subsea, arrangeres hver mandag.
3. **Veiledningsmøter:** Møte med veilder og bi veileder.
4. **Teammøter:** Møte for autonom kjøring, siden begge sitter på lab sammen mesteparten av tiden er disse møtene ved behov.

1.6.3 Arbeidsfordeling

Arbeidsfordelingen er satt opp som følger:

- **Aleksander:** Oppsett av programvare(Ubuntu, ROS, Gazebo) og utvikling av programvare til autonom kjøring for *oppdrag arbeidsbenk*
- **Bjørn Magnus:** Algoritmer for bildebehandling og utvikling av programvare til autonom kjøring for *oppdrag rørledning*.

1.6.4 GitHub

I dette prosjektet samarbeider gruppemedlemmene om å utvikle programvare, til dette formål brukes GitHub. Slik kan kode skrives til skyen fra forskjellige maskiner. GitHub har egne funksjoner for sammensying av kode dersom flere medlemmer redigerer samme fil samtidig. Link til GitHub mappen for prosjektet

Kapittel 2

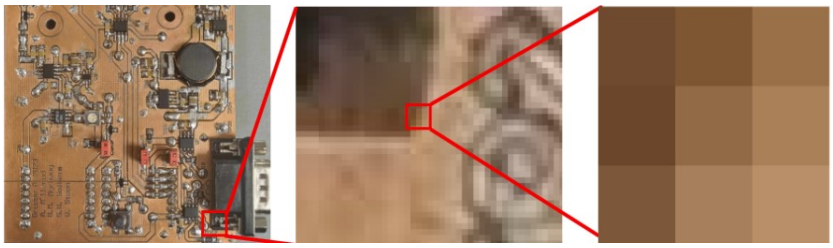
Bildebehandling

Innhold

Når ROV-en skal kjøre autonomt brukes kunstig syn for å lokalisere objekter og ArUco-koder. Dette kapittelet tar derfor for seg teorien om sammensetningen til digitale bilder, hvordan biblioteket OpenCV brukes i kunstig syn, og avlesning av ArUco-koder med kunstig syn.

2.1 Digitale bilder

Digitale bilder bygges opp av individuelle piksler som tildeles hver sin farge. Oppløsningen til et bilde defineres som antall piksler bildet består av. Dersom et bilde har en oppløsning på 1280x720, tilsvarer det en høyde på 720 piksler og en bredde på 1280 piksler, totalt $1280 * 720 = 921600$ piksler.
Figur 2.1 viser tre bilder med forskjellige oppløsninger: 1681x1508, 42x46 og 3x3.

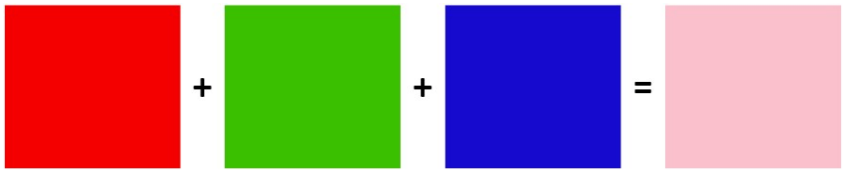


Figur 2.1: Nr.2 og 3 er forstørrelser av bilde 1.

2.1.1 Fargemodeller

RGB

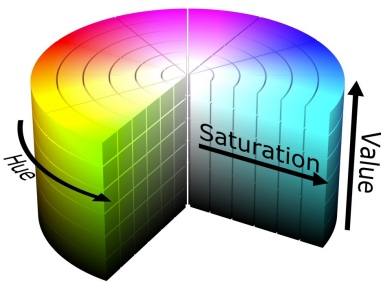
Det finnes flere forskjellige fargemodeller som kan brukes til å representere digitale fargebilder. Den mest brukte fargemodellen er RGB(Rødt, Grønt og Blått), denne kan representere enhver farge ved å beskrive hvor mye av av hver grunnfarge en piksel inneholder. Mengden grunnfarge representeres av en verdi fra 0-255. Rosa har f.eks RGB kode (255, 192, 203), den består av en kombinasjon av rød, grønn og blåfargen fra figur 2.2.



Figur 2.2: Rosa beskrevet av rød(255), grønn(192) og blå(203)

HSV

HSV(fra engelsk: *hue, saturation, value*) er en fargemodell som brukes hovedsaklig i bildebehandling og maskinlæring. Oversatt til norsk står HSV for fargetone(H), metning(S) og verdi(V). Fargetone beskriver ren farge, metningen beskriver hvor lys fargen er, mens verdi forteller hvor sterk fargen er. For å beskrive en farge i HSV spekteret brukes tallom-

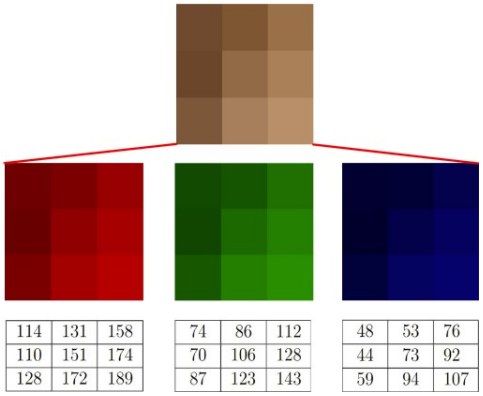


Figur 2.3:
HSV spekteret[9]

rådene (0-179, 0-255, 0-255). Figur 2.3 viser HSV spekteret representert som et sylinder.
Grunnen til at HSV blir brukt i bildebehandling er at HSV verdiene kan stilles inn til å filtrere bort farger slik at en kun beholder ønsket informasjon fra bildet. For en mer detaljert forklaring se [8] og et eksempel på bruk av HSV [2].

2.1.2 Sammensetning av et farget digitalt bilde

Et digitalt bilde som bruker RGB fargemodellen er satt sammen av 3 matriser. Hver av matrisene beskriver hvor mye rødt, grønt eller blått hver piksel inneholder. Figur 2.4 viser hvordan et 3x3 bilde består av ett rødt, ett grønt og ett blått bilde som representeres av hver sin matrise. De tre matrisene kan altså kombineres for å beskrive det øverste bildet i figur 2.4.



Figur 2.4: Sammensetningen RGB matrisene som brukes for å utgjøre et fullstendig bilde

2.2 Kunstig syn

Kunstig syn(engelsk: *computer vision*) er et felt innen kunstig intelligens, som tillater datamaskiner å identifisere objekter, hendelser og aktiviteter i bilder og bildestrøm(video). Kunstig syn kan brukes til blant annet: deteksjon av objekter og deres posisjon, gjenkjenning av farger, samt lesing av tekst og koder.

2.2.1 OpenCV

OpenCV(se figur 2.5) er et åpen kildekode bibliotek produsert for kunstig syn og maskinlæring. OpenCV er originalt skrevet i C++ men er kompatibelt med både Python, Java

Figur 2.5: OpenCV sin logo

og MATLAB. Biblioteket består av mer enn 2500 funksjoner som er godt dokumentert på OpenCV sin nettside https://docs.opencv.org/3.4/.

Under er en liste over alle funksjoner brukt til objekt deteksjon i dette kapittelet. Ved å trykke på en funksjon under åpnes dokumentasjonen til funksjonen

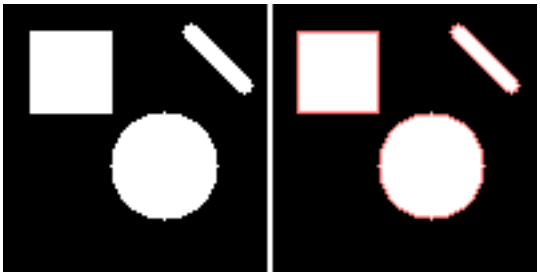
- **cv2.cvtColor()**: Endrer fargemodellen til bildet.
- **cv2.inRange()**: Returnerer et svart-hvitt bilde der fargen innen fir området er hvitt.
- **cv2.findContours()**: Finner konturene til bildet.
- **cv2.boundingRect()**: Returnerer hjørnene til en rektangel rundt en kontur.
- **cv2.minEnclosingCircle()**: Returnerer sentrum og radiusen til en sirkel rundt en kontur.
- **cv2.minAreaRect()**: Returnerer hjørnene til en vinklet rektangel rundt en kontur.
- **cv2.drawContours()**: Tegner konturene.
- **cv2.contourArea()**: Beregner arealet til en kontur.

2.2.2 Objekt deteksjon

Objekt deteksjon er en teknikk som gjenkjenner/lokaliserer objekter ved hjelp av kunstig syn.

Konturer

En måte å detektere et objekt på er ved å finne konturer. For å finne konturer i et bilde, må bildet konverteres til et binært bilde(svarthvitt bilde). Konturene oppdages deretter ved å detektere hvor i bildet pikslene går fra svart til hvitt. Med OpenCV kan man bruke **cv2.findContours()** for å finne konturene i et bilde. Figur 2.6 viser to bilder, bildet til høyre viser konturene i bildet til venstre.
cv2.findContours() returner koordinatene til kantene tilhørende hvert objekt. Funksjonen registrerer at firkanten har 4 kanter, linjen 18 kanter og sirkelen 68 kanter.



Figur 2.6: Eksempel på bruk av cv2 for å finne konturene i et bilde

Approksimasjon av konturer

OpenCV har flere forskjellige funksjoner for å approksimere konturer. Her ser vi på tre forskjellige metoder, avgrenset firkant, firkant med minst areal, og sirkel med minst areal.

Funksjonen **cv2.boundingRect()** returnerer hjørnene til en avgrenset firkant, der alle sidene er parallelle med kantene til bildet. I tillegg har firkanten minst mulig areal samtidig som den dekker den originale konturen. Bruk av denne vises i figur 2.7.

Figur 2.7: Approksimasjon av kontur fra figur 2.6 med avgrensede firkanter

For å approksimere konturen med en firkant som har minst mulig areal, brukes **cv2.minAreaRect()**. Funksjonen returnerer som **cv2.boundingRect()** hjørnene til firkanten. Her trenger ikke kantene være parallelle med kantene til bildet, som vist i figur 2.8.

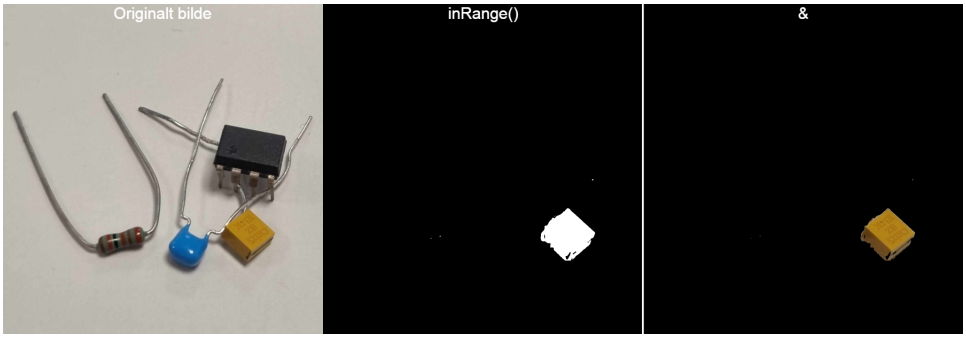
Figur 2.8: Approksimasjon av kontur fra figur 2.6 med firkanter med minst mulig areal

For å approksimere konturen som er en sirkel med minst mulig areal brukes **cv2.minEnclosingCircle()**. Funksjonen returnerer koordinatene til senter og radiusen til sirkelen. Bruk av funksjonen vises i figur 2.9.

Figur 2.9: Approksimasjon av kontur fra figur 2.6 med avgrenset sirkler

Fargemaskering

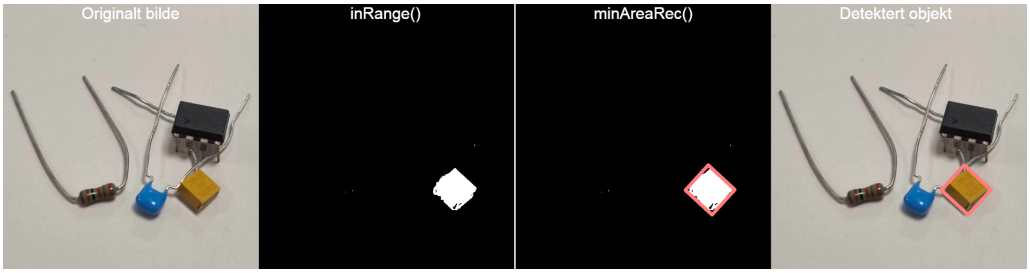
Fargetmaskering brukes i bildebehandling for å isolere en farge og dermed filtrere vekk unødvendig informasjon(som regel de delene av bildet som ikke inneholder spesifisert farge). I OpenCV brukes funksjonen **cv2.inRange()** til fargemaskering, denne tar inn nedre og øvre HSV-område sammen med et bilde og returnerer et maskert(svart-hvitt) bilde. På figur 2.10 brukes **cv2.inRange()** på bildet til venstre med et HSV-område på [15,72,88]-[23,255,189], bildet lengst til høyre er produktet av de to andre bildene kombinert med en bitvis og-operator &.



Figur 2.10: Eksempel på fargemaskering

2.2.3 Eksempel på objekt deteksjon

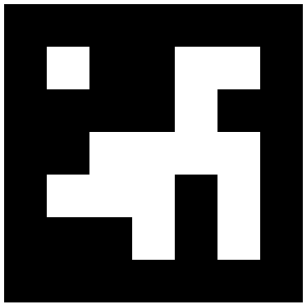
Figur 2.11 viser steg for steg for hvordan kondensatoren i figur 2.10 detekteres. Først maskeres originalbildet med fargeområdet til kondensatoren for å isolere denne. Deretter brukes **cv2.minAreaRect()** i svart-hvitt bildet, for å finne et rektangel med størrelse større enn 100 piksler.



Figur 2.11: Eksempel objekt detektering

2.3 ArUco koder

En ArUco-kode er en binær matrisekode originalt produsert for utvidet virkelighet(engelsk: *augmented reality*). Standarden ble utviklet i 2014 ved universitet i Cordoba og dokumentert i artikkel [5]. Målet med ArUco-kodene er at en datamaskin skal kunne lese av ett tall ved hjelp av kunstig syn.



Figur 2.12: Tallet 9 for en 5x5 ArUco-kode

Koden består av en svart ramme med en bredde på 1 enhet(en svart eller hvit firkant er en enhet), inni rammen finnes en matrisekode representert av binære verdier, størrelsen på matrisekodene kan variere fra 4x4 til 7x7 enheter. verdien til ArUco-koden kan variere fra 0-999, uavhengig av størrelsen, koden kan leses av uavhengig av rotasjonen til koden. På figur 2.12 viser et eksempel på en ArUco-kode som representerer tallet 9 med en størrelse på 5x5. En ArUco-kode kan beskrives som en forenklet versjon av QR-koden. Informasjonen i en ArUco-kode kan kun representere ett tall, mens en QR-kode kan representere større mengder informasjon.

For å kunne både generere og lese ArUco-koder har Rafael Muñoz og Sergio Garrido utviklet et bibliotek til OpenCV som heter ArUco. Biblioteket er basert på artiklene [5] og [6] og en annen rapport hvor programmet har blitt videre utviklet for å blir raskere [20]. Alle artiklene er skrevet av de samme som skrev [5] i 2014. Hele biblioteket for lesing av ArUco-koder er dokumentert i [3]

2.3.1 Lesing av ArUco-koder med OpenCV

Følgende funksjoner brukes til avlesning av ArUco-koder:

- **cv2.imread()**:
- **cv2.cvtColor()**:
- **ArUco.getPredefinedDictionary()**:

- **ArUco.detectMarkers()**:

Med bibliotekene fra OpenCV er det veldig enkelt å lese ArUco-kodene. Det gjøres med en enkel funksjon **ArUco.detectMarkers()**. Funksjonen vil returnere 3 matriser:

1. Hjørnene til de detekterte ArUco-kodene formatert som en 1xn matrise. Der n er antall ArUco-koder detekter. Verdiene i matrisen en 2x4 matrise med koordinatene til hjørnene.
2. Id'ene som er detektert som en matrise som er 1xn.Der n er antall ArUco-koder detekter
3. Diskvalifiserte avlesninger likt som hjørnene bare koordinatene er til kvadrater som ikke innholdt noe ArUco-koder.

Eksempel kode for avlesning av ArUco-koder:

```
1 import cv2
2 import cv2.ArUco as ArUco
3
4 Image = cv2.imread("Image_With_ArUco_Code.png")
5 Grayscale_Image = cv2.cvtColor(Image,cv2.COLOR_BGR2GRAY)
6 Dictionary = ArUco.getPredefinedDictionary(ArUco.DICT_5X5_100)
7 corners, ids, rejected = ArUco.detectMarkers(Grayscale_Image, Dictionary)
```

Tabell 2.1 forklarer kort hvordan avlesning av ArUco-koder fungerer i OpenCV.

<table>
<tr><td>1. Originalbildet.
  </td><td>2. Gråskalering av bildet gjort med funksjonen cv2.cvtColor()
  </td></tr>
<tr><td colspan="2">Steg 3-6 er alle gjort i funksjonen ArUco.detectMarkers()</td></tr>
<tr><td>3. Kvadrater i bildet identifiseres.
  </td><td>4. Identifiserer den ytre rammen som potensielt innholder en ArUco-kode
  </td></tr>
<tr><td>5. Matrisekoden i bildet gjøres om til en matrise som deretter prosesseres for å oppdage et predefinert mønster som tilsvarer en tallverdi.
<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table></td><td>6. Gråskalert bilde med markering av detektert ArUco-kode med tilhørende verdi.
  </td></tr>
</table>

Tabell 2.1: Avlesning av ArUco koder med OpenCV

Kapittel 3

Matematiske modeller

Innhold

Dette kapittelet tar for seg en gjennomgang av teori, beregninger og testing av overføringsfunksjonene som anvendes i simuleringen av ROV-en. Ved simulering responderer ROV-kontrolleren nesten perfekt og umiddelbart. Dette gir et urealistisk grunnlag for justering av PID-regulatorene som styrer bevegelsene til ROV-en. Derfor modelleres ROV-ens bevegelser først som overføringsfunksjoner, som deretter konverteres til diskrete tilstandsmodeller. Disse diskrete tilstandsmodellene implementeres i simulatoren og gir et mer realistisk grunnlag for PID-regulering.
Store deler av kapittelet baserer seg på informasjon fra vedlegg A, vedlegget gir en detaljert forklaring av fremgangsmåte for beregning og implementasjon av overføringsfunksjoner.

3.1 Valg av frihetsgrader

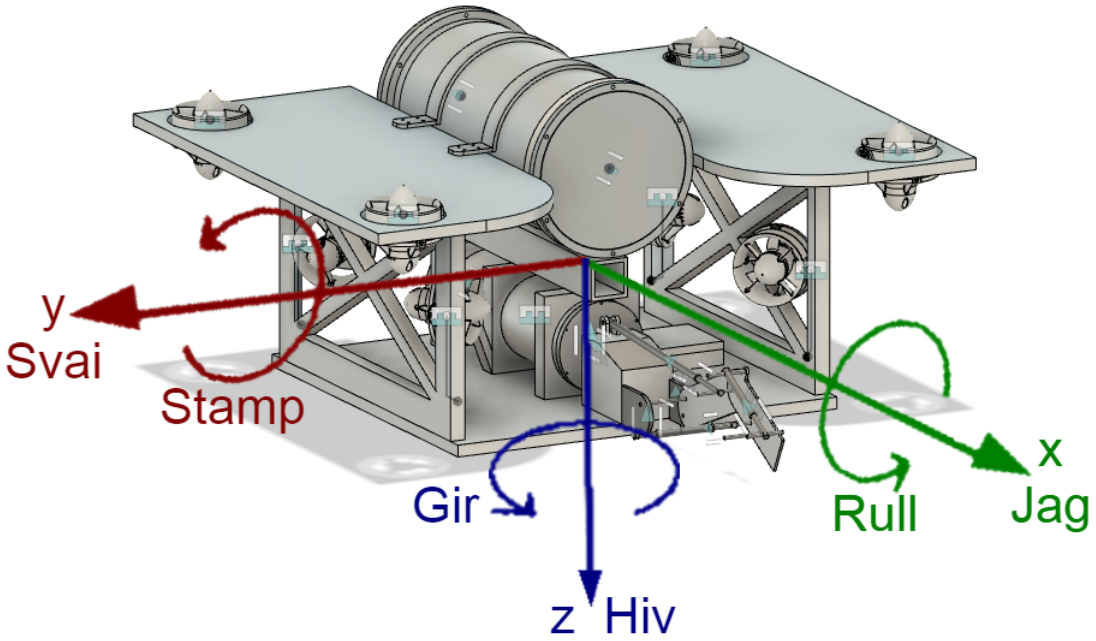
For å utvikle matematiske modeller må det bestemmes hvilke frihetsgrader som skal inkluderes, deretter modelleres hver frihetsgrad separat. Det å modellere flere frihetsgrader samtidig krever hensyn til alle aspektene ved væskedynamikk og er utenfor omfanget av denne oppgaven.

3.1.1 Frihetsgradene til ROV-en

Bevegelse hos en ROV defineres vanligvis ved seks frihetsgrader. Tabell 3.1 beskriver alle frihetsgradene brukt i ROV-en til UIS Subsea. Figur 3.1 gir en grafisk fremstilling av frihetsgradene.

Frihetsgrad	Bevegelse	Navn	Måleenhet
1	Lineær langs x-aksen	Jag	m
2	Lineær langs y-aksen	Svai	m
3	Lineær langs z-aksen	Hiv	m
4	Rotasjon om x-aksen	Rull	rad
6	Rotasjon om y-aksen	Stamp	rad
7	Rotasjon om z-aksen	Gir	rad

Tabell 3.1: ROV-ens frihetsgrader



Figur 3.1: Visualisering av ROV-ens frihetsgrader

3.1.2 Frihetsgrad simulert ROV

Det er behov for å kunne bevege ROV-en lineært langs x,y og z-aksen, samt rotere rundt z-aksen. Oppgavene som skal utføres krever ingen rotasjon rundt x eller y-aksen. Det planlegges derfor kun bruk av følgende frihetsgrader i simulatoren:

- Jag
- Svai
- Hiv
- Gir

Før disse frihetsgradene modelleres, gis det en gjennomgang av fremgangsmåte for hvordan modellere en frihetsgrad og konvertere denne til en diskret tilstandsmodell.

3.2 Overføringsfunksjoner

En overføringsfunksjon beskriver forholdet mellom inngang og utgang i s-planet. Overføringsfunksjoner brukes istedenfor differensialligninger som er i tidsplanet. For å gjøre om fra en differensialligning til en overføringsfunksjon brukes Laplace transformen. Ligning (3.1) viser den ensidige Laplace transformen.

$$F(s) = \mathcal{L}\{f(t)\} = \int_{0^-}^{\infty} e^{-st} f(t) dt \tag{3.1}$$

For anvendelse av modellene til frihetsgradene utvikles det en overføringsfunksjon for hver av de fire bevegelsene. Overføringsfunksjonene er lineariserte modeller av de originale differensialligningene. Grunnlaget for bruk av overføringsfunksjoner er at de er enklere å anvende enn differensialligninger.

3.3 Tilstandsmodeller

En tilstandsmodell representerer flerordens differensialligninger som et sett med førsteordens differensialligninger. En standard tilstandsmodell kan beskrives som:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{3.2}$$

$$y(t) = Cx(t) + Du(t) \tag{3.3}$$

Ligning (3.2) er tilstand ligningen og (3.3) er utgangs ligningen, der $u(t)$ er inngangen og $y(t)$ er utgangen.

Fra ligningene (3.2) og (3.3) er A,B,C og D

- **A:** Tilstands matrisen med en størrelse på $n \times n$, der n er antall tilstandsvariabler.
- **B:** Inngangs matrisen med en størrelse på $n \times m$, der m er antall innganger.
- **C:** Utgangs matrisen med en størrelse på $p \times n$, der p er antall utganger
- **D:** Gjennomførings matrisen med en størrelse på $p \times m$, vanligvis en nullmatrise.

Hver modell omgjøres fra overføringsfunksjon til tilstandsmodell for senere å anvendes i simulatoren.

3.3.1 Diskret tilstandsmodell

Simulering foregår i diskret tid, for at tilstandsmodellen skal kunne anvendes må denne også være på diskret form. For å gjøre om tilstandmodellen brukes ZOH (Zero-Order Hold)-metoden. For å diskretisere A, B, C og D brukes formelene:

$$A_d = e^{A \cdot t_s} \tag{3.4}$$

$$B_d = A^{-1}(A_d - I)B \tag{3.5}$$

$$C_d = C \tag{3.6}$$

$$D_d = D \tag{3.7}$$

Hvis $A \cdot t_s$ er diagonaliserbar kan ligning (3.8) brukes, da er kolonnene til P og P^{-1} er eigenvektorene til matrisen $A \cdot t_s$ og λ_1,λ_2 til $\lambda_n$er eigenverdiene.

$$A \cdot t_s = PDP^{-1} = \begin{bmatrix} \mathbf{v}_{11} & \mathbf{v}_{21} & \cdots & \mathbf{v}_{n1} \\ \mathbf{v}_{12} & \mathbf{v}_{22} & \cdots & \mathbf{v}_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{v}_{1m} & \mathbf{v}_{2m} & \cdots & \mathbf{v}_{nm} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} \mathbf{v}_{11} & \mathbf{v}_{21} & \cdots & \mathbf{v}_{n1} \\ \mathbf{v}_{12} & \mathbf{v}_{22} & \cdots & \mathbf{v}_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{v}_{1m} & \mathbf{v}_{2m} & \cdots & \mathbf{v}_{nm} \end{bmatrix}^{-1} \tag{3.8}$$

A_d regnes så ut med ligning (3.9).

$$e^{A \cdot t_s} = Pe^{D}P^{-1} = \begin{bmatrix} \mathbf{v}_{11} & \mathbf{v}_{21} & \cdots & \mathbf{v}_{n1} \\ \mathbf{v}_{12} & \mathbf{v}_{22} & \cdots & \mathbf{v}_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{v}_{1m} & \mathbf{v}_{2m} & \cdots & \mathbf{v}_{nm} \end{bmatrix} \begin{bmatrix} e^{\lambda_1} & 0 & \cdots & 0 \\ 0 & e^{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{\lambda_n} \end{bmatrix} \begin{bmatrix} \mathbf{v}_{11} & \mathbf{v}_{21} & \cdots & \mathbf{v}_{n1} \\ \mathbf{v}_{12} & \mathbf{v}_{22} & \cdots & \mathbf{v}_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{v}_{1m} & \mathbf{v}_{2m} & \cdots & \mathbf{v}_{nm} \end{bmatrix}^{-1} \tag{3.9}$$

Ved hjelp av MATLAB eller *control* biblioteket i Python kan variablene for de diskrete tilstandsmodellene regnes ut med funksjonene **c2d**(MATLAB) og **control.c2d**(Python).

3.4 Eksempel

I dette delkapittelet vises utregninger for et eksempel på omgjøring fra overføringsfunksjon til tilstandsmodell, og deretter diskretisering av tilstandsmodellen.

3.4.1 Linearisert system

For å linearisere modellene fra testrapport A.2 brukes System Identification programmet i MATLAB. Programmet trenger inngangs- og utgangsverdier fra systemene for å lage en linearisert modell. Hele utledningen av lineariseringen vises i testrapporten A.3.

Ved å følge stegene beskrevet i delkapittel A.3.4 for ROV-ens bevegelser, oppnås resultatene som presenteres i tabell 3.2

	overføringsfunksjon		overføringsfunksjon
Hiv	$\frac{4.308s^2+101.2s+141.5}{s^3+11.02s^2+133.8s+142}$	Jag	$\frac{3.606s^2+40.12s+1421}{s^3+16.98s^2+273.3s+1411}$
Svai	$\frac{4.014s^2+83.39s+2173}{s^3+31.09s^2+357.9s+2170}$	Gir	$\frac{3.74s^2+61.48s+2546}{s^3+38.65s^2+519.6s+2533}$

Tabell 3.2

Videre i dette eksempelet brukes kun overføringsfunksjonen til hiv fra tabell 3.2, verdiene fra denne brukes i ligning (3.10).

$$H(s) = \frac{b_2s^2 + b_1s + b_0}{s^3 + a_3s^2 + a_2s + a_1} = \frac{4.308s^2 + 101.2s + 141.5}{s^3 + 11.02s^2 + 133.8s + 141} \tag{3.10}$$

3.4.2 Omgjøring til tilstandsmodell

Overføringsfunksjonen for hiv konverteres videre til en tilstandsmodell, tilstandsmatrisen i likning 3.11 bruker derfor verdiene fra nevneren i likning (3.10).

$$A = \begin{bmatrix} -a_3 & -a_2 & -a_1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} -11.02 & -133.8 & -142 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.11}$$

Inngangsmatrisen har form som i ligning (3.12), systemet er et SISO system.
SISO(eng:*Single-Input Single-Output*) altså: en inngang og en utgang.

$$B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.12}$$

Utgangsmatrisen bruker verdiene fra telleren til overføringsfunksjonen i likning 3.10:

$$C = \begin{bmatrix} b_2 & b_1 & b_0 \end{bmatrix} = \begin{bmatrix} 4.308 & 101.2 & 141.5 \end{bmatrix} \tag{3.13}$$

Det er ingen gjennomføring i systemet så gjennomføringsmatrisen blir en nullmatrise.

$$D = \begin{bmatrix} 0 \end{bmatrix} \tag{3.14}$$

3.4.3 Diskretisering av tilstandsmodellen

Tilstandsmatrisen fra 3.4.2 diskretiseres ved å anvende ZOH-metoden, da endres kun tilstands- og inngangsmatrisen, ved å bruke formlene i ligning (3.4) og (3.5).

3.4.4 Resultat

Resultatet av diskretiseringen av overføringsfunksjonen i likning (3.10), med et tidsskritt lik 0.01 sekund ble følgende matriser:

$$A_d \approx \begin{bmatrix} 0.89 & -1.23 & -1.34 \\ 9.5 \cdot 10^{-3} & 0.99 & -6.9 \cdot 10^{-3} \\ 4.8 \cdot 10^{-5} & 9.8 \cdot 10^{-3} & 1.00 \end{bmatrix} \tag{3.15}$$

$$B_d \approx \begin{bmatrix} 9.5 \cdot 10^{-3} \\ 4.8 \cdot 10^{-5} \\ 5 \cdot 10^{-7} \end{bmatrix} \tag{3.16}$$

$$C = \begin{bmatrix} 4.31 & 101.2 & 141.5 \end{bmatrix} \tag{3.17}$$

$$D = \begin{bmatrix} 0 \end{bmatrix} \tag{3.18}$$

3.5 Modellering av bevegelsene

Som beskrevet i delkapitel 3.1.2, skal jag, svai, gir og hiv bevegelsene modelleres for implementasjon i simulatoren. Disse modellene tar utgangspunkt i modeller hentet fra bachelorgruppen ansvarlig for styring og regulering på Subsea ROV-en i 2022 [14].

PID-parametrene justeres og modellene endres slik at de tar inn hastighet isteden for posisjon. I Vedlegg A.2 vises en detaljert gjennomgang av hvordan endringene implementeres og hvordan de forskjellige PID-parameterene beregnes.

Den nye modellen for hiv-bevegelsen vises i figur A.3. Modellene for de andre bevegelsene har blitt endret på samme måte.

Tabell 3.3 viser PID-parametrene som brukes i modellene. PID tuning kolonnen representerer parametrene hentet fra *PID-tuning* programmet i MATLAB. Kolonnen Justerte PID-parametere representerer justerte parametere for modellene som skal implementeres i simulatoren.

Hiv		
	PID Tuning	Justert PID-parametere
P	1.71	2.71
I	0.55	6.55
D	0.06	0.06
Svai		
	PID Tuning	Justert PID-parametere
P	3.42	3.42
I	1.10	10.10
D	0.12	0.12

Gir		
	PID Tuning	Justert PID-parametere
P	0.23	0.23
I	0.07	0.57
D	0.02	0.02
Jag		
	PID Tuning	Justert PID-parametere
P	3.22	3.23
I	1.01	6.51
D	0.29	0.29

Tabell 3.3: PID parametrene fra PID tuning og justert verdier.

3.6 Linearisering av bevegelsene

Modellen for hiv fra delkapitel 3.5 samt de tre andre modellene skal lineariseres. Dette ble gjort med tilleggsprogrammet System Identification Toolbox i MATLAB. System Identification trenger inngangs- og utgangs verdiene til et system, for å lage en linearisert modell. Fremgangsmåten for linearisering av bevegelse er beskrevet i større detalj i vedlegg. A.3.
Tabell 3.4 viser de lineariserte modellene til de forskjellige bevegelsene beskrevet som overføringsfunksjoner.

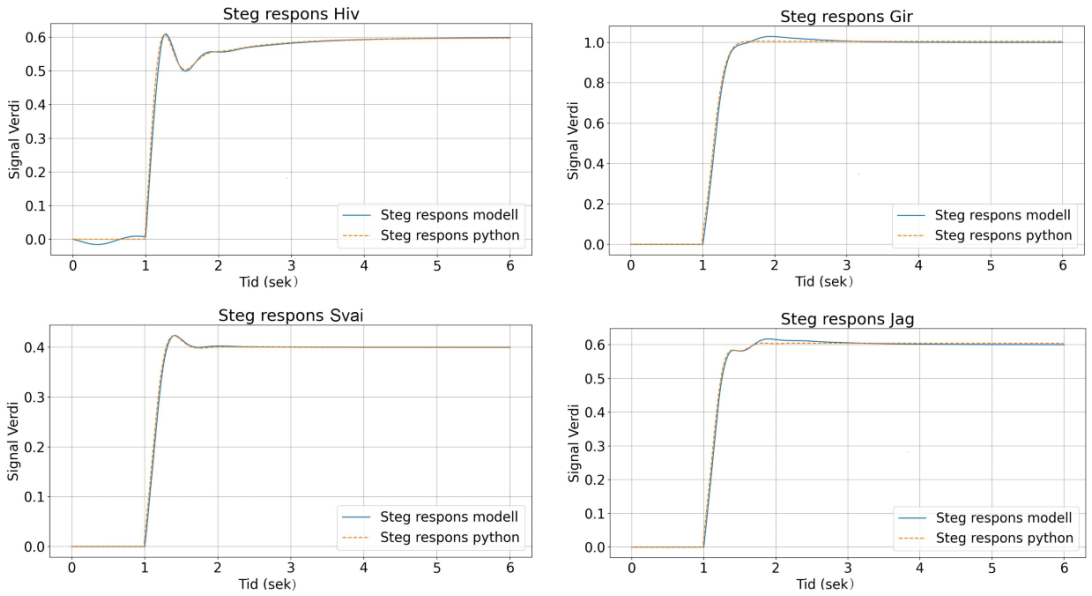
	Overføringsfunksjon		Overføringsfunksjon
Hiv	$\frac{4.308s^2+101.2s+141.5}{s^3+11.02s^2+133.8s+142}$	Jag	$\frac{3.606s^2+40.12s+1421}{s^3+16.98s^2+273.3s+1411}$
Svai	$\frac{4.014s^2+83.39s+2173}{s^3+31.09s^2+357.9s+2170}$	Gir	$\frac{3.74s^2+61.48s+2546}{s^3+38.65s^2+519.6s+2533}$

Tabell 3.4

3.7 Testing av overføringsfunksjonene

Overføringsfunksjonene fra delkapittel 3.6 kan ikke implementeres direkte i simulatoren da Python sine kontroll-biblioteker ikke støtter direkte bruk av overføringsfunksjoner, de støtter derimot bruk av tilstandsmodeller. For anvendelse av overføringsfunksjonene i Python, konverteres de først til tilstandsmodeller, for så å diskretiseres. Fremgangsmåten for konvertering til diskret tilgangsmodell forklares i vedlegg A.4.
Figur 3.2 viser stegresponsene til de forskjellige bevegelsene. Heltrukken linje viser den ulineære modellen fra MATLAB anvendt i Python. Stiplet linje viser den lineariserte modellen

anvendt i Python.



Figur 3.2: Stegrespons fra overføringsfunksjonene til brukte bevegelser

3.8 Konklusjon

For å danne et mer realistisk grunnlag for PID-regulering, implementeres overføringsfunksjonene til ROV-ens bevegelse fra 2022. De gamle overføringsfunksjonene har blitt modifisert, linearisert, konvertert til tilstandsmodell og deretter diskretisert. De diskretiserte tilstandsmodellene resulterte i tilnærmet samme respons som de originale overføringsfunksjonene, med minimalt avvik i hiv og svai, og et mer signifikant avvik i jag og gir. Avviket kunne vært redusert ved bruk av høyere ordens systemer, på bekostning av maskinvarens prosseseringskraft. Med hensyn til datamaskinens kapasitet falt valget på bruk av tredje ordens overføringsfunksjoner i simulatoren.
De eksisterende avvikene mellom de nye tilstandsmodellene og de gamle overføringsfunksjonene er relativt små, og de gamle overføringsfunksjonene var heller ikke 100% nøyaktige, avviket regnes derfor som akseptabelt.

Kapittel 4

Simulering

Innhold

Dette kapittelet tar for seg en grundig gjennomgang av programvaren for simulering og regulering av ROV-en. Igjennom simulering utvikles programvare for autonom kjøring av ROV-en, denne programvaren skal senere modifiseres og implementeres i den virkelige ROV-en.

4.1 Hvorfor simulere

Under utvikling av algoritmer for autonom styring av et farttøy, legger simulering til rette for lett tilgjengelig testing på ROV-en der dette ellers ikke ville vært mulig grunnet mangel på følgende gjenstander:

- Ferdigstilt ROV
- Basseng
- Rørledning
- Arbeidsbenk

Målet med simuleringen i denne oppgaven er å etterlikne et undervannsmiljø som tillater ROV-en å inspisere objektene fra oppgavene beskrevet i delkapittel 1.5.1.
Ved bruk av miljøet i simulatoren skal det utvikles programvare som bruker bildebehandling og DVL-sensor sammen med PID-regulatorer. Denne kombinasjonen av sensordata og regulatorer anvendes for å regulere farten til ROV-en basert på posisjonen til objekter som skal inspiseres, og ROV-ens egen posisjon.
En virkelighetsnær simulering legger til rette for utvikling av virkelighetsnær og robust programvare for kjøring. Dette fører igjen til mindre arbeid når programvaren skal integreres til den fysiske ROV-en.

4.2 Rammeverk for programvare

Det utvidede prosjektarbeidet på ROV-en innebærer et samarbeid mellom 10 forskjellige grupper. Flere individuelle moduler skal settes sammen for å produsere et sluttprodukt. Dette krever koordinasjon mellom gruppene. Enkelte moduldesign og tester er avhengige av enten delvis, eller fullstendig ferdigstilte moduler fra andre grupper.

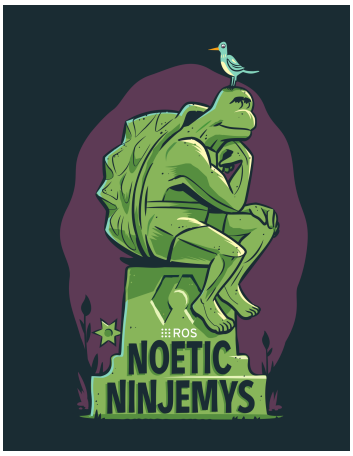
For sammensying av programvaren fra de forskjellige gruppene som jobber med ROV-en, har UiS Subsea i år bestemt seg for å bruke ROS(Robot Operating Systems), et rammeverk for robotikkapplikasjoner. Kjernefunksjonaliteten til ROS er node/topic nettverket deres, dette legger til rede for sømløs kommunikasjon mellom individuelle komponenter i et system. ROS nettverket er modulært, fleksibelt og legger til rette for effektiv skalering og videreutvikling av projekter.

4.2.1 Valg av ROS versjon

ROS 1 ble først lansert i 2007, 10 år senere i 2017 ble ROS 2 lansert. Den viktigste forskjellen mellom ROS 1 og 2 er mellomvaren, ROS 1 bruker en egendefinert mellomvare som heter *Rosmaster*.

ROS 2 bruker *DDS(Data Distribution Service)*, denne mellomvaren er standardisert og kan brukes opp i mot de fleste Windows-baserte applikasjoner. Bruk av DDS gir fleksibilitet for utvidelse av systemet og integrasjon mot eksisterende produkter. ROS2 ble utviklet for bruk i større applikasjoner.

Prosjektgruppene på UiS Subsea har i år måttet velge en felles distribusjon av ROS. Oppgaven autonom kjøring er svært avhengig av simulering og dermed ROS. Derfor har gruppen tatt hovedansvaret for å undersøke og velge ROS distribusjon for alle gruppene i år. Nedefor representeres de aktuelle distribusjonene i kronologisk rekkefølge, sammen med sine individuelle fordeler og ulemper.



Figur 4.1: Noetic logo [17]

Noetic Ninjemys

- **Pros:**
  - Mye brukt og mange eksempler
  - Mild læringskurve
- **Cons:**
  - ROS 1 er gammelt og bruken reduseres årlig
  - Færre funksjoner og mindre fleksibelt uten 'DDS'
  - Kjører kun stabilt på eldre versjon av ubuntu (20.04 Focal)

Figur 4.2: Foxy logo [4]

Foxy Fitzroy

- **Pros:**
  - Den langtisstøttede versjonen av ROS 2 med lengst levetid
  - De fleste open source ROS 2 projekter med undervannsmiljø er utviklet i Foxy.
- **Cons:**
  - Support avsluttet i 2023, anbefales derfor ikke av utviklerene.
  - Kjører kun stabilt på eldre operativsystemer ubuntu (20.04 Focal)

Figur 4.3: Humble logo [10]

Humble Hawksbill

- **Pros:**
  - Nyeste Langtidsstøttede distribusjon
  - Flere open source projekter tilgjengelig
  - Kjører på nyeste Ubuntu(22.04 Jammy)
- **Cons:**
  - Mindre open source projekter enn hos Foxy og Noetic
  - Mangler de nyeste funksjonene i Iron

Figur 4.4: Iron logo [11]

Iron Irwini

- **Pros:**
  - Nyeste distribusjon av ROS 2
  - Kjører på nyeste Ubuntu(22.04 Jammy)
- **Cons:**
  - Nærmest ingen open source projekter utviklet i Iron
  - Korttidsstøttet(Støtte avsluttes november 2024)

Valget av distribusjon falt til slutt på ROS 2 Humble.
Humble er den nyeste langtidsstøttede distribusjonen av ROS 2(videre omtalt som ROS) og har derfor kontinuerlig oppdatering av dokumentasjon og støttes fortsatt av utvikler.

4.2.2 Valg av simulator

Ved valg av simulator ble det funnet to aktuelle kandidater: *Gazebo Classic* og *Gazebo Fortress*. ROS og *Gazebo Classic* ble originalt produsert av samme utvikler: *Open Robotics*. I nyere tid har *Gazebo* og et annet simuleringsprogram med navn *Ignition* blitt sammenslått til*Gazebo Ignition*. *Gazebo Ignition* har flere versjoner, anbefalt versjon for bruk med ROS 2 Humble er *Gazebo Fortress*.

Gazebo Fortress er objektivt nyere og raskere enn *Gazebo Classic*. Likevel er *Classic* bedre støttet for bruk med ROS.
I ROS finnes det grensesnitt som gjør at ROS og *Classic* samhandler sømløst.
I *Fortress* må bruker selv sette opp en *ROS Ignition bridge* per emne, for å oppnå kommunikasjon mellom ROS og *Igniton*.

En annen nevneverdig funksjon i *Classic* er programvareutvidelser. Utviklerne av *Classic* inkluderer en lang liste med programvareutvidelser. Disse tillater brukere å legge til spesielle funksjonaliteter i en simulering, f.eks kamerastrøm eller kontrollere.

Før valg av simulator ble det utført tester av både *Classic* og *Fortress*. Etter testing ble følgende konklusjoner trukket angående simulatorene.

- *Fortress* er raskere og mer brukervennlig
- Programvareutvidelser er lavterskel å integrere i *Classic*, høy terskel i Fortress.
- Programvareutvidelser er nødvendige for å simulere et undervannsmiljø .
- *Classic* er utviklet spesifikt for bruk sammen med ROS.
- *Classic* er eldre enn Fortress, derfor finnes det mer dokumentasjon for *Classic*.

Etter testing av de forskjellige versjonene ble valget til slutt *Gazebo Classic*(videre omtalt som *Gazebo*). Den største faktoren for valget viste seg å være *Classic* sin programvareutvidelse for gravitasjonsløs bevegelse i 2D. Programvareutvidelsen blir brukt som en enkel løsning til implementasjon av en simplifisert modell av ROV-en.

4.2.3 Valg av operativsystem

En av de mest omfattende valgene for prosjektet er valg av operativsystem, ROS Humble gir følgende valgmuligheter for OS:

- Windows 10
- Ubuntu 22.04 Jammy
- Ubuntu på Windows 11 med virtuell maskin

Under testing av valgmulighetene ovenfor ble følgende konklusjoner trukket:

- Virtuell maskin er generelt krevende og tregt. ROS kjører ikke pålitelig på virtuell maskin.
- Selvom ROS kan kjøre på Windows 10, finnes det få dokumenterte tilfeller av dette.
- Docker som virtuell maskin vil gi størst gevinst over lengre tid, men krever for mye opplæring og dermed tid.

Valget for OS ble til slutt: Ubuntu 22.04 Jammy.
Ubuntu er lett å installere, krever ingen lisens og tar lite lagringsplass(25 Gb).
De fleste ressurser for opplæring i ROS bruker Ubuntu, for ROS Hubmle kreves Ubuntu versjon 22.04.

Figur 4.5: Ubuntu Logo [1]

4.3 Gazebo simulator

For simulering brukes simulatorverktøyet *Gazebo* sammen med *ROS*. *ROS* fungerer som hjernen til ROV-en, *Gazebo* simulerer kroppen og sensorene til ROV-en, og undervannsmiljøet ROV-en skal plasseres i. *ROS* og *Gazebo* kommuniserer sammen via emner.
ROV modellen i simuleringen er en forenklet versjon av den virkelige ROV-en. Den simulerte ROV-en representes av en simpel kube, i en verden uten gravitasjon.

4.3.1 Planar Move Plugin

For å styre en ROV i *Gazebo*, kreves en kontroller. Programvareutvidelsen *Planar Move Plugin*(PMP) viste seg å være en passende kontroller til dette formålet. *PMP* er designet til bruk i miljøer uten gravitasjon, en tilstand som ligner på bevegelser under vann. *PMP* abonnerer på emnet *cmd_vel* og styrer ROV-ens bevegelser basert på dataene fra dette emnet. *PMP* har også den innebygde funksjonen at den publiserer odometridataen til ROV-en.

4.3.2 Sensorer

ROV-en er utstyrt med diverse sensorer, dataen fra disse sensorene behandles for å produsere informasjon om ROV-ens posisjon relativt til verden rundt seg. *Gazebo* har et bibliotek med programvareutvidelser som tillater simulering av sensorer og sensordata. Den virkelige ROV-en er utstyrt med tre kameraer og en DVL-sensor. For utførelse av *TAC Challenge* oppgavene fra delkapittel 1.5.1, brukes to av kameraene og DVL-sensoren.

DVL A50

DVL-sensoren produserer mye data og har mange bruksområder, sensoren publiserer blant annet en *Dead reckoning report*(*DRP*) som er svært nyttig for navigasjon av ROV-en. *DRP-rapporten* inneholder inneholder sensorens(og dermed ROV-ens) posisjon og orientasjon, denne oppdateres med en fast frekvens på 5 Hz. *Gazebo* har ikke innebygget programvareutvidelse for DVL-sensor. I simuleringen brukes heller odometrien til ROV-en for å etterligne en DVL-sensor[1].

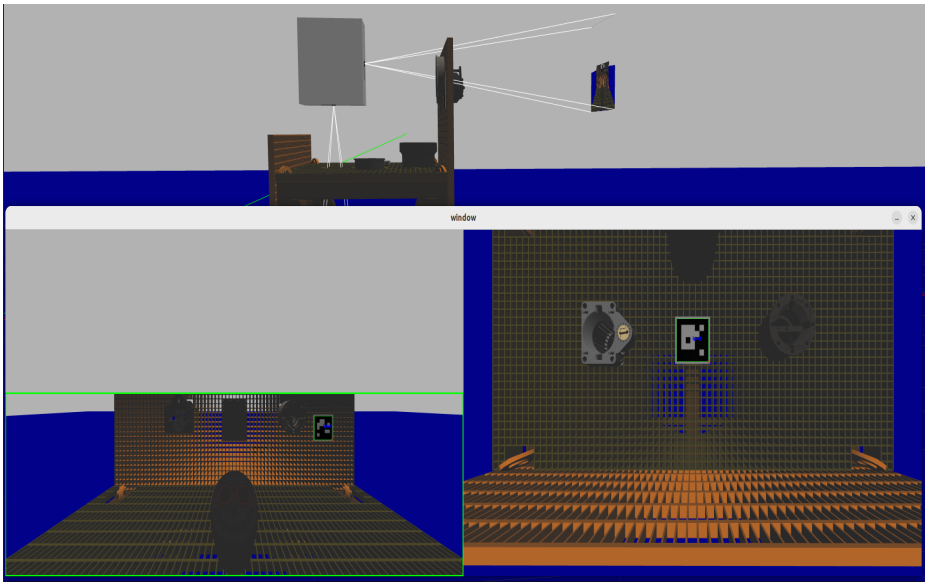
Kamera

De kameraene som brukes til å lokalisere objekter ved inspeksjon, er plassert på forsiden og undersiden av ROV-en. For å simulere kameraer med samme spesifikasjoner som de på den virkelige ROV-en, brukes *Gazebo* sin programvareutvidelse for kameraer, med følgende parametre:

- Oppløsning: 1920x1080(full HD)
- FOV: 150°
- Maks FPS: 30
- Minstedistanse for fokus: 20-30 cm

Figur 4.6 viser den simulerte ROV-en med tilhørende kameraer og deres bildestrøm.

[1] For dokumentasjon på sensoren som skal brukes i virkeligheten se: [25].



Figur 4.6: Simulert ROV øverst, bildestrømmen fra kameraene nederst

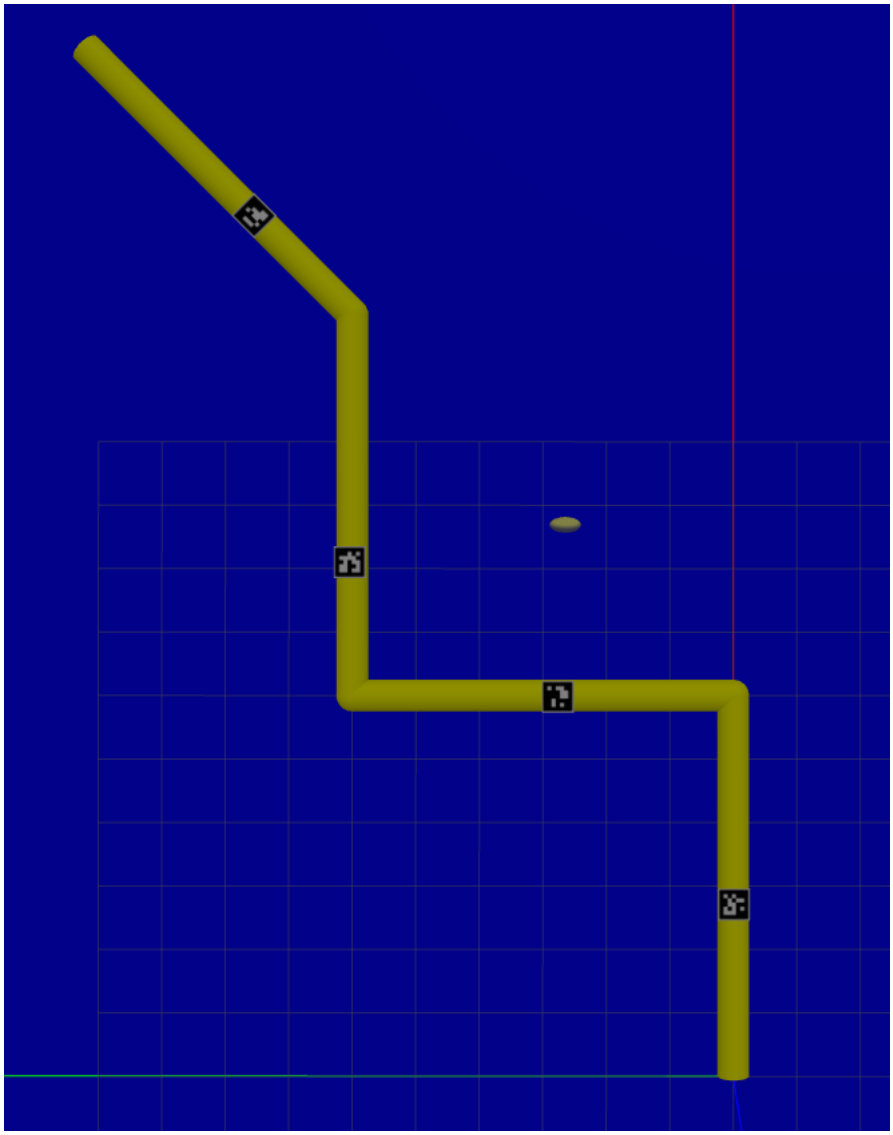
4.3.3 Terreng og miljø

Autonom kjøring skal utvikles for inspeksjon av to forskjellige objekter, i to forskjellige oppdrag: *oppdrag rørledning* og *oppdrag arbeidsbenk*. To forskjellige .world[2] filer utvikles derfor til simuleringen, en fil for rørledningen og en for arbeidsbenken.

Pipeline.World

I .world filen for rørledningen brukes en modell av et gult sylinder, dette kopieres, vinkles og plasseres kant til kant for å etterlikne rørledningen i *TAC challenge*. ArUco-koder er plassert diverse steder på oversiden av rørledningen, som vist i figur 4.7

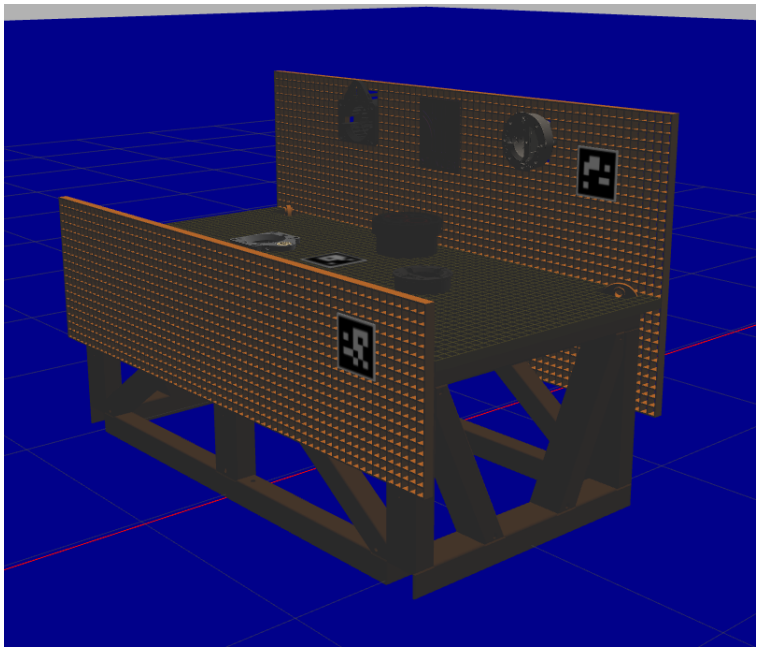
[2].world filer skrives i SDF format, mer om SDF-formatet finnes i vedlegg I.3.



Figur 4.7: Pipeline.world avbildet fra fugleperspektiv

Bench.world

I .world filen for arbeidsbenken brukes en .step utlevert av *TAC Challenge* teamet. Filen implementeres i *Gazebo*, for å etterlikne arbeidsbenken som brukes i *TAC Challenge*. ArUco-koder er plassert diverse steder på arbeidsbenken, som vist i figur 4.8



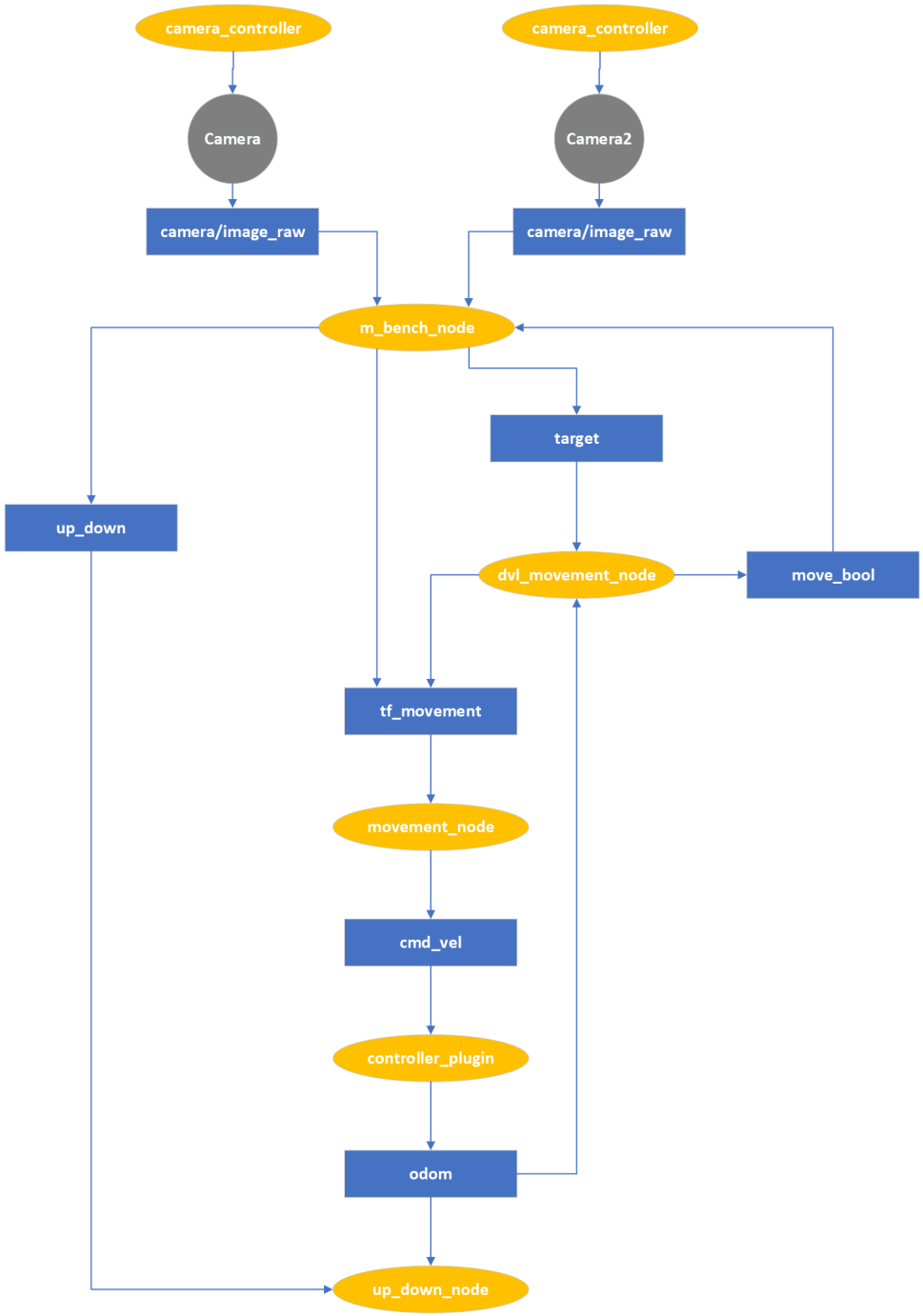
Figur 4.8: Arbeidsbenk modell fra Bench.world

4.4 ROS - programvare og struktur

Dette underkapittelet tar for seg en gjennomgang av nøkkelfunksjonalitetene i ROS2, deretter en gjennomgang av hvordan noder og emner struktureres i programvaren for autonom kjøring.

4.4.1 Informasjonsflyt

Figur 4.9 visualiserer informasjonsflyten mellom noder(gul) og emner(blå) som brukes i simulering av *oppdrag arbeidsbenk*. Ved simulering av *oppdrag rørledning* erstattes m_bench_node med m_pipe_node .



Figur 4.9: Grafisk fremstilling av noder og emner i ROS nettverket

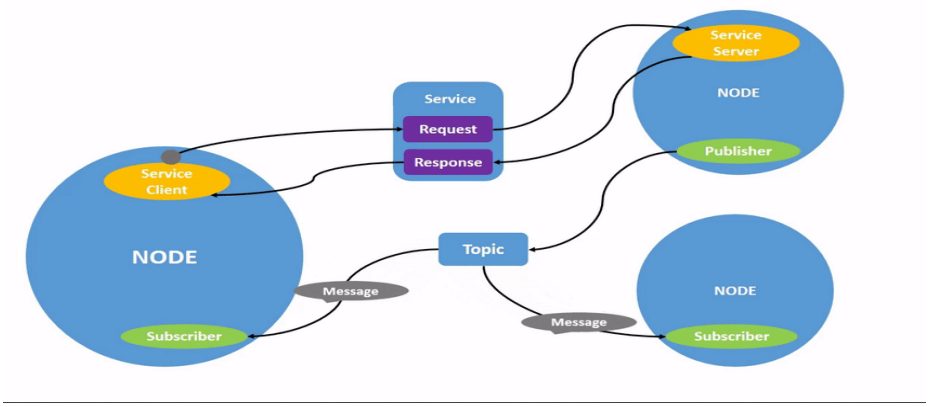
4.4.2 Emner

(Den offisielle dokumentasjonen for emner i ROS)[22] fremhever tre nøkkelord som beskriver emner(Englesk: Topics).

- **Sender *(Eng: Publisher)* / Abonnent *(Eng: Subscriber)***
  Dersom et emne eksisterer kan alle noder sende til eller abonnere på emnet. For eksempel kan to noder sende til et emne, mens ti noder abonnerer på det samme emnet.

- **Anonymt**
  Emner fungerer anonymt, dette vil si at når en abonnent leser fra et emne, er den likegyldig til hvem som har sendt til emnet. Slik kan sendere/abonnenter byttes ut uten å påvirke resten av nettverket.

- **Sterke typer**
  For å kommunisere via et emne brukes en meldingstype. Det er da nødvending at sender og abonnent er enige om meldingstype. Det finnes predefinerte meldingstyper i ROS, en kan også definere egne meldingstyper.
  Et eksempel på en forhåndsdefinert meldingstype fra ROS er *Twist*.
  Twist meldinger ser slik ut: {linear.(x,y,z), angular.(x,y,z)}, disse er designet for å sende både posisjon og rotasjonsdata for alle tre akser i en melding.



Figur 4.10: Trykk på bildet for å se hvordan et emne brukes for å publisere data fra en node til to separate noder. [16]

I tabell 4.1 vises emnene som brukes i nettverket som utgjør programvaren for autonom kjøring, de fleste emnene bruker meldingstypen *Twist*. Denne meldingstypen brukes fordi den kan sende posisjonsdata for alle de seks bevegelsene til ROV-en i en enkelt melding.

Emne	Type	Bruk
Camera/image_raw	Image	Bildestrøm kamera(nedover)
Camera2/image_raw	Image	Bildestrøm kamera(forover)
target	Twist	Koordinater for DVL-bevegelseskommandoer
tf_movement	Twist	Kommandoer for ønsket bevegelse
odom	Odometry	Informasjon om ROV-ens orientasjon
cmd_vel	Twist	PMP lytter til denne og flytter ROV-en deretter
up_down	Float32	Koordinat for teleportering langs Z aksen
move_bool	Boolean	Tilstandsvariabel for blind bevegelse

Tabell 4.1: Tabell over emnene i nettverket

4.4.3 Noder

(Den offisielle dokumentasjonen for noder i ROS)[23], oppfordrer til å designe systemer slik at en node utfører en spesifikk oppgave.
I et slikt design kjører noder uavhengig av hverandre og systemet er derfor modulært.

Noder kan settes opp til å ha sin egen livssyklus(aktiv, pauset, inaktiv). Disse kan utvikles i Python og C++. Noder utviklet i forskjellige språk kan fortsatt kommunisere mellom hverandre. Dette er fordi nodene ikke forholder seg til hverandre direkte men kommuniserer igjennom topics.

Programvare for autonom kjøring skal utvikles til to forskjellige oppdrag, ROV-en trenger mange av de samme funksjonalitetene i begge oppgavene. Derfor struktureres nodene i ROS-nettverket slik at felles funksjoner samles i felles noder. Til de to spesifikke oppdragene(arbeidsbenk og rørledning) utvikles to individuelle oppdragsnoder.
Oppdragsnodene kombinerer fellesnodene med programvare for bildebehandling og regulering, med det formål å navigere ROV-en slik at den fullfører sitt oppdrag.

Videre forklares de egenproduserte nodene fra tabell 4.2. Nodene rsp, gazebo, camera_controllerx og object_controller er ikke egenproduserte, disse er del av gazebo og programvareutvidelsene i prosjektet.

Node	Funksjonalitet
rsp	Publiserer informasjon om ROV
gazebo	Kjører gazebo via ROS
movement_node	Simulerer transfer functions for virkelighetsnær bevegelse
dvl_movement_node	Regulerer ROV-ens posisjon i forhold til seg selv
camera_controllerx	Generer bildestrøm med *Gazebo camera-plugin*
Object_controller	Kontrollerer ROV
mission_bench_node	Oppdragsnode for *oppdrag arbeidsbenk*
mission_pipeline_node	Oppdragsnode for *oppdrag rørledning*
up_down_node	Teleporterer ROV langs z-aksen

Tabell 4.2: Oversikt over nodene i nettverket.

4.4.4 movement_node

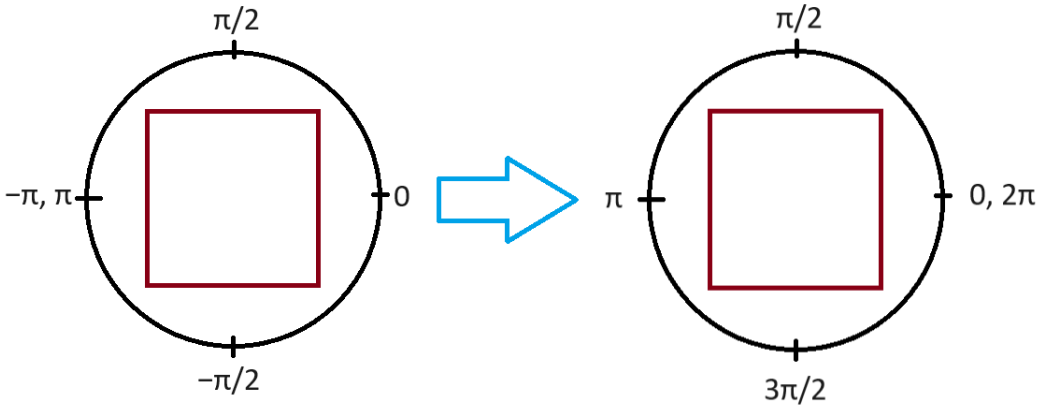
Uten gravitasjon eller avanserte modeller for simulert vann blir bevegelsene i simuleringen ideelle og gir et dårlig grunnlag for å simulere realistisk PID-regulering.
Denne noden fungerer som en overføringsfunksjon, da er inndataen ønsket bevegelse og utdataen er et mer realistisk substitutt for denne bevegelsen.
Noden implementerer klassen **transfer_funtion_class**[3], for å transformere ønskede bevegelser til mer realistiske bevegelser. Noden henter pådragsdataen for ROV-en fra emnet *tf_movement*, og kjører dataen igjennom de aktuelle tilstandsmodeller[4]. Deretter sendes den behandlede dataen videre til emnet *cmd_vel*. ROV-ens kontroller leser direkte fra *cmd_vel*.

[3]Klassen transfer_funtion_class blir beskrevet i delkapittel 4.6.2.
[4]Se vedlegg A.4.4 for en forklaring på implementasjon av tilstandsmodeller i Python.

4.4.5 dvl_movement_node

Denne noden har som oppgave å regulere ROV-ens posisjon og orientasjon ved bruk av DVL-sensor. For å utføre denne oppgaven brukes følgende:

1. **Lister for posisjoner og målposisjoner:** listen **pos** holder posisjonsdata for ROV-en, listen **target_pos** holder målposisjoner. Disse listene bruker samme rekkefølge: $[x, y, z, rull, stamp, gir]$.
2. **En funksjon som henter ROV-ens odometri:** for å hente ROV-ens odometri brukes funksjonen **odom_callback()** for å abonnere på emnet *odom*[5]
   Funksjonen fungerer som følger:
   (a) Henter posisjonen og rotasjonen til ROV-en.
   (b) Gjør om rotasjonsdata fra *quaternions* til radianer.
   (c) Endrer rekkevidden for rotasjon fra $(-\pi, \pi)$ til $(0, 2\pi.)$ som vist i figur 4.11



Figur 4.11: Visuell demonstrasjon av omgjøringen i rekkevidde for rotasjon.

3. **Omregning i enhetssirkelen for forenklet regulering av rotasjon:** Rekkevidden for rotasjonsdata endres for å unngå sprang i PID-regulering når ROV-ens rotasjon ruller over fra π til $-\pi$.
   Disse utgjør samme posisjon på enhetssirkelen, men et snudd fortegn oppfattes av PID-regulatoren som et stort sprang i avvik.
   Hjelpefunksjonen **reset_pi()** trekker 2π fra målverdier over 2π når ROV-ens rotasjon overskrider verdien 2π og ruller over til 0.
   Slik ROV-en kan rotere forbi punktet 2π på enhetssirkelen istedenfor å rotere unødvendig langt i motsatt retning for å nå samme posisjon.
4. **En funksjon som gjør om fra globale til lokale koordinater:** funksjonen **xyt-rig()** brukes for å gjøre om globale koordinater til lokale koordinater. Som vist i likning

[5]odom emnet publiserer ROV-ens odometri, og er en del av kontrolleren beskrevet i delkapittel 4.3.1.

(4.1) vekter funksjonen globale x og y koordinater basert på ROV-ens gir. Denne funksjonen brukes når nye reisemål settes og når ROV-ens posisjon reguleres i forhold til reisemålene.

$$\begin{aligned} local_x &= global_x \cdot cos(gir) + global_y \cdot sin(gir) \\ local_y &= -global_x \cdot sin(gir) + global_y \cdot cos(gir) \end{aligned} \tag{4.1}$$

5. **En funksjon som setter reisemål:** funksjonen **move_pos_callback()**[6] bruker meldingen motatt på emnet *target* for å sette mål for ROV-en i listen **target_pos**.

6. **En funksjon som beveger ROV-en mot reisemålene sine:** funksjonen **check_goal_pose()**[7] beregner differansen mellom ROV-ens posisjon og mål og bruker differansen som referanse i PID-regulator, for flytte ROV-en mot sitt mål. Funksjonen tar inn frihetsgrad som argument og anvender PID-regulatoren til den spesifikke frihetsgraden.

7. **En hjem-funksjon:** når **home1()**[8] funksjonen kalles kjører ROV-en til bestemt hjemposisjon. I simuleringen regnes hjem for ROV-en som $x = 0$, $y = 0$ og $gir = 0$.

4.4.6 up_down_node

I *oppdrag arbeidsbenk* kreves bevegelse langs z-aksen for ROV-en, slik at denne kan scanne både rundt og oppå benken. Da ROV-ens kontroller i Gazebo ikke støtter bevegelse langs z-aksen anvendes en alternativ løsning på dette problemet. En løsning som fungerer til tross for kontrolleren er å teleportere ROV-en langs z-aksen med egne *service calls* i *Gazebo*.
Når denne noden oppdager en endring i verdi på emnet *up_down* bruker den ROV-ens odometri for å teleporterte til akkurat samme x,y lokasjon men med z verdi lik den fra meldingen i up_down emnet.

4.4.7 m_bench_node

Denne noden fungerer som oppdragsnoden til *oppdrag arbeidsbenk*. Noden sørger for å navigere ROV-en med den hensikt å utføre *oppdrag arbeidsbenk*. I tabell 4.3 finnes en oversikt over funksjonene til noden.

[6] Flytskjemaet i vedlegg G.4 viser flyten i funksjonen **move_pos_callback()**.
[7] Flytskjemaet i vedlegg G.5 viser flyten i funksjonen **check_goal_pose**.
[8] **home1()** kalles ved å sende en bevegelseskommando med distanse lik 0, som vist i flytskjema G.4.

Funksjon	Kommentar.
cam1_callback	Oppdaterer bildestrømmen fra kamera 1 til imageHandler.
cam2_callback	Oppdaterer bildestrømmen fra kamera 2 til ImageHandler.
bool_callback	Tilbakestiller move_bool til False, etter fullført bevegelse i dvl_movement_node.
timer_callback	Kaller bench_sequence funksjonen hvert femte sekund.
cam_info_get	Bruker funksjonen find_bench fra figur G.3 for å finne benkens posisjon.
bench_sequence	Funksjon som kaller forskjellige bevegelseskommandoer for ROV-en basert på tilstand.
move_pos(axis,distance)	Setter move_bool = True og sender bevegelseskommandoer til dvl_movement_node.
camera_regulator(key, accuracy)	Regulerer ROV-ens posisjon og orientasjon i forhold til benken.
publish_z	Sender verdi til up_down_node for å teleportere langs z aksen .
send_movement	Sender bevegelser direkte til ROV-en, brukes av camera_regulator.

Tabell 4.3: Oversikt over funksjonene i m_bench_node.

Sekvens: visuell inspeksjon av arbeidsbenk

For å identifisere alle ArUco-koder plassert på benken kreves det at ROV-en reiser rundt hele benken med kamera rettet mot denne. For å oppnå dette går ROV-en igjennom en sekvens[9] med bevegelser. For gjennomgang av sekvensen brukes variabelen mode, i funksjonen **bench_sequence()**, som er en forenklet endelig tilstandsmaskin. I **bench_sequence()** kalles forskjellige bevegelseskommandoer med enten kameraregulering eller DVL-regulering, basert på verdien til mode variabelen, som inkrementeres når en bevegelse fullføres.

move_pos

Denne funksjonen kalles for å sende en forespørsel om DVL-basert bevegelse til dvl_movement_node. Når forespørselen sendes inkrementes variabelen mode med 1, og move_bool blir satt til True. Når bevegelsen er fullført tilbakestilles move_bool til False.

camera_regulator

Denne funksjonen brukes for å regulere posisjonen til ROV-en basert på informasjonen hentet ut med funksjonen **cam_info_get**. Denne funksjonen tar argumentene *key* og *accuracy*. Følgende streng-verdier for *key* fører til PID-regulering mot følgende referanser:

- align: Regulerer gir mot vinkelen på boksen i forhold til kameraet på ROV-en, dette gjøres for å sørge for at ROV-en ser direkte mot benken og dermed har ett godt utgangspunkt for videre regulering.
- slide_in: Regulerer svai og teleporterer for å posisjonere ROV-en slik at den kan skli inn over benken uten å kræsje i den.

[9] Flytskjemaet i vedlegg G.6 viser sekvensen av bevegelser som anvendes i funksjonen **bench_sequence** for å kjøre ROV-en rundt benken.

- center, middle_left eller middle_right: regulerer svai slik at ønsket del av benken plasseres i midten av kameraets horisontale akse.
- distance: regulerer jag med størrelsen til benken som referanse, slik holder ROV-en en bestemt distanse fra benken.

Når avviket i regulatoren aktivert via **camera_regulator** har regulert vekk avviket til sin referanse, inkrementeres variabelen mode med 1.

4.4.8 m_pipeline_node

Oppdragsnoden for *oppdrag rørledning* fungerer ikke på samme måte som den for *oppdrag arbeidsbenk*. Denne bruker likevel mange av de samme prinsippene, emnene og nodene som i *oppdrag arbeidsbenk*. ROV-en befinner seg direkte over rørledningen ved start.

Noden anvender følgende logikk for å gjennomføre oppdraget:

- Bildestrøm sendes kontinuerlig til ImageHandler
- Vinkel og rørposisjon hentes fra ImageHandler
- ROV-en gis en konstant fart forover(jag).
- ROV-ens gir reguleres mot vinkelen på røret, slik vil ROV-en prøve å følge røret og svinge når røret svinger.
- Ved å samtidig regulere svai i forhold til midten av røret vil ROV-en forsøke å holde seg på midten av røret.
- Når det ikke lengre finnes rør i bildestrømmen sendes bevegelsen (0,0) til *dvl_movement_node* og trigger **home1()** funksjonen.

4.5 Programvare for bildebehandling

Store deler av reguleringen i prosjektet baserer seg på lokalisering av objekter ved bruk av bildebehandling. Det å lokalisere objekter innebærer som regel kombinert bruk av mange funksjoner, derfor utvikles det et bibliotek med statiske metoder basert på openCV. I tillegg utvikles det en klasse som kombinerer disse metodene for å utvinne relevant informasjon fra bilder.

4.5.1 Statiske metoder

Biblioteket for Statiske metoder kalles ImageMethods[10], tabell 4.4 gir en oversikt over metodene og hvordan de brukes.

Funksjon	Parametere	Returnerer	Bruksområde
scale_image	image, scale_factor	Skalert bildet	Skalerer bildet
fix_hsv	image	Fikset hsv	Gjør om fra HSV til BGR for display av HSV bildet sammen med BGR bildet
saveImage	image	N/A	Lagrer bildet
showImage	image, scale	N/A	Viser bilde, kaller saveImage ved tastetrykk 's'
close_image	image, rate	Filtrert bildet	Gjevner ut svarthvitt bilde.
stack_images	images	Ett bilde	Setter flere bilder sammen til ett bilde.
color_filter	image_inn, range:list	Svart hvitt bilde	Fjerner unødvendig informasjon i bilder.
find_boxes	hsv_image, original_image, min_box_size, draw	Liste over bokser	Finner bokser(firkanter) i bildet
find_biggest_box	image, boxes, draw	Største boks	Finner størst boks i liste
find_highest_box	box_list	box	Finner boks høyest i bildet
get_box_info	box	positions, area	Finner størrelse og posisjonen til og mellom hjørner i en boks.
find_center	image, box	center_x, center_y	Finner sentrum av boks
find_angle_box	box, offset	angle_deg	Finner vinkel mellom lengste vektor og den horisontale/vertikale akse i bildet
angle_cooldown	angle_deg, cooldown	angle_deg, cooldown	Hindrer flimring ved 90° vinkler
read_ArUco	image, id_list	id_list	Finner ArUco koder i bildet
filtered_ids_list	list	filtered_list	Fjerner duplikater i listen
quaternion_to_euler	x,y,z,w	roll_x, pitch_y, yaw_z	Konverterer fra kvaternionen til Eulervinkler

Tabell 4.4: Tabell over statiske metoder

4.5.2 ImageHandler

Klassen som anvender de statiske metodene fra tabell 4.4 kalles ImageHandler. Denne initialiseres som et objekt, og mates deretter kontinuerlig med bildestrømmen fra ROV-ens kameraer. I oppdragsnodene kalles funksjoner fra ImageHandler for å lokalisere objekter i bildestrømmen til ROV-en.

ImageHandler har 4 funksjoner:

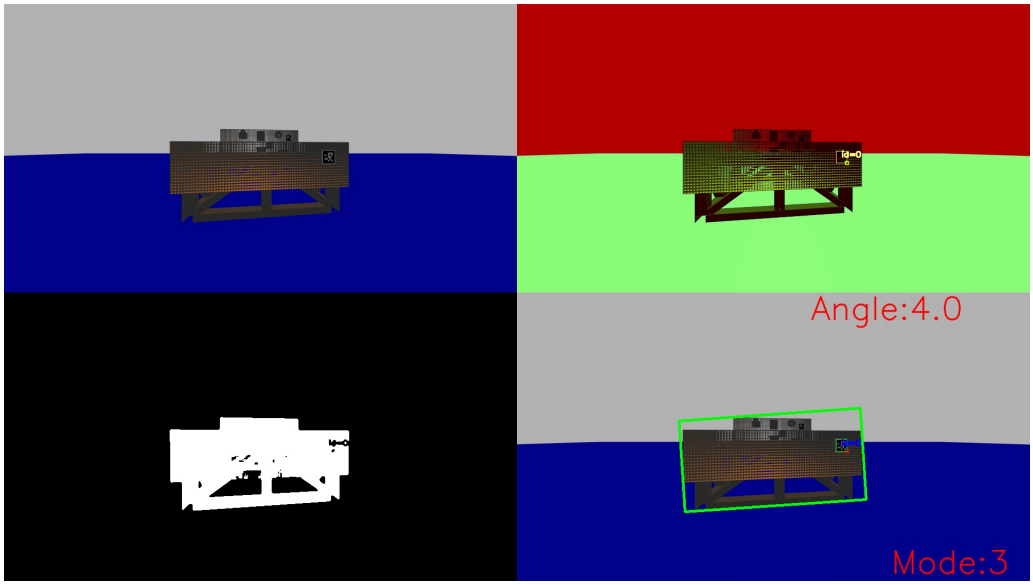
- **ArUco_handler():** Bruker funksjonen **read_ArUco()** fra tabell 4.4, for å sjekke bildestrømmen til ROV-en for ArUco-koder.
- **filter_ArUcos():** Fjerner alle duplikater i en liste
- **find_bench()**[11]**:** Brukes i *oppdrag arbeidsbenk*, funksjonen beregner estimater for benkens avstand, vinkel og posisjon i forhold til ROV-en, disse beregningene baserer seg på størrelsen, vinkelen og posisjonen til boksen vist i figur 4.12.

[10] For programkode se vedlegg J.7

[11] Flytskjemaet i vedlegg G.3 illustrerer prosessflyten og informasjonsflyten til funksjonen **find_bench()**.

- **find_pipe()**[12]**:** Brukes i *oppdrag rørledning*, funksjonen beregner estimater for rørets posisjon og vinkel i forhold til ROV-en, disse beregningene baserer seg på vinkelen og posisjonen til boksen i figur 4.13.

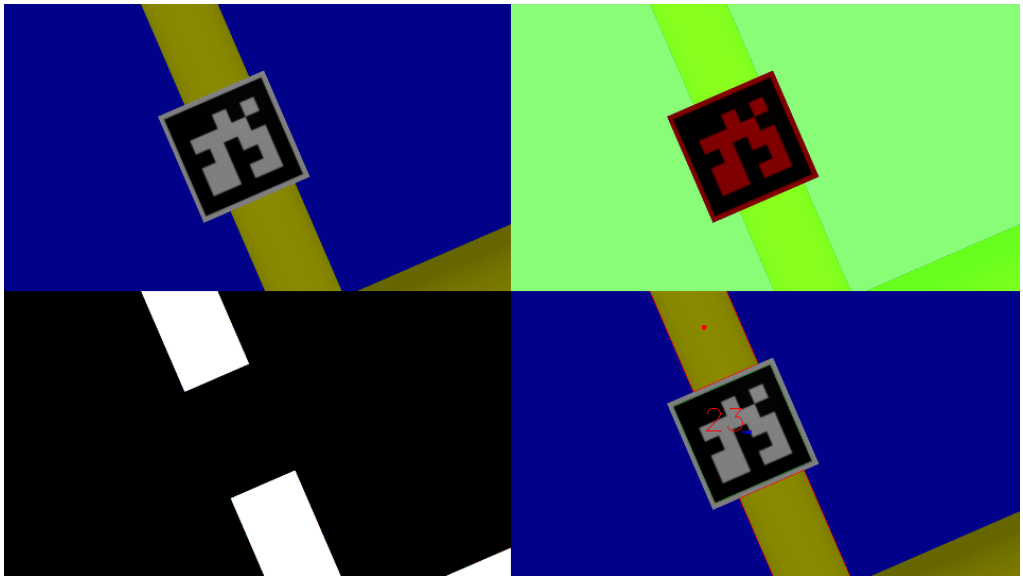
Figur 4.12 illustrerer de ulike stegene i bildebehandlingsprosessen som anvendes for å lokalisere arbeidsbenk med funksjonen **find_bench()**. Under testing gir denne funksjonen stabile og robuste resultater. Benken oppdages med korrekt lokasjon og avstand så lenge den er innenfor kamerarekkevidden.



Figur 4.12: Bildebehandling arbeidsbenk, steg for steg.

Figur 4.13 illustrerer de ulike stegene i bildebehandlingsprosessen som anvendes for å lokalisere rørledning med funksjonen **find_pipe()**. Under testing gir denne funksjonen stabile og robuste resultater. Rørlinjen oppdages med korrekt lokasjon så lenge den er innenfor kamerarekkevidden.

[12]Flytskjemaet i vedlegg G.2 illustrerer prosessflyten og informasjonsflyten til funksjonen **find_pipe()**.



Figur 4.13: Bildebehandling rørledning, steg for steg.

Under testing av funksjonene **find_bench()** og **find_pipe()** viser funksjonen **ArUco_handler()** seg å være meget robust, det vil si at alle ArUco-koder alltid detekteres, så lenge de er innenfor kamerarekkevidde.

4.6 Implementasjon av Kontrollere og overføringsfunksjoner

PID-kontrollere brukes for å regulere ROV-ens posisjon, overføringsfunksjoner anvendes for å simulere realistiske responser til pådrag, som igjen gir et mer realistisk grunnlag for justering av PID-parametere.

4.6.1 PID-kontroller

PID-regulator implementeres med klassen PidController som består av funksjonene:

1. **calculate_offset():** Beregner avvik, tabell 4.5 beskriver inn- og utdata for funksjonen.
2. **Pid_Controller():** Brukes for å implementere en diskret PID-regulator. Tabell 4.6 beskriver inn- og utdata for funksjonen.

calculate_offset	
Argumenter	
meassured_value	målt verdi
set_point	settpunkt
Returnerer avvik	

Tabell 4.5: Parametere for calculate_offset

4.6

Pid_Controller	
Argumenter	
e	Avvvik
P	P-ledd
I	I-ledd
D	D-ledd
T_f	T_F er skaleringsfaktoren for filteret til D-leddet
scale_devide	Skalering som P,I og D-leddende blir dividert med
margin	Avvik hvor regulatoren ikke vil gi ut 0 pådrag
Returnerer fart	

Tabell 4.6: Parametere for PID_controller

Logikken for **Pid_Controller()** funksjonen, er basert på ligning 5.11 i kapittel 5.1. Der PID-regulatoren er diskret, med diskrete P, I og D-verdier.

4.6.2 transfer_funtion_class

Overføringsfunksjoner brukes for å simulere realistiske bevegelser til ROV-en. Disse implementeres i klassen transfer_funtion_class. Klassen initialiseres med argumenter for teller og nevner i overføringsfunksjon. Funksjonen **implement_transfer_function()** brukes deretter med inndata som parameter for å returnere utdata påvirket av overføringsfunksjonen. Kapittel 3 forklarer hvordan overføringsfunksjonene i simulatoren beregnes og implementeres.

4.7 Konklusjon

Simulatoren fungerer som en god plattform for utvikling av programvare som senere kan brukes i den virkelige ROV-en. I løpet av prosjektet, har simulatoren stått som rammeverk for mesteparten av all utvikling og testing utført i for emnet autonom kjøring. De fleste løsninger i prosjektet er blitt mere robuste fordi de har blitt optimaliserte via omfattende

testing i simulator. Løsninger for regulering og behandling av sensordata har sett spesielt stor forbedring etter feilsøking i simulator.

Kapittel 5

Regulering

Innhold

Dette kapittelet gir en gjennomgang av den matematiske oppbygningen til PID-regulatoren. Videre forklares utledningen av PID-parametre ved bruk av Skogestad-metoden, samt manuell justering av disse parametrene i simulator.

5.1 PID-regulator

For regulering av farten til ROV-en i de forkskjellige frihetsgradene brukes PID-regulatorer. En PID-regulator gir ut pådrag basert på avvik fra et referansepunkt, og består av tre forskjellige ledd:

- Proporsjonal ledd(P-ledd)

- Integral ledd(I-ledd)
- Derivasjons ledd(D-ledd)

Pådraget beregnes som summen av P,I og D-leddene.

5.1.1 Proporsjonal ledd

P-leddet ser kun på avviket, der pådraget er en skalerings faktor K_p multiplisert med avviket. Det finnes regulatorer som kun bruker P-leddet (P-regulator). Ligning (5.1) beskriver den tidskontinuerlige versjonen av P-leddet.

$$u_p(t) = K_p \cdot e(t) \tag{5.1}$$

Diskret proporsjonal ledd

Simuleringen er ikke tidskontinuerlig, den er diskret, ligning (5.2) viser den diskrete versjonen av P-leddet.

$$u_p(k) = K_p \cdot e(k) \tag{5.2}$$

5.1.2 Integral ledd

I-leddet ser på avviket integrert over tid multiplisert med en skalerings faktor K_i. I-leddet bestemmes av størrelsen på avviket over tid, med et konstant avvik vil I-leddet økes proporsjonalt med tiden. I-leddet sin hovedoppgave er å fjerne restavvik. Ligning (5.3) beskriver den tidskontinuerlig versjonen av I-leddet.

$$u_i(t) = K_i \cdot \int_0^t e(\tau)d\tau \tag{5.3}$$

Diskret integral ledd

For numerisk integrasjon brukes trapesformellen som er beskrevet i ligning (5.4).

$$\int_a^b x(t) \approx (b-a)\frac{x(a)+x(b)}{2} \tag{5.4}$$

I-leddet for den diskrete PID-reglatoren blir som i ligning (5.5). T_s er tidsskrittet til sytemet.

$$u_i(k) = u_i(k-1) + K_i \cdot T_s \cdot \frac{e(k) + e(k+1)}{2} \tag{5.5}$$

For å ha en øvre grense for pådraget fra integral leddet implementeres ofte en $U_{I_{max}}$ som begrenser maks pådrag fra I-leddet.

5.1.3 Derivasjons ledd

D-leddet ser på avviket derivert multiplisert med en skalerings faktor K_i. D-leddet bestemmes av hvor fort avviket endres, jo raskere endring jo større pådrag fra D-leddet. Ligning (5.6) beskriver den tidskontinuerlige versjonen av D-leddet.

$$u_d(t) = K_d \cdot \frac{d}{dt} e(t) \tag{5.6}$$

Diskret derivasjons ledd

For numerisk derivasjon brukes definisjonen av derivasjon beskrevet i ligning (5.7).

$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{5.7}$$

Gjør ligning (5.7) om til en som kan brukes diskret der T_s er tidsskrittet vist i ligning (5.8).

$$\frac{dx(k)}{dt} = \frac{x(k) - x(k-1)}{T_s} \tag{5.8}$$

For å forhindre at D leddet blir for stort ved et sprang i avvik eller ved støy, brukes en filtrert verdi av avviket $e_f(k)$. Filteret er et IIR-filter er et digitalt filter som fungerer som et lavpass filter. IIR(infinite impulse response) står for uendelig impulsrespons og er et rekursivt filter som ser på nåværende inngang(er) og tidligere utgang(er).
Ligning (5.9) beskriver hvordan IIR-filteret er implementert. T_F er skaleringsfaktoren som bestemmer vektingen av forrige filtrerte verdi mot nyeste målt verdi.

$$e_f(k) = \frac{1}{1 + \frac{T_s}{T_f}} e_f(k-1) + \frac{\frac{T_s}{T_f}}{1 + \frac{T_s}{T_f}} e(k) \tag{5.9}$$

D-leddet for den diskrete PID-reglatoren blir som i ligning (5.10)

$$u_d(k) = K_d \cdot \frac{e_f(k) - e_f(k-1)}{T_s} \tag{5.10}$$

5.1.4 Diskret PID-regulator

Det totale pådraget fra PID-regulatoren blir da summen av pådragene fra P,I og D-leddene som vist i ligning (5.11).

$$u(k) = u_p(k) + u_i(k) + u_d(k) \tag{5.11}$$

5.2 Valg av regulatorparametere

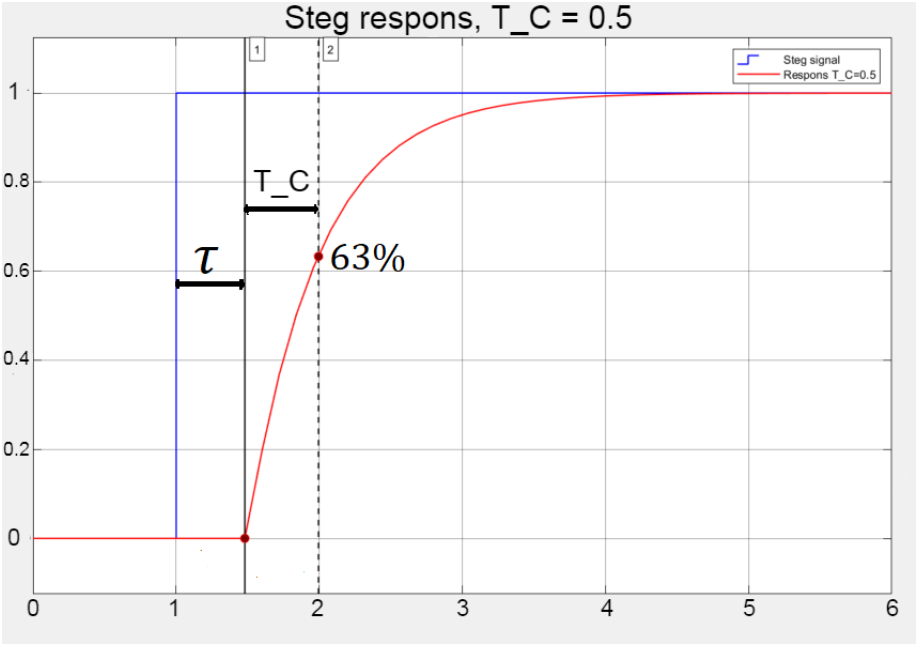
For at PID-regulatoren skal fungere optimalt er det nødvendig å bruke passende regulatorparametere.

5.2.1 Skogstad-metoden

Skogestad-metoden er metode for å finne parametereverdiene til en PID-regulator ut ifra overføringsfunksjonene i et lukket sløyfe system. Metoden forutsetter en lineær modell av systemet. Hvis systemet er av andre orden må overføringsfunksjonen kunne faktoriseres. Målet med å bruke Skogestad-metoden er å få responsene til å oppføre seg som en første-ordens respons som vist i ligning (5.12). Figur 5.1 viser et eksempel på en respons for et første-ordens system,hvor:

- τ er systemet tidsforsinkelse
- T_C er tidskonstanten til systemet, tidskonstanten defineres som: tiden det tar for systemet å nå 63% av ønsket verdi.

$$T(s) = \frac{1}{T_C s + 1} e^{-\tau s} \tag{5.12}$$



Figur 5.1: Eksempel på en første-ordens respons

PID-parameterverdiene for systemet bestemmes deretter ved hjelp av tabell 5.1 utledet av Sigurd Skogestad. Tabellen kan brukes for overføringsfunksjoner skrevet på fem forskjellige former, som vist i prosess-kolonnen.
K_1 er en variabel faktor utarbeidet av Skogestad, denne kan endres for å modifisere oppførselen til systemets respons. En høy verdi for $K_1(4)$ gir gode følge-egenskaper ved endring av referanse. En lav verdi for $K_1(1{,}4)$ gir gode egenskaper for kompensering ved forstyrrelse i referansen.

Prosess	K_p	T_i	T_d
$\frac{K}{s}e^{-\tau s}$	$\frac{1}{K(T_C+\tau)}$	$k_1(T_C+\tau)$	0
$\frac{K}{Ts+1}e^{-\tau s}$	$\frac{T}{K(T_C+\tau)}$	$min[T, k_1(T_C+\tau)]$	0
$\frac{K}{(Ts+1)s}e^{-\tau s}$	$\frac{1}{K(T_C+\tau)}$	$k_1(T_c+\tau)$	T
$\frac{K}{(T_1s+1)(T_2s+1)}e^{-\tau s}$	$\frac{T_1}{K(T_C+\tau)}$	$min[T_1, k_1(T_C+\tau)]$	T_2
$\frac{K}{s^2}e^{-\tau s}$	$\frac{1}{4K(T_C+\tau)^2}$	$4(T_C+\tau)$	$4(T_C+\tau)$

Tabell 5.1: Tabell for bestemmelse av regulatorparametere utledet av Sigurd Skogestad

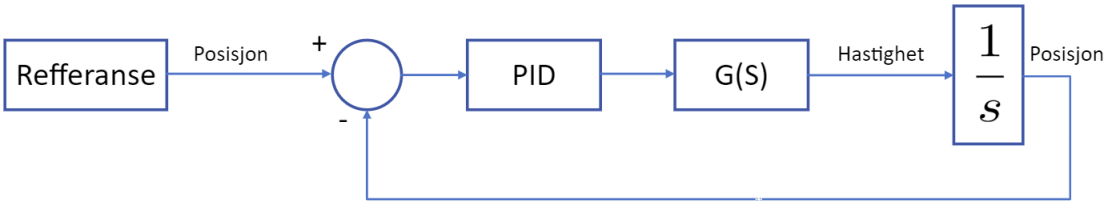
Tabell 5.1 gir parametere for en PID-regulator på seriell form, som de brukt i dette prosjektet. For å omforme fra seriell til parallel form, anvendes følgende formler:

$$K_{pp} = K_p\left(1 + \frac{T_d}{T_i}\right) \tag{5.13}$$

$$T_{ip} = T_i\left(1 + \frac{T_d}{T_i}\right) \tag{5.14}$$

$$T_{dp} = T_d \frac{1}{1 + \frac{T_d}{T_i}} \tag{5.15}$$

Overføringsfunksjonene utledet i kapittel 3.6 kan ikke konverteres til noen av formene i prosess-kolonnen i tabell 5.1, fordi de er av for høy orden. Det ble derfor linearisert nye første-ordens modeller[1] for å finne systemets PID-parametre ved bruk av Skogestad-metoden. Systemet hadde posisjon som referanse og hastighet som utgang, det er derfor lagt til en integrator. Figur 5.2 viser et skjema for prosessen der G(s) er overføringsfunksjonen til en gitt bevegelse.



Figur 5.2: Prosessskjema for bevegelse

Overføringsfunksjonene for systemene til de forskjellige bevegelsene skrives om til samme form som på rad tre i tabell 5.1, der τ er systemets dødtid. Dødtiden[2] til systemet estimeres å være avrundet 0.9 sekunder, altså er $\tau = 0.9$. Det er brukt $T_C = \tau$ som anbefalt i *Finn Haugen: PID Control*[7] og $k_1 = 1.5$, for bedre følge-egenskaper. Standardiseringen av systemene i henhold til rad tre i prosess-kolonnen i tabell 5.1 beregnes i ligningene nedenfor:

Hiv:

$$\frac{9.526}{s + 9.808} \cdot \frac{1}{s} e^{-0.9s} \approx \frac{0.97}{(0.10s + 1)s} e^{-0.9s} \tag{5.16}$$

Gir:

$$\frac{5.21}{s + 5.16} \cdot \frac{1}{s} e^{-0.9s} \approx \frac{1.01}{(0.19s + 1)s} e^{-0.9s} \tag{5.17}$$

Jag:

$$\frac{5.95}{s + 5.89} \cdot \frac{1}{s} e^{-0.9s} \approx \frac{1.01}{(0.17s + 1)s} e^{-0.9s} \tag{5.18}$$

Svai:

$$\frac{7.09}{s + 7.04} \cdot \frac{1}{s} e^{-0.9s} \approx \frac{1.01}{(0.14s + 1)s} e^{-0.9s} \tag{5.19}$$

Tabell 5.2 viser parametrene utledet ved bruk av Skogestad-metoden. Den inneholder verdier for seriell og parallel kontroller.

[1] De nye første-ordens systemene ble utledet med samme fremgangsmåte som i testrapport A.
[2] Testrapport F viser utledningen av dødtiden til systemet.

	K_p	T_i	T_d	K_{pp}	K_{ip}	K_{dp}
Hiv	0.57	2.7	0.1	0.59	2.80	0.10
Gir	0.52	2.7	0.19	0.56	2.89	0.18
Jag	0.52	2.7	0.17	0.55	2.87	0.16
Svai	0.52	2.7	0.14	0.55	2.84	0.13

Tabell 5.2: PID-parametre utledet med Skogestad-metoden

5.2.2 Manuell Justering

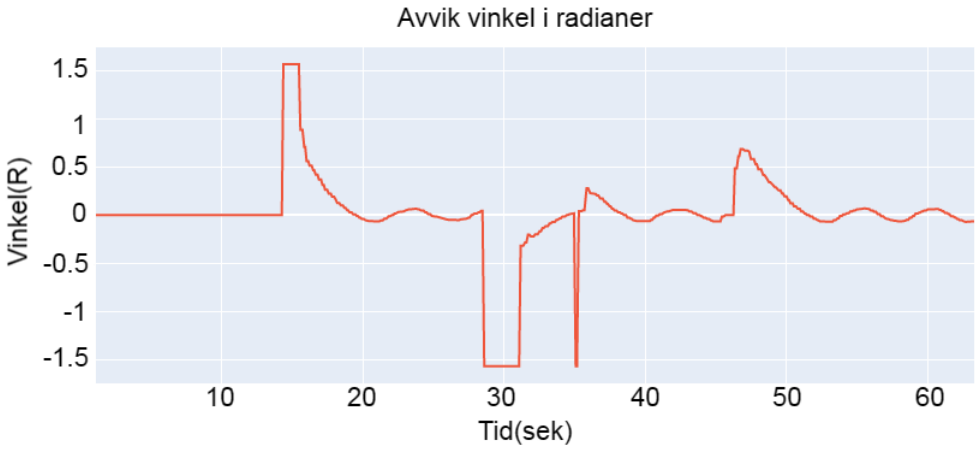
PID-parametrene i tabell 5.2 gir et godt utgangspunkt, men testing og justering av parameterene er nødvendig for å optimalisere reguleringen. I *Oppdrag rørledning* og *opprag arbeidsbenk* reguleres ROV-ens fart, med hensyn til forskjellige referanser og mål, alle regulatorene må derfor finjusteres individuelt.

5.2.3 Inspeksjon av rørlinje

ROV-en skal inspisere rørledningen vist i fra 4.7, rørledningen inneholder flere svinger med opptil 90° krapphet For automatisk inspeksjon av rørlinje kjører ROV-en med konstant jag på 0.2 $\frac{m}{s}$, mens gir og svai reguleres slik at ROV-en følger rørledningen.

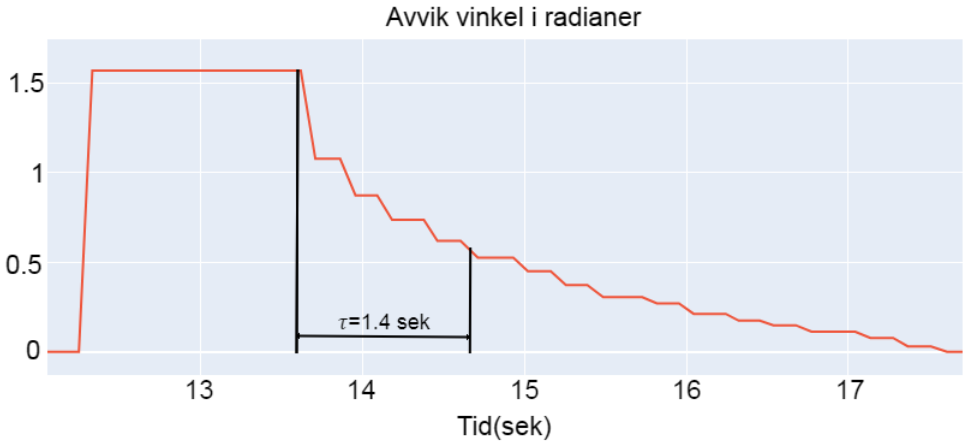
Gir

Gir regulatoren bruker vinkelen til røret under ROV-en som referanse, denne sørger for å rotere x-aksen til ROV-en, slik at den er parallelt med røret. Avviket blir da vinkelen mellom røret og den vertikale kanten i bildestrømmen. Figur 5.3 viser avvik for gir under kjøring av *oppdrag rørledning*, med bruk av PID-parametere fra tabell 5.2 og en $u_{i_{max}}$ lik 0.02.



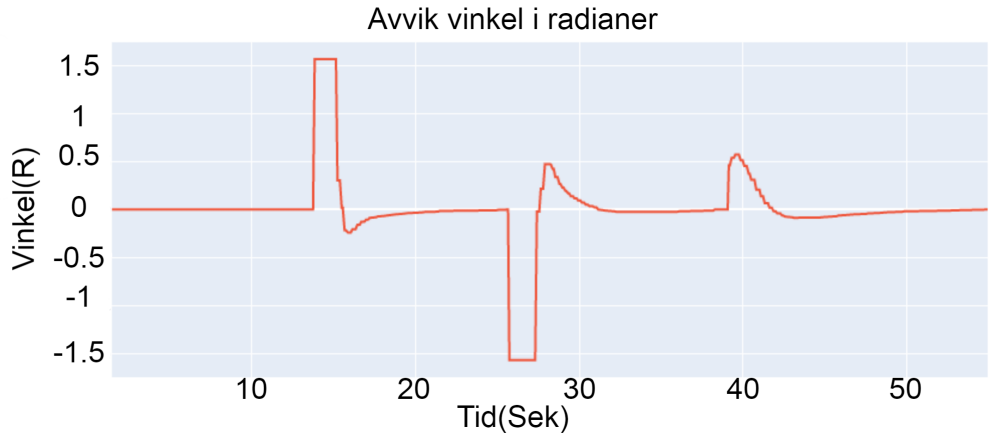
Figur 5.3: Logging av avvik av med PID-parametere fra tabell 5.2

Figur 5.4 viser sprangresponsen i gir ved bruk av Skogestad-parametre. Fra figuren kommer det frem at $\tau = 1.4$, som er 0.5s tregere enn $\tau = 0.9$ anvendt ved beregning av PID-parametre med Skogestad-metoden. Med såpass stor tidsforsinkelse klarer ikke ROV-en å fullføre *oppdrag rørledning*, derfor justeres heller PID-parametrene manuelt.



Figur 5.4: Tidskonstant for gir ved bruk av PID-parametrene fra Skogestad-metoden

Figur 5.3 viser treg stigningstid og oscillering etter sprang. For å minke stigningstiden økes P-leddet. Samtidig reduseres I-leddet kraftig for å redusere oscilleringen. Figur 5.5 viser avvik for gir under kjøring, med $K_p = 1$, $K_i = 0.19$ og $K_d = 0.19$. Med de justerte parametrene blir reguleringen raskere og får lavere innsvingningstid.

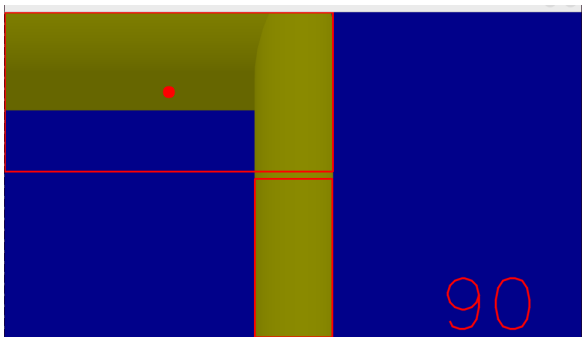


Figur 5.5: Logging av avvik av med manuelt justerte PID-parametre

Svai

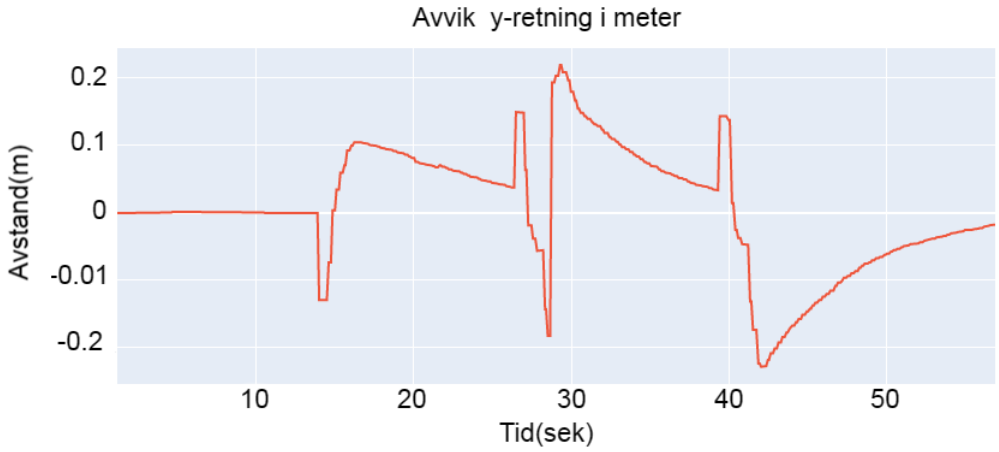
Svai-regulatoren bruker senter av røret under ROV-en som referanse, regulatoren forflytter ROV-en langs y-aksen, slik at den flyter direkte langs senter av røret. Figur 5.6 viser hvordan referansepunkt for svai-regulering feilaktig velges som røret til venstre, der riktig referanse

er røret i midten. For å unngå feilaktig svai-regulering i krappe svinger, begrenses pådraget fra svai-regulatoren når gir-avviket er større enn 45°.



Figur 5.6: En 90° sving fra simulatoren

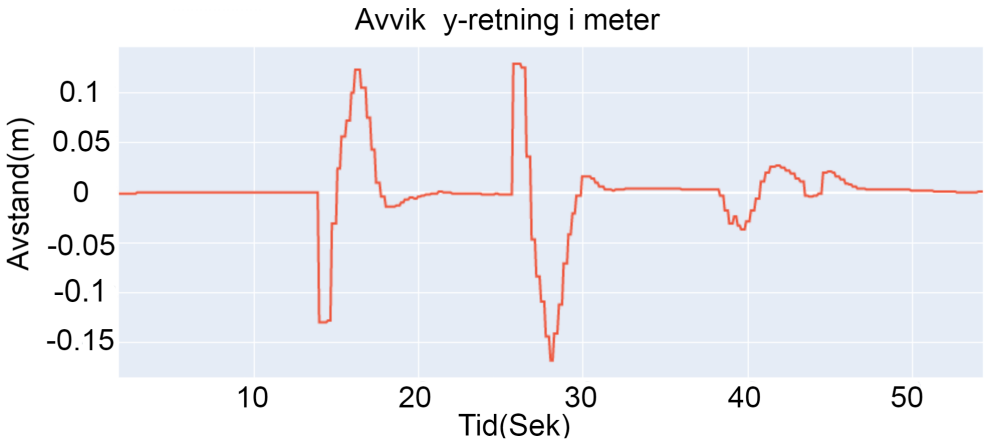
Figur 5.7 viser avviket for ROV-ens svai under kjøring av *oppdrag rørledning*, med PID-parametrene fra tabell 5.2. Under kjøring responderte reguleringen med treg stigningstid. ROV-en klarte heller ikke å fullføre inspeksjonen da den begynte å rotere ukontrollerbart ved sving nummer to.



Figur 5.7: Logging av avvik av med PID-parameter verdiene fra Skogestad-metoden

Ved anvendelse av parametrene utledet fra Skogestad-metoden i svai-regulatoren, blir responsen store oversvingninger og oscillering. Det konkluderes med at systemet fortsatt ikke responderer som forventet av et første-ordens system. Bruk av Skogestad-metoden har altså ikke gitt gode resultater for svai-reguleringen.

For å minke stigningstiden økes P-leddet, I-leddet reduseres samtidig kraftig for å redusere oversving. Figur 5.8 viser avviket mellom ROV-ens svai og midten av røret under kjøring, med $K_p = 6$, $K_i = 0.19$ og $K_d = 0.14$. Med de justerte parametrene er reguleringen raskere og har akseptabel innsvingningstid.



Figur 5.8: Logging av avvik av med justerte PID-parameter verdiene

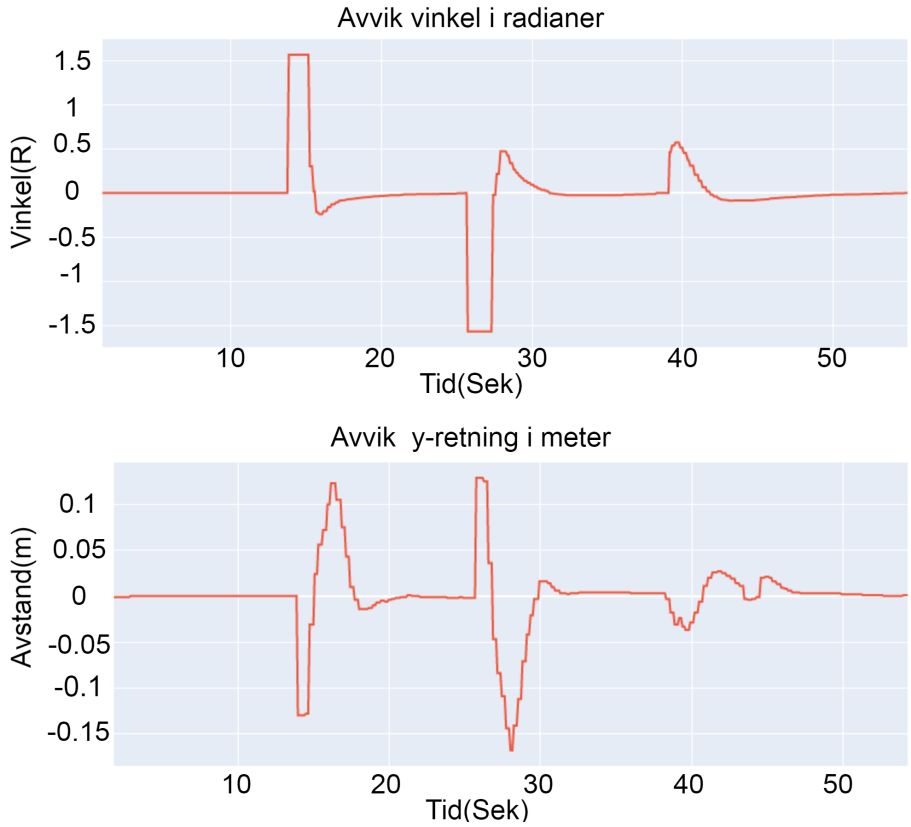
Tabell 5.3 viser parametrene fra Skogestad-metoden og de justerte parametrene som brukes i simulatoren.

	Skogestad			Justert		
	K_p	K_i	K_d	K_p	K_i	K_d
Gir	0.56	2.89	0.18	1	0.19	0.1
Jag	0.55	2.87	0.16	-	-	-
Svai	0.55	2.84	0.13	6	0.19	0.14

Tabell 5.3: PID-parametere for inspeksjon av rørlinjen

5.2.4 Resultat rørlinje

Figur 5.9 viser avviket i gir og svai regulatorene under *oppdrag rørledning*. I figuren illustreres det at gir-regulatoren ser et sprang i referanse ved ca t = 13s, 25s og 40s. Disse sprangene tilsvarer sving venstre 90°, sving høyre 90°og sving venstre 30°. Sprangene korrigeres for det meste i løpet av 2 sekunder, fullstendig korreksjon kan ta opptil 5 sekunder. Ved sammenlikning av grafene kommer det frem at de to regulatorene påvirker hverandre. Gir-regulatoren påvirkes litt av svai-regulatoren, mens svai-regulatoren reagerer ganske kraftig på gir-regulatoren. Når gir reguleres, svinger ROV-en samtidig som svai-regulatoren endrer referanse fra røret før svingen, til røret etter. Denne kombinasjonen fører til svingninger på opptil 15cm i svai-regulatoren, svigningene korrigeres innen 4 sekunder, og representerer derfor ikke et problem.



Figur 5.9: Logging av avvik av vinkel og avstand i y-retning mellom rør og ROV

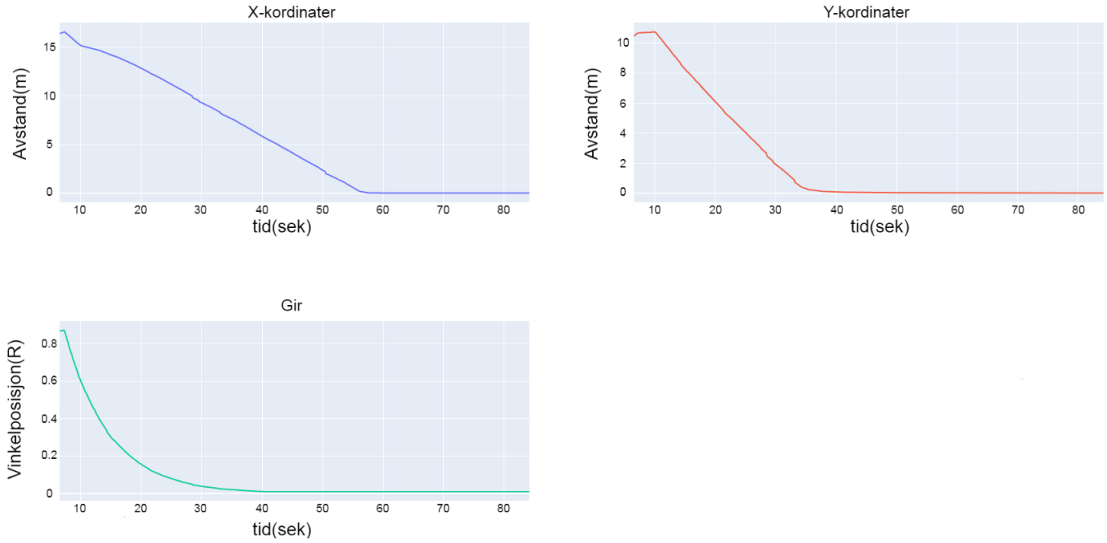
5.2.5 Retur til hjem-posisjon

Etter at ROV-en har gjennomført rørinspeksjon, skal den returnere til hjem-posisjon(x=0, y=0, gir=0). Navigering til nullpunktene utføres ved å bruke posisjondata fra DVL-sensor som referanser for jag, svai og gir. I virkeligheten vil ROV-en kjøre til overflaten før den returnerer til hjem-posisjon, for ikke å kollidere med hindringer i havet. I simulatoren returnerer ROV-en hjem uten å endre høyde. Ved aktivering av hjem-funksjonen, oppstår et sprang i avvik på 10-15m, da ROV-en er relativt langt unna hjem. Skogestad-parametrene er beregnet for avvik på 0.4-0.6m. PID-regulatorene som brukes til retur hjem, justeres derfor manuelt. ROV-en skal returnere med en lav fart, så maks fart i x og y-retning settes til $0.4\frac{m}{s}$. På grunn av stort avvik brukes lave K_p verdier. I-ledd brukes ikke da dette vil vokse for stort fordi avviket er stort over lengre tid. D-leddet settes til 0, fordi selv et lite D-ledd begrenset akselerasjon mer enn det som var gunstig for en effektiv regulering.

	Skogestad			Justert		
	K_p	K_i	K_d	K_p	K_i	K_d
Gir	0.56	2.89	0.18	0.14	0	0
Jag	0.55	2.87	0.16	1	0	0
Svai	0.55	2.84	0.13	1	0	0

Tabell 5.4: PID-parametrene for bevegelse med DVL

Logging av ROV-ens jag, svai og gir ved retur hjem vises i figur 5.10. Fra figuren kommer det frem at jag og svai regulatorene korrigerer avvik ved å holde konstant fart mot referanseposisjonen 0. Disse er rene p-regulatorer og holder konstant fart fordi de begrenses av en forhåndsbestemt toppfart før de gir ut pådrag. Da avstanden er såpass stor forsøker regulatorene å gi enorme verdier for pådrag, dette ville ikke vært verken sikkert eller robust. Gir-regulatoren roterer ROV-en med ROV-ens rotasjon lik 0 som referanse, samtidig som ROV-en kjører hjem.
Resultatet av disse tre regulatorene er en treg, men trygg navigasjon hjem.



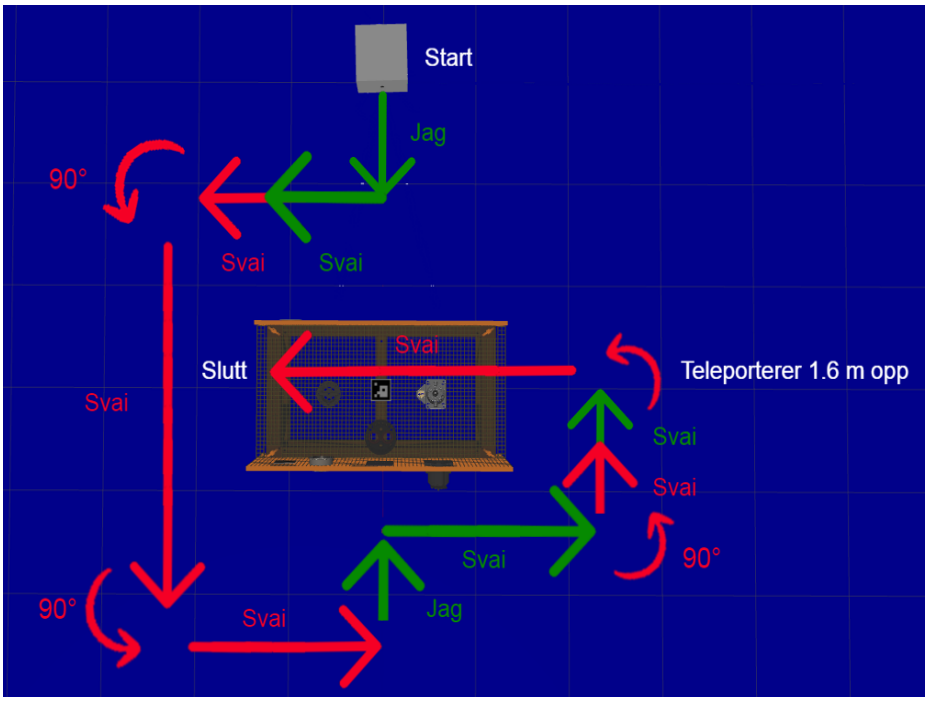
Figur 5.10: Logging av avvik av med justerte PID-parametre

Video av simulatoren for inspeksjon av rørlinje, opptaket av simulatoren er mer hakkete enn ved kjøring uten opptak.

5.2.6 Inspeksjon av arbeidsbenk

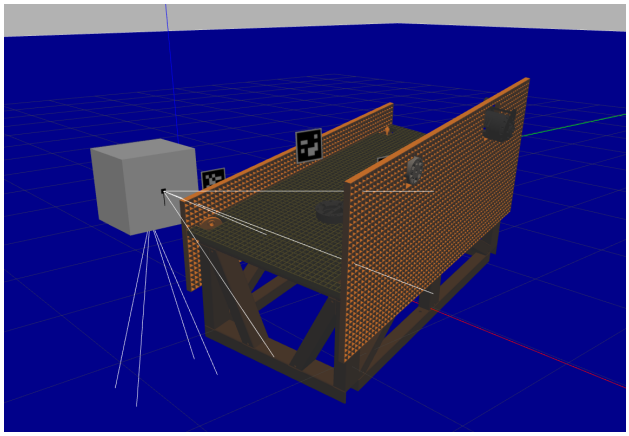
ROV-en skal inspisere arbeidsbenken vist i figur 4.8, inspeksjon utføres rundt, og over benken. Det medfører at ROV-en skal kjøre rundt hele benken med frontkamera rettet mot benken, etter rundturen skal ROV-en også kjøres langs oversiden av benken for å inspeksjon av ArUco-koder.

For å utføre oppgaven brukes fem forskjellige regulatorer, tre av regulatorene bruker posisjonen til arbeidsbenkenm detektert via bildebehandling(videre omtalt som kamera-regulering). To av regulatorene bruker ROV-ens posisjon som referanse fra DVL-sensoren (videre omtalt som DVL-regulering). Figur 5.11 visualiserer hvilken regulator som brukes hvor i oppdraget, der grønn pil indikerer kamera-regulering, rød pil indikerer DVL-regulering.



Figur 5.11: Bildet kartlegger hvilken regulator som brukes hvor. Grønn pil indikerer kamera-regulering, rød pil indikerer DVL-regulering.

ROV-en kan kollidere med benken ved for stor oversving i jag-regulator og ved unøyaktighet i DVL-regulering. Dersom ROV-en reguleres med parametrene fra Skogestad-metoden i tabell 5.3, fører dette til stor oversving og dermed kollisjon, som vist i figur 5.12.



Figur 5.12: ROV-en som kolliderer med benken ved bruk av PID-parametrene fra Skogestad-metoden

Manuell justering av PID-parametre resulterte i parametrene i tabell 5.5.

Referanse	Kamera			DVL	
Frihetsgrad	Jag	Svai	Gir	Svai	Gir
P	0.4	0.4	1.5	6	1.5
I	0.3	0.3	0.3	0.19	0.3
D	0.02	0.02	0.1	0.13	0.1
$u_{i_{max}}$	0.03	0.1	0.01	0.1	0.01

Tabell 5.5: Oversikt over manuelt justerte PID-parametre for *oppdrag arbeidsbenk*.

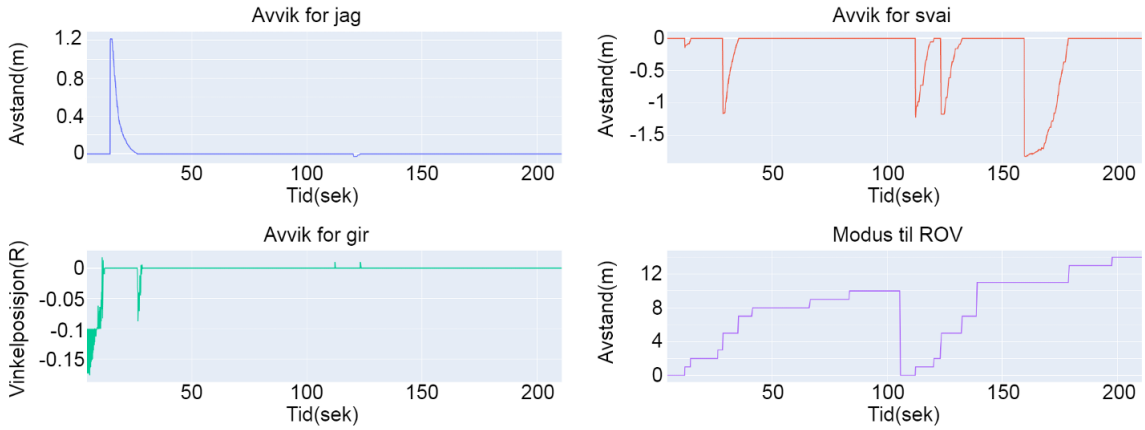
5.2.7 Resultat benk

Under kjøring av operasjon arbeidsbenk logges avvik for regulering av frihetsgradene fra tabell 5.5. Regulatorene som beveger ROV-en kjøres individuelt i en sekvens, når en regulator er aktiv, er de fire andre inaktive. Variabelen mode[3] bestemmer rekkefølgen regulatorene aktiveres i. Når mode inkrementeres sendes en forespørsel om bevegelse til en bestemt posisjon, som resulterer i endring i referanse i tilsvarende regulator. Når avviket til regulatoren reguleres tilbake til 0 er forespørselen om bevegelse til bestemt posisjon fullført, regulatoren deaktiveres og mode inkrementeres. Deaktivering av regulator etter oppnådd målposisjon fører til null oversving. Null oversving er noe urealistisk og må trolig tas høyde for ved rejustering av parametre på den virkelige ROV-en senere.

Sekvensen fra figur 5.11, gir grunnlag for loggedataen i figur 5.13 og 5.14. Ved å sammenlikne verdien av mode med avvikene i de fem forskjellige regulatorene, bygges det en forståelse av hvilken regulator som er aktiv, hvor langt den beveget ROV-en, og hvor lang tid det tok. Fra grafene i figur 5.13 gjøres følgende observasjoner om kamera-regulering:

- Kamera-regulering av jag utføres to ganger for å sørge for riktig avstand mellom ROV-en og benkens fram og bakside.
- Kamera-regulering av gir utføres kun før og etter kamera-regulering av jag, for å sikre at ROV-en peker rettvinklet mot benken og dermed får bedre utgangspunkt til senere DVL-regulering av gir. Dette sørger også for at sekvensen fungerer selvom ROV-en plasseres skeivt mot benken før aktivering av autonom kjøring.
- Kamera-regulering av svai utføres fem ganger, dette gjøres først for å posisjonere ROV-en ved midten av benken, for så å kjøre langs benken, både foran og bak. Til slutt brukes denne regulatoren til å posisjonere ROV-en på midten av siden av benken, før den skal kjøre over benken.

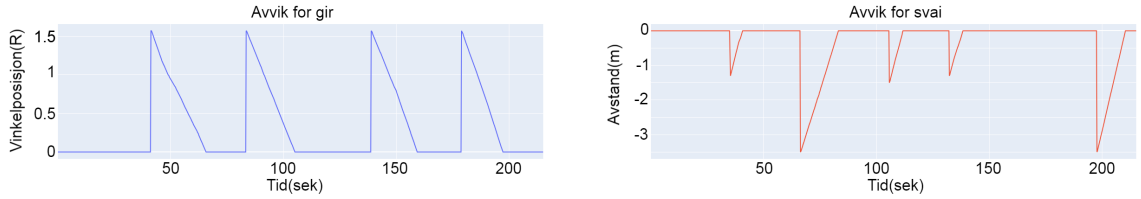
[3]ROV-en følger sekvensen fra figur G.6, som følger forskjellige kommandoer basert på mode.



Figur 5.13: Logging av ROV-ens avvik ved kamera-regulering

Fra grafene i figur 5.14 gjøres følgende observasjoner om DVL-regulering:

- DVL-regulering av Gir viser at ROV-en ved fire forskjellige tidspunkt, roterer 90°, slik at frontkameraet holdes rettet mot benken.
- DVL-regulering av svai brukes for å runde hjørner, eller plassere ROV-en nærmere midten av benken etter å ha rundet et hjørne, den siste svai bevegelsen i grafen, tilsvarer ROV-en som sklir sidelengs over arbeidsbenken.



Figur 5.14: Logging av ROV-ens avvik ved DVL-regulering

Video av simulatoren for inspeksjon av benk, opptaket av simulatoren er mer hakkete enn ved kjøring uten opptak.

Kapittel 6

Fra simulering til virkelighet

Innhold

Dette kapittelet tar for seg integrasjon av programvare fra simulering til virkelighet. Når ROV-en ikke var ferdig til planlagt tid medfører dette alternativ testing av kunstig syn. Objekt deteksjon av rørlinje og arbeidsbenk testes ved hjelp av videopptak fra *TAC Challenge 2023*. Videopptakene brukes også til å finne HSV-områder til bildebehandling under vann. Bildebehandlingen til *oppdrag rørledning* testes på land med håndholdt kamera.

6.1 HSV-område rørlinje

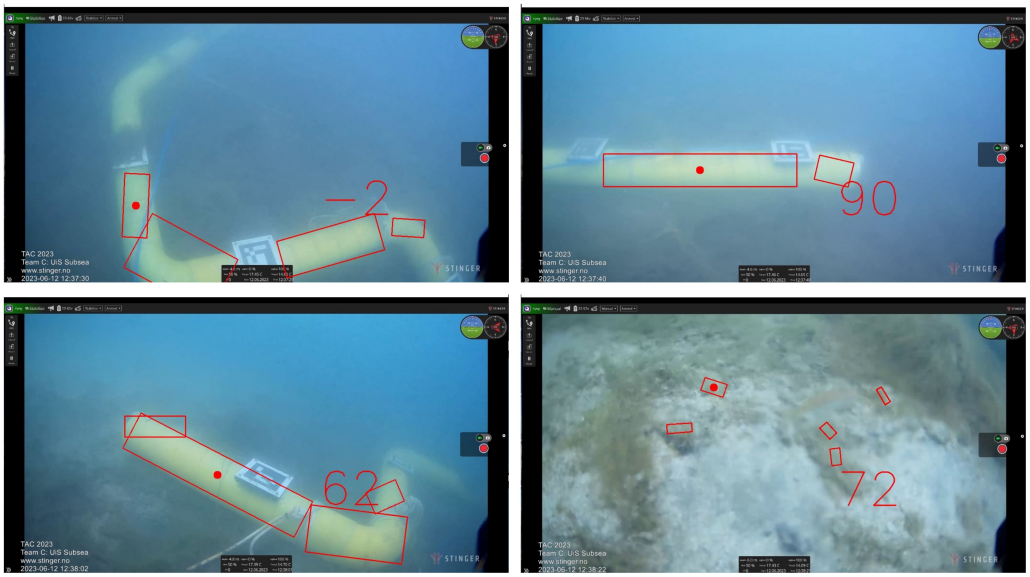
Dette underkapittelet tar for seg en kort oppsummering av testrapporten fra vedlegg B, som viser utledningen av HSV-området til rørlinjen.

6.1.1 Hensikt

Funksjonen **find_pipe()** trenger et HSV-område for fargemaskering. Området brukes senere for å finne rørledningen i *TAC challenge* 2024 beskrevet i kapittel 1.5.1.

6.1.2 Testresultater

([46,35,158],[82,101,229]) er best egnet til rørdeteksjon[1]. Bruk av dette området resulterer i deteksjonen vist i figur 6.1. Ved å anvende dette området detekteres røret nøyaktig og uten støy, men på bekostning av at rør som befinner seg langt utenfor senter av bildet ikke detekteres. Deler av omgivelsene kan forsatt blir gjenkjent som rør dersom ROV-en kjører veldig nærme bakken.

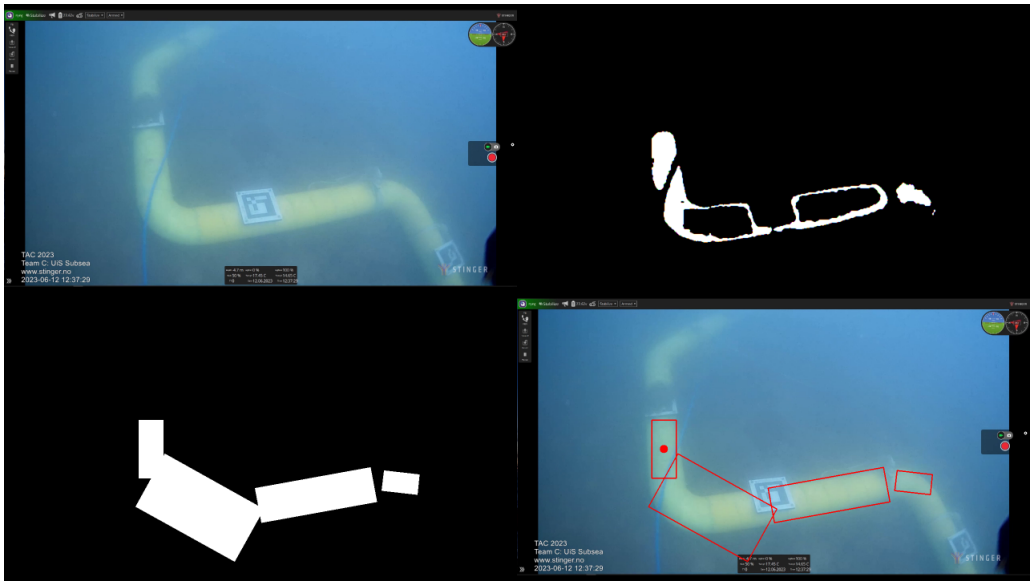


Figur 6.1: Viser rørdeteksjon med anvendt HSV-område ([46,35,158],[82,101,229])

[1]I delkapittel B.6 uføres det tester av forskjellige HSV-områder for deteksjon av rørledningen

6.2 Deteksjon av rørlinje - steg for steg

Figur 6.4 illustrerer steg for steg hvordan **find_pipe()**[2] bruker HSV-maske for å isolere rørledningen og tegne bokser rundt rørene.



Figur 6.2: Bildebehandling for *oppdrag rørledning* steg for steg. HSV-masken anvendes for å produsere bildet øverst til høyre.

6.3 HSV-område arbeidsbenk

Dette underkapittelet tar for seg en kort oppsummering av testrapporten fra vedlegg C, som viser utledningen av HSV-området til benken.

6.3.1 Hensikt

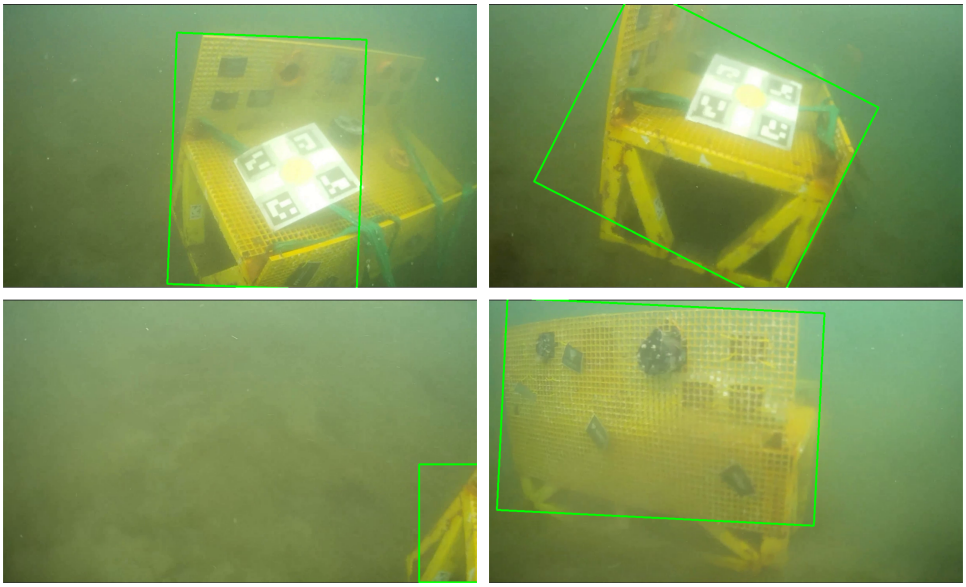
Funksjonen **find_bench()** bruker et HSV-område til fargemaskering. Fargemaskeringen brukes til å lokalisere benken som skal brukes under *TAC challenge* 2024, som beskrevet i delkapittel 1.5.2.

[2]Flytskjema G.2 beskriver dypere hvordan funksjonen fungerer.

6.3.2 Testresultater

([0,87,150],[38,255,248]) er best egnet til deteksjon av benk[3]. Bruk av dette området resulterer i deteksjonen vist i figur 6.3. Området bidrar til robust deteksjon dersom ROV-en ser vinkelrett mot benken. Dersom ROV-en ikke ser vinkelrett mot benken kan deteksjonen bli unøyaktig, og dermed føre til at ROV-en mistolker posisjoneringen av benken.

Videoene fra *TAC Challenge* teamet er meget begrenset, som igjen begrenser testing av bildebehandling av arbeidsbenk. Det ville vært optimalt med videostrøm der ROV-en ser vinkelrett mot langsidene til benken. Under *TAC Challenge* vil det sannsynligvis bli mulighet for finjustering av HSV-området.

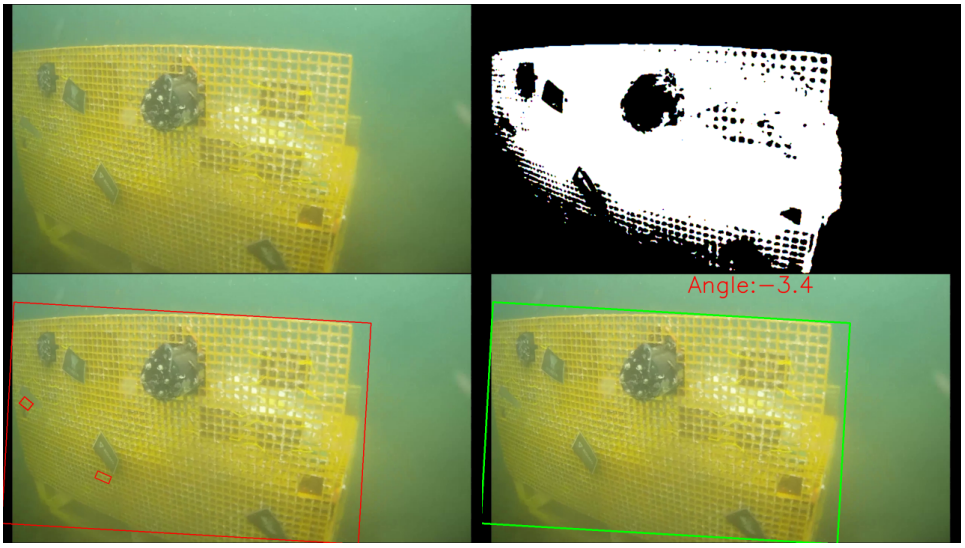


Figur 6.3: Deteksjon av benk med HSV-området [0,87,150],[38,255,248]

6.4 Deteksjon av benk - steg for steg

Figur 6.4 illustrerer steg for steg hvordan **find_bench()**[4] bruker HSV-maske for å isolere rørledningen og tegne bokser rundt rørene.

[3] I delkapittel C.6 uføres det tester av forskjellige HSV-områder for deteksjon av benken
[4] Flytskjema G.3 beskriver dypere hvordan funksjonen fungerer.



Figur 6.4: Bildebehandling for *oppdrag rørledning* steg for steg

6.5 Kalibrering av kameraet

Kameraene ROV-en bruker har fiskeøye-linser, disse skaper linseforvrengning som gjør at rette linjer fremstår som buet. Figur 6.5 viser et eksempel der en rett dør fremstår som buet. For å bli kvitt denne forvrengingen brukes en kalibreringsmatrise med tilhørende forvrengningskoeffisient. Dette underkapittelet tar for seg en kort oppsummering av test-rapporten fra vedlegg D, som viser framgangsmåten for å finne kalibreringsmatrisene og forvrengningskoeffisientene.

Figur 6.5: Originalt bildet fra exploreHD 3.0

6.5.1 Resultater

Forskjellige oppløsninger gir forskjellige forvrengninger. Kalibreringen utføres derfor for tre forskjellige oppløsninger: 640x480, 1280x720 og 1920x1080.

640x480

KalibrerKamera.py med 26 forskjellige bilder av sjakkbrettet, resulterte i kalibreringsmatrisen:

$$\begin{bmatrix} 426.2 & 0 & 320.4 \\ 0 & 429.8 & 225.3 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

og forvrengningskoeffisienten:

$$\begin{bmatrix} -0.357 & 0.117 & 0.011 & 3.763 \cdot 10^{-3} & -1.392 \cdot 10^{-2} \end{bmatrix} \tag{6.2}$$

1280x720

KalibrerKamera.py med 34 forskjellige bilder av sjakkbrettet, resulterte i kalibreringsmatrisen:

$$\begin{bmatrix} 611.1 & 0 & 595.4 \\ 0 & 618.5 & 343.1 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.3}$$

og forvrengningskoeffisienten:

$$\begin{bmatrix} -0.397 & 0.189 & 9.87 \cdot 10^{-3} & 2.94 \cdot 10^{-3} & -4.81 \cdot 10^{-2} \end{bmatrix} \tag{6.4}$$

1920x1080

KalibrerKamera.py med 36 forskjellige bilder av sjakkbrettet, resulterte i kalibreringsmatrisen:

$$\begin{bmatrix} 942.6 & 0 & 998.5 \\ 0 & 940.4 & 483.6 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.5}$$

og forvrengningskoeffisienten:

$$\begin{bmatrix} -0.400 & 0.210 & 7.31 \cdot 10^{-3} & -6.25 \cdot 10^{-3} & -7.03 \cdot 10^{-2} \end{bmatrix} \tag{6.6}$$

Figur 6.6 viser forskjellen mellom et originalt bilde og et bilde der forvrengingene er fjernet. Det er tegnet på en rød strek for å illustrere effekten av forvrengingene.

Figur 6.6: Sammenligning av originalt bildet fra kamera og med kalibrerings matrise

6.5.2 Konsekvenser av kalibrering

Kalibreringsmatrisen og forvrengningskoeffisienten fjerner mesteparten av forvrengingene. Kalibreringsmatrisen kunne vært mer nøyaktig ved bruk av enda flere bilder av sjakkbrettet, men disse resultatene er gode nok til sine formål.
En ulempe ved å fjerne forvrengningen er at deler av bildet forsvinner. Før beskjæring ser bildet ser ut som i figur 6.7, etter beskjæring finnes litt mindre informasjon i av bildet.

Figur 6.7: Bildet før beskjæring

6.6 Test av bildebehandling med direkte bildestrøm

Dette underkapittelet tar for seg en kort oppsummering av testrapporten i vedlegg E, der bildebehandlingen for deteksjon av rørlinjen testet.

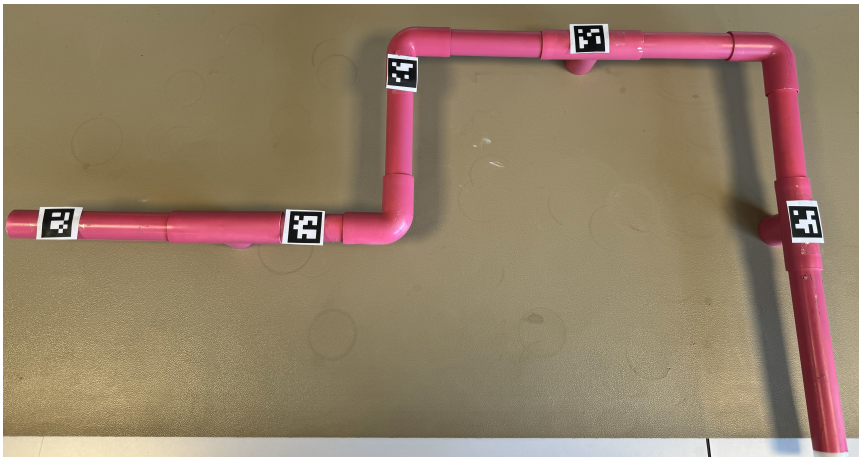
6.6.1 Hensikt

Funksjonen **find_pipe** fra underkapittel 4.5.2 skal testes med bildestrøm fra kameraet som skal brukes på ROV-en.

6.6.2 Fremgangsmåte

For testing av bildebehandlingsprogramvaren utviklet for rørlinje inspeksjon, utvikles en modifisert versjon av m_pipeline_node.py fra underkapittel 4.4.8 og en ny oppstartsfil. Her brukes bildebehandlings metodene fra (image_handler.py) beskrevet i underkapittel 4.5.2.

Gulfarget rør var ikke tilgjengelig, og substitueres derfor med rosafarget rør. Fem ArUco-koder teipes til røret, resultatet blir rørledningen i figur 6.8.



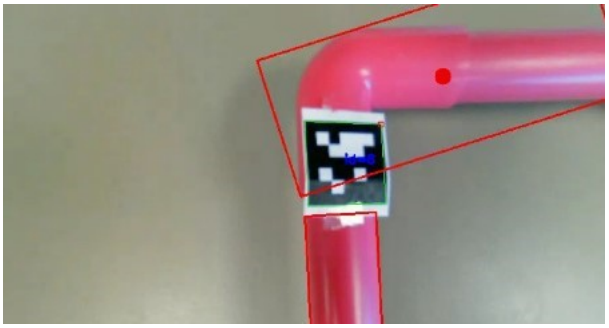
Figur 6.8: Rørledningen som brukes til testing av bildebehandling

HSV-området som brukes til fargemaskering for den rosa rørlinjen ble: [126, 45, 87] - [179, 255, 255][5].

6.6.3 Testresultater

Programvaren detekterer den rosa rørlinjen og analyserer den øverste delen av røret, ved å finne senter og vinkel til den røde boksen. Figur 6.9 illustrerer at røret og ArUco-koden ble identifisert og markert av programvaren.

[5]For å finne HSV-området til det rosa røret brukes samme metode som i testrapport B



Figur 6.9: Bildestrøm hvor røret er detektert og avlesning av ArUco-koden.Trykk her for å se en YouTube video av testen

Etter inspeksjon av røret printes listen med ArUco-koder i terminalen som vist i figur 6.10. Listen printes i riktig rekkefølge og uten duplikater i henhold til kravene for *TAC Challenge*.

```
[mission_pipeline_node]: Aruco List:[11, 2, 8, 5, 9]
```

Figur 6.10: Liste med detekterte ArUco-koder

Funksjonen **find_bench()** viser tilfredstillende resultater under manuell test med direkte video. Testen skulle helst vært utført under vann og med gult rør, for å gjenskape den dårlige sikten ROV-en i møter i virkeligheten[6].

6.7 Endring fra simulator

I ROS strukturen delegeres forskjellige arbeidsoppgaver til forskjellige noder, den modulære programstrukturen fører til enkel integrasjon av programvare fra simulering til virkelighet. Ved integrasjon gjøres følgende endringer:

- **Ny node utvikles for inntak av bildestrømmen fra ROV-ens kameraer:** Den rå bildestrømmen fra ROV-ens kameraer er forvrengt, derfor utvikles camera_node [7] som anvender kalibreringsmatrise[8] og forvrengningskoeffisient, for deretter å sende ut den korrigerte bildestrømmen på emnet *usb_camera*. Noden kan også brukes til å kalibrere[9] kameraer , samt lagre opptak av den korrigerte bildestrømmen.
- **Aktivering av oppdragsnodene** Under *TAC Challenge* skal ROV-en kjøres manuelt frem til benk/rør, før oppdragsnodene tar over og sørger for automatisk kjøring av ROV-en. Dette løses ved at topside-sjåføren sier ifra via *modus* emnet som utløser

[6] I vedlegg B og underkapittel 6.1 utføres tester for å finne HSV-området til den virkelige rørlinjen, testene baseres på bildestrømmen fra *TAC Challenge 2023*.
[7] Koden for camera_node finnes i vedlegg J.14
[8] Kalibreringsmatrise og forvrengningskoeffesient utledes i testrapport D
[9] Fremgangsmåte for kalibrering av kamera med camera_node forklares i vedlegg D.6

oppdragsnodene, som ligger i dvale frem til de får beskjed. Når et oppdrag fullføres gir oppdragsnoden beskjed tilbake via *modus*.

- **Finjustering av PID-parametre:** Selvom simuleringen er relativt realistisk når det gjelder bevegelse, er overføringsfunksjonene som brukes basert på en annen ROV. Manuell finjustering av PID-parametre blir sannsynligvis nødvendig for å oppnå god regulering.
- **Emner som må erstattes:** Emner som kommuniserer simulert data erstattes med emnene fra GUI-gruppen som overfører ekte sensordata. I skrivende stund er ikke navnene på de nye emnene bestemt, men emnene som skal erstattes er:
  - camera2/image_raw
  - camera/image_raw
  - /odom
  - /tf_movement

6.8 Videre arbeid

Da ROV-en ikke var ferdigstilt til planlagt tid, ble det lite dokumentasjonsgrunnlag for integrasjon fra simulering til virkelighet. Bruk av bildestrøm fra tidligere *TAC Challenge* resulterte i akseptable og HSV-områder til bruk i *TAC Challenge 2024*. Manuell kalibrering av kamera resulterte i kalibreringsmatriser og forvrengningskoeffisienter, som er nødvendige for å kunne anvende bildestrømmen til kameraene på ROV-en. Håndkjøring av kameraet med rosa rørledning beviser at camera_node og **find_bench()** funksjonen fungerer som de skal.

Videre etter levering av bacheloren vil det være mulighet for testing av programvaren på den faktiske ROV-en. Under testing planlegges manuell justering av PID-regulatorene for bevegelse ettersom at ROV-en tvilsomt kommer til å bevege seg akkurat som i simuleringen.

1. **Bassengtest av røroppdraget:** På denne måten kan regulatorer og hjem funksjon, testes og optimaliseres før konkurransen. Testen utføres med de de tilgjengelige rosa rørene.
2. **Bassengtest av DVL-regulering:** Her skal ROV-en beveges i en firkant og roteres 360°grader, med målet om å deretter befinne seg i samme posisjon som ved start. Testen utføres for å verifisere ROV-ens regulering mot egen posisjon.
3. **Bassengtest av høyderegulering:** Regulator gruppen fra prosjektarbeidet hos UiS Subsea tar seg av regulering av høyde, denne funksjonen må integreres for autonom kjøring og deretter testes.
4. **Bassengtest av *oppdrag arbeidsbenk*:** På denne måten kan regulatorer og sekvens optimaliseres før konkurransen. Her planlegges det å konstruere et objekt med form som et firkantet prisme for å etterligne benken.

Kapittel 7

Diskusjon

Innhold

I dette delkapittelet diskuteres praktiske problemer som har vist seg i løpet av prosjektarbeidet, og potensielle forbedringer av ROV-en i fremtiden. Det blir også gjennomgodt en analyse av tidsbruket for bacheloren.

7.1 Problemer og løsninger

I løpet av bacheloren har det oppstått flere problemer, går igjennom de største problemene og løsningene deres.

7.1.1 Testing av ROV

I starten av prosjektarbeidet ble det kommunisert at ROV-en skulle ha sin første vanntest i April. Når det gjelder utviklingen av programvare for autonom kjøring krever dette en ferdigstilt ROV, som igjen vil medføre at denne gruppen kun får testet programvare siste fjerdedel av tidsfristen for oppgaven. Løsningen på dette problemet ble å simulere ROV-en og miljøet rundt. Bruk av ROS og Gazebo tillater også sømløs integrasjon mot den virkelige ROV-en når den ferdigstilles. Hovedfokuset for oppgaven har derfor blitt å produsere best

mulig simulering av ROV-en. Produksjon av simulering har resultert i lav terskel for testing, da dette kan gjøres på PC istedenfor i basseng. Ved utvikling og testing i simulator, ble programvare og dokumentasjonsgrunnlag produsert i god tid før innleveringfrist.

7.1.2 Mangel på simpel og fungerende kontroller i simulator

For bevegelse av ROV-en i simulator foretrekkes bruk av en innebygget ROS-kontroller. Ingen av de eksisterende kontrollerene i ROS/Gazebo var designet for simpel 3D bevegelse i vann. Løsningen ble å heller bruke en programvareutvidelse[1] for bevegelse i planet. Denne tillot thruster-liknende bevegelser i alle frihetsgrader, bortsett fra hiv, ROV-en kunne derfor ikke beveges verken opp eller ned. Dette problemet ble tungt å løse da programvareutvidelsen aktivt sperret all form for bevegelse langs z aksen. Løsningen på det nye problemet ble å bruke enda en programvareutvidelse som sørget for å teleportere ROV-en om nødvendig, samt utviklingen av en egen node som tar seg av teleportering. Heldigvis trengtes dette kun når ROV-en skulle opp på arbeidsbenken for å inspisere oversiden av denne.

7.1.3 Perfekt respons i en perfekt verden

Simuleringen bruker en simplifisert versjon av ROV-en, med en simpel kontroller, uten gravitasjon eller vanndynamikk. Bruken av simulator resulterer derfor i nærmest umiddelbar og perfekt respons når posisjonen til ROV-en reguleres. Disse ideelle bevegelsene gir et meget tynt grunnlag for justering av PID-parametre. Da det er ønskelig å justere regulatorene så bra som mulig så langt det lar seg gjøre i simulatoren, ble denne problemstillingen en prioritet. Løsningen på problemet ble å ta i bruk overføringsfunksjoner fra en tidligere produsert ROV. Overføringsfunksjonene beskriver bevegelsen av ROV-en, det brukes da en overføringsfunksjon per relevante frihetsgrad. Disse overføringsfuksjonene ble linearisert, diskretisert og deretter implementert i simulatoren. All bevegelse sendt til ROV-kontrolleren, må da igjennom en overføringsfunksjon som sørger for at ROV-en utviser mer realistiske responser ved regulering av fart, men ikke perfekt.

7.1.4 Utydelig sikt under vann

Grunnlaget for utvikling av autonom kjøring til ROV-en er at den skal samle så mange poeng som mulig i *TAC Challenge 2024*. Konkurransen foregår i havet og bildestrømmen fra tidligere konkurranser viser meget dårlig sikt i vannet. I simuleringen er sikten så god at bildebehandling kan utføres med meget stort HSV-område. Ved utvikling av metoder for bildebehandling ønskes et mer realistisk utgangspunkt. Det har derfor blitt utført en rekke tester på bildestrømmen fra tidligere konkurranser, for å finne realistiske HSV-områder, og utvikle robuste bildebehandlingsalgoritmer. Disse testene viser gode resultater og verifiserer funksjonen til bildebehandlingsalgoritmene som skal brukes i konkurransen.

[1] Se underkapittel 4.3.1 for informasjon om denne.

7.2 Potensielle forbedringer

I løpet av prosjektarbeidet var det enkelte løsninger og forbedringer som ble nedprioritert, enten grunnet mangel på tid, eller mangel på utstyr, disse er som følger:

- **Optimalisering av ytelsen til simuleringen:** Kjøring av simulatoren krever svært mye prosesseringskraft. Med flere arbeidstimer tilgjengelig, ville det vært større fokus på å forbedre programkoden med mål om å redusere belastningen på datamaskinen som driver simuleringen.
- **Sonar, stereokamera og sensorfusjon:** En god løsning for å måle avstand er bruk av stereokamera, eventuelt paret med sonar i sensorfusjon. Stereokamera og sonar kan brukes til å måle avstander. Fusjon av disse sensorene ville resultert i robuste og nøyaktige data for avstandsvurdering og deteksjon av objekter. Selv uten fusjon med sonar, ville et stereokamera fortsatt gitt gode avstandsestimater. Disse kameraene koster en del og ble ikke prioriert i budsjettet til UiS Subsea i år.
- **SLAM-teknologi:** For automatisk kjøring av ROV-en kunne SLAM(Simultanious Localisation And Mapping)-teknologi blitt en del av løsningen. Ved bruk av Visual-SLAM kartlegges miljøet på havbunnen under kjøring, slik at ROV-en kan navigere i kartet for å inspisere objekter. Bruk av V-SLAM krever stereokamera og mange arbeidstimer med opplæring, løsningen ble derfor nedprioritert.
- **Pinger:** En av deloppgavene i *oppdrag rørledning* går ut på lokalisering av en pinger, ved bruk av av en akustisk sensor. Denne akustiske sensoren ble desverre ikke prioritert i budsjettet til UiS Subsea i år, maksimal poengsum i *oppdrag rørledning* vil derfor være uoppnåelig.
- **Smart kodeavlesning:** ArUco-koder kan være vanskelige å lese av ved dårlig sikt. En løsning på dette problemet er å justere kamerainnstillinger midlertidig når en ArUco-kode oppdages, men ikke kan dekodes. Ved oppdagelse av kvadratet rundt koden kan synsfeltet til kameraet reduseres og konsentreres mot koden, samtidig kan lukketiden for linsen økes for dette spesifikke bildet. Kombinasjon av disse to teknikkene vil gi et bilde med mer nøyaktig informasjon, i en mindre ramme, som igjen øker sannsynligheten for å dekode en ArUco-kode.

7.2.1 Analyse av timeliste

Totalt har det gått med litt færre arbeidstimer enn det som var prosjektert[2] i starten av bacheloroppgaven. Arbeidstimene har heller ikke blitt distribuert akkurat som forventet. Timefordelingen visualiseres i GANT-skjemaet i figur 7.1, lilla bokser representerer timebruk som avviker fra det originale GANT-skjemaet. Punktene som har sett størst avvik fra det originale GANT-skjemaet er som følger:

[2]Se det originalet GANT-skjemaet for oppgaven i vedlegg G.1

- **Møter:** Tid brukt på møter ble kraftig redusert da gruppen fikk fritak fra mandagsmøtene til UiS-Subsea. Ved å anvende simulator ble det ikke lengre nødvendig med ukentlige oppdateringer rundt arbeidet på ROV-en. Fritaket har resultert i ca 1.5 time mindre møtetid hver uke.
- **Opplæring ROS:** Opplæring i ROS tok mye lengre tid enn forventet da rammeverket viste seg å være mer tyngre å sette seg inn i enn først antatt. Her endte gruppen på 60 timer istedenfor de planlagte 20, dette viste seg etterhvert å være tid vel brukt.
- **Programvare:** I utviklingen av programvare har mange av oppgavene resultert i bruk av omtrentlig planlagt antall timer. Da oppgaven har vist seg å kreve kontinuerlig oppdatering og optimalisering av programvaren, har disse timene blitt jevnere fordelt enn først planlagt.
- **Rapport skriving:** Utover dokumentasjon av bacheloroppgaven, har denne rapporten som mål å tjene som en brukermanual for simuleringen som ble utviklet i løpet av prosjektet. Autonom kjøring er ett førstegangsemne for UiS Subsea, og videre arbeid innen emnet vil sannsynligvis bygge videre på arbeidet utført i denne oppgaven. Derfor har over 50% av alle timene brukt på denne bacheloroppgaven gått med til rapportskriving. Rettskriving og forbedring av rapporten har fått tildelt dobbelt så mange timer som planlagt.
- **Testing og optimalisering**: ROV-en ble ikke ferdigstilt før innlevering av bacheloren, det ble derfor ikke brukt tid på testing av den faktiske ROV-en, derfor står denne posten tom i GANT-skjemaet.

Er nå i uke:	20															Vanntest							
Uke Nr	2023	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Beregnet	Faktisk Tid brukt
Prosjekt styring																							
Planlegging	11	9	6	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	55
Møter	0	6,5	6,5	0	2	3,5	5	0	3	3,5	2,5	0	0	0	2,5	2	1	1	0	2	0	100	41
Opplæring	4	0	2	0	2	22	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	36
Totalt	15	15,5	14,5	29	4	25,5	11	0	3	3,5	2,5	0	0	0	2,5	2	1	1	0	2	0	170	96
Planlegging																							
Behovsspesefikasjon	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Funksjonsspesefikasjon	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	1
Blokkskjema	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Totalt	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32	3
Programvare																							
Opplæring ROS	0	0	36,5	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	60,5
Bildebehandling	0	0	15	0	0	0	28	9	0	0	0	0	0	2	15	15,5	0	0	0	0	0	100	84,5
Simulering	0	0	0	0	18	13	20	6,5	30	17	3	7	0	0	2	0	0	0	0	0	0	75	116,5
Pipeline inspeksjon	0	0	0	13,5	2,5	6	1	27,5	0	0	4	0	6	0	7,5	8	0	0	0	0	0	100	76
Retur til hjem posisjon	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	30	2
Objekt inspeksjon	0	0	0	0	0	0	7	0	20	9	9	11	0	0	0	0	10,5	10	0	3	0	100	79,5
Ut data	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0
Integrering	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	2
Videreutvikling	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0
Totalt	0	0	51,5	37,5	22,5	19	56	43	50	26	17	19	6	2	24,5	23,5	10,5	10	0	3	0	475	421
Testing og Opptimalisering																							
Ut data	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0
Pipeline inspeksjon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
Objekt inspeksjon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
Totalt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	130	0
Uke Nr	2023	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Beregnet	Faktisk Tid brukt
Rapport skriving/teori																							
Oppsett	0	0	0	0	7,5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	10	8,5
Innleding og intro subsea	0	0	0	0	6	0	0	0	1	0	0	0	7,5	0	0	0	0	0	0	5	0	16	19,5
Planlegging	0	0	0	0	8	11,5	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	4	22,5
Bildebehandling	0	0	5,5	22	6	9	17	0	4	0	0	0	14	0	0	4	0	0	0	0	0	50	81,5
Simulering	0	0	0	0	0	0	0	0	0	21	52	47,5	15	0	0	0	14	0	3	3	0	50	155,5
Objekt inspeksjon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	21,5	0	0	0	50	25,5
Resultatanalyse	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	0	0	50	8
Diskusjon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	3	5	0	50	15
Konklusjon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	50	10
Rettskriving og Forbedring	0	0	0	0	0	0	0	0	0	0	0	0	17	17	11,5	11	30	32	28	64	0	80	208,5
Totalt	0	0	5,5	22	27,5	20,5	17	0	7	23	52	47,5	53,5	17	11,5	22	48	57,5	36	87	0	410	564,5
Totalt	15	18,5	71,5	51	54	46	84	43	60	26,5	71,5	66,5	59,5	17	38,5	24	59,5	58,5	36	89	0	1217	1074,5

Figur 7.1: Revidert GANT-skjema

Kapittel 8

Konklusjon

Denne bacheloroppgaven har beskrevet løsninger for autonom kjøring av ROV-en Draugen. Målet for oppgaven har vært å muliggjøre at ROV-en, uten hjelp fra en pilot, selvstendig skal kunne utføre inspeksjon av en arbeidsbenk og en rørledning under vann. Arbeidsoppgavene har hovedsaklig bestått av følgende:

- Velge fremgangsmåte for utvikling av programvare. Valget falt på bruk av simulator for å lage ett kunstig miljø som testgrunnlag for bildebehandling av objekter, og fartsregulering av Draguen.
- Velge simulator og rammeverk. Valget falt på Gazebo som simulator, og ROS 2 Humble som rammeverk. Bruk av ROS2 har lagt til rette for utviklingen av modulær programvare, som effektivt kan integreres til den virkelige ROV-en.
- Utvikle programvare for behandling av sensordata. Bildebehandlingen som brukes til objekt deteksjon viser god funksjonalitet i alle testede situasjoner.
- Implementering av overføringsfunksjoner i simulator for å gi et mer realistisk grunnlag for justering av PID-parametere. Disse er ikke helt nøyaktige, men gir bedre grunnlag enn simulatorens perfekte respons.
- Utvikle programvare for regulering av frihetsgradene jag, svai og gir. Realisering av ROV-ens bevegelse gjøres ved å bruke en PID-regulator per frihetsgrad. For forskjellige referanser brukes forskjellige PID-regulatorer selv om de regulerer samme frihetsgrad. Ved justering av parametere viste Skogestad-metoden seg å gi et utgangspunkt for noen av parametrene, men manuell justering ble i høyeste grad nødvendig for å produsere en robust og effektiv regulering.
- *Oppdrag rørledning* ble løst ved å holde konstant jag, og bruk av rørposisjonering som referanse for regulatorer til frihetsgradene svai og gir. Løsningen ga tilfredstillende resultater slik at ROV-en klarte å lese av alle ArUco-kodene langs røret.
- *Oppdrag arbeidsbenk* ble løst ved å benytte fem forskjellige regulatorer aktivert i sekvens. Regulatorene virket på frihetsgradene jag, svai og gir, med enten benkens posi-

sjon, eller ROV-ens posisjon som referanse. Dette ga tilfredstillende resultater slik at ROV-en klarte å lese av alle ArUco-kodene plassert på benken.

Et sidemål for denne oppgaven har vært å gjøre programvaren modulær og velstrukturert, slik at denne kan brukes videre av UiS Subsea. Nå som et rammeverk og simuleringsmiljø for ROV er utviklet, kan UiS Subsea bruke dette rammeverket under videreutviklingen av autonom kjøring for ROV. Funksjoner for bildebehandling og regulering er abstrahert slik at disse enkelt kan gjenbrukes eller videreutvikles. ROS2 prosjektstrukturen er utviklet slik at ROV-en og terrenget rundt enkelt kan modifiseres eller erstattes.

Den viktigste delen av prosjektet, som ikke kunne fullføres innen leveringsfristen for denne rapporten, er integrasjon og testing av programvaren på den virkelige ROV-en. Så langt har alle tester blitt gjort i simulator, eller ved manuell håndføring av kamera. Disse testene har vist gode resultater. Det planlegges fortsatt å integrere programvaren når ROV-en ferdigstilles, slik at ROV-en kan oppnå høyest mulig poengsum under *TAC Challenge 2024*.

Simuleringen har vist gode resultater og det forventes at ROV-en, etter litt finjustering, også vil yte godt når den er ferdigstilt.

Bibliografi

[1] *768px-Logo-ubuntu_ cof-orange-hex.svg.png (768×768)*. URL: https://upload.wikimedia.org/wikipedia/commons/thumb/a/ab/Logo-ubuntu_cof-orange-hex.svg/768px-Logo-ubuntu_cof-orange-hex.svg.png (sjekket 18.03.2024).

[2] Javier Abellán Abenza. *HSV color model*. en. Des. 2018. URL: https://medium.com/neurosapiens/segmentation-and-classification-with-hsv-8f2406c62b39 (sjekket 23.01.2024).

[3] *ArUco Marker Detection*. English. Jan. 2024. URL: https://docs.opencv.org/3.4/d9/d6a/group__aruco.html.

[4] *foxy-small.png (150×178)*. URL: https://docs.ros.org/en/foxy/_static/foxy-small.png (sjekket 18.03.2024).

[5] Sergio Garrido-Jurado mfl. "Automatic generation and detection of highly reliable fiducial markers under occlusion". I: *Pattern Recognition* 47 (jun. 2014), s. 2280–2292. DOI: 10.1016/j.patcog.2014.01.005.

[6] Sergio Garrido-Jurado mfl. "Generation of fiducial marker dictionaries using Mixed Integer Linear Programming". I: *Pattern Recognition* 51 (okt. 2015). DOI: 10.1016/j.patcog.2015.09.023.

[7] Finn Haugen. "Finn Haugen: PID Control". I: ().

[8] *HSL and HSV*. en. Page Version ID: 1194710171. Jan. 2024. URL: https://en.wikipedia.org/w/index.php?title=HSL_and_HSV&oldid=1194710171 (sjekket 22.01.2024).

[9] *HSL bilde*. URL: https://upload.wikimedia.org/wikipedia/commons/3/33/HSV_color_solid_cylinder_saturation_gray.png.

[10] *humble-small.png (150×183)*. URL: https://docs.ros.org/en/humble/_static/humble-small.png (sjekket 18.03.2024).

[11] *iron-small.png (150×172)*. URL: https://docs.ros.org/en/iron/_static/iron-small.png (sjekket 18.03.2024).

[12] Flatheim Jesper, Andreas Eng og Wagner Frida. "Regulering og styring av motorsystem for fjernstyrt undervannsfartøy (ROV)". I: (). URL: https://uis.brage.unit.no/uis-xmlui/handle/11250/3073584.

[13] Åse Jortveit og Vebjørn Riiser. "Operatørgrensesnitt og topside-kommunikasjon for fjernstyrt undervannsfartøy". I: (). URL: https://uis.brage.unit.no/uis-xmlui/handle/11250/3004283.

[14] Birkeland Kristian Ljosdal Otto og Royal Tomaas. "Regulering og styring av motorsystem for fjernstyrt undervannsfartøy (ROV)". I: (). URL: https://uis.brage.unit.no/uis-xmlui/handle/11250/3002120.

[15] *Mission Booklet TAC-challage*. English. URL: https://tacchallenge.com/wp-content/uploads/2024/02/Mission-Booklet-2024-rev.1.pdf.

[16] *Nodes-TopicandService.gif (854×480)*. URL: https://docs.ros.org/en/humble/_images/Nodes-TopicandService.gif (sjekket 18.03.2024).

[17] *noetic (256×330)*. URL: https://wiki.ros.org/noetic?action=AttachFile&do=get&target=noetic.png (sjekket 18.03.2024).

[18] *OpenCV: Camera Calibration*. URL: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html (sjekket 01.04.2024).

[19] *PID Controller Tuning in Simulink - MATLAB & Simulink - MathWorks Nordic*. URL: https://se.mathworks.com/help/slcontrol/gs/automated-tuning-of-simulink-pid-controller-block.html (sjekket 01.03.2024).

[20] Francisco Romero-Ramirez, Rafael Muñoz-Salinas og Rafael Medina-Carnicer. "Speeded Up Detection of Squared Fiducial Markers". I: *Image and Vision Computing* 76 (jun. 2018). DOI: 10.1016/j.imavis.2018.05.004.

[21] *SDFormat Home*. URL: http://sdformat.org/ (sjekket 24.01.2024).

[22] *Topics — ROS 2 Documentation: Humble documentation*. URL: https://docs.ros.org/en/humble/Concepts/Basic/About-Topics.html (sjekket 23.02.2024).

[23] *Understanding nodes — ROS 2 Documentation: Humble documentation*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html# (sjekket 30.01.2024).

[24] *URDF*. en-US. URL: https://formant.io/urdf/ (sjekket 24.01.2024).

[25] waterlinked. *Protocol - Documentation*. URL: https://waterlinked.github.io/dvl/dvl-protocol/ (sjekket 27.02.2024).

[26] *Xml - Social media & Logos Icons*. en. URL: https://icon-icons.com/icon/xml/21811 (sjekket 24.01.2024).

[27] *XML Introduction*. en-US. URL: https://www.w3schools.com/xml/xml_whatis.asp (sjekket 24.01.2024).

Vedlegg A

Modellering av overføringsfunksjoner

A.1 Hensikt

Formålet er å finne overføringsfunksjonene til bevegelsene i de forskjellige frihetsgradene til ROV-en. Overføringsfunksjonene brukes i simulatoren for å gi ROV-en mer realistiske bevegelser.

A.2 Del 1 modellering

Del 1 går igjennom hvordan modellene for bevegelsene er endret fra modellene brukt av en bachelor gruppe i 2022.

A.2.1 Programvare

- MATLAB R2023b
- Simulink

A.2.2 MATLAB Add-ons

- Control system toolbox
- Simscape
- Simscape Electrical
- Simulink Controll Design
- System Identification Toolbox

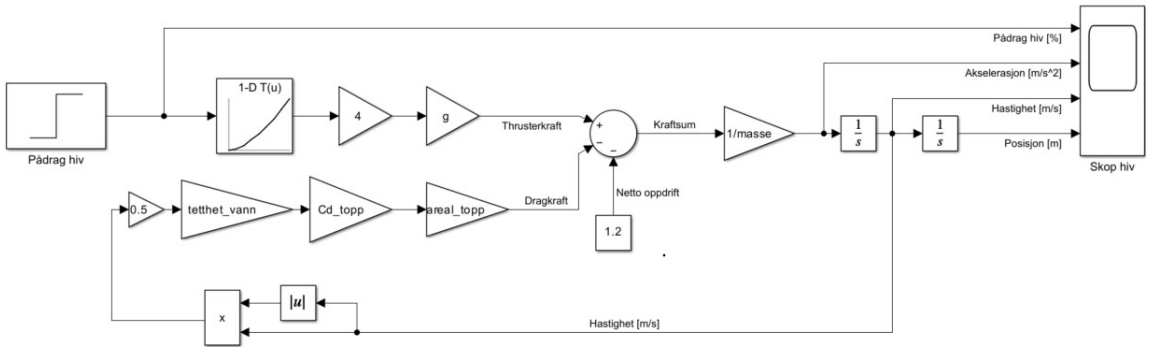
A.2.3 Programfiler

Link til GitHub mappe med filene:

- modell_koeff.m
- styremodell.slx
- pid_hiv_forskjellige.slx
- ModeleringBevegelserSim.slx

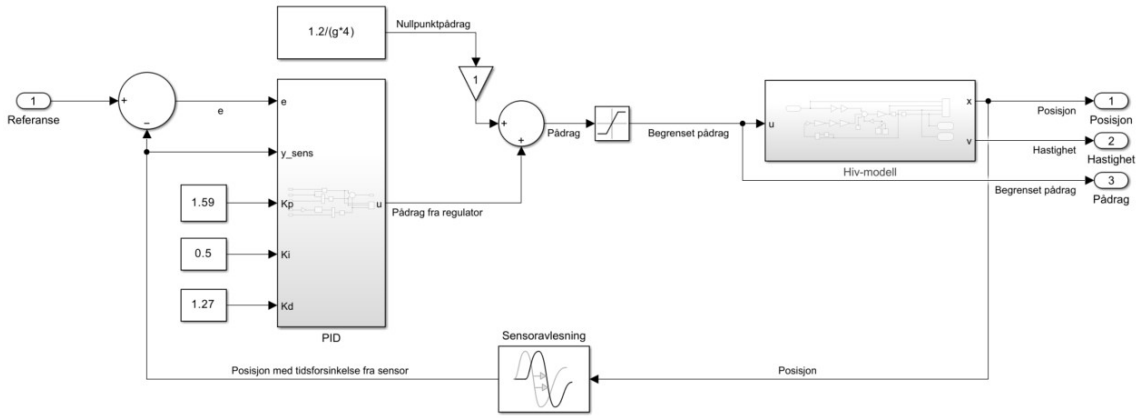
A.2.4 Utgangspunkt

De lineariserte modellene er basert på modeller fra bachelor'en som ble skrevet i 2022 for UiS Subsea [14] (kapittel 7).
Gruppen har utviklet modeller for bevegelse i de forskjellige frihetsgradene. Modellene tar inn et pådrag til thrusterene og gjør det om til hastighet i $\frac{m}{s}$ for de lineære bevegelsene, for rotasjons bevegelsene gjøres pådrag om til vinkelhastighet i $\frac{^\circ}{s}$. Figur A.1 viser modellen for jag i Simulink fra filen styremodell.slx.



Figur A.1: Utdrag fra styremodell.slx modell av hiv bevegelsen

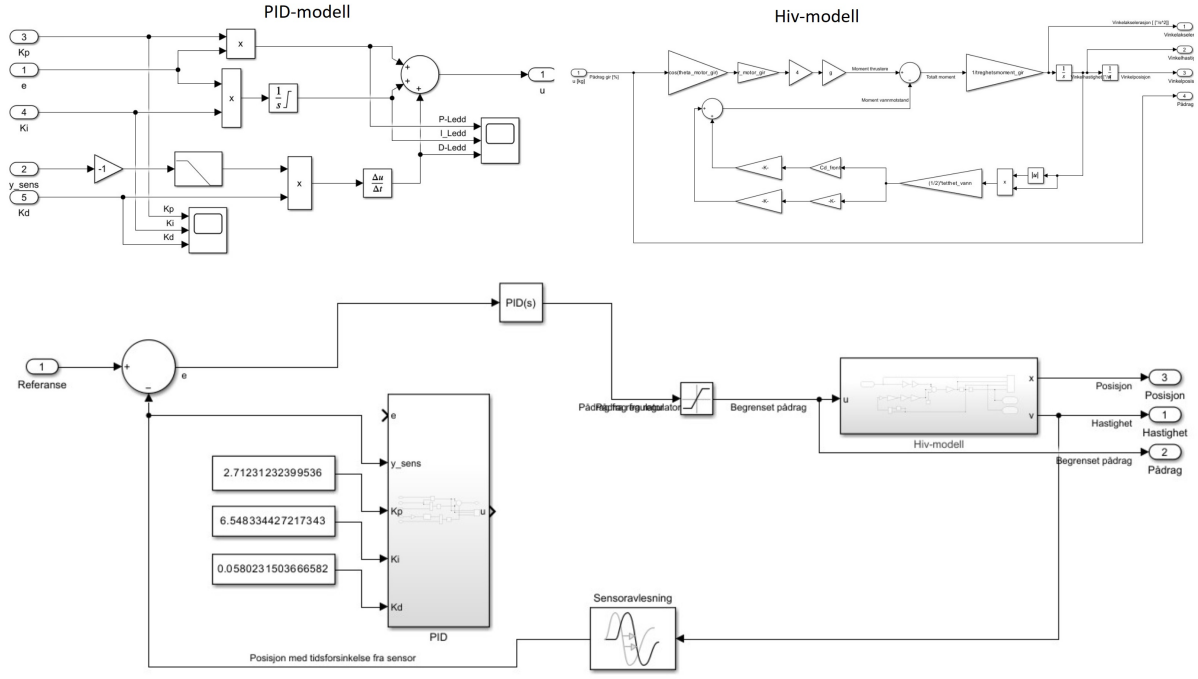
Gruppen fra 2022 har også utviklet en modell med en PID-regulator. Modellen tar inn en referanse i posisjon og gir ut en hastighet. Figur A.2 viser den originale modellen.



Figur A.2: Utdrag fra pid_hiv_forskjellige.slx som hviser PID modellen.

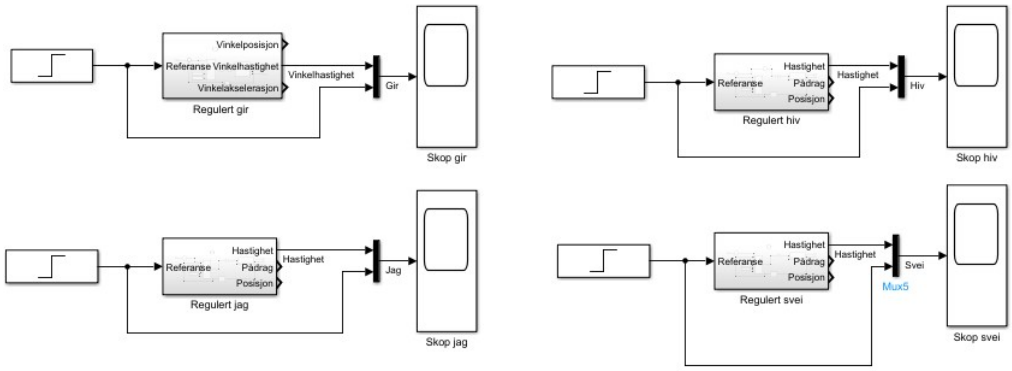
A.2.5 Endring av den originale modellen

Referanseverdiene som skal brukes til ROV-en i år er fart og ikke posisjon. Den forrige modellen må da endres litt på for bruk til simulering av årets ROV. Referanseverdien endres fra hastighet og ikke posisjon, og nullpunkts-pådraget blir fjernet. Figur A.3 viser endringene i hiv modellen.



Figur A.3: Utdrag fra RegulertHiv.slx som viser PID modellen.

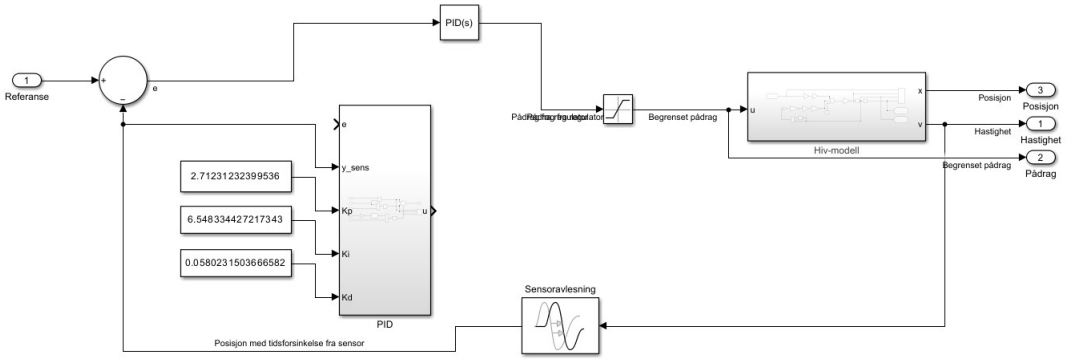
Figur A.4 viser hvordan alle modellene ser ut i filen RegulertHiv.slx.



Figur A.4: Bildet av RegulertHiv.slx

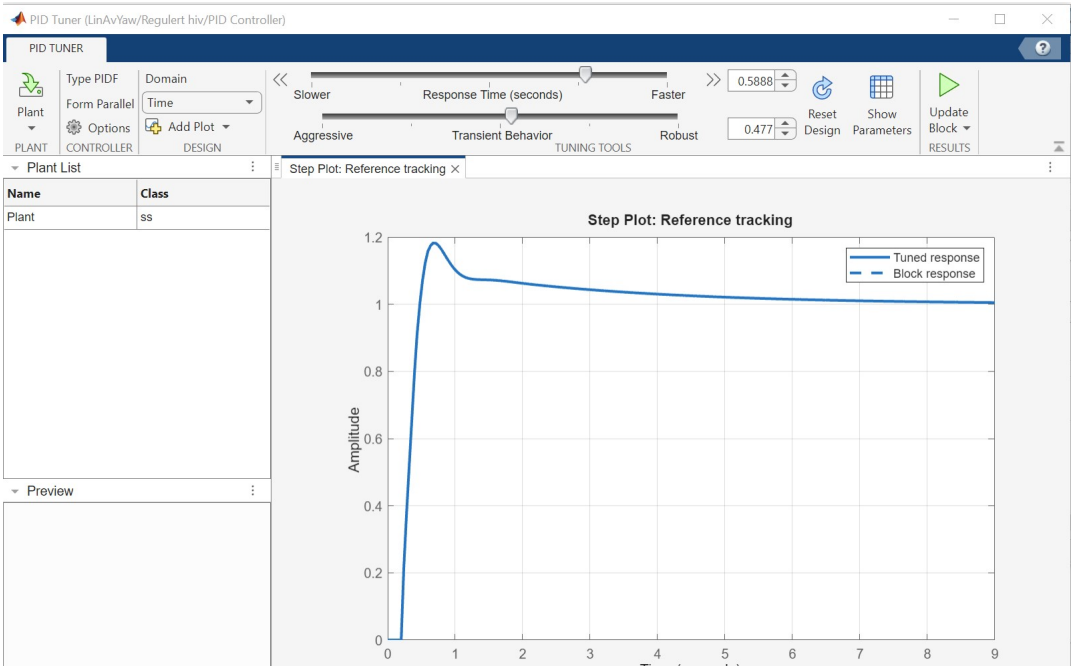
A.2.6 Nye PID-parametere

For å finne parameterene til PID-en brukes en PID controller"(simulink/continuous). Oppsettet til PID-blokken vies i figur A.5.



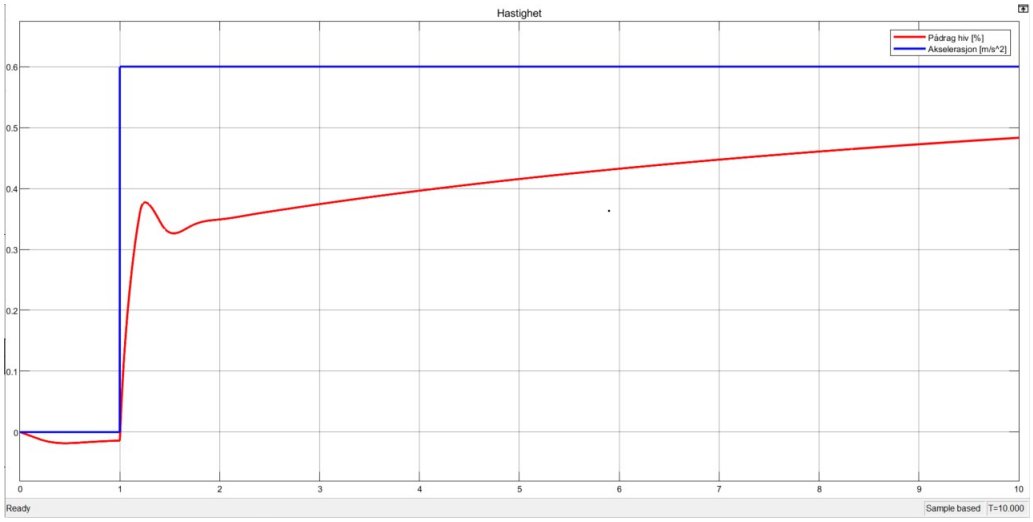
Figur A.5: Utdrag fra ModeleringBevegelserSim.slx viser hvordan PID blokken er oppkoblet

Åpne tuning inne på menyen til PID blokken. Dokumentasjonen for *PID Tuner* finnes her [19]. Ved bruk av *PID tuner* velges oppførselen til systemet. Siden dette systemet skal kjøres autonomt ønskes et raskt system. Oversving kan derfor godtas dersom det fører til kortere innsvingningstid.
Ved testing av forskjellige verdier for Response Time og Transient Behavior, viste den mest egnede responsen en responstid på 0.5888 sek og en transient oppførsel på 0.477 sek.



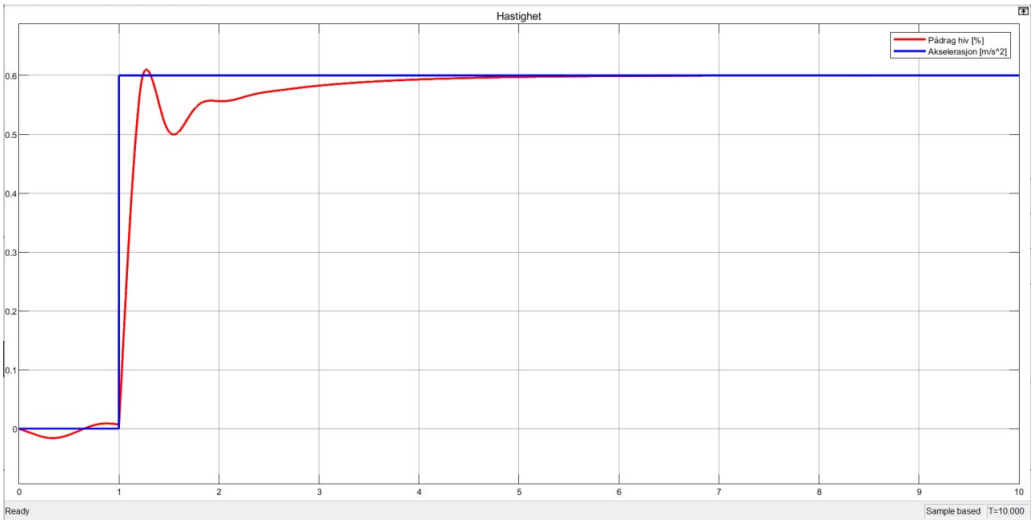
Figur A.6: Utdrag fra RegulertHiv.slx som hviser PID modellen.

Med ønsket respons ble PID-parameterene fra PID tuning programmet: P 1.7123, I 0.5483 og D 0.0580. Bruk av parametrene i PID blokken fra den originale modellen, resulterte i responsen vist i figur A.7. Steget som blir brukt går fra 0 til 0.6 $\frac{m}{s}$



Figur A.7: Utdrag fra RegulertHiv.slx som hviser PID modellen.

Skop bildet fra figur A.7 ga ikke helt optimal regulering da den ble alt for treg. For å oppnå raskere regulering ble P og I verdien justert til en P=2.7123 og I=6.5483. De justerte parametrene ga responsen på figur A.8.



Figur A.8: Utdrag fra RegulertHiv.slx som hviser PID modellen.

A.2.7 Resultat

For alle de andre bevegelsene er det gjort det samme som stegene over. Innsvingningstid og transient tide brukt i PID-tuning er i tabell A.1.

	Innsvingningstid	Transient oppførsel
Hiv	0.5888	0.477
Gir	0.6008	0.6
Svai	0.5888	0.477
Jag	0.5888	0.621

Tabell A.1: Verdiene brukt i PID tuner for de forskjellige bevegelsene

Tabell A.2 viser PID-parametrene som brukes i modellene. PID tuning kolonnen representerer parametrene hentet fra *PID-tuning* programmet i MATLAB. Kolonnen Justerte PID-parametere representerer justerte parametere for modellene som skal implementeres i simulatoren.

Hiv		
	PID Tuning	Justert PID-parametere
P	1.7123	2.7123
I	0.5483	6.5483
D	0.0580	0.0580
Svai		
	PID Tuning	Justert PID-parametere
P	3.4246	3.4246
I	1.0966	10.0966
D	0.1160	0.1160

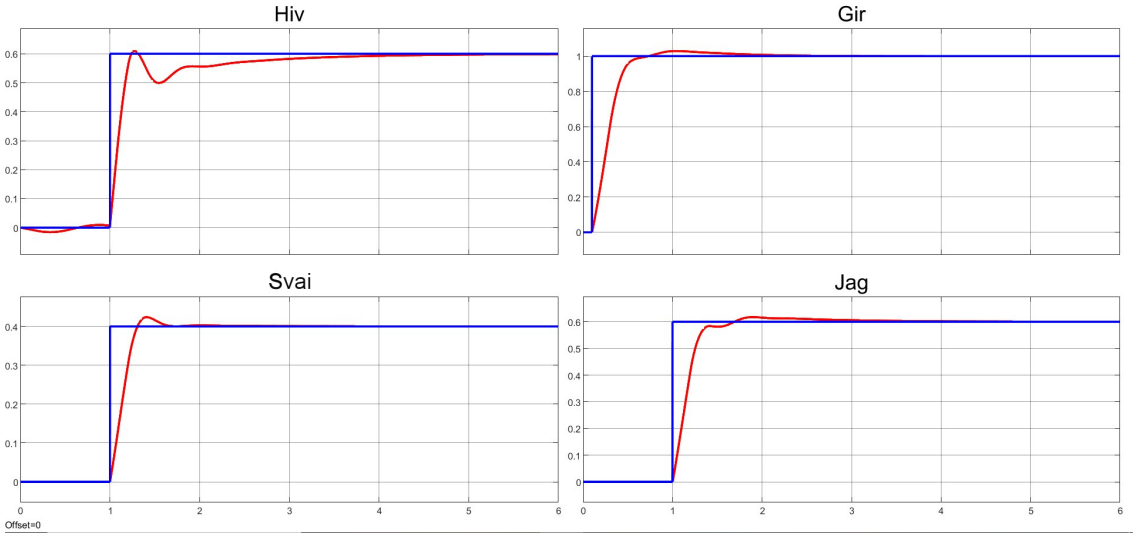
Gir		
	PID Tuning	Justert PID-parametere
P	0.2261	0.2261
I	0.0700	0.5700
D	0.0170	0.0170
Jag		
	PID Tuning	Justert PID-parametere
P	3.2225	3.225
I	1.012	6.5120
D	0.2921	0.2921

Tabell A.2: PID parametrene fra PID tuning og justert verdier.

Stegene til de forskjellige bevegelsen er bestemt ut ifra hvilke fart som brukes i simulatoren. F.eks svai bevegelsen trenger ikke å være modellert raskere enn $0.4\frac{m}{s}$, og i simuleringen er det en begrensing på rotasjonen til Gir på $1\frac{rad}{s}$. Stegene for de forskjellige bevegelsene er da:

- Hiv 0 til 0.6
- Svai 0 til 0.4
- Jag 0 til 0.6
- Gir 0 til 1

Figure A.9 viser den justerte steg responsen for alle bevegelsen. Dette blir da den responsen som skal lineariseres for implementasjon i simulatoren.



Figur A.9: Stegrespons for modelen av alle bevegelsene som skal brukes i simulatoren

A.3 Del 2 Linearisering av modellene

Modellen fra kapittel A.2 skal lineariseres implementasjon i simuleringen.

A.3.1 Programvare

- MATLAB R2023b
- Simulink

A.3.2 MATLAB Add-ons

- Control system toolbox
- Simscape
- Simscape Electrical
- Simulink Controll Design
- System Identification Toolbox

A.3.3 Programfiler

Link til GitHub mappe med filene:

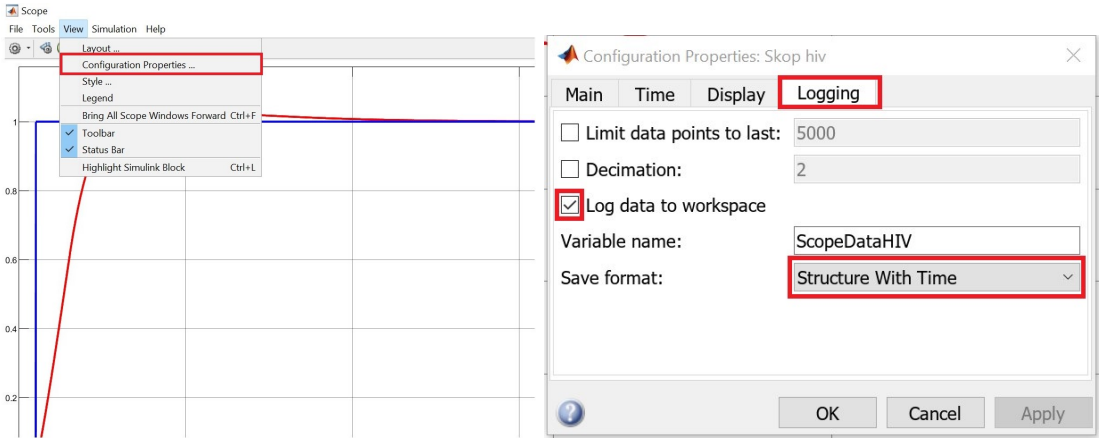
- modell_koeff.m
- SkopData.m
- ModeleringBevegelserSim.slx

A.3.4 Fremgangsmåte

For å linearisere modellene fra kapittel A.2 brukes System Identification programmet fra MATLAB. Den trenger inngangs- og utgangsverdier fra et system for å lage en linearisert modell.

For å bruke dataen fra simulink må de eksporteres til arbeidsområdet(eng:*workspace*) i MATLAB. Setter opp skopene som leser av verdiene for modellene utviklet i delkapittel A.2 til å logge data til MATLABs arbeidsområdet. Åpne skopene som er i filen RegulertHiv.slx vist i figur A.4. Etter det kan du:

1. Gå inn i view fanen
   (a) Trykk på Configuration Properties ...
2. Inne på Configuration Properties gå inn på Logging fanen
   (a) Huk av Log data to workspace
   (b) Endre variable navnet i dette eksempelet endres det til ScopeDataHIV
   (c) Endre Save format til Structure With Time
3. Figure A.10 illustrer stegene og hviser hva som er endret



Figur A.10: Innstillinger for å logge

For å bruke dataen i System Identification programmet lages det lister for hver verdi som brukes. Program filen SkopData.m gjør hele koden kan leses i A.1.

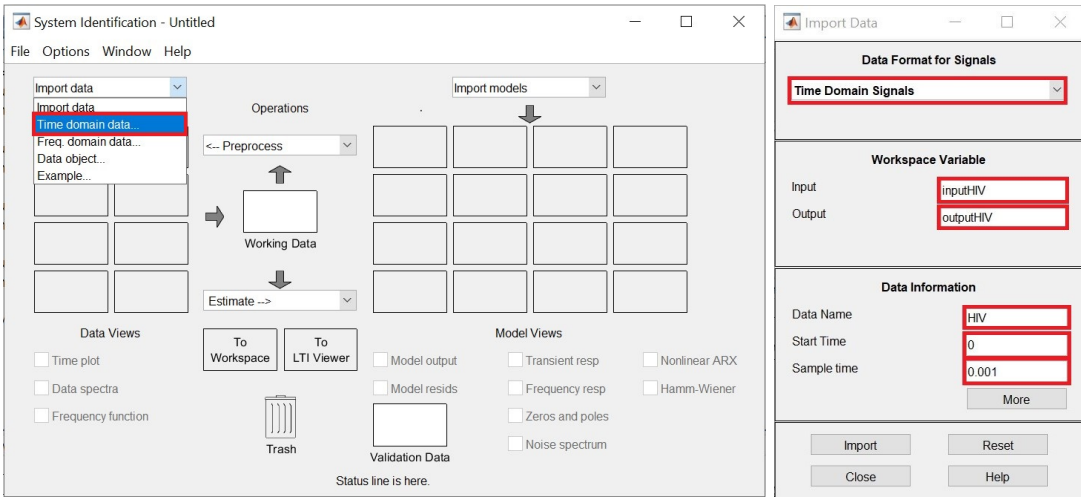
Kode A.1: Koden fra TransferFunction.py

```
t=out.tout; %Makes a time variable from simulink
outputHIV=out.ScopeDataHIV.signals.values(:,1); %Makes an output varible ...
    for hiv motion
inputHIV=out.ScopeDataHIV.signals.values(:,2); %Makes an input varible ...
    for hiv motion

outputGIR=out.ScopeDataGIR.signals.values(:,1); %Makes an output varible ...
    for gir motion
inputGIR=out.ScopeDataGIR.signals.values(:,2); %Makes an input varible ...
    for git motion

outputJAG=out.ScopeDataJAG.signals.values(:,1); %Makes an output varible ...
    for jag motion
inputJAG=out.ScopeDataJAG.signals.values(:,2); %Makes an input varible ...
    for jag motion

outputSVEI=out.ScopeDataSVEI.signals.values(:,1); %Makes an output ...
    varible for svei motion
```

```
12 inputSVEI=out.ScopeDataSVEI.signals.values(:,2); %Makes an input varible ...
       for sveiv motion
13
14 systemIdentification
```

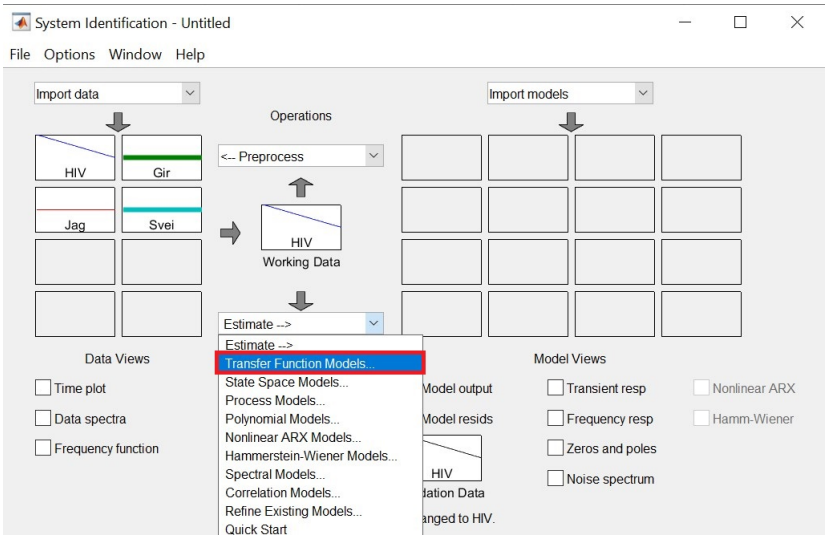
Når koden over er kjørt vil System Identification programmet åpnes. Først må signal dataen importeres på følgende måte:

1. Åpner Time domain data fra flervalgs menyen oppe til høyre hjørne-
   (a) I fanen Import Data skriver du inn navnet på listene med inngangs verdier og utgangs verdier. Her er input: inputHIV og output: outputHIV.
   (b) Data Name kan være hva enn du vil valgt her i kalle data'en HIV.
   (c) Start time er 0 da tiden begynner på 0 i simulink modellen
   (d) Sample time blir satt til det samme som i simulink modellen, i ModeleringBevegelserSim.slx er samplings tiden 0.001.
2. Figure A.11 illustrer stegene og hviser hva som er endret



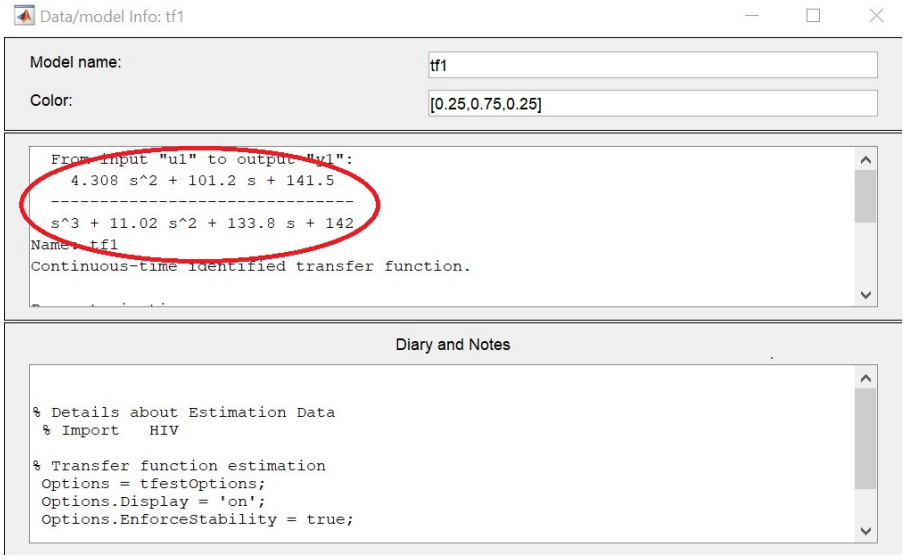
Figur A.11: Innstillinger for å importere data til System Identification

For å lage en overføringsfunksjon utifra inngangs og utgangsverdien trykker man på flervalgsmenyen som heter Estimate -> trykker så på Transfer Function Models som vist på figur A.12.



Figur A.12: Estimere en overførings funksjon ut ifra importert data

Dobbeltklikk på modellen som har blitt opprettet for å så å lese av overføringsfunksjonen. overføringsfunksjon er markert med en rød sirkel på figure A.13



Figur A.13: Se overføringsfunksjonen til den lineariserte modellen

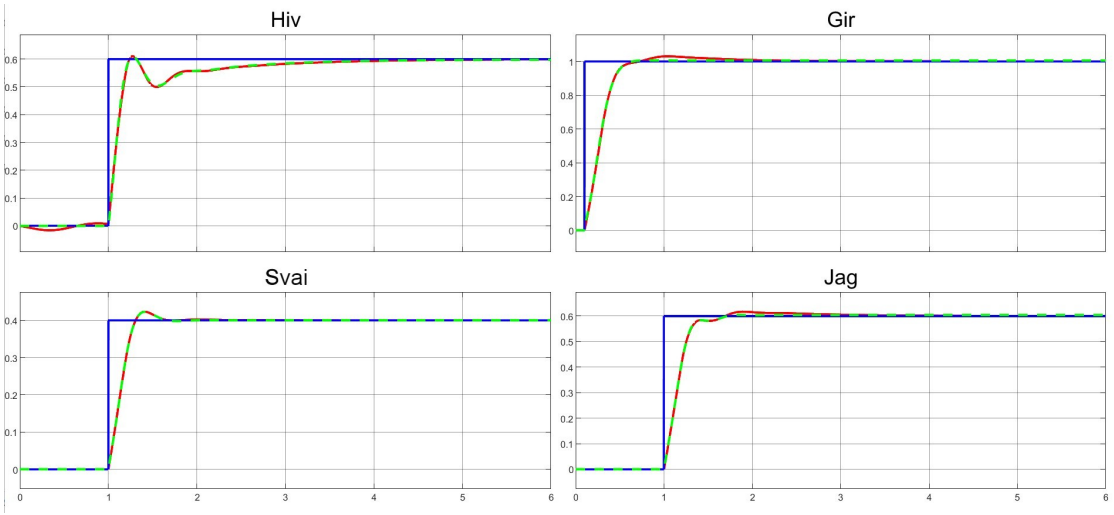
A.3.5 Resultat

Resultatet i tabell A.3 kommer fra å gjennomføre stegene fra kapittel A.3.4 for alle bevegelsene.

	overføringsfunksjon		overføringsfunksjon
Hiv	$\frac{4.308s^2+101.2s+141.5}{s^3+11.02s^2+133.8s+142}$	Jag	$\frac{3.606s^2+40.12s+1421}{s^3+16.98s^2+273.3s+1411}$
Svai	$\frac{4.014s^2+83.39s+2173}{s^3+31.09s^2+357.9s+2170}$	Gir	$\frac{3.74s^2+61.48s+2546}{s^3+38.65s^2+519.6s+2533}$

Tabell A.3

Figur A.14 viser stegresponsen til det vanlige systemene og lineariserte systemet der den grønne dottet linjen er det lineariserte systemet.



Figur A.14

A.4 Del 3 testing av overføringsfunksjonene i Python

Simulatoren som skal anvende overføringsfunksjonene, bruker programmeringsspråket Python. I denne testrapporten testes et program som anvender overføringsfunksjonen og returnerer utgangsverdier basert på inngangsverdi og overføringsfunksjon.

A.4.1 Programvare

- Python

A.4.2 Python biblioteker

- control
- numpy
- matplotlib

A.4.3 Programfiler

Link til GitHub mappe med filene:

- TransferFunction.py

A.4.4 Fremgangsmåte

For å kunne teste overføringsfunksjonene i Python må de gjøre om til en State Space modell. State Space modellene må så gjøre om til en diskret versjon av State Space modellene som tar hensyn til tidsskrittet.
Bruker State Space variablene til å regne ut utgangs signalet basert på inngangs signalet og den tidligere x matrisen.
Inngangs signalet er et steg signal, ved testing av hiv bevegelsens overføringsfunksjon går steget fra 0 til 0.6.
Resultatet blir plottet i en figur ved hjelp av matplotlib.
Koden som ble brukt for å lage grafen til hiv bevegelsen vises i kodeutdrag A.2.

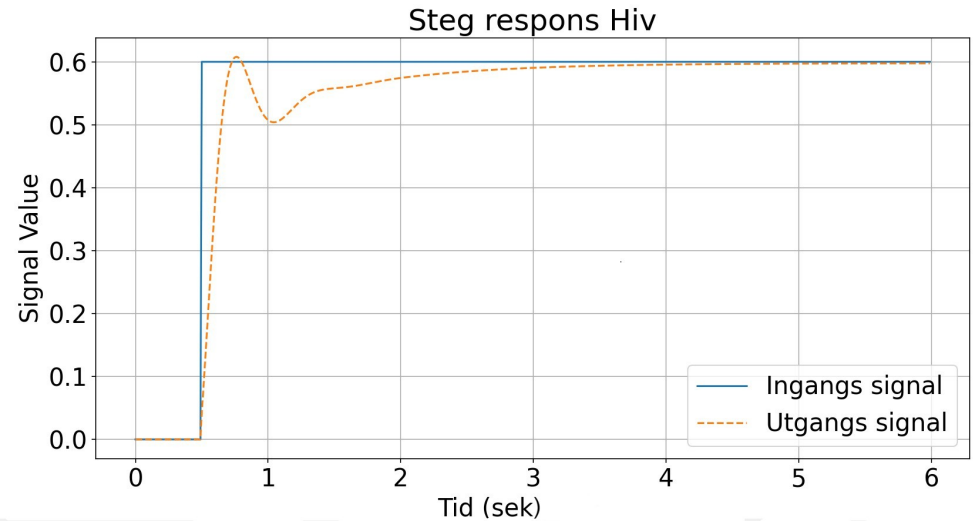
Kode A.2: Koden fra TransferFunction.py

```
1  import control as ctl
2  import numpy as np
3  import matplotlib.pyplot as plt
4  # Define the transfer function
5  numerator = [4.308, 101.2, 141.5]
6  denominator = [1, 11.02, 133.8, 142]
7
8  # Display the state-space matrices
9  dt = 0.01  # Time step (seconds)
10 t_end = 6  # End time (seconds)
11 t_array = np.arange(0, t_end, dt) #list with all the times
12 input_signal = [] #innit for input signal
13
14
15 ss = ctl.tf2ss(numerator,denominator) # Convert the transfer function to ...
       state-space representation
16 ss= ctl.c2d(ss,dt,"foh") #Makes the state model discret
17
18 # Access the state-space matrices directly
19 A = ss.A
20 B = ss.B
21 C = ss.C
22 D = ss.D
23 x = np.zeros((A.shape[0],))
24
25 for t in range(0, len(t_array), 1):
26     if t < 50: input_signal.append(0)
27     else: input_signal.append(0.6)
28
29 # Initialize output storage
30 output_signal = []
```

```
# Real-time simulation loop
for u in input_signal:
    # State update equation: x(k+1) = Ax(k) + Bu(k)
    x = np.dot(A, x) + np.dot(B, u)  # Directly update for discrete ...
        system, no dt multiplication
    # Output equation: y(k) = Cx(k) + Du(k)
    y = np.dot(C, x) + D * u
    output_signal.append(y[0][0])  # Assuming y_disc is scalar

#plots data
plt.figure(figsize=(10, 6))
plt.plot(t_array, input_signal, label='Ingangs signal')
plt.plot(t_array, output_signal, label='Utgangs signal', linestyle='--')
plt.title('Steg respons hiv')
plt.xlabel('Tid (sekkunder)')
plt.ylabel('Signal Value')
plt.legend()
plt.grid(True)
plt.show()
```

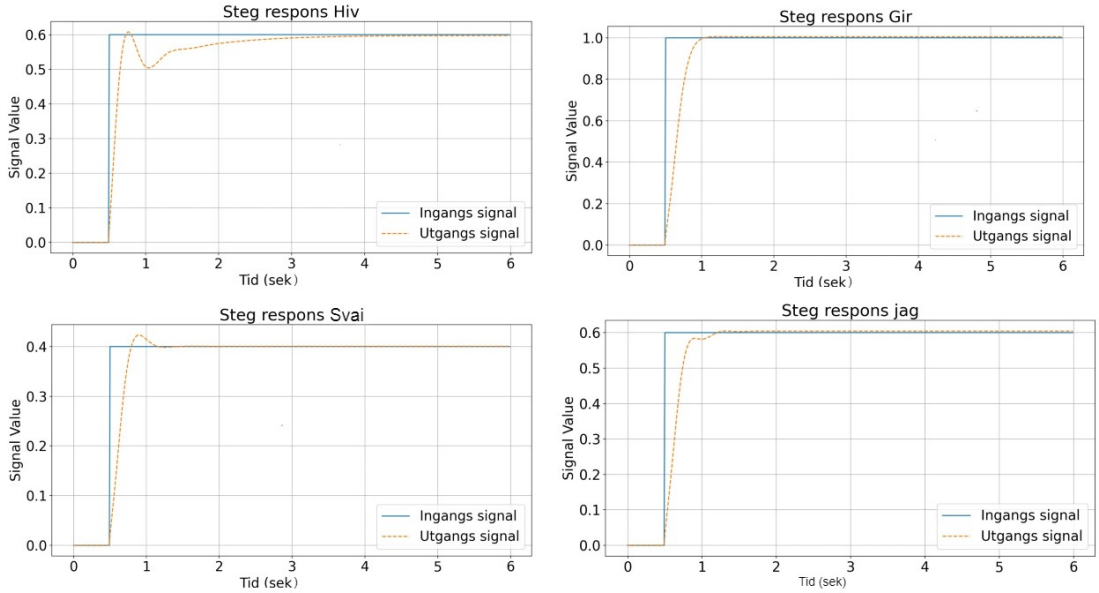
A.4.5 Resultat

Koden fra A.2 åpner grafen på figur A.15. Dette er samme respons som ble linearisert i A.3.4.



Figur A.15: Steg respons fra overføringsfunksjonen til hiv bevegelsen i Python

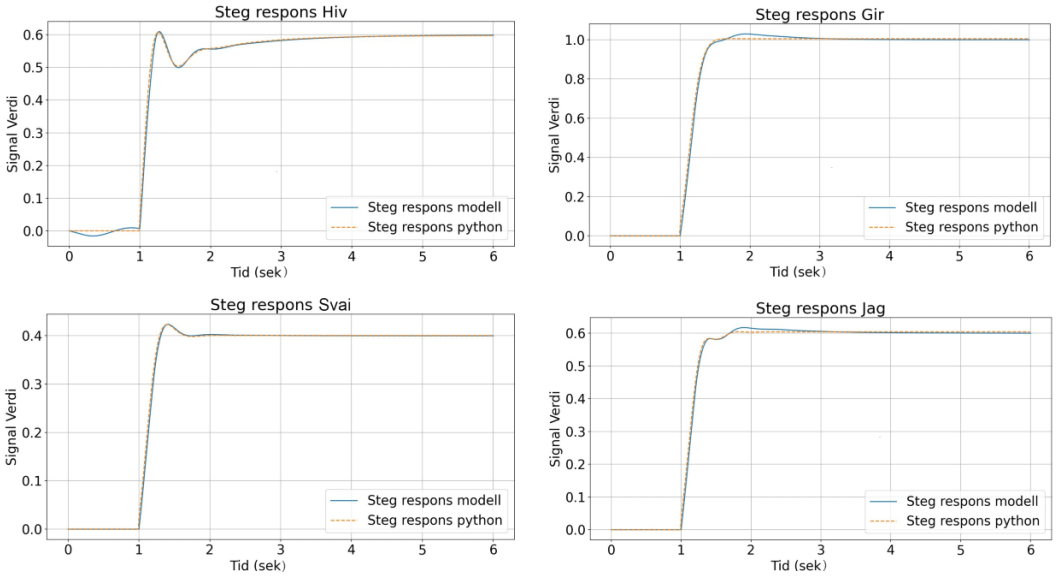
Figur A.16 viser responsen til alle bevegehelsene basert på overføringsfunksjonen i Python. Alle responsene ser like ut som de lineariserte responsene i A.3.4.



Figur A.16: Steg respons fra overføringsfunksjonene til alle bevegelsene

A.5 Konklusjon

Overføringsfunksjonene som har blitt funnet for bevegelsene har litt avvik, spesielt på gir og jag bevegelsene. overføringsfunksjonene kunne vært mere nøyaktige, dersom de var av høyere orden. Problemet med å høyere ordens likninger er at det kreves mere prosesseringskraft for å håndtere overføringsfunksjonene i simulatoren. Med hensyn til prosesseringskraft har valget vært å bruke overføringsfunksjoner av tredje orden. Modellene som er utviklet er heller ikke helt nøyaktige. Dermed godtas avviket til de lineariserte overføringsfunksjonene. Figur A.17 viser alle steg responsene til de forskjellige bevegelsene, der steg respons modell, er steg responsen til de originale modellene fra Simulink modellen.



Figur A.17: Steg respons fra overføringsfunksjonene til alle bevegelsene

Vedlegg B

Finne HSV-området for deteksjon av rørlinje

B.1 Programvare

- Python

B.2 Python biblioteker

- cv2
- cv2.aruco
- numpy

B.3 Programfiler

Link til GitHub mappe med filene:

- Find_hsv_vindeo.py
- Find_hsv_range.py
- image_handler_windows.py

B.4 Hensikt

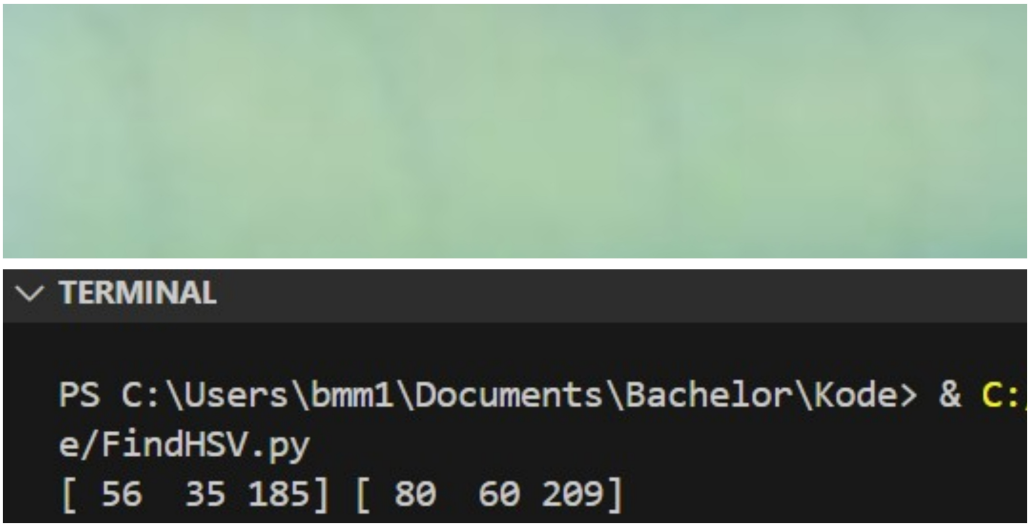
Funksjonene **find_pipe()** trenger et HSV-område til fargemaskering. Fargemaskeringen brukes til å filtrere vekk unødvendig bildeinformasjon, ved lokalisering av rørledningen som skal brukes under *TAC-challenge 2024* 1.5.1.

B.5 Fremgangsmåte

For å finne HSV-området til røret brukes to forskjellige Python program.

B.5.1 Finne HSV-området til et helt bilde

For å få hele HSV-området til et helt bilde brukes filen Find_hsv_range.py. Koden går igjennom HSV verdiene til hver piksel til bildet som blir angitt. Koden printer den laveste og høyste verdien for H, S og V.
Figur B.1 viser bildet Find_hsv_range.py har brukt og terminalen der HSV minimum og maksimum har blitt printet.

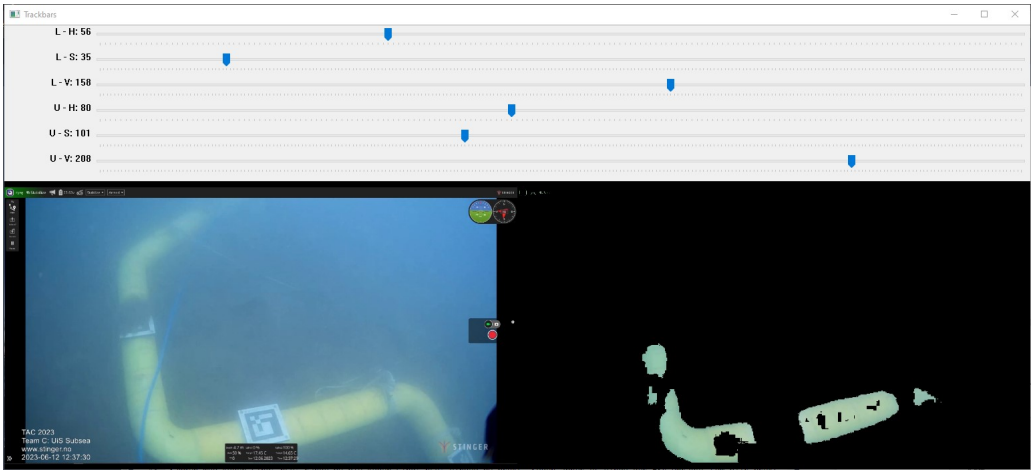


Figur B.1: Utskrift av HSV-området gitt i bilde

B.5.2 Finne HSV-området ved bruk av glide meny(eng:*trackbar*)

Koden find_hsv_vindeo.py blir brukt sammen med find_hsv_range.py. find_hsv_vindeo.py åpner et vindu der HSV-området som blir brukt til fargemaskeringen kan endres mens videoen blir spilt av. HSV-området endres ved å dra i glide menyene.

Videoen kan settes på pause ved å trykke på spacebar.
Et skjermutklipp av vinduet som åpnes av koden find_hsv_vindeo.py er vist i figur B.2.



Figur B.2: Utklipp fra programmet tilhørende find_hsv_video

B.5.3 Testing av HSV-området

For å teste HSV-området brukes image_handler_windows.py, som er en modifisert versjon av ImageHandler fra delkapittel 4.5.2. Koden har blitt modifisert for å fungere direkte på videofiler i windows og looper funksjonen **find_pipeline()**.

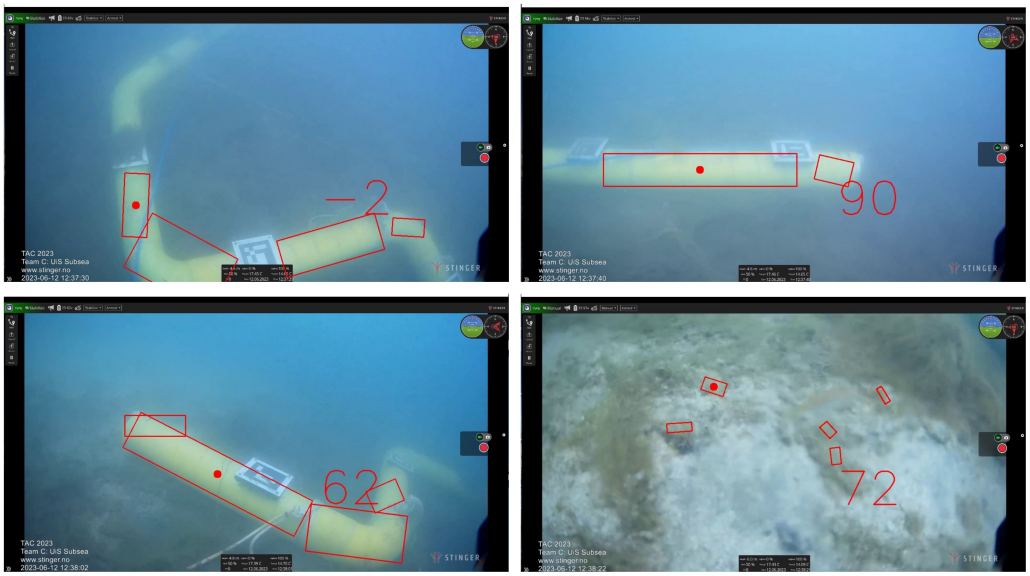
Figur B.3: Bildebehandling fra image_handler_windows.py med et HSV-område fra [46,35,158] til [82,101,229]

B.6 Resultat

Sammenligner resultatet med fem forskjellige HSV-områder. Det som som tas i betraktning er hvor mye rør som blir detektert og hvor mye som ikke er rør(støy) som blir detektert. Det er sammenlignet 4 forskjellige bilder fra videoen VideoRor.mp4.

B.6.1 HSV-området:[46,35,158],[82,101,229]

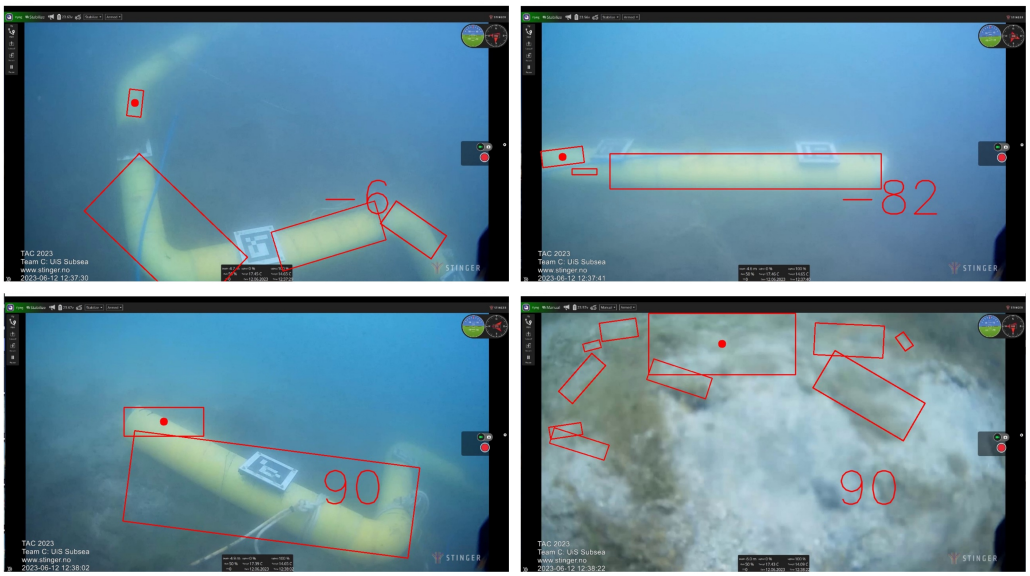
HSV-området [46,35,158],[82,101,229] detekterer røret men ikke hele, andre ting en røret ble detektert men begrenset. Figur B.4 viser resultatet.



Figur B.4: Gjenkjenning av rør med HSV-området [46,35,158],[82,101,229

B.6.2 HSV-området:[46,35,158],[89,101,229]

HSV-området [46,35,158],[89,101,229] detekterer store deler av røret, men også mye støy. Figur B.5 viser resultatet.



Figur B.5: Gjenkjenning av rør med HSV-området [46,35,158],[89,101,229]

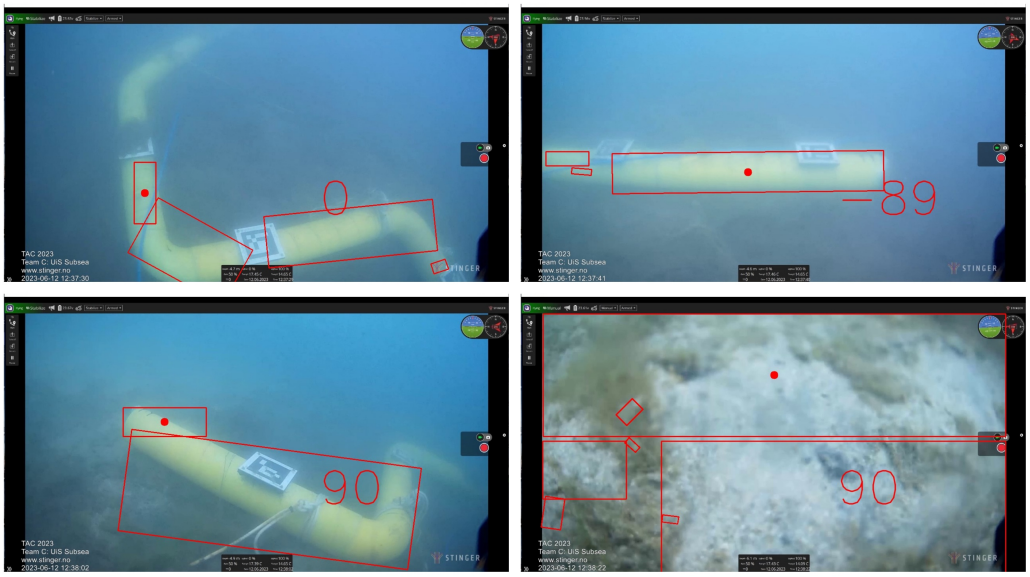
B.6.3 HSV-området:[46,33,133],[100,101,229]

HSV-området [46,33,133],[100,101,229] svært mye støy. Figur B.6 viser resultatet.

Figur B.6: Gjenkjenning av rør med HSV-området [46,33,133],[100,101,229]

B.6.4 HSV-området:[46,33,133],[100,74,228]

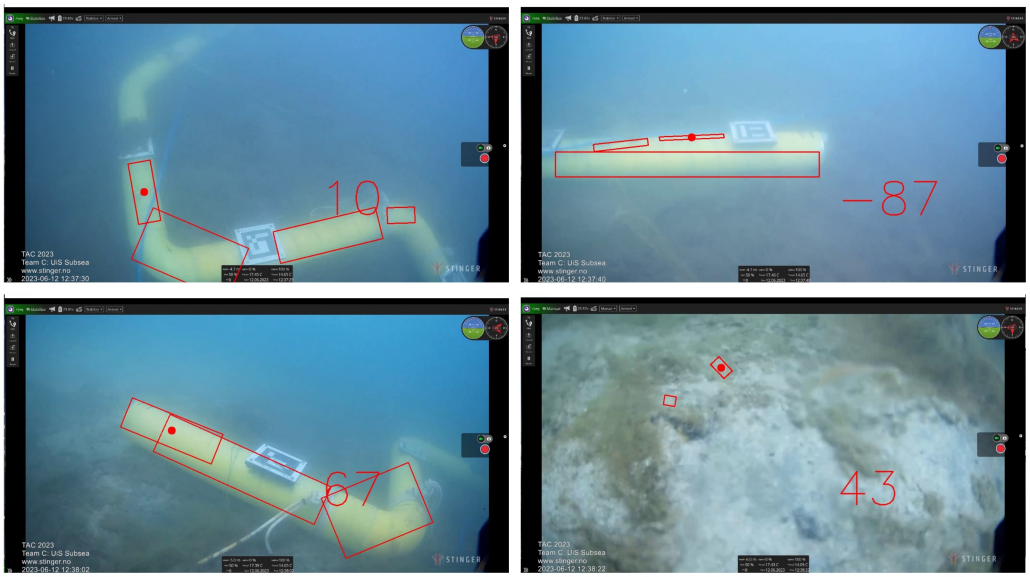
HSV-området [46,33,133],[100,101,229] detekterer litt rundt røret, men mye støy nærme bakken. Figur B.7 viser resultatet.



Figur B.7: Gjenkjenning av rør med HSV-området [46,33,133],[100,74,228]

B.6.5 HSV-området:[46,35,183],[89,101,229]

HSV-området [46,33,133],[100,101,229] detekterer røret, men deler det opp i flere bokser, ved bruk av dette området detekteres mindre støy enn de andre områdene. Figur B.8 viser resultatet.



Figur B.8: Gjenkjenning av rør med HSV-området [46,35,183],[89,101,229]

B.7 Konklusjon

Ved å sammenligne alle resultatene fra B.6, viser det seg at det første HSV-området ([46,35,158],[82,101,229]) er best egnet til å filtrere rørlinjen. Resultatet vises i figur B.4. Med dette HSV-området detekteres lite eller ingen støy. Røret blir ikke dektert når det eller deler av det er for langt opp eller til siden i bildet. Dette godtas, da det er bedre enn alternative områder med mere støy. Støy kan forsatt detekteres, men kun hvis ROV-en kjører veldig nærme bakken. Det vil være mulighet for testing under *TAC challenge 2024*, kan da området finjusteres enda mer med hensyn til dagsforholdene.

Vedlegg C

Finne HSV-området for deteksjon av benk

C.1 Programvare

- Python

C.2 Python biblioteker

- cv2
- cv2.aruco
- numpy

C.3 Programfiler

Link til GitHub mappe med filene:

- Find_hsv_vindeo.py
- Find_hsv_range.py
- image_handler_windows2.py

C.4 Hensikt

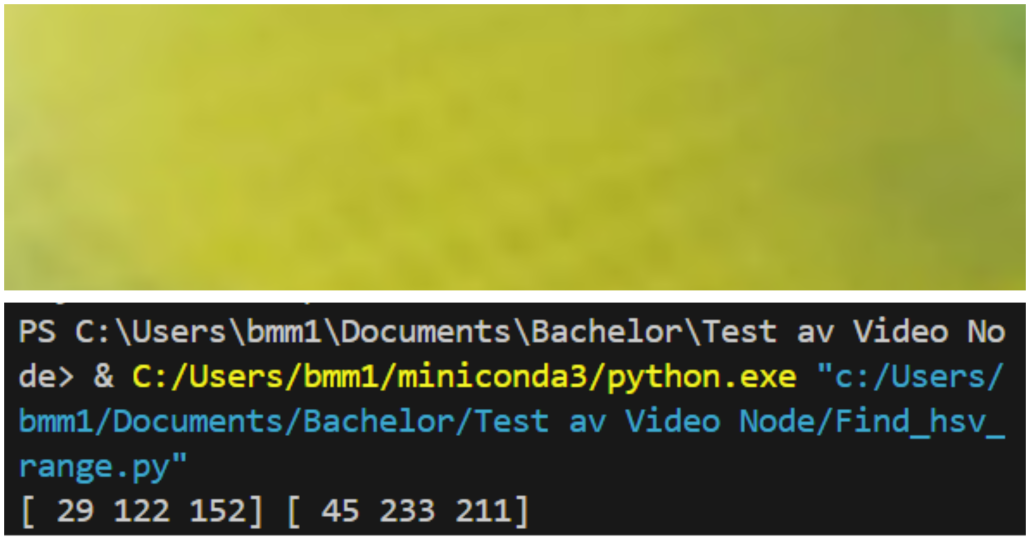
Funksjonen **find_bench()** trenger et HSV-område til fargemaskering. Fargemaskeringen brukes til å filtrere vekk unødvendig bildeinformasjon, ved lokalisering av benken som skal brukes under *TAC-challenge 2024*[1]

C.5 Fremgangsmåte

For å finne HSV-området til benken brukes to forskjellige Python program.

C.5.1 Finne HSV-området til et helt bilde

For å få hele HSV-området til et helt bilde brukes filen Find_hsv_range.py. Koden går igjennom HSV verdiene til hver piksel til bildet som blir angitt. Koden printer den laveste og høyste verdien for H, S og V.
Figur C.1 viser bildet Find_hsv_range.py har brukt og terminalen der HSV minimum og maksimum har blitt printet.

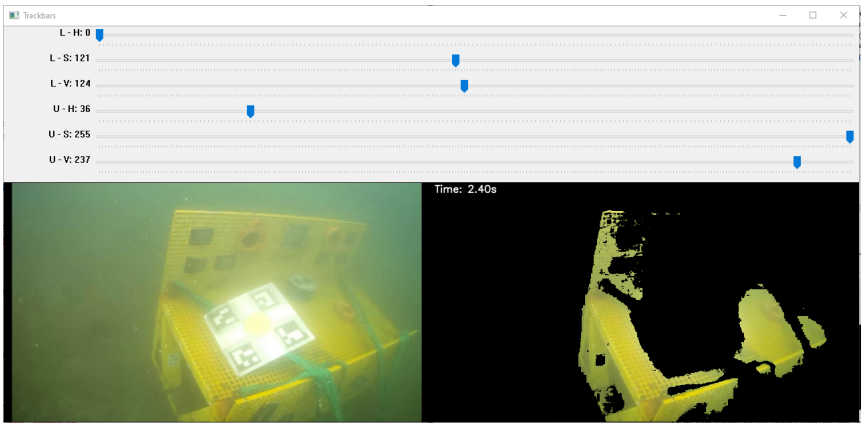


Figur C.1: Utskrift av HSV-området gitt i bildet

[1]Benken er beskrevet i underkapittel 1.5.2.

C.5.2 Finne HSV-området ved bruk av glide meny(eng:*trackbar*)

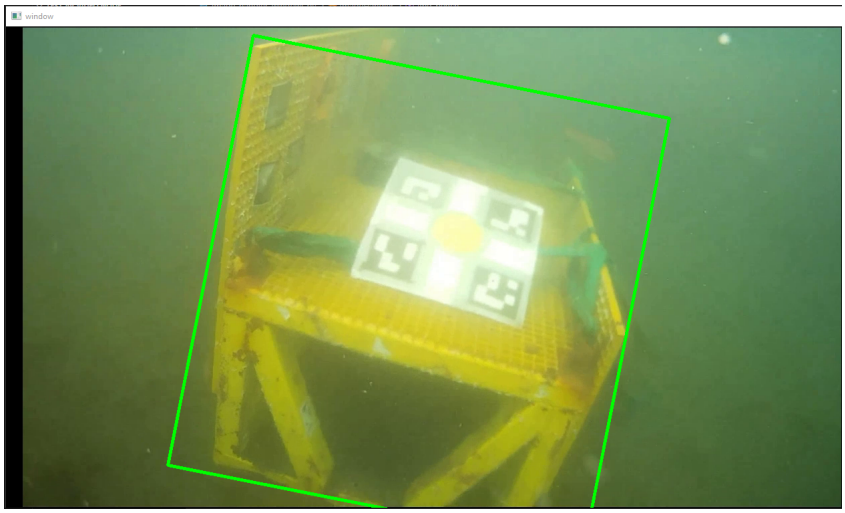
Koden find_hsv_vindeo.py blir brukt sammen med find_hsv_range.py. find_hsv_vindeo.py åpner et vindu der HSV-området som blir brukt til fargemaskeringen kan endres mens videoen blir spilt av. HSV-området blir endret ved å dra i glide menyene.
Videoen kan settes på pause ved å trykke på mellomroms knappen.
Et skjermutklipp av vinduet som åpnes av koden find_hsv_vindeo.py er vist i figur C.2.



Figur C.2: Utklipp fra programmet tilhørende find_hsv_video

C.5.3 Testing av HSV-området

For å teste HSV-området brukes image_handler_windows2.py, som er en modifisert versjon av ImageHandler fra delkapittel 4.5.2. Koden har blitt modifisert for å fungere direkte på videofiler i windows og looper funksjonen **find_bench()**.



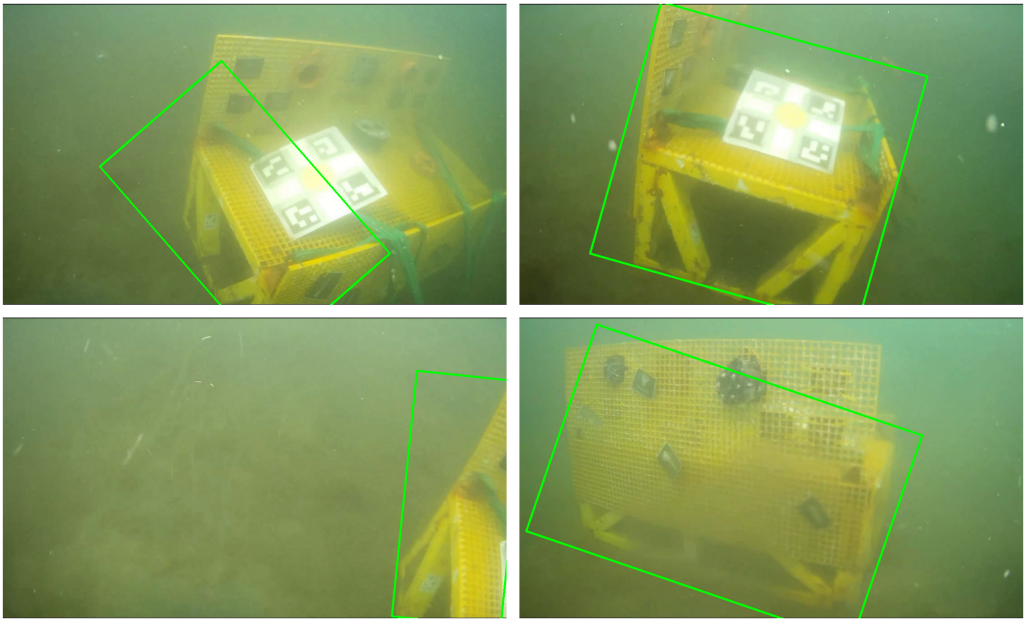
Figur C.3: Bildebehandling fra image_handler_windows2.py med HSV-området fra [0,121,124] til [36,255,237]

C.6 Resultat

Sammenligner resultatet med fem forskjellige HSV-området. Det som som tas i betraktning er hvor mye av benken som blir detektert og hvor mye som ikke er benken som blir detektert. Det er sammenlignet fire forskjellige bilder fra videoen Benk.mp4.

C.6.1 HSV-området:[0,121,183],[36,255,237]

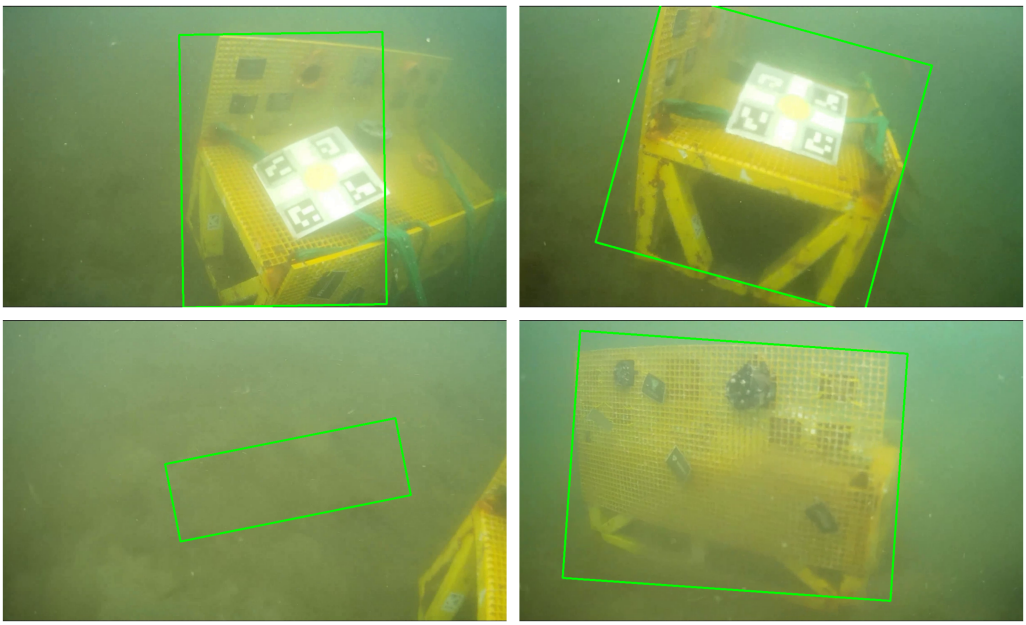
HSV-området på [0,121,183],[36,255,237] detekterer benken men ikke hele ved skrå vinkel, området utenfor benken blir også detektert. Figur C.4 viser resultatet.



Figur C.4: Gjenkjenning av benk med et HSV-området på [0,121,183],[36,255,237]

C.6.2 HSV-området:[0,85,141],[38,255,237]

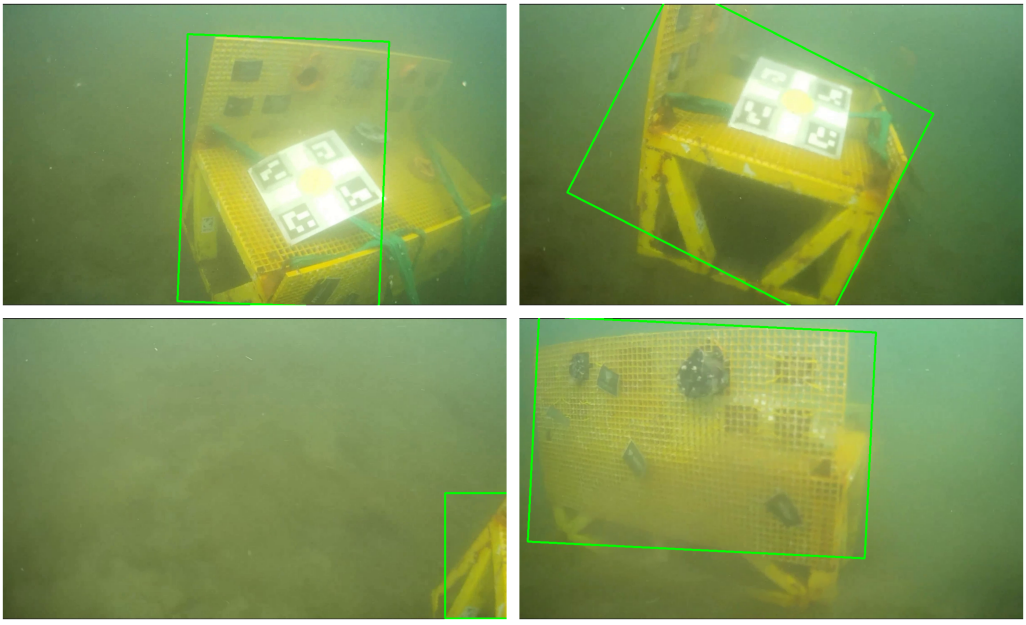
HSV-området på [0,85,141],[38,255,237] detekterer benken men ikke hele ved skrå vinkel, havbunnen blir også detektert. Figur C.5 viser resultatet.



Figur C.5: Gjenkjenning av benk med et HSV-området på [0,85,141],[38,255,237]

C.6.3 HSV-området:[0,87,150],[38,255,248]

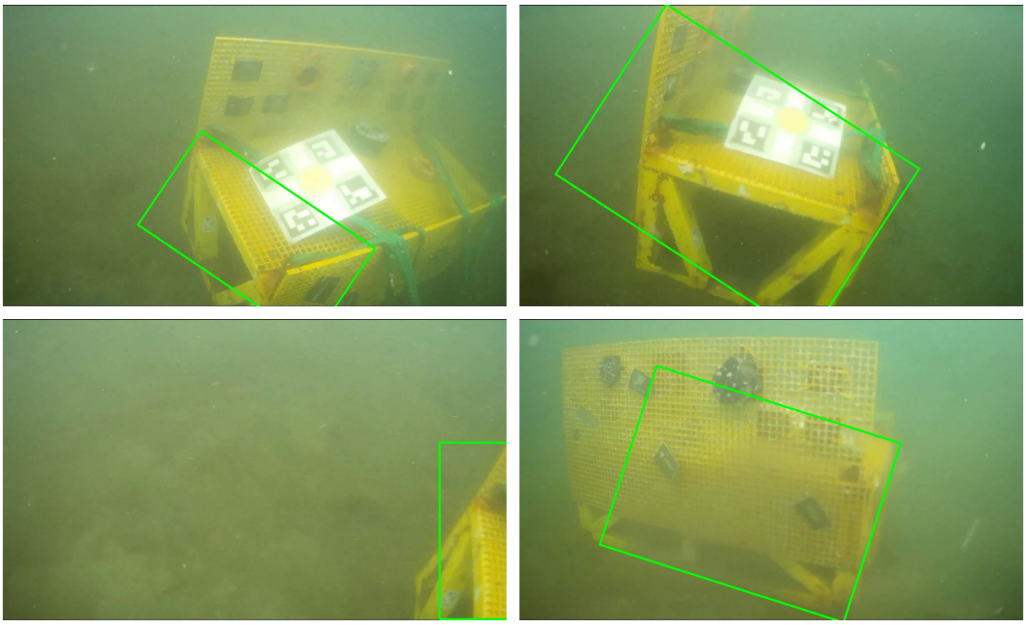
HSV-området på [0,85,141],[38,255,237] detekterer benken men ikke hele ved skrå vinkel. Detekterer områder rundt benken, men meget begrenset. Veldig lignende resultat som i delkapittel C.6.1 bare med litt forbedring i deteksjon av benken. Figur C.6 viser resultatet.



Figur C.6: Gjenkjenning av benk med et HSV-området på [0,87,150],[38,255,248]

C.6.4 HSV-området:[0,38,38],[31,255,255]

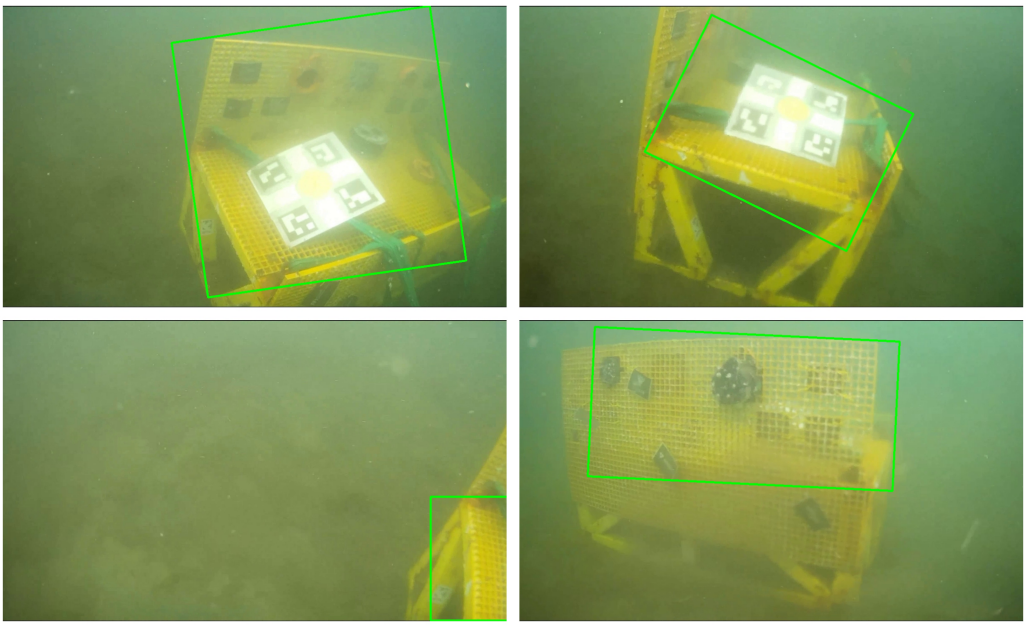
HSV-området på [0,38,38],[31,255,255] er endret slik at det ikke skal detekteres så stort fargespekter men heller flere variasjoner av få farger. Dette fungererte ikke som tenkt da hele benken ikke ble detektert fra vinkel eller fra baksiden. Figur C.7 viser resultatet.



Figur C.7: Gjenkjenning av benk med et HSV-området på [0,38,38],[31,255,255]

C.6.5 HSV-området:[0,24,179],[49,255,238]

HSV-området på [0,24,179],[49,255,238] detekterer store deler av benken fra skrå vinkel men ikke hele benke bakfra. Figur C.8 viser resultatet.



Figur C.8: Gjenkjenning av benk med et HSV-området på [0,24,179],[49,255,238]

C.7 Konklusjon

Ved å sammenligne alle resultatene fra kapittel C.6, viser det seg at det tredje HSV-området ([0,87,150],[38,255,248]) er best egnet. Resultatet er vist på figur C.6.

Med dette HSV-området detekteres store deler av benken. Deler av benken blir ikke dektert når ROV-en er på siden av benken. Omgivelsene blir ikke detektert. Videoen fått fra *TAC challenge* teamet har ikke videostrøm av benken fra alle vinkler dette gjør at testingen ikke er optimal. Det hadde vært optimalt med videostrøm der ROV-en ser rett på forsiden av benken. Det vil være mulighet for testing på *TAC challenge* å, kan da gjøres enda flere justeringer til HSV-området

Vedlegg D

Kalibrering av kamera

D.1 Utstyr

- Kamera (exploreHD 3.0)

D.2 Programvare

- Python

D.3 Python biblioteker

- cv2
- numpy

D.4 Programfiler

Link til GitHub mappe med filene:

- KalibrererKamera.py
- TestKalibreringsMatrise.py

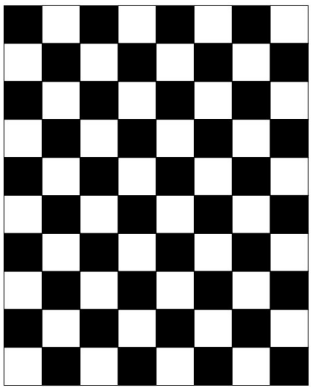
D.5 Hensikt

Kameraene på ROV-en har fiskeøye linser dette skaper linseforvrengning. Denne forvrengningen gjør at rette linjer fremstår buet. Figur D.1 viser et eksempel på en dør som fremstår buet. For å bli kvitt denne forvrengingen brukes en kalibrerings matrise og tilhørende forvrengning koeffisient.

Figur D.1: Originalt bildet fra exploreHD 3.0

D.6 Fremgangsmåte

Framgangsmåten er sterkt inspirert av OpenCv sin kamera kalibrerings artikkel [18]. Programmet beskrevet er har blitt redigert til å fungere med exploreHD 3.0 og det er brukt et sjakkbrett som er 7×9 som vist på figur D.2.



Figur D.2: Sjakkbrett brukt til kalibrering

Python programmet KalibrerKamera.py tar inn bildestrøm fra kameraet koblet til PC-en med en USB-kabel. Hvis sjakkbrettet blir registert vil programmet:

- Legge hjørnene til en liste
- Tegne på hjørnene på bildet som vises
- kjøre **cv2.calibrateCamera()** som gir oss kalibrerings matrisen og forvrengning koeffisient

For at kalibrerings matrisen skal bli så nøyaktig som mulig trengs det flere bilder av sjakkbrettet fra forskjellige vinklet. Når KalibrerKamera.py kjøres, beveges USB-kamera rundt for å få bilder av sjakkbrettet fra forskjellige vinkler.

Programmet avsluttes ved taste Q på tastaturet, deretter vill kalibrerings matrisen, forvrengning koeffisient og antall hjørner som har blitt brukt printes til terminalen.

D.6.1 Implementering av kalibrerings matrisen

Kalibrerings matrisen og forvrengning koeffisienten blir implementert med koden i TestKalibreringsMatrise.py. Hovedfunksjonene er

- **cv2.getOptimalNewCameraMatrix():** Optimaliserer kalibrerings matrisen.
- **cv2.initUndistortRectifyMap()**: Lager to variabler for å fjerne forvrenginger i x og y retning
- **cv2.remap()**: Bruker variablene fra **cv2.initUndistortRectifyMap()**: for å fjerne forvrengingene fra bilde.

Bildet blir også beskjært etter at forvrengingen er fjernet.

D.7 Resultater

Det er forskjellige forvrenginger for forskjellige oppløsning. Det er derfor kalibrert til tre forskjellige oppløsninger: 640x480, 1280x720 og 1920x1080.

D.7.1 640x480

Fra KalibrerKamera.py hvor den brukte 26 forskjellige bilder av sjakkbrettet, ble kalibrering matrisen:

$$\begin{bmatrix} 426.2 & 0 & 320.4 \\ 0 & 429.8 & 225.3 \\ 0 & 0 & 1 \end{bmatrix} \tag{D.1}$$

Forvrengning koeffisienten ble:

$$\begin{bmatrix} -0.357 & 0.117 & 0.011 & 3.763 \cdot 10^{-3} & -1.392 \cdot 10^{-2} \end{bmatrix} \tag{D.2}$$

D.7.2 1280x720

Fra KalibrerKamera.py hvor den brukte 34 forskjellige bilder av sjakkbrettet, ble kalibrering matrisen:

$$\begin{bmatrix} 611.1 & 0 & 595.4 \\ 0 & 618.5 & 343.1 \\ 0 & 0 & 1 \end{bmatrix} \tag{D.3}$$

Forvrengning koeffisienten ble:

$$\begin{bmatrix} -0.397 & 0.189 & 9.87 \cdot 10^{-3} & 2.94 \cdot 10^{-3} & -4.81 \cdot 10^{-2} \end{bmatrix} \tag{D.4}$$

D.7.3 1920x1080

Fra KalibrerKamera.py hvor den brukte 36 forskjellige bilder av sjakkbrettet, ble kalibrering matrisen:

$$\begin{bmatrix} 942.6 & 0 & 998.5 \\ 0 & 940.4 & 483.6 \\ 0 & 0 & 1 \end{bmatrix} \tag{D.5}$$

Forvrengning koeffisienten ble:

$$\begin{bmatrix} -0.400 & 0.210 & 7.31 \cdot 10^{-3} & -6.25 \cdot 10^{-3} & -7.03 \cdot 10^{-2} \end{bmatrix} \tag{D.6}$$

Figur D.3 viser originalt bildet og bildet hvor forvrengingene er fjernet ved en oppløsning på 640x480 piksler. Det er tegnet på en rød strek for å se effekten av å fjerne forvrengingene.

Figur D.3: Sammenligning av originalt bildet fra kamera og med kalibrerings matrise

D.8 Konklusjon

Kalibrerings matrisen og forvrengning koeffisienten fjerner meste parten av forvrengingene. Kalibrerings matrisen kunne vært mere nøyaktig ved bruk av enda flere bilder av sjakkbrettet. For formålet som kameranene skal brukes til, er det tilstrekkelig med matrisen og koeffisienten som er brukt.
En ulempe med å fjerne forvrengningen er at deler av bildet blir borte. Dette fordi bildet ser ut som figur D.4 men blir beskjært. Dette gjør at du får litt mindre informasjon ut av bilde.

Figur D.4: Bildet som ikke er beskjært

Vedlegg E

Test av bildebehandling med direkte bildestrøm

E.1 Utstyr

- Kamera (exploreHD 3.0)

E.2 Programvare

- Python

E.3 Python biblioteker

- cv2
- numpy
- signal
- ROS2 biblioteker
- time

E.4 Programfiler

Link til GitHub mappe med filene:

- KalibrererKamera.py
- TestKalibreringsMatrise.py

I tillegg til de vedlagte filene brukes filene fra ROS som beskrevet i simulerings kapittelet 4.

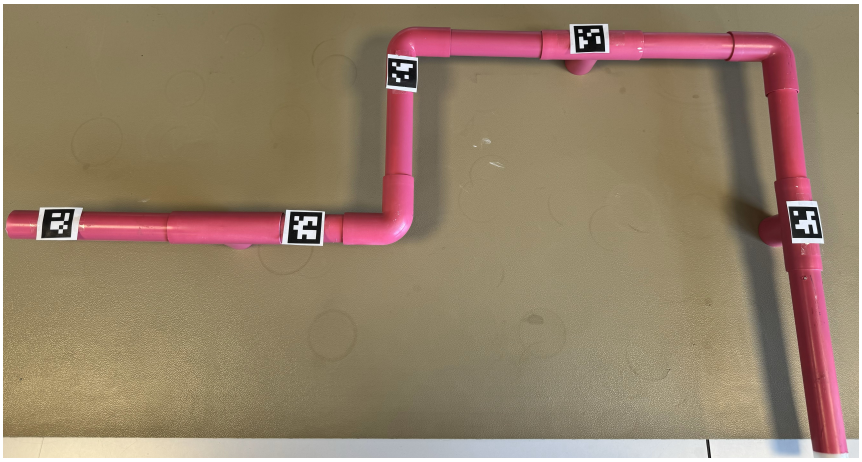
E.5 Hensikt

Bildebehandlingen som er utviklet for rørlinje inspeksjon skal testes med bildestrøm som kommer fra kameraet som brukes på ROV-en.

E.6 Fremgangsmåte

For testing av bildebehandlingen for rørlinje inspeksjon lages det en modifisert versjon av m_pipeline_node.py beskrevet i delkapittel 4.4.8 og en ny oppstarts fil. Bruker forstatt samme bildebehandlings algoritmer (image_handler.py) beskrevet i delkapittel 4.5.2.

Det var et rosarør tilgjengelig så dette brukes med fem AruCo-koder teipet til seg. Rørledningen som er brukt er vist i figur E.1

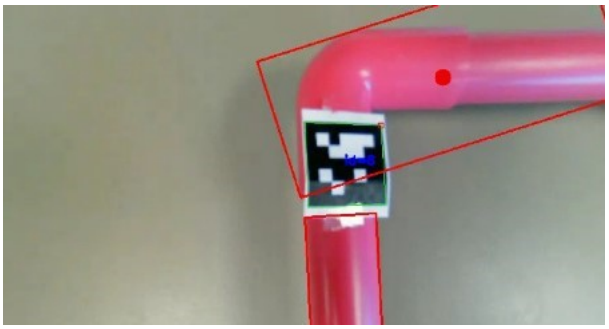


Figur E.1: Røret som skaø brukes til testing av bildebehandlingen

For å finne HSV-området til det rosa røret brukes samme metode som i testrapport B. HSV-området som brukes til fargemaskering: [126, 45, 87] - [179, 255, 255].

E.7 Resultat

Bildebehandlingen detekterer den rosa rørlinjen og analyserer den øverste delen av røret, for å finne center og vinkelen til den røde boksen. Figur E.2 viser et programmet detektert røret og leser av AruCo-koden. Hele videoen av rørlinje inspeksjonen kan ses på videoen her.



Figur E.2: Bildestrøm hvor røret er detektert og avlesning av AruCo-koden

Når programmet ikke detekterte røret, ble listen med AruCo-koder printet til terminalen som vist i figur E.3. Listen ble printet i riktig rekkefølge og uten duplikater.

```
[mission_pipeline_node]: Aruco List:[11, 2, 8, 5, 9]
```

Figur E.3: Liste med detekterte AruCo-koder

E.8 Konklusjon

Bildebehandlingen implementert for rørlinje inspeksjonen viser tilfredstillende resultater. Problemet med denne testen er at det ikke var mulig å teste under lignende omstendigheter. Det er gjort en test av å detektere det den faktiske røret i testrapporten Finne HSV-området for rørlinjen (vedlegg B). Der videoen som ble brukt kom fra *TAC challange* teamet som ble innspilt i fjor sommer.

Vedlegg F

Dødtid for ROV-en

F.1 Programvare

- Python

F.2 Filer

Brukt hele koden for autonom kjøring beskrevet i kapittel 4 utenom noden for implementering av overføringsfunksjonene. GitHub mappen for prosjektet

F.3 Hensikt

Skogestad-metoden krever dødtiden til systemet. Finner dermed dødtiden utifra tider fra tidligere år og fra koden utviklet i år.
Regulerings gruppen fra 2023 [12] estimerete en treghet på reguleringssystemer på 0.2 sekunder beskrevet i delkapittel 8.1.1(parameterbestemmelse). De som skrev bacheloren for 2023 for Subsea skrev ikke noe om forsinkelse i bildestrøm, bruker derfor heller resultatet 0.13sek fra delkapittel 5.1.2(Test av videooverføring) i følgende rapport fra 2022: [13].

F.4 Fremgangsmåte

For å finne tregheten til programmet for autonom kjøring brukes et sprang på 90° for gir bevegelsen. Noden for implantering av overføringsfunksjoner er tatt vekk siden den ikke vil brukes ved kjøring av den ordentlige ROV-en. Fremgangsmåten blir å loggføre pådra-

get(Ang_vel) gitt til ROV-en sammen med den faktiske målingen(cmd_vel) av farten til ROV-en.

F.5 Resultat

Figur F.1 viser terminalen ved spranget på 90°. Ser hvor lang tid det tar fra det ønskete pådraget blir sendt til ROV-en.

```
[1715258688.201777062] [mission_pipeline_node]: angle:-0.0, ang_vel:0.0, cmd_vel 0.0, odom -0.0
[1715258688.322579782] [mission_pipeline_node]: angle:1.57, ang_vel:4.04, cmd_vel 0.0, odom -0.0
[1715258688.400322636] [mission_pipeline_node]: angle:1.57, ang_vel:1.61, cmd_vel 0.0, odom -0.0
[1715258688.495149751] [mission_pipeline_node]: angle:1.57, ang_vel:1.64, cmd_vel 0.0, odom -0.0
[1715258688.557752997] [mission_pipeline_node]: angle:1.57, ang_vel:1.66, cmd_vel 0.0, odom -0.0
[1715258688.624399977] [mission_pipeline_node]: angle:1.57, ang_vel:1.67, cmd_vel 0.0, odom -0.0
[1715258688.692319922] [mission_pipeline_node]: angle:1.57, ang_vel:1.67, cmd_vel 0.0, odom -0.0
[1715258688.803210901] [mission_pipeline_node]: angle:1.57, ang_vel:1.67, cmd_vel 0.0, odom -0.0
[1715258688.870044841] [mission_pipeline_node]: angle:1.57, ang_vel:1.67, cmd_vel 4.04, odom -0.0
```

Figur F.1: Konsollen for testing av dødtid

For å finne tregheten i programmet, brukes tiden programmet faktisk sendte ut et pådrag minus tiden avviket oppstod. Ligning (F.1) kalkulerer denne tiden, obs bruker kun siste heltall fra tiden og to desimaler.

$$8.87 - 8.32 = 0.55 \tag{F.1}$$

Legger sammen tiden koden bruker på å sende ut et pådrag pluss treghetene til reguleringen og videostrømmen.Estimat for dødtid for hele systemet beregnes i ligning (F.2).

$$0.55 + 0.2 + 0.13 = 0.88 \tag{F.2}$$

F.6 Konklusjon

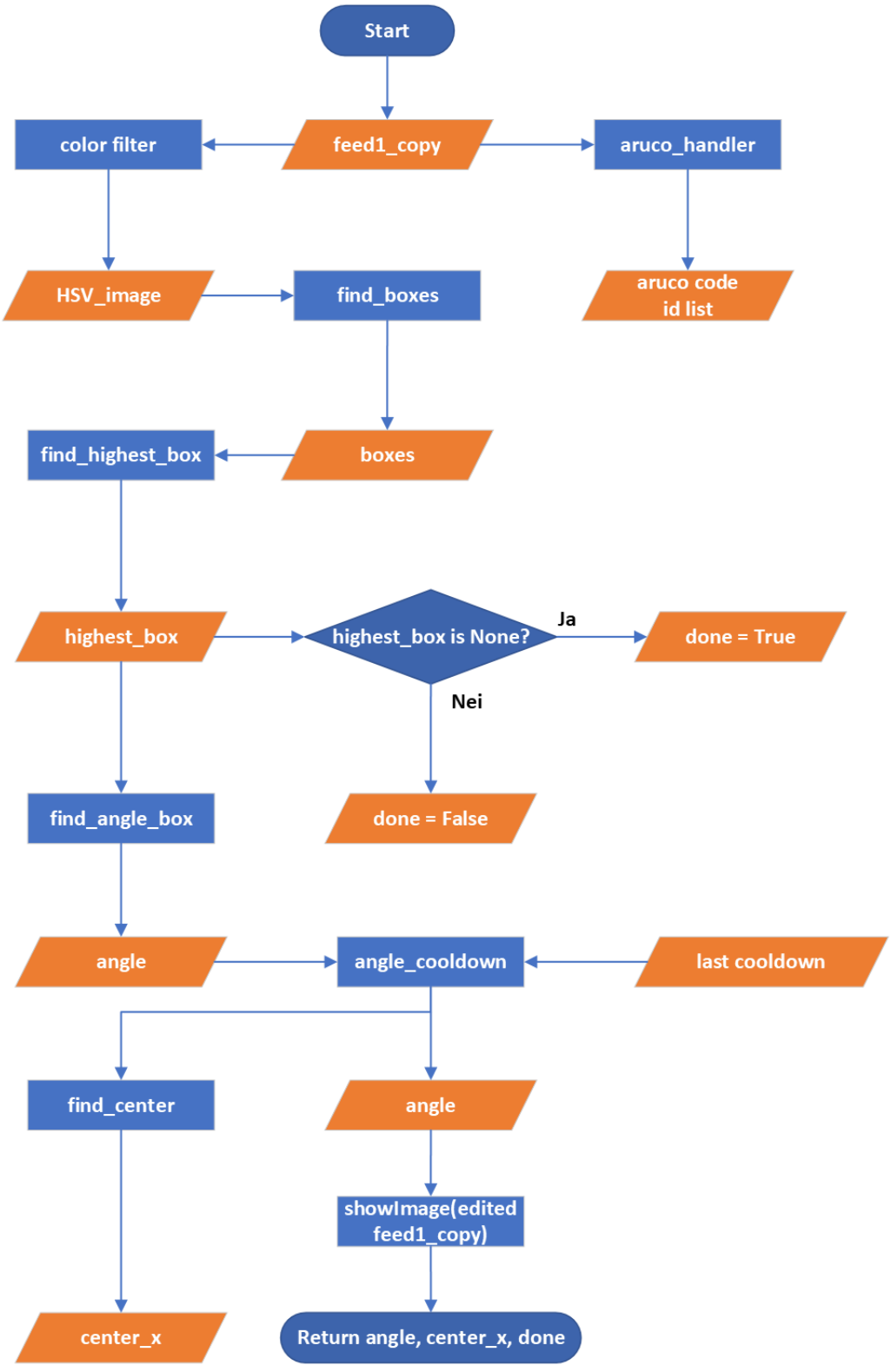
Dødtiden til systemet estimeres til 0.88 sekunder, som objektivt er meget tregt. Hovedkilden til denne dødtiden viser seg å være den delen av programmet som beregner og sender ut pådrag.

Vedlegg G

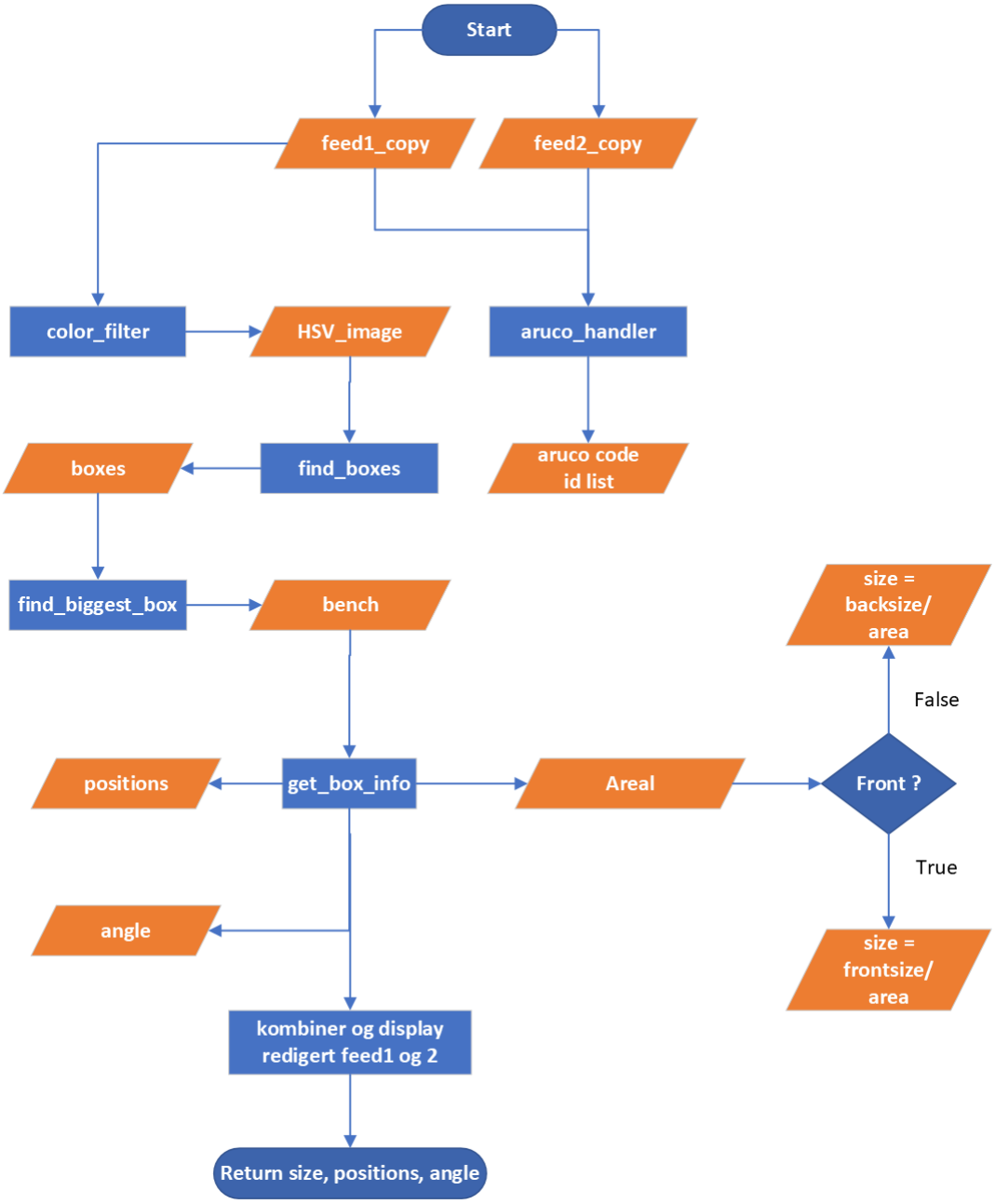
Figurer

Er nå i uke:	12														Vanntest								
Uke Nr	2023	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Beregnet	Faktisk Tid brukt	Fullført?
Prosjekt styring																								
Planlegging	11	9	6	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	55	
Møter	0	6,5	6,5	0	2	3,5	5	0	3	3,5	2,5	0	0	0	0	0	0	0	0	0	0	100	32,5	
Opplæring	4	0	2	0	2	22	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	36	
Totalt	15	15,5	14,5	29	4	25,5	11	0	3	3,5	2,5	0	0	0	0	0	0	0	0	0	0	170	87,5	
Planlegging																								
Behovsspesefikasjon	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	JA
Funksjonsspesefikasjon	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	1	JA
Blokkskjema	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	JA
Totalt	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32	3	
Programvare																								
Opplæring ROS	0	0	36,5	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	60,5	
Bildebehandling	0	0	15	0	0	0	28	9	0	0	0	0	0	0	0	0	0	0	0	0	0	100	52	
Simulering	0	0	0	0	18	13	20	6,5	30	17	3	7	0	0	0	0	0	0	0	0	0	75	114,5	
Pipeline inspeksjon	0	0	0	13,5	2,5	6	1	27,5	0	0	4	0	0	0	0	0	0	0	0	0	0	100	54,5	
Retur til hjem posisjon	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	30	2	
Objekt inspeksjon	0	0	0	0	0	0	7	0	20	9	9	11	0	0	0	0	0	0	0	0	0	100	56	
Ut data	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0	
Integrering	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	2	
Videreutvikling	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0	
Totalt	0	0	51,5	37,5	22,5	19	56	43	50	26	17	19	0	0	0	0	0	0	0	0	0	475	341,5	
Testing og Optimalisering																								
Ut data	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	
Pipeline inspeksjon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	
Objekt inspeksjon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	
Totalt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	130	0	
Uke Nr	2023	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Beregnet	Faktisk Tid brukt	Fullført?
Rapport skriving/teori																								
Oppsett	0	0	0	0	7,5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	10	8,5	
Innleding og intro subsea	0	0	0	0	6	0	0	0	1	0	0	0	7,5	0	0	0	0	0	0	0	0	16	14,5	
Planlegging	0	0	0	0	8	11,5	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	4	22,5	
Bildebehandling	0	0	5,5	22	6	9	17	0	4	0	0	0	0	0	0	0	0	0	0	0	0	50	63,5	
Simulering	0	0	0	0	0	0	0	0	0	21	52	47,5	0	0	0	0	0	0	0	0	0	50	120,5	
Objekt inspeksjon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	
Resultatanalyse	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	
Diskusjon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	
Konklusjon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	
Rettskriving og Forbedring	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80	0	
Totalt	0	0	5,5	22	27,5	20,5	17	0	7	23	52	47,5	7,5	0	0	0	0	0	0	0	0	410	229,5	
Totalt	15	18,5	71,5	51	54	46	84	43	60	26,5	71,5	66,5	7,5	0	0	0	0	0	0	0	0	1217	661,5	

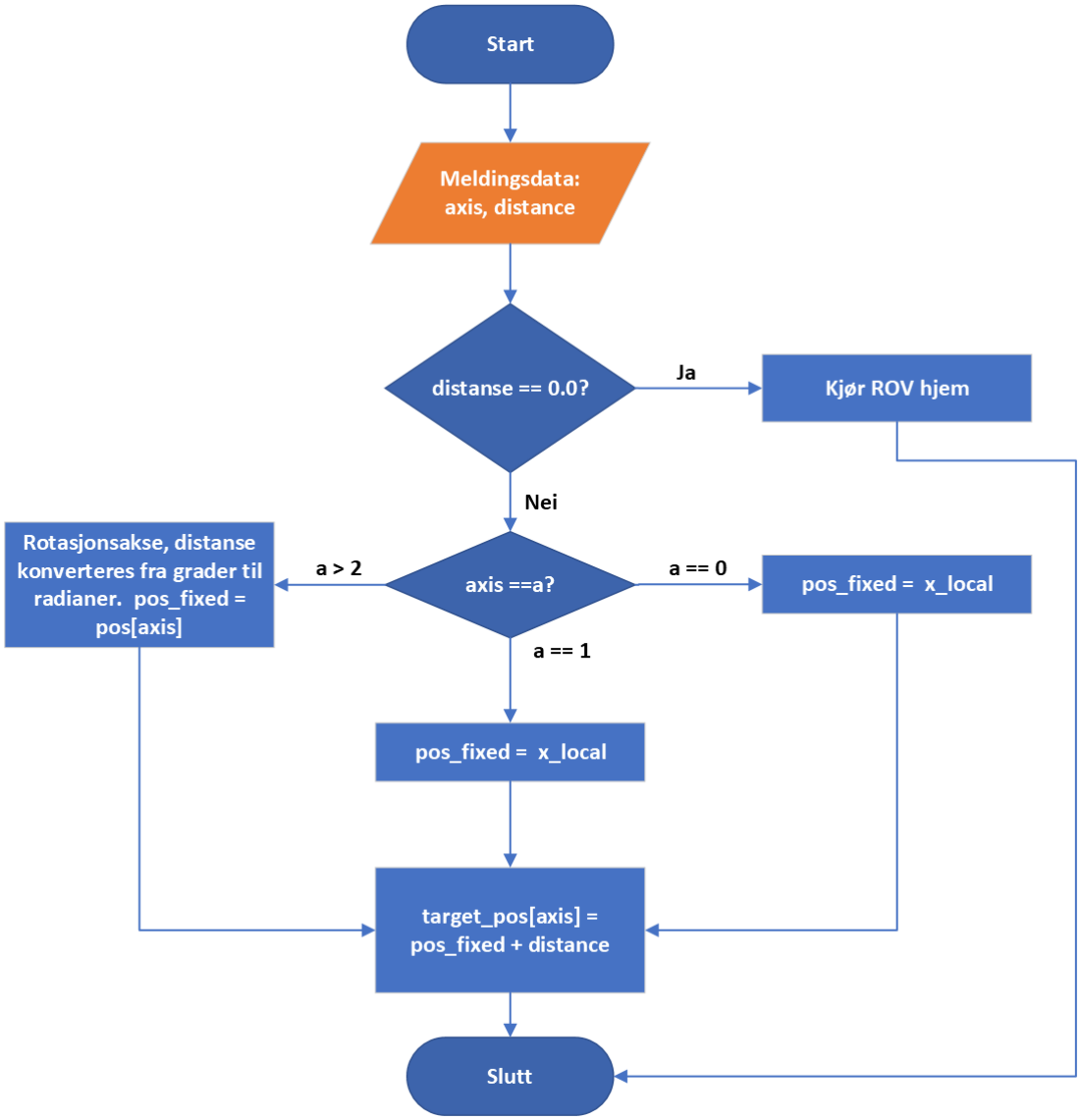
Figur G.1: GANT skjema for prosjektet



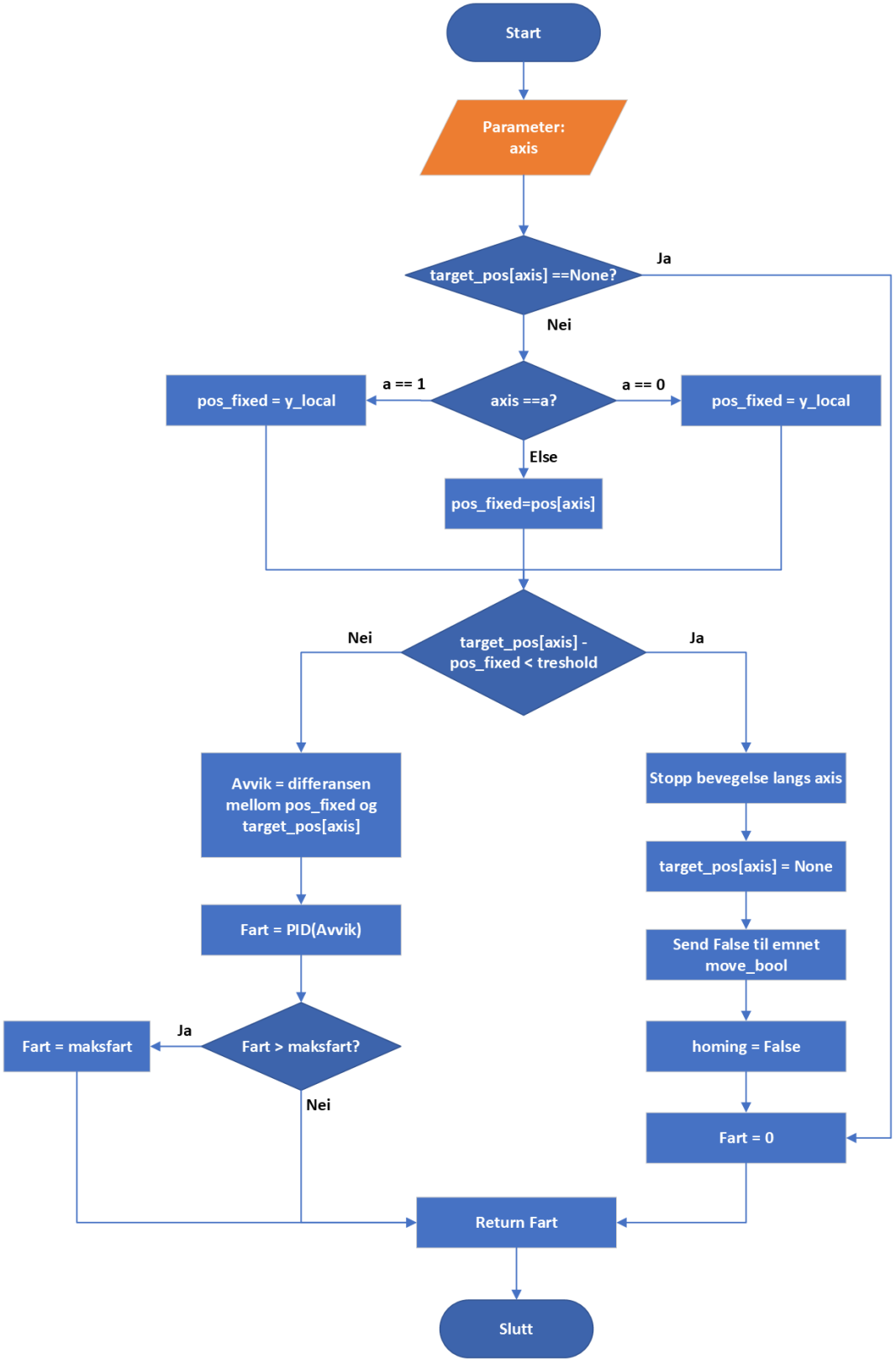
Figur G.2: Prossesflyt for find_pipe. Funksjonene i flytskjemaet forklares i tabell 4.4.



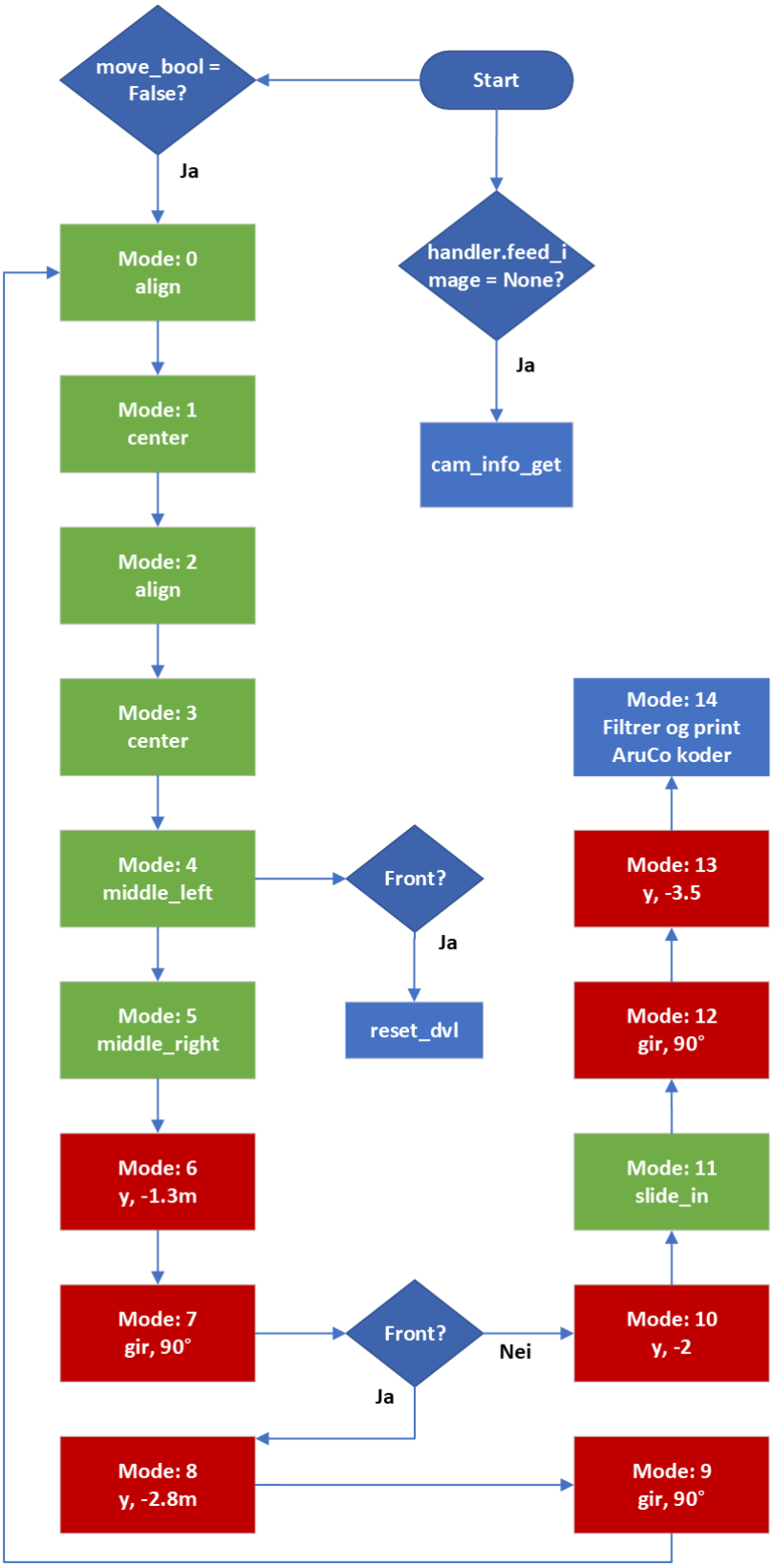
Figur G.3: Prossesflyt for find_bench. Blå representerer funksjoner/betingelser, oransje representerer data. Funksjonene i flytskjemaet forklares i tabell 4.4.



Figur G.4: Flytskjema for funksjonen move_pos_callback



Figur G.5: Flytskjema for funksjonen check_goal_pose

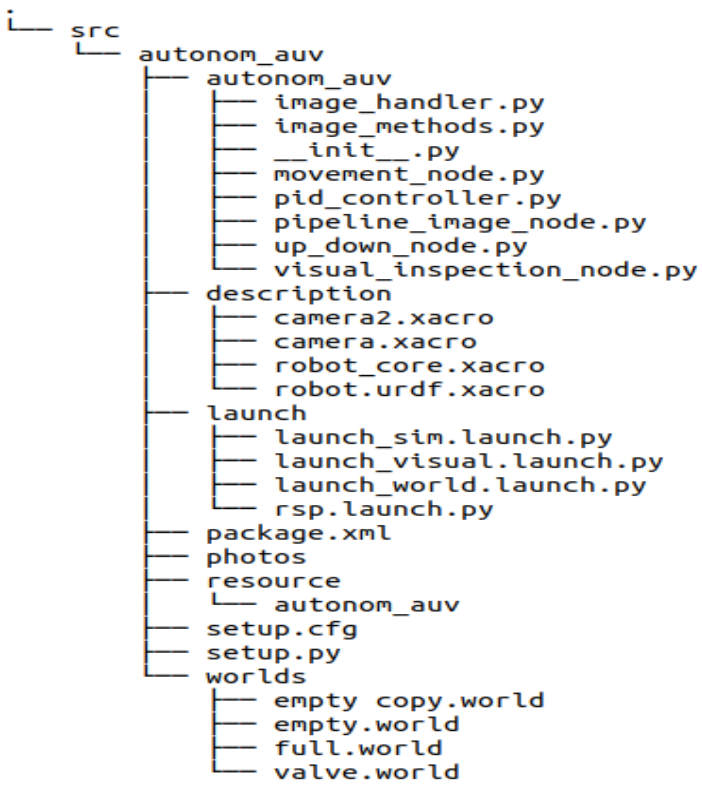


Figur G.6: Flytskjema for funksjonen bench_sequence, her er grønn boks kamera-regulering, rød boks er DVL-regulering

Vedlegg H

Filstruktur

I ett så stort prosjekt er det viktig å strukturere programvaren slik at en selv og andre kan forstå og videreutvikle prosjektet. I dette prosjektet brukes ROS som rammeverk og Python som språk. For prosjekter i ROS finnes egne regler og normer for prosjektstruktur. Dette prosjektet følger standarden for Python pakker i ROS. På figur H.1 vises mappestrukturen for prosjektet.



Figur H.1: Mappestruktur for prosjektet

- autonom_auv: Her plasseres kodefiler for noder og klasser
- description: Her plasseres filene for modellen til ROV-en
- launch: Her plasseres launch kodefiler
- worlds: Her plasseres .world filer
- photos: Her lagres bilder tatt under simulering
- setup.py og .cfg: Disse forteller *Colcon* hvor den finner nodene og mappene vist i bildet.
- package.xml: Her inluderes pakker som prosjektet trenger for å kjøre
- resource: Denne mappen med sin undermappe må eksistere for at pakken skal kunne bygges.

Ved å navigere til mappen over src og deretter bruke kommandoen **colcon build**, bygges en pakke basert på disse filene. *Colcon* er byggeverktøyet som brukes av ROS, dette lager tre nye filer ved siden av src: build, install og log. Kodefilene kan deretter kjøres via ros2 kommandoer i terminal.

H.0.1 Oppstartsfiler

For å starte simulatoren er det flere filer som kjøres, det lages derfor oppstartsfiler som kjører alle nødvendige filer samtidig.
I simuleringen brukes to forskjellige oppstartsfiler: launch_pipeline.launch.py til *oppdrag rørledning*, launch_bench.launch.py til *oppdrag arbeidsbenk*. Tabell H.1 viser hvilke noder som kjøres ved bruk av disse oppstartsfilene. Den eneste forskjellen mellom de to oppstartsfilene er oppdragsnoden, for *oppdrag rørledning* brukes m_pipeline_node, mens i *oppdrag arbeidsbenk* brukes m_bench_node.

Node	Funksjonalitet
rsp	Publiserer informasjon om ROV
gazebo	Kjører gazebo via ROS
spawn_entity	Genererer ROV-en i Gazebo miljøet
movement_node	Simulerer transfer functions for virkelighetsnær bevegelse
dvl_movement_node	Regulerer ROV-ens posisjon i forhold til seg selv
oppdragsnode	Gir passende oppførsel for valgt oppdrag.

Tabell H.1: Noder som kjøres ved bruk av launch kodefiler.

Vedlegg I

Filformater

For å simulere trengs det et miljø/terreng og modeller som kan plasseres i denne. I Gazebo Classic brukes XML formatet sammen med undergrupper av dette, for å produsere en verden som holder objekter.

I.1 XML - Extensible Markup Language

XML er ett "Markup Language", dette betyr at formatet er designet for enkel forståelse av både menneske og maskin. I XML filer brukes nøsting av data for å lage ett hierarki som er enkelt å forstå. XML alene gjør ingenting, dette er ikke en kjørbar fil(engelsk: Executable). XML må brukes sammen med programvare som tolker innholdet. Se [26] for mer om XML.

Figur I.1: XML-logo [27]

Kode I.1: Utdrag fra fil med XML format

```
25    <link name="base_footprint">
26     <visual>
27        <origin xyz="0 0 0" rpy="0 0 0" />
28        <geometry>
29          <box size="0.001 0.001 0.001" />
30        </geometry>
31      </visual>
32    </link>
```

I.2 URDF - Unified Robot Description Format

En URDF-fil benytter XML-syntaks for å beskrive en robot og dens tilhørende elementer. en URDF-fil er en 3D model av en robot som inneholder informasjon om ledd, motorer, masse og utseende. Ved bruk av Xacro(XML Makroer), kan man lage forskjellige deler av en robot i forskjellige xacro-filer, for så å kombinere disse i en URDF-fil. Når en URDF-fil brukes i Gazebo, vil Gazebo automatisk reformatere den til en SDF-fil. Se [24] for mer om URDF.

I.3 SDF - Simulation Description Format

Offisiell nettside for SDF: [21] En SDF fil benytter XML-format til å beskrive både roboter og miljøet rundt de. SDF filer ble originalt utviklet som en del av Gazebo og kan brukes for å lage en virkelighetsnær simulering av roboter og miljøet rundt de. Dette inkluderer:

- Fysikk
- Statiske og dynamiske objekter
- Lys
- Terreng

Figur I.2: XML-logo [27]

Vedlegg J

Kode

Kode J.1: Koden fra m_bench_node.py

```
1  import rclpy
2  from rclpy.node import Node
3  from sensor_msgs.msg import Image
4  from cv_bridge import CvBridge
5  import cv2
6  import numpy as np
7  import os
8  from std_msgs.msg import Bool, Float32
9  from geometry_msgs.msg import Twist
10 from nav_msgs.msg import Odometry
11 import time
12 from .image_handler import ImageHandler, logging_data
13 from .image_methods import ImageMethods
14 from .trackbar_hsv import DynamicDisplay
15 from .controller import PidController
16 import signal
17 from example_interfaces.srv import AddTwoInts
18 import math
19
20 class MBenchNode(Node):
21     def __init__(self):
22         super().__init__('mission_bench_node')
23         #Topic pub/sub + timer
24         self.create_subscription(Image,'/camera2/image_raw',  ...
               self.cam2_callback,10)
25         self.create_subscription(Image,'/camera/image_raw',  ...
               self.cam1_callback,10)
26         self.create_subscription(Bool, '/move_bool', self.bool_callback, 10)
27         self.create_timer(0.05, self.timer_callback)
28         self.publisher1 = self.create_publisher(Twist, '/tf_movement', 10)
29         self.publisher2 = self.create_publisher(Float32, '/up_down', 10)
30         self.publisher3 = self.create_publisher(Twist, '/target', 10)
31         #object declarations
32         self.bridge = CvBridge()
33         self.handler = ImageHandler()
34         self.x_controller = PidController()
```

```
35          self.y_controller = PidController()
36          self.y_controller2 = PidController()
37          self.y_controller3 = PidController()
38          self.yaw_controller = PidController()
39          self.logger = logging_data()
40          self.logger_slide = logging_data()
41          self.logger_align = logging_data()
42          self.logger_else = logging_data()
43          self.logger_dvl = logging_data()
44          #variable declarations
45          self.desired_distance = 0.35
46          self.mode = 0
47          self.move_bool = False
48          self.front = True
49          self.size = None
50          self.positions = None
51          self.angle = None
52          self.angle_list = []
53          self.dvl_zeroed = False
54          self.found_bench = False
55          pid_gir = [1.5, 0.3, 0.1]
56          pid_jag = [0.4, 0.3, 0.02]  #[0.1, 0.3, 0.01]
57          pid_svai = [0.4, 0.3, 0.02]
58          self.pid = [pid_jag,pid_svai,pid_svai,pid_gir,pid_gir,pid_gir]
59
60      def custom_cleanup(self):
61          """Cleans up certain error messages when closing node"""
62          self.logger.plot_data("Cam" ,["Jag","Svai", "Gir", "Mode"])
63          self.get_logger().info(f'I ran')
64
65      def zero_dvl(self):
66          """Pretend zeroing DVL"""
67          self.dvl_zeroed = True
68          self.get_logger().info("ZERO DVL")
69
70      def send_movement(self, x=0.0, y=0.0, z=0.0, roll=0.0, pitch=0.0, ...
            yaw=0.0):
71          """Sends movements which are then processed by the movement node"""
72          msg = Twist()
73          msg.linear.x = x
74          msg.linear.y = y
75          msg.linear.z = z
76          msg.angular.x = roll
77          msg.angular.y = pitch
78          msg.angular.z = yaw
79          self.publisher1.publish(msg)
80
81      def publish_z(self, value):
82          """Teleports on the z axis"""
83          msg = Float32()
84          msg.data = value
85          self.publisher2.publish(msg)
86
87      def bool_callback(self, msg):
88          #Changes movebool if DVL movement node is finished moving
89          self.move_bool = msg.data
90          self.get_logger().info(f"BOOL CALLBACK: {msg.data}")
91
92      #Fetch image feeds
```

```
93      def cam1_callback(self,data):
94          self.handler.feed_image2 = self.bridge.imgmsg_to_cv2(data, "bgr8")
95
96      def cam2_callback(self, data):
97          self.handler.feed_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
98
99      def timer_callback(self): #Runs the main sequence on a timer
100         self.run_image_strategy()
101
102     def cam_info_get(self):
103         """Retrieves positional data on the bench relative to the ROV"""
104         try:
105             self.size, self.positions, self.angle = ...
                    self.handler.find_bench(self.front, self.mode)
106             self.angle_list.append(self.angle)
107             self.found_bench = True
108         except Exception as e:
109             self.found_bench = False
110
111     def camera_regulator(self, key:str, accuracy = 1.0):
112         """Regulates postion relative to the bench using imageprocessing ...
                data.
113         Different key arguements lead to different methods of regulation ...
                and different offsets.
114         Increases mode by 1"""
115         if self.found_bench == True:
116             bench_width_pix = abs(self.positions['middle_left'][0] - ...
                    self.positions['middle_right'][0])
117             pix_per_m = bench_width_pix/2.485
118             m_per_pix = 2.485/bench_width_pix
119             if key == 'slide_in':
120                 y_offset = (self.handler.dims[1]/2 - ...
                        self.positions['center'][0] - 20)*m_per_pix
121                 self.logger.log_data(0,y_offset,0,self.mode)
122                 y_vel = self.y_controller.PID_controller(
123                     y_offset,*self.pid[1], max_out = 0.2, u_I_max=0.1)
124                 self.get_logger().info(f"mode:{self.mode} slide y_offset ...
                        = {y_offset}, y_vel = {y_vel}")
125                 self.send_movement(y=y_vel)
126                 if abs(y_offset) < 0.01:
127                     self.publish_z(1.6)
128                     self.mode += 1
129
130             elif key == 'align':
131                     yaw_vel = ...
                            self.yaw_controller.PID_controller(self.angle, ...
                            *self.pid[5], max_out = 0.1, u_I_max=0.01)
132                     self.logger.log_data(0,0,self.angle,self.mode)
133                     self.get_logger().info(f"mode:{self.mode} align angle ...
                            = {self.angle}, yaw_vel = {yaw_vel}")
134                     self.send_movement(yaw = yaw_vel)
135                     self.logger.log_data(0,0,yaw_vel,self.mode)
136                     if len(self.angle_list) > 10:
137                         filtered_angle = sum(self.angle_list[-10:]) / 10
138                         if abs(filtered_angle) < 0.01:
139                             self.mode+=1
140
141             elif key =='distance':
142                 distance_offset = self.size - self.desired_distance
```

```
143                 self.logger.log_data(distance_offset,0,0,self.mode)
144                 x_vel = self.x_controller.PID_controller(
145                     distance_offset,*self.pid[0], u_I_max=0.03)
146                 self.send_movement(x=x_vel)
147                 if abs(distance_offset) < 0.01/accuracy:
148                         self.mode += 1
149 
150             else:
151                 y_offset = (self.handler.dims[1]/2 - ...
                        self.positions[key][0])*m_per_pix
152                 self.logger.log_data(0,y_offset,0,self.mode)
153                 y_vel = self.y_controller.PID_controller(y_offset, ...
                        *self.pid[1], u_I_max=0.1)
154                 self.get_logger().info(f"mode:{self.mode} key = {key}, ...
                        y_offset = {round(y_offset,4)}, y_vel = ...
                        {round(y_vel,4)}")
155                 self.send_movement(y=y_vel)
156                 if abs(y_offset) < 0.05/accuracy:
157                     self.mode += 1
158 
159 
160 
161     def run_image_strategy(self):
162             """Contains and controls the sequence for the mission, mode ...
                    variable indicates where in the sequence the ROV is"""
163             if self.handler.feed_image is not None:
164                 self.cam_info_get()
165             if self.move_bool == False:
166                 if self.mode == 0:
167                     self.camera_regulator('align')
168                 elif self.mode == 1:
169                     self.camera_regulator('center')
170                 elif self.mode == 2:
171                     self.camera_regulator('distance', 5)
172                 elif self.mode == 3:
173                     self.camera_regulator('align')
174                 elif self.mode == 4:
175                     if self.front:self.zero_dvl()
176                     self.mode +=1
177                     #self.camera_regulator('middle_left') kan trenges ...
                            dersom vi leser nermere benken
178                 elif self.mode == 5:
179                     self.camera_regulator('middle_right')
180                 elif self.mode == 6:
181                     self.move_pos(1,-1.3)
182                     #Slide further to the right after finding the right ...
                            side of the bench
183                 elif self.mode == 7:
184                     self.move_pos(5,90)
185                     # turn 90 deg
186                     if not self.front:self.mode += 3
187                     #Check if were behind the bench, if so skip to mode 10.
188                 elif self.mode == 8:
189                     self.move_pos(1,-3.5)
190                     #Slide to the right to position behind the bench #-2.8
191                 elif self.mode == 9:
192                     self.move_pos(5,90)
193                     #rotate 90 deg
194                     self.front = False
```

```
195                         #Now the ROV is behind the bench
196                     elif self.mode ==10:
197                         self.move_pos(1, -1.5)
198                         #sideways slide to get closer to middle of the side ...
                                of the bench before trying to locate it via ...
                                imageprocessing.
199                         self.mode = 0
200                     elif self.mode == 11:
201                         self.camera_regulator('slide_in')
202                         #Find an exact position for sliding in between the ...
                                bench and teleport upwards
203                     elif self.mode == 12:
204                         self.move_pos(5, 90)
205                         # turn around 90 deg to face the bench top front.
206                     elif self.mode ==13:
207                         self.move_pos(1,-3.5)
208                         #Slide along the top of the bench to find the last codes.
209                     elif self.mode ==14:
210                         aruco_list = self.handler.filter_arucos()
211                         #filter codes
212                         self.get_logger().info(f"Aruco list: {aruco_list}")
213                         #print codes
214                 else:self.logger.log_data(0,0,0,self.mode)
215
216
217     def move_pos(self, axis, distance):
218         """Sends movement command to the dvl_movement_node and increase ...
                mode by 1"""
219         self.move_bool = True
220         msg = Twist()
221         msg.linear.x = float(axis)
222         msg.linear.y = float(distance)
223         self.publisher3.publish(msg)
224         self.mode += 1
225
226
227 def main(args=None):
228     rclpy.init(args=args)
229     node = MBenchNode()
230     signal.signal(signal.SIGINT, lambda sig, frame: node.custom_cleanup())
231     rclpy.spin(node)
232     node.custom_cleanup()
233     cv2.destroyAllWindows()
234     node.destroy_node()
235     rclpy.shutdown()
236
237
238 if __name__ == '__main__':
239     main()
```

Kode J.2: Koden fra m_pipeline_node.py

```
1 import rclpy
2 from rclpy.node import Node
3 from sensor_msgs.msg import Image
4 from cv_bridge import CvBridge
5 import cv2
```

```
6  from ament_index_python.packages import get_package_share_directory
7  from .image_methods import ImageMethods
8  from .controller import PidController
9  from .image_handler import ImageHandler
10 from .image_handler import logging_data
11 import signal
12 from geometry_msgs.msg import Twist
13 from nav_msgs.msg import Odometry
14 import time
15 import math
16
17 class PipelineImageNode(Node):
18     def __init__(self,mode):
19         super().__init__('mission_pipeline_node')
20         #Topic pub/sub and timer declaration
21         self.create_subscription(Image,'/camera/image_raw',  ...
               self.listener_callback,10)
22         self.create_subscription(Odometry, '/odom', self.odom_callback, 10)
23         self.create_subscription(Twist, '/cmd_vel', self.cmd_vel_callback ...
               ,10)
24         self.publisher = self.create_publisher(Twist, '/target', 10)
25         self.publisher1 = self.create_publisher(Twist, '/cmd_vel', 10) ...
               #/tf_movement
26         self.create_timer(0.05, self.timer_callback1)
27
28         #Variable declaration
29         self.bridge = CvBridge()
30         self.angular_controller = PidController()
31         self.y_controller = PidController()
32         self.handler = ImageHandler()
33         self.logger = logging_data()
34         self.logger_y = logging_data()
35         self.logger_dvl = logging_data()
36         self.mode=mode
37         self.time_start = time.time()
38         self.cv_image = None
39         self.state = 0
40         self.pid_gir =[1, 0.19, 0.19] #[0.56, 2.89, 0.18] [1, 0.19, 0.19]
41         self.pid_svai =[6,0.19, 0.14] #[0.55, 2.84, 0.13]
42         self.super_start = time.time()
43         self.colum1 = ["P","I","D","Acceleration","min area box"]
44         self.colum2 = [17,0.1,0,0.1,.4654,75000]
45         self.yaw = 0
46
47     def move_pos(self, axis, distance):
48         """Request movement through the DVL_movement_node"""
49         msg = Twist()
50         msg.linear.x = float(axis)
51         msg.linear.y = float(distance)
52         self.publisher.publish(msg)
53
54     def custom_cleanup(self):
55         """Cleans up certain error messages when closing node"""
56         self.logger.plot_data_table("gir", ...
               self.colum1,self.pid_gir,self.handler.filter_arucos(),self.plot_names)
57         self.logger_y.plot_data_table("svai" ...
               ,self.colum1,self.pid_svai,self.handler.filter_arucos(),self.plot_names_y)
58         self.logger_dvl.plot_data("xy" ,self.plot_names_dvl)
59         self.get_logger().info(f'I ran')
```

```
60
61      def send_movement(self,ang_vel=0.0,linear_y_vel=0.0):
62          """Request movement via the Movement_node"""
63          movement = Twist()
64          movement.linear.x = 0.6
65          movement.angular.z = ang_vel
66          movement.linear.y = linear_y_vel
67          self.testmeg = movement.angular.z
68          self.publisher1.publish(movement)
69
70      def odom_callback(self, msg):
71          """Fetch the odom of the ROV"""
72          self.odom_x = msg.pose.pose.position.x
73          self.odom_y = msg.pose.pose.position.y
74          self.odom_z = msg.pose.pose.position.z
75          self.odom_roll = msg.pose.pose.orientation.x
76          self.odom_yaw ,a,b= ImageMethods.quaternion_to_euler(
77                msg.pose.pose.orientation.z,0,0,msg.pose.pose.orientation.w)
78          self.angular_yaw = -msg.twist.twist.angular.z
79          self.velocity_y = msg.twist.twist.linear.y
80          self.velocity_x = msg.twist.twist.linear.x
81
82      def cmd_vel_callback(self, msg):
83          self.yaw = msg.angular.z
84
85
86      def listener_callback(self, data): #Get imagefeed.
87          self.handler.feed_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
88
89      def timer_callback1(self): #Timer callback to run the tasks for the ...
            mission.
90          if self.handler.feed_image is not None:
91              time_elapsed = time.time() - self.super_start
92              self.time_start = time.time()
93              angle, center_x, done = self.handler.find_pipeline(75000) ...
                    #Get info about the pipeline position'
94              distance_pipe = self.odom_z-0.2
95              center_x_meters = ImageMethods.pixles_to_meters(
96                    center_x,distance_pipe,self.handler.dims[1])
97
98              if self.state == 1:
99                  self.plot_names_dvl=["X cordinates","Y cordinates","Yaw",""]
100                 self.logger_dvl.log_data(
101                     self.odom_x,self.odom_y,self.odom_yaw)
102                 self.colum1 = ["P","I","D","Acceleration","min area box"]
103                 self.colum2 = [17,0.1,0,0.1,.4654,75000]
104                 return #if done stop camera based movement
105
106             if done and time_elapsed > 4:
107                 self.get_logger().info(f"Aruco ...
                        List:{self.handler.filter_arucos()}")
108                 self.state = 1
109                 self.move_pos(0.0,0.0) #tell the DVl movement node you ...
                        want to go home
110                 return
111
112             set_point = ImageMethods.pixles_to_meters(
113                 self.handler.dims[1]/2,distance_pipe,self.handler.dims[1])
114             offsett_x = ...
```

```
                    -PidController.calculate_offset((center_x_meters),set_point)
115             if self.mode ==1:
116                 angle_vel = self.angular_controller.PID_controller(
117                     angle,*self.pid_gir, scale_devide=1, u_I_max=0.10)  ...
                        #PID-regulate the ROV yaw
118                 linear_y_vel = self.y_controller.PID_controller(
119                     -offsett_x,*self.pid_svai, ...
                        scale_devide=1,u_I_max=0.02) #Do the same for ROV ...
                        Y posistion
120                 if linear_y_vel > 1:linear_y_vel=1.0
121                 if abs(angle)> math.pi/4: linear_y_vel = 0.0
122                 self.send_movement(angle_vel,linear_y_vel)   #send ...
                    regulated values
123             else:
124                 angle_vel= self.angular_controller.PID_controller
125                 (offsett_x,(7.8125),0.05,0.05,0.5,10000)  #PID-regulate ...
                    the ROV yaw
126                 self.send_movement(linear_y_vel) #send regulated value
127             # gir = yaw, svai = y, jag = x
128             #push logging data once per tick
129
130             self.plot_names=["","Angle offset in degrees","Ideal angular ...
                    velocity","Real angular velocity"]
131             self.logger.log_data(angle,-angle_vel, self.angular_yaw )
132
133             self.plot_names_y=["","Offsett meters","Ideal Velocity","Real ...
                    Velocity"]
134             self.logger_y.log_data(offsett_x,linear_y_vel,self.velocity_y)
135
136 def main(args=None):
137     rclpy.init(args=args)
138     image_processor = PipelineImageNode(1)
139     signal.signal(signal.SIGINT, lambda sig, frame: ...
            image_processor.custom_cleanup())
140     rclpy.spin(image_processor)
141     image_processor.custom_cleanup()  # Ensure cleanup is called if exit ...
            wasn't due to SIGINT
142     rclpy.shutdown()
143     cv2.destroyAllWindows()
144     image_processor.destroy_node()
145
146
147
148 if __name__ == '__main__':
149     main()
```

Kode J.3: Koden fra controller.py

```
1  import numpy as np
2  import time
3  import control as ctl
4  import numpy as np
5  from collections import deque
6  import asyncio
7  import math
8
9  class PidController:
```

```
10      def __init__(self,Pre_offset=None,Pre_time=None,Pre_I=None,Pre_D=None):
11             self.Pre_offset = Pre_offset
12             self.Pre_time = Pre_time
13             self.Pre_I = Pre_I
14             self.Pre_D = Pre_D
15
16      @staticmethod
17      def calculate_offset(measured_value,set_point):
18          """Calcualates offset"""
19          Offset = set_point-measured_value
20          return Offset
21
22
23      def PID_controller(self,e,P=0.0,I=0.0,D=0.0,T_f=0.5,scale_devide=1, ...
            margin=0, u_I_max=100, max_out = 1000000):
24           """Discrite PID controller"""
25           time_now = time.time()
26           P = P/scale_devide
27           I = I/scale_devide
28           D = D/scale_devide
29           margin = margin/scale_devide
30           if self.Pre_time is None:
31                Output = P*e
32                u_I = 0
33                u_D = 0
34                e_f=e
35           else:
36                T_s = time_now-self.Pre_time
37                u_P = P*e
38                u_I=self.Pre_I+I*T_s*(e+self.Pre_offset)/2
39                if abs(u_I)>u_I_max: u_I=u_I_max*np.sign(u_I)
40                e_f = (1/(1+(T_s/T_f)))*self.Pre_e_f ...
                      +((T_s/T_f)/(1+(T_s/T_f)))*e
41                u_D = D*(e-self.Pre_offset)/T_s
42                Output = u_P+u_I+u_D
43
44           #sets the newest values as the preious values
45           self.Pre_offset = e
46           self.Pre_time = time_now
47           self.Pre_e_f = e_f
48           self.Pre_I = u_I
49           self.Pre_D = u_D
50
51           if abs(Output) > max_out:
52                Output = max_out*np.sign(Output)
53           return Output
54
55  class transfer_funtion_class:
56      def __init__(self,numerator,denominator):
57          self.numerator = numerator
58          self.denominator = denominator
59          TF = ctl.TransferFunction(numerator,denominator)
60          ctl.c2d(TF,0.25,method="zoh")
61          self.ss = ctl.tf2ss(TF)
62          self.A = self.ss.A
63          self.x = np.zeros(self.A.shape[0],)
64          self.pre_time = time.time()
65
66
```

```
67      def implement_transfer_function(self,input):
68          "Implements transfer function"
69          time_now = time.time()
70          t_s = time_now-self.pre_time
71          ss = ctl.c2d(self.ss,t_s,"foh")
72          self.A =ss.A
73          self.B =ss.B
74          self.C =ss.C
75          self.D =ss.D
76          # State update equation: x(k+1) = Ax(k) + Bu(k)
77          self.x  = np.dot(self.A,self.x) + np.dot(self.B,input)
78          # output equation: y(k) = Cx(k) + Du(k)
79          output = np.dot(self.C,self.x) + self.D * input
80          self.pre_time=time_now
81          return output[0][0]
```

Kode J.4: Koden fra dvl_movement_node.py

```
1  import rclpy
2  from rclpy.node import Node
3  from sensor_msgs.msg import Image
4  from cv_bridge import CvBridge
5  import cv2
6  import numpy as np
7  import os
8  from std_msgs.msg import Bool, Float32
9  from geometry_msgs.msg import Twist
10 from nav_msgs.msg import Odometry
11 import time
12 from .image_handler import ImageHandler, logging_data
13 from .image_methods import ImageMethods
14 from .trackbar_hsv import DynamicDisplay
15 from .controller import PidController
16 import signal
17 from example_interfaces.srv import AddTwoInts
18 import math
19
20 class DvlMovementNode(Node):
21     def __init__(self):
22         super().__init__('dvl_movement_node')
23         #Topic pub/sub + timer declaration
24         self.create_subscription(Odometry, '/odom', self.odom_callback, 40)
25         self.create_subscription(Twist, '/target', ...
               self.move_pos_callback, 40)
26         self.publisher1 = self.create_publisher(Twist, '/tf_movement', 40)
27         self.publisher2 = self.create_publisher(Bool, '/move_bool', 40)
28         self.timer = self.create_timer(0.2, self.timer_callback)
29         #pid controller list declaration
30         self.blind_pid = [PidController() for _ in range(6)]
31         self.logger = logging_data()
32         #Node variable declarations
33         self.pos = [None,None,None,None,None,None]
34         self.target_pos = [None,None,None,None,None,None]
35         self.speed_scale = 1
36         self.treshold = [0.05,0.05,0.05,0.001,0.001,0.01]
37         self.zero_yaw = False
38         self.homing = False
```

```
39          self.top_speed = [0.3, 0.3, 0.3, 0.1, 0.1, 0.1]
40          pid_gir = [0.14, 0.0, 0.0]
41          pid_jag = [1,0.00, 0.0]
42          pid_svai = [1,0.0, 0.0]
43          pid_gir = [1.5, 0.3, 0.1]
44          pid_jag = [1,0.19, 0.19]
45          pid_svai = [6,0.19, 0.13]
46          self.pid = [pid_jag,pid_svai,pid_svai,pid_gir,pid_gir,pid_gir]
47          self.margin = [0.02,0.02,0.02, 0.001, 0.001, 0.001]
48          self.u_I_max = [0.1, 0.1, 0.1, 0.1, 0.1, 0.01]
49
50
51      def custom_cleanup(self):
52          """Cleans up certain error messages when closing node"""
53          self.logger.plot_data("DVL" ,["Gir","Svai","", ""])
54          self.get_logger().info(f'I ran')
55
56
57
58
59      def send_movement(self,x = 0.0, y = 0.0, yaw =0.0, axis = 6, ...
            magnitude = 0.0):
60          """Sends movements to the movement node"""
61          msg = Twist()
62          msg.linear.x = x
63          msg.linear.y = y
64          msg.angular.z = yaw
65
66          if axis == 0:msg.linear.x = magnitude
67          elif axis == 1:msg.linear.y = magnitude
68          elif axis == 2:msg.linear.z = magnitude
69          elif axis == 3:msg.angular.x = magnitude
70          elif axis == 4:msg.angular.y = magnitude
71          elif axis == 5:msg.angular.z = magnitude
72          self.publisher1.publish(msg)
73
74      def odom_callback(self, msg):
75          """Fetches odometry of the ROV"""
76          self.pos[0] = msg.pose.pose.position.x
77          self.pos[1] = msg.pose.pose.position.y
78          self.pos[2] = msg.pose.pose.position.z
79          quaternion = msg.pose.pose.orientation
80          self.pos[3], self.pos[4], self.pos[5] = ...
                ImageMethods.quaternion_to_euler(quaternion.x, quaternion.y, ...
                quaternion.z, quaternion.w)
81          if self.pos[5] < 0:self.pos[5] = 2*math.pi - abs(self.pos[5])
82
83      def send_false(self):
84          """Tells the m_bench_node that movement command has been executed"""
85          msg = Bool()
86          msg.data = False
87          self.publisher2.publish(msg)
88
89      def home1(self):
90          """Sets yaw x and y targets to 0 to return home"""
91          self.target_pos[5] = 0.0
92          self.target_pos[0] = 0.0
93          self.target_pos[1] = 0.0
94          self.get_logger().info(f"HOMING")
```

```
95
96      def reset_pi(self):
97          """Corrects for simulator trigonometry to allow ROV to fully ...
                rotate further than 360 degrees"""
98          if self.target_pos[5] is not None:
99              if self.target_pos[5] > 2*math.pi and self.pos[5] < 1:
100                 self.get_logger().info("RESET PI")
101                 self.target_pos[5] = self.target_pos[5] - 2*math.pi
102
103     def timer_callback(self):
104         """Calls appropriate functions on a timer to regulate the ROV ...
                postition towards its targets"""
105         self.reset_pi()
106         x, ox = self.check_goal_pose(0) #sjekk x
107         y, oy = self.check_goal_pose(1) #sjekk y
108         yaw, oyaw = self.check_goal_pose(5) #sjekk yaw
109         self.send_movement(x=x, y=y, yaw=yaw)
110         self.logger.log_data(oyaw,oy)
111
112     def xytrig(self):
113         """Remaps x and y coordinates using trigonometry"""
114         yaw, x, y = self.pos[5], self.pos[0], self.pos[1]
115         local_x = x * math.cos(yaw) + y * math.sin(yaw)
116         local_y = -x * math.sin(yaw) + y * math.cos(yaw)
117         return local_x, local_y
118
119     def move_pos_callback(self, msg):
120         """Sets targets when called"""
121         axis = int(msg.linear.x); distance = msg.linear.y
122         if distance == 0.0:
123             self.home1()
124             return
125         pos_fixed = self.pos[axis]
126         if axis > 2:
127             distance = math.radians(distance)
128         elif axis == 0: pos_fixed, _ = self.xytrig()
129         elif axis == 1: _, pos_fixed = self.xytrig()
130         self.target_pos[axis] = pos_fixed + distance
131         self.get_logger().info(f"new target = {self.target_pos[axis]}, ...
                pos = {pos_fixed}, distance = {distance}")
132
133     def check_goal_pose(self, axis):
134         """Checks targets vs position when called, call continously to ...
                regulate so that target = position."""
135         if self.target_pos[axis] is None:return 0.0, 0.0
136         if axis == 0:pos_fixed, _ = self.xytrig()
137         elif axis == 1:_, pos_fixed = self.xytrig()
138         else:pos_fixed = self.pos[axis]
139
140         if abs(self.target_pos[axis] - pos_fixed) < self.treshold[axis]: ...
                # Check if we are close to the target
141             self.send_movement(axis=axis, magnitude =0.0)
142             self.target_pos[axis] = None
143             self.get_logger().info(f"reached goal in axis {axis}")
144             self.send_false()
145             return 0.0, 0.0
146         else:
147             offset = round(self.target_pos[axis] - pos_fixed, 4)
148             vel = self.blind_pid[axis].PID_controller(
```

```
149                     offset,*self.pid[axis],self.u_I_max[axis]
150                      ,margin=self.margin[axis])
151                 if abs(vel) > self.top_speed[axis]:vel = ...
                        self.top_speed[axis]*np.sign(vel)
152                 self.get_logger().info(f"axis = {axis} goal = ...
                        {self.target_pos[axis]}, offset = {offset}, odom = ...
                        {round(pos_fixed, 4)}, vel = {round(vel,4)}, rot = ...
                        {self.pos[5]}")
153                 return vel, offset
154
155
156
157
158 def main(args=None):
159     rclpy.init(args=args)
160     node = DvlMovementNode()
161     signal.signal(signal.SIGINT, lambda sig, frame: node.custom_cleanup())
162     rclpy.spin(node)
163     node.custom_cleanup()
164     cv2.destroyAllWindows()
165     node.destroy_node()
166     rclpy.shutdown()
167
168 if __name__ == '__main__':
169     main()
```

Kode J.5: Koden fra trackbar_hsv.py

```
 1 import cv2
 2 import numpy as np
 3 from ament_index_python.packages import get_package_share_directory
 4 import os
 5 from .image_methods import ImageMethods
 6 def nothing(x):
 7     pass
 8
 9 class DynamicDisplay:
10
11     @staticmethod
12     def save_parameters(my_parameters):
13         """Saves current parameters in tex file in config folder under ...
               install"""
14         config_path = ...
               os.path.join(get_package_share_directory('autonom_auv'), ...
               'config', 'image_settings.tex')
15         my_parameters_str = str(my_parameters)
16         with open(config_path, 'a') as file:
17             file.write(my_parameters_str + '\n')
18
19     @staticmethod
20     def trackbar_init():
21         """Initialises trackbars"""
22         cv2.namedWindow("Trackbars")
23         cv2.createTrackbar("L - H", "Trackbars", 0, 179, nothing)
24         cv2.createTrackbar("L - S", "Trackbars", 0, 255, nothing)
25         cv2.createTrackbar("L - V", "Trackbars", 0, 255, nothing)
26         cv2.createTrackbar("U - H", "Trackbars", 179, 179, nothing)
```

```
27          cv2.createTrackbar("U - S", "Trackbars", 255, 255, nothing)
28          cv2.createTrackbar("U - V", "Trackbars", 255, 255, nothing)
29
30      @staticmethod
31      def find_hsv(image):
32          """use this to find HSV ranges live using trackbars"""
33          hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
34          l_h = cv2.getTrackbarPos("L - H", "Trackbars")
35          l_s = cv2.getTrackbarPos("L - S", "Trackbars")
36          l_v = cv2.getTrackbarPos("L - V", "Trackbars")
37          u_h = cv2.getTrackbarPos("U - H", "Trackbars")
38          u_s = cv2.getTrackbarPos("U - S", "Trackbars")
39          u_v = cv2.getTrackbarPos("U - V", "Trackbars")
40          lower_range = np.array([l_h, l_s, l_v])
41          upper_range = np.array([u_h, u_s, u_v])
42          mask = cv2.inRange(hsv, lower_range, upper_range)
43          res = cv2.bitwise_and(image, image, mask=mask)
44          # Converting the binary mask to 3 channel scaled_image
45          mask_3 = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)
46          stacked = np.hstack((mask_3, image, res))
47          cv2.imshow('Trackbars', cv2.resize(stacked, None, fx=0.3, fy=0.3))
48          key = cv2.waitKey(1)
49          if key == ord('s'):
50              thearray = [[l_h,l_s,l_v],[u_h, u_s, u_v]]
51              DynamicDisplay.save_parameters(thearray)
```

Kode J.6: Koden fra image_handler.py

```
1  try:
2      from .image_methods import ImageMethods  # Attempt relative import ...
           for package context
3  except ImportError:
4      from image_methods import ImageMethods  # Fallback to direct import ...
           for standalone execution
5
6  import cv2
7  import numpy as np
8  from ament_index_python.packages import get_package_share_directory
9  import time
10 import subprocess
11 from plotly.subplots import make_subplots
12 import plotly.graph_objects as go
13 import cv2.aruco as aruco
14
15
16 class ImageHandler:
17     def __init__(self):
18         #HSV range library
19         self.hsv_range_bib = {
20             "pipeline_sim" : [[30,114,114],[30,255,255]],
21             "visual_short_distance" : [[0, 0, 30],[86, 0, 120]],
22             "visual_long_distance" : [[0, 0, 0],[40, 200, 170]],
23             "pink_pipeline" : [[126, 45, 87],[179, 255, 255]]
24         }
25         #Object declarations
26         self.feed_image = None
27         self.feed_image2 = None
```

```
28          self.show_hsv = None
29          self.cooldown = 0
30          self.Id_list= []
31          self.aruco_printed = 0
32          self.bench_box_image = None
33          self.scale_factor = 0.5
34
35
36      def show_image(self, double):
37          """Shows 1 or 2 image feeds, takes a boolean value, false = 1 ...
                feed, true = 2 feeds"""
38          if not double:
39              ImageMethods.showImage(self.feed_image)
40          else:
41              showImage = ...
                    ImageMethods.stack_images([self.feed_image,self.feed_image2])
42              ImageMethods.showImage(showImage)
43
44      def find_bench(self, front, mode, test = False):
45          #Copy and scale current imagefeeds
46          original = ImageMethods.scale_image(self.feed_image.copy(), ...
                scale_factor=self.scale_factor)
47          image_edit = ImageMethods.scale_image(self.feed_image.copy(), ...
                scale_factor=self.scale_factor)
48          image_edit2 = ImageMethods.scale_image(self.feed_image2.copy(), ...
                scale_factor=self.scale_factor)
49          #Check for AruCo-codes
50          self.aruco_handler(image_edit, image_edit2)
51          #retrieve image dimentions
52          self.dims = image_edit.shape
53          #Apply HSV mask to make a binary image
54          hsv_range = self.hsv_range_bib["visual_long_distance"]
55          masked, hsv = ImageMethods.color_filter(image_edit , hsv_range, True)
56          fixed_hsv = ImageMethods.fix_hsv(masked)
57          #Try to find bench by finding the biggest box.
58          #Find the angle and size of the box to decide distance and yaw ...
                relative to bench
59          try:
60              boxes = ImageMethods.find_boxes(masked, image_edit, 500, False)
61              bench = ImageMethods.find_biggest_box(image_edit, boxes, True)
62              positions, area = ImageMethods.get_box_info(bench)
63              angle, angle_deg = ImageMethods.find_angle_box(bench,180, ...
                    self.dims[1])
64              #Back of bench is smaller than front, this makes for a ...
                    different distance measure in pixels.
65              if front:size = (self.scale_factor**2)*80000/area
66              else: size = (self.scale_factor**2)*160000/area
67          except Exception as e:_ = e #Display camerafeeds no matter if ...
                bench is found or not.
68          cv2.putText(image_edit, f"Mode:{mode}",[700,525], ...
                cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 255), 2, cv2.LINE_AA)
69          cv2.putText(image_edit, f"Angle:{round(angle_deg, 1)}",[550,50], ...
                cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 255), 2, cv2.LINE_AA)
70          if test:showImage = ...
                ImageMethods.stack_images([original,hsv,fixed_hsv,image_edit])
71          else:showImage = ImageMethods.stack_images([image_edit,image_edit2])
72          ImageMethods.showImage(showImage,0.7)
73          return size, positions, angle #return relevant positional data.
74
```

```
75
76      def find_pipeline(self,min_box_sice):
77          #Copy and scale current image feed
78          image_edit = ImageMethods.scale_image(self.feed_image.copy(), ...
                scale_factor=self.scale_factor)
79          self.dims = image_edit.shape
80          #Make binary picture using HSV mask
81          hsv_range = self.hsv_range_bib["pipeline_sim"]
82          hsv_img = cv2.cvtColor(image_edit, cv2.COLOR_BGR2HSV)
83          self.hsv_image = ImageMethods.color_filter(image_edit , hsv_range)
84          #Find AruCo-codes
85          self.aruco_handler(image_edit)
86          #Try to find pipe(s)
87          try:
88              #paint a line to split the pipe
89              cv2.line(self.hsv_image,(0,int(self.dims[0]/2))
90                   ,(self.dims[1],int(self.dims[0]/2)),(0,0,0),10)
91              box_list = ImageMethods.find_boxes(self.hsv_image, ...
                    image_edit, (self.scale_factor**2)*min_box_sice, True)
92              highest_box = ImageMethods.find_highest_box(box_list)
93              #Find the box highest in the picture as desired pipe to ...
                    trace, if no pipe then alert the mission node that ...
                    mission is done.
94              if highest_box is None:done = True
95              else:done=False
96              #find angle and y position of pipe relative to ROV
97              angle, angle_deg = ...
                    ImageMethods.find_angle_box(highest_box,90, self.dims[1])
98              angle, self.cooldown = ...
                    ImageMethods.angle_cooldown(angle,self.cooldown)
99              center_x,center_y = ...
                    ImageMethods.find_Center(image_edit,highest_box, True)
100             cv2.putText(image_edit, f"{int(angle_deg)}",[725,525], ...
                    cv2.FONT_HERSHEY_SIMPLEX, 4, (0, 0, 255), 2, cv2.LINE_AA)
101         except Exception as e: #Display imagefeed no matter if pipe is found.
102             angle, center_x = 0, 0
103         ImageMethods.showImage(image_edit, 1)
104         return angle,center_x, done #return relevant positional data.
105
106
107
108
109     def aruco_handler(self,image1, image2=None):
110         """Detects AruCo-codes up to two imagefeeds"""
111         self.Id_list= ImageMethods.read_AruCo(image1,self.Id_list)
112         if image2 is not None:
113             self.Id_list = ImageMethods.read_AruCo(image2,self.Id_list)
114
115     def filter_arucos(self):
116         """Deletes duplicate AruCo-codes """
117         filtered_list = []
118         filtered_list = ImageMethods.filtered_ids_list(self.Id_list)
119         return filtered_list
120
121
122
123
124 class logging_data:
125     def __init__(self):
```

```
126         self.time = []
127         self.data1 = []
128         self.data2 = []
129         self.data3 = []
130         self.data4 = []
131         self.markers1 = []
132         self.markers2 = []
133         self.markers3 = []
134         self.markers4 = []
135         self.start_time = time.time()
136 
137     def log_data(self,value1=None,value2=None,value3=None,
138                  value4=None,marker1=None,marker2=None
139                  ,marker3=None,marker4=None):
140         """Appends new data for up to 4 datasets"""
141         time_now = time.time()-self.start_time
142         time_now = round(time_now,3)
143         self.time.append(time_now)
144         value1 = round(value1,3)
145         self.markers1.append(f"mode: {marker1}")
146         self.data1.append(value1)
147         if value2 is not None:
148             value2 = round(value2,3)
149             self.data2.append(value2)
150             self.markers2.append(f"mode: {marker2}")
151         if value3 is not None:
152             value3 = round(value3,3)
153             self.data3.append(value3)
154             self.markers3.append(f"mode: {marker3}")
155         if value4 is not None:
156             value4= round(value4,3)
157             self.data4.append(value4)
158             self.markers4.append(f"mode: {marker4}")
159 
160 
161 
162 
163     def plot_data(self, name, plot_names=["","","",""]):
164         """Plots up to 4 datasets from existing data"""
165         fig = make_subplots(rows=2, cols=2,
166         subplot_titles=(plot_names[0],plot_names[1]
167                 ,plot_names[2],plot_names[3]))
168         fig.add_trace(go.Scatter(x=self.time, y=self.data1),row=1, col=1,)
169         if self.data2 is not None:
170             fig.add_trace(go.Scatter(x=self.time, y=self.data2),row=1, col=2)
171         if self.data3 is not None:
172             fig.add_trace(go.Scatter(x=self.time, y=self.data3),row=2, col=1)
173         if self.data4 is not None:
174             fig.add_trace(go.Scatter(x=self.time, y=self.data4),row=2, col=2)
175         fig.write_html(name + ".html" )
176         image_path = name + ".html"
177         subprocess.run(['xdg-open', image_path], check=True)
178 
179     def plot_data_table(self, name, colum1, ...
        colum2,colum3=[],plot_names=["","","",""]):
180         """Makes table and plot for several datasets"""
181         fig = make_subplots(
182             rows=2, cols=2,
183             specs=[[{"type": "table"}, {"type": "scatter"}],
```

```
                [{"type": "scatter"}, {"type": "scatter"}]],
        subplot_titles=(plot_names[0],plot_names[1]
                    ,plot_names[2],plot_names[3]))
        fig.add_trace(go.Table(header=dict(values=['Name', 'Value',"AruCo ...
            Codes"]),
                               cells=dict(values=[colum1, colum2,colum3])),
                               row=1, col=1)
        fig.add_trace(go.Scatter(x=self.time, y=self.data1),row=1, col=2)
        if self.data2 is not None:
            fig.add_trace(go.Scatter(x=self.time, y=self.data2),row=2, col=1)
        if self.data3 is not None:
            fig.add_trace(go.Scatter(x=self.time, y=self.data3),row=2, col=2)
        fig.write_html(name + ".html")
        image_path = name + ".html"
        subprocess.run(['xdg-open', image_path], check=True)

    def plot_data_markers(self, name, plot_names=["","","",""]):
        """Plots up to 4 datasets from existing data"""
        fig = make_subplots(rows=2, cols=2,
        subplot_titles=(plot_names[0],plot_names[1],
                        plot_names[2],plot_names[3]))
        fig.add_trace(go.Scatter(x=self.time, ...
            y=self.data1,text=self.markers1,),row=1, col=1,)
        if self.data2 is not None:
            fig.add_trace(go.Scatter(x=self.time, ...
                y=self.data2,text=self.markers2),row=1, col=2)
        if self.data3 is not None:
            fig.add_trace(go.Scatter(x=self.time, ...
                y=self.data3,text=self.markers3),row=2, col=1)
        if self.data4 is not None:
            fig.add_trace(go.Scatter(x=self.time, ...
                y=self.data4,text=self.markers4),row=2, col=2)
        fig.write_html(name + ".html")
        image_path = name + ".html"
        subprocess.run(['xdg-open', image_path], check=True)
```

Kode J.7: Koden fra image_methods.py

```
import math
import cv2
import cv2.aruco as aruco
import numpy as np
from ament_index_python.packages import get_package_share_directory
import os
import time
class ImageMethods:

    @staticmethod
    def scale_image(image, scale_factor = 0.5):
        """Scales image"""
        new_width = int(image.shape[1] * scale_factor)
        new_height = int(image.shape[0] * scale_factor)
        resized_image = cv2.resize(image, (new_width, new_height))
        return resized_image

    @staticmethod
    def fix_hsv(image):
```

```
20          """Gives HSV image a third dimention so it can be merged with ...
                color pictures for dual display"""
21          fixed_hsv = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
22          return fixed_hsv
23
24      @staticmethod
25      def saveImage(image):
26          "saves image"
27          photos_path = ...
                os.path.join(get_package_share_directory('autonom_auv'), ...
                'photos', f"cam2_{time.time()}.jpg")
28          cv2.imwrite(photos_path, image)
29
30      @staticmethod
31      def showImage(image,scale=1):
32          """Saves displayed image by pressing 's' """
33          image = cv2.resize(image, (0, 0),fx=scale,fy=scale)
34          cv2.imshow("window", image)
35          key = cv2.waitKey(1)
36          if key == ord('s'):
37              ImageMethods.saveImage(image)
38
39      @staticmethod
40      def close_image(image, rate):
41          """Fills holes in image"""
42          kernel = np.ones((rate, rate), np.uint8)  # Kernel size affects ...
                the amount of merging
43          closed_image = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)
44          return closed_image
45
46      @staticmethod
47      def stack_images(images,):
48          """Takes a list of images and stacks them"""
49          images = [img for img in images if img is not None]
50          rows = (len(images) + 1) // 2
51          cols = min(2, len(images))
52          row_images = []
53          for i in range(0, len(images), 2):
54              imgs_to_stack = images[i:i+2]
55              if len(imgs_to_stack) == 1:
56                  imgs_to_stack.append(np.zeros_like(images[0]))
57              row_images.append(np.hstack(imgs_to_stack))
58          image_show = np.vstack(row_images) if len(row_images) > 1 else ...
                row_images[0]
59          return image_show
60
61
62      @staticmethod
63      def color_filter(image, range:list, debug = False):
64          "Takes in an image and a HSV range, return a black and white image"
65          image_HSV = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
66          HSV_lower = np.array(range[0])
67          HSV_upper = np.array(range[1])
68          mask = cv2.inRange(image_HSV,HSV_lower,HSV_upper)
69          maskM = cv2.medianBlur(mask, 5)
70          if debug:return maskM, image_HSV
71          else:return maskM
72
73
```

```
74
75      @staticmethod
76      def find_boxes(hsv_image, Original_image, min_box_size, draw:bool):
77          """
78          Takes in a black and white image and the original image, returns ...
                the original image with drawn boxes,
79          an image with box approximation, and a list of boxes that have an ...
                area greater than min_box_size.
80          """
81          box_list = []
82          contours, _ = cv2.findContours(hsv_image, cv2.RETR_EXTERNAL, ...
                cv2.CHAIN_APPROX_SIMPLE)
83          if contours is not None:
84              for cnt in contours:
85                  rect = cv2.minAreaRect(cnt)
86                  box = np.intp(cv2.boxPoints(rect))
87                  area = cv2.contourArea(box)
88                  if area > min_box_size:
89                      box_list.append(box)
90                      if draw:
91                          cv2.drawContours(Original_image, [box], -1, (0, ...
                                0, 255), thickness=2)
92          return box_list
93
94
95      @staticmethod
96      def find_biggest_box(image, boxes:list, draw:bool):
97          """Finds biggest box in a list of boxes"""
98          max_area = 0
99          biggest_box = None
100         for box in boxes:
101             area = cv2.contourArea(box)
102             if area > max_area:
103                 biggest_box = box
104                 max_area = area
105
106         if biggest_box is not None and draw:
107             cv2.drawContours(image, [biggest_box], -1, (0, 255, 0), ...
                    thickness=3)
108         return biggest_box
109
110
111
112     @staticmethod
113     def find_highest_box(boxes):
114         "Takes in a list of boxes return the boxes highest in the picture"
115         min_y_value=1920
116         if len(boxes)>0:
117             for i in range(len(boxes)):
118                 y_value_list = boxes[i][:,1]
119                 if min(y_value_list)≤min_y_value:
120                     min_y_value = min(y_value_list)
121                     box_index = i
122             box=boxes[box_index]
123             return box
124
125         else: return None
126
127     @staticmethod
```

```
128     def get_box_info(box):
129         """
130         Takes a box as an argument and returns the positions of each of its four corners and the middle points of each of its four lines.
131         Parameters:
132         - box: A NumPy array of shape (4, 2) representing the box's four corners.
133         Returns:
134         - A dictionary with keys 'top_left', 'top_right', 'bottom_right', 'bottom_left'
135         corresponding to the coordinates of each corner and 'middle_top', 'middle_right', 'middle_bottom', 'middle_left' for the middle points of each line.
136         """
137         area = cv2.contourArea(box)
138         # Sort the box points based on their x-coordinates (helps in identifying left/right)
139         sorted_box = sorted(box, key=lambda x: x[0])
140         # Split the sorted points into leftmost and rightmost
141         left_points = sorted_box[:2]
142         right_points = sorted_box[2:]
143         # Sort the left_points and right_points by their y-coordinates to separate top/bottom
144         top_left, bottom_left = sorted(left_points, key=lambda x: x[1])
145         top_right, bottom_right = sorted(right_points, key=lambda x: x[1])
146 
147         # Calculate the middle points of each side
148         middle_top = ((top_left[0] + top_right[0]) / 2, (top_left[1] + top_right[1]) / 2)
149         middle_right = ((top_right[0] + bottom_right[0]) / 2, (top_right[1] + bottom_right[1]) / 2)
150         middle_bottom = ((bottom_left[0] + bottom_right[0]) / 2, (bottom_left[1] + bottom_right[1]) / 2)
151         middle_left = ((top_left[0] + bottom_left[0]) / 2, (top_left[1] + bottom_left[1]) / 2)
152 
153         # Calculate the center of the box
154         center_x = (top_left[0] + top_right[0] + bottom_left[0] + bottom_right[0]) / 4
155         center_y = (top_left[1] + top_right[1] + bottom_left[1] + bottom_right[1]) / 4
156         center = (center_x, center_y)
157 
158         # Return the corners and middle points in a structured dictionary
159         positions = {
160             'top_left': tuple(top_left),
161             'top_right': tuple(top_right),
162             'bottom_right': tuple(bottom_right),
163             'bottom_left': tuple(bottom_left),
164             'middle_top': middle_top,
165             'middle_right': middle_right,
166             'middle_bottom': middle_bottom,
167             'middle_left': middle_left,
168             'center' : center
169         }
170         return positions, area
171 
172 
173 
```

```
174
175
176     @staticmethod
177     def find_Center(image, box, draw:bool):
178         "Draw a dot in the center of a box, return Center cordinates"
179         if box is not None:
180             center=((box[0]+box[2])/2)
181             Center_X=int(center[0])
182             Center_Y=int(center[1])
183             cv2.circle(image,(Center_X,Center_Y),10,(0,0,255),-1)
184             return Center_X,Center_Y
185         else: return 960,600
186
187
188     @staticmethod
189     def find_angle_box(box,offset=0, width= 960):
190         if box is not None:
191             "Finds the angle of the object between the horizontal frame ...
                    and the longest vector"
192             Vec_1=box[0]-box[1]
193             Vec_2=box[1]-box[2]
194             lenght_Vec_1=math.sqrt(Vec_1[0]**2+Vec_1[1]**2)
195             lenght_Vec_2=math.sqrt(Vec_2[0]**2+Vec_2[1]**2)
196
197             if lenght_Vec_1>lenght_Vec_2:
198                 diff = (box[0][0]+box[1][0]-width)
199                 Vector = Vec_1
200                 lenght_Vec = lenght_Vec_1
201             else:
202                 diff=(box[1][0]+box[2][0]-width)
203                 Vector = Vec_2
204                 lenght_Vec = lenght_Vec_2
205
206             direction=np.sign(Vector[1])*(-1)
207             if direction==0:
208                 if diff>0:
209                     direction=-1
210                 else: direction=1
211             angle=math.acos(Vector[0]/lenght_Vec)
212             angle_deg=(angle*360/2/math.pi-offset)*direction
213             angle = math.radians(angle_deg)
214             return angle, angle_deg
215         else: return 0
216
217     @staticmethod
218     def angle_cooldown(angle,Cooldown):
219         """makes sure angle calculating functions dont confuse themselves ...
                about 90 and -90 degrees"""
220         if Cooldown == 0:
221             if angle==-math.pi/2:
222                 Cooldown = -40
223             elif angle == math.pi/2:
224                 Cooldown = 40
225
226         if angle==-math.pi/2 or angle ==math.pi/2:
227             if Cooldown > 0:
228                 angle=abs(angle)
229                 Cooldown -= 1
230             if Cooldown < 0:
```

```
231                 angle=-abs(angle)
232                 Cooldown += 1
233         else:
234             if Cooldown > 0:
235                 Cooldown -= 1
236             elif Cooldown < 0:
237                 Cooldown += 1
238             else:
239                 Cooldown = 0
240         return angle, Cooldown
241 
242     @staticmethod
243     def read_AruCo(image,id_list):
244         "Takes the image and reads the aruco code and adds the Id to the ...
                 given list"
245         gray= cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
246         dict= aruco.getPredefinedDictionary(aruco.DICT_5X5_100)
247         corners, ids, rejected = aruco.detectMarkers(gray, dict)
248 
249         if ids is not None and len(ids) > 0:
250             aruco.drawDetectedMarkers(image,corners,ids)
251             id_list.append(ids[0][0])
252         return id_list
253 
254 
255 
256     @staticmethod
257     def filtered_ids_list(list):
258         "Filter the list and gives back a list without duplicates"
259         filtered_list=[]
260         for i in range(len(list)):
261             if filtered_list.count(list[i]) < 1:
262                     filtered_list.append(list[i])
263         return filtered_list
264 
265 
266     @staticmethod
267     def quaternion_to_euler(x, y, z, w):
268           """
269           Convert a quaternion into euler angles (roll, pitch, yaw)
270           roll is rotation around x in radians (counterclockwise)
271           pitch is rotation around y in radians (counterclockwise)
272           yaw is rotation around z in radians (counterclockwise)
273           """
274           t0 = +2.0 * (w * x + y * z)
275           t1 = +1.0 - 2.0 * (x * x + y * y)
276           roll_x = math.atan2(t0, t1)
277           t2 = +2.0 * (w * y - z * x)
278           t2 = +1.0 if t2 > +1.0 else t2
279           t2 = -1.0 if t2 < -1.0 else t2
280           pitch_y = math.asin(t2)
281           t3 = +2.0 * (w * z + x * y)
282           t4 = +1.0 - 2.0 * (y * y + z * z)
283           yaw_z = math.atan2(t3, t4)
284           return roll_x, pitch_y, yaw_z
285 
286 
287     @staticmethod
288     def pixles_to_meters(pixles, distance_from_object, dimension):
```

```
289         meters = pixles * distance_from_object / dimension
290         return meters
291
292     @staticmethod
293     def pixles_to_meters69(pixles):
294         meters = pixles * 0.01355
295         return meters
```

Kode J.8: Koden fra movement_node.py

```
1  import rclpy
2  from rclpy.node import Node
3  from geometry_msgs.msg import Twist
4  from std_msgs.msg import Float32
5  import time
6  import numpy as np
7  from .controller import PidController
8  from .controller import transfer_funtion_class
9  from nav_msgs.msg import Odometry
10
11 class MovementNode(Node):
12     def __init__(self):
13         super().__init__('movement_node')
14         #Topic pub/subs
15         self.publisher = self.create_publisher(Twist, '/cmd_vel', 40)
16         self.create_subscription(Twist,'/tf_movement', ...
               self.movement_callback,40)
17
18         #Object declarations
19         self.yaw_controller = PidController()
20         self.yaw_tf =  transfer_funtion_class([3.74, 61.48, 2546], [1, ...
               38.65, 519.6, 2533])
21         self.x_tf = transfer_funtion_class([3.606,40.12,1421],[1, 16.98, ...
               273.3, 1411])
22         self.y_tf = transfer_funtion_class([4.014, 83.39, 2173],[1, ...
               31.09, 357.9, 2170])
23
24         #Variable declarations
25         self.x = 0.0
26         self.y = 0.0
27         self.z = 0.0
28         self.roll = 0.0
29         self.pitch = 0.0
30         self.yaw = 0.0
31         self.counter = 0
32
33
34     def publish_movement(self):
35         """Publishes TF corrected movement"""
36         cmd_vel = Twist()
37         self.yaw = self.yaw_tf.implement_transfer_function(self.yaw)
38         self.x = self.x_tf.implement_transfer_function(self.x)
39         self.y = self.y_tf.implement_transfer_function(self.y)
40         cmd_vel.linear.x = self.x
41         cmd_vel.linear.y = self.y
42         cmd_vel.linear.z = self.z
43         cmd_vel.angular.x = self.roll
```

```
44          cmd_vel.angular.y = self.pitch
45          cmd_vel.angular.z = self.yaw
46          self.publisher.publish(cmd_vel)
47
48
49      def movement_callback(self,msg):
50          """Reads external movement commands"""
51          self.x = msg.linear.x
52          self.y = msg.linear.y
53          self.z = msg.linear.z
54          self.roll = msg.angular.x
55          self.pitch = msg.angular.y
56          self.yaw = msg.angular.z
57          self.publish_movement()
58
59
60
61  def main(args=None):
62      rclpy.init(args=args)
63      movement = MovementNode()
64      rclpy.spin(movement)
65
66      # Destroy the node explicitly
67      movement.destroy_node()
68      rclpy.shutdown()
69
70
71
72  if __name__ == '__main__':
73      main()
```

Kode J.9: Koden fra up_down_node.py

```
1  import rclpy
2  from rclpy.node import Node
3  from nav_msgs.msg import Odometry
4  from gazebo_msgs.srv import SetEntityState
5  from std_msgs.msg import Float32
6
7  class UpDownNode(Node):
8      def __init__(self):
9          super().__init__('up_down_node')
10         #variable declaration
11         self.last_z = None
12         self.current_x = None
13         self.current_y = None
14         self.current_orientation = None
15         #Topics
16         self.create_subscription(Odometry, '/odom', self.odom_callback, 10)
17         self.create_subscription(Float32, '/up_down', ...
               self.up_down_callback, 10)
18
19         #Services
20         self.set_state_client = self.create_client(SetEntityState, ...
               '/demo/set_entity_state')
21         while not self.set_state_client.wait_for_service(timeout_sec=1.0):
22             self.get_logger().info('Waiting for /demo/set_entity_state ...
```

```
                service...')
23
24
25      def odom_callback(self, msg): #Fetches odometry
26          self.current_x = msg.pose.pose.position.x
27          self.current_y = msg.pose.pose.position.y
28          self.current_orientation = msg.pose.pose.orientation
29
30      def up_down_callback(self, msg): #Checks if movement command has been ...
            recieved in z axis and teleports bot accordingly
31          if self.last_z != msg.data:
32              self.last_z = msg.data
33              self.update_my_bot_position()
34
35      def update_my_bot_position(self):
36          """Copies the position of the ROV but uses a given Z value to ...
                teleport the bot"""
37          if None in (self.current_x, self.current_y, ...
                self.current_orientation):
38              self.get_logger().error('Current position or orientation not ...
                    yet received.')
39              return
40          set_req = SetEntityState.Request()
41          set_req.state.name = 'my_bot'
42          set_req.state.pose.position.x = self.current_x
43          set_req.state.pose.position.y = self.current_y
44          set_req.state.pose.position.z = self.last_z
45          set_req.state.pose.orientation = self.current_orientation
46          future = self.set_state_client.call_async(set_req)
47          future.add_done_callback(self.set_state_callback)
48
49      def set_state_callback(self, future):
50          """Tries to update the ROV posisiton"""
51          try:
52              response = future.result()
53              if response.success:
54                  self.get_logger().info('Successfully updated position of ...
                        my_bot.')
55              else:
56                  self.get_logger().error('Failed to update position of ...
                        my_bot.')
57          except Exception as e:
58              self.get_logger().error(f'Exception while updating position: ...
                    {e}')
59
60
61
62
63  def main(args=None):
64      rclpy.init(args=args)
65      node = UpDownNode()
66      rclpy.spin(node)
67      node.destroy_node()
68      rclpy.shutdown()
69
70  if __name__ == '__main__':
71      main()
```

Kode J.10: Koden fra launch_bench.launch.py

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch_ros.actions import Node
from launch.substitutions import LaunchConfiguration
from launch.actions import IncludeLaunchDescription, SetEnvironmentVariable


def generate_launch_description():

    package_name='autonom_auv'
    #Configure RSP
    rsp = IncludeLaunchDescription(
                PythonLaunchDescriptionSource([os.path.join(
                    get_package_share_directory(package_name),
                    'launch','rsp.launch.py'
                    )]), launch_arguments={'use_sim_time': 'true',
                    'log_level': 'WARN'}.items()
    )
    #Pick world
    world_file_path = ...
        os.path.join(get_package_share_directory('autonom_auv'), ...
        'worlds', 'bench.world')

    #Configure Gazebo
    gazebo = IncludeLaunchDescription(
                PythonLaunchDescriptionSource([os.path.join(
                    get_package_share_directory('gazebo_ros'), 'launch', ...
                        'gazebo.launch.py')]),
                    launch_arguments={'world': world_file_path, ...
                        'log_level': 'WARN'}.items()
             )

    #spawn ROV
    spawn_entity = Node(package='gazebo_ros', executable='spawn_entity.py',
                        arguments=['-topic', 'robot_description', ...
                            '-entity', 'my_bot', '-z', '1', '-Y', '0.4', ...
                            '--ros-args', '--log-level', 'WARN'], ...
                            output='screen')


    #Configure/include other nodes
    up_down_node = Node(
        package = package_name,
        executable= 'up_down_node',
        output='screen'
    )

    m_bench_node = Node(
        package = package_name,
        executable= 'm_bench_node',
        output='screen'
    )

    dvl_movement_node = Node(
```

```
51          package = package_name,
52          executable= 'dvl_movement_node',
53          output='screen'
54      )
55
56      movement_node = Node(
57          package = package_name,
58          executable= 'movement_node',
59          output='screen'
60      )
61
62  #Launch nodes
63      return LaunchDescription([
64          rsp,
65          gazebo,
66          spawn_entity,
67          movement_node,
68          up_down_node,
69          dvl_movement_node,
70          m_bench_node
71      ])
```

Kode J.11: Koden fra launch_pipeline.launch.py

```
1  import os
2  from ament_index_python.packages import get_package_share_directory
3  from launch import LaunchDescription
4  from launch.actions import IncludeLaunchDescription
5  from launch.launch_description_sources import PythonLaunchDescriptionSource
6  from launch_ros.actions import Node
7  from launch.substitutions import LaunchConfiguration
8  from launch.actions import IncludeLaunchDescription, SetEnvironmentVariable
9
10
11 def generate_launch_description():
12
13     package_name='autonom_auv'
14     #Configure RSP
15     rsp = IncludeLaunchDescription(
16                 PythonLaunchDescriptionSource([os.path.join(
17                     get_package_share_directory(package_name),
18                     'launch','rsp.launch.py'
19                     )]), launch_arguments={'use_sim_time':
20                    'true'}.items()
21     )
22     #pick world
23     world_file_path = ...
           os.path.join(get_package_share_directory('autonom_auv'), ...
           'worlds', 'pipeline.world')
24     #configure Gazebo
25     gazebo = IncludeLaunchDescription(
26                 PythonLaunchDescriptionSource([os.path.join(
27                     get_package_share_directory('gazebo_ros'), 'launch', ...
                        'gazebo.launch.py')]),
28                     launch_arguments={'world': world_file_path}.items()
29             )
30
```

```
31      #spawn ROV
32      spawn_entity = Node(package='gazebo_ros', executable='spawn_entity.py',
33                          arguments=['-topic', 'robot_description',
34                                     '-entity', 'my_bot','-x','0', '-z', ...
                                           '0.8',"-y","0"],
35                          output='screen')
36
37
38      #configure/include other nodes
39      fake_controller_node = Node(
40          package = package_name,
41          executable= 'fake_controller_node',
42          output='screen'
43      )
44
45      up_down_node = Node(
46          package = package_name,
47          executable= 'up_down_node',
48          output='screen'
49      )
50
51    #  valve_image_node = Node(
52     #      package = package_name,
53      #     executable= 'valve_image_node',
54       #    parameters=[{'save_images': True}],
55        #   output='screen'
56      #)
57
58      m_pipeline_node = Node(
59          package = package_name,
60          executable= 'm_pipeline_node',
61          output='screen'
62      )
63
64
65      movement_node = Node(
66          package = package_name,
67          executable= 'movement_node',
68          output='screen'
69      )
70
71      dvl_movement_node = Node(
72          package = package_name,
73          executable= 'dvl_movement_node',
74          output='screen'
75      )
76
77
78       #launch nodes
79      return LaunchDescription([
80          rsp,
81          gazebo,
82          spawn_entity,
83          m_pipeline_node,
84          movement_node,
85          dvl_movement_node,
86      ])
```

Kode J.12: Koden fra launch_world.launch.py

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch_ros.actions import Node
from launch.substitutions import LaunchConfiguration
from launch.actions import IncludeLaunchDescription, SetEnvironmentVariable

def generate_launch_description():

    package_name='autonom_auv'

    world_file_path = ...
        os.path.join(get_package_share_directory('autonom_auv'), ...
        'worlds', 'bench.world')

    gazebo = IncludeLaunchDescription(
                PythonLaunchDescriptionSource([os.path.join(
                    get_package_share_directory('gazebo_ros'), 'launch', ...
                        'gazebo.launch.py')]),
                    launch_arguments={'world': world_file_path}.items()
             )

    return LaunchDescription([
        gazebo
    ])
```

Kode J.13: Koden fra rsp.launch.py

```
import os

from ament_index_python.packages import get_package_share_directory

from launch import LaunchDescription
from launch.substitutions import LaunchConfiguration, Command
from launch.actions import DeclareLaunchArgument
from launch_ros.actions import Node

import xacro


def generate_launch_description():

    # Check if we're told to use sim time
    use_sim_time = LaunchConfiguration('use_sim_time')

    # Process the URDF file
    pkg_path = os.path.join(get_package_share_directory('autonom_auv'))
    xacro_file = os.path.join(pkg_path,'description','robot.urdf.xacro')
    # robot_description_config = xacro.process_file(xacro_file).toxml()
    robot_description_config = Command(['xacro ', xacro_file, ' ...
        sim_mode:=', use_sim_time])

    # Create a robot_state_publisher node
    params = {'robot_description': robot_description_config, ...
```

```
            'use_sim_time': use_sim_time}
    node_robot_state_publisher = Node(
        package='robot_state_publisher',
        executable='robot_state_publisher',
        output='screen',
        parameters=[params]
    )


    # Launch!
    return LaunchDescription([
        DeclareLaunchArgument(
            'use_sim_time',
            default_value='false',
            description='Use sim time if true'),
        node_robot_state_publisher
    ])
```

Kode J.14: Koden fra Camera_node.py

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
import numpy as np
import signal


class USB_Camera(Node):
    def __init__(self):
         super().__init__('usb_camera_node')
         self.publisher = self.create_publisher(Image, 'usb_camera', 10)
         self.timer = self.create_timer(0.1, self.timer_callback)
         self.cap = cv2.VideoCapture(0)
         self.cv_bridge = CvBridge()
         self.calibrate = False
         self.record = False
         if self.calibrate:
            self.criteria = (cv2.TERM_CRITERIA_EPS + ...
                cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
            self.chessboard_size = (7, 9)  # 7 corners in width, 6 in height
            # Prepare object points based on the chessboard size
            self.objp = np.zeros((self.chessboard_size[0] * ...
                self.chessboard_size[1], 3), np.float32)
            self.objp[:, :2] = np.mgrid[0:self.chessboard_size[0], ...
                0:self.chessboard_size[1]].T.reshape(-1, 2)
            # Arrays to store object points and image points from all the ...
                images.
            self.objpoints = [] # 3d point in real world space
            self.imgpoints = [] # 2d points in image plane.
         self.video_writer = None

    def custom_cleanup(self):
            if self.video_writer:
                self.video_writer.release()
            if self.calibrate:
```

```
34                 self.get_logger().info(f"{self.mtx}")
35                 self.get_logger().info(f"{self.dist}")
36 
37     def impliment_matrix(self,camera_port):
38         ret, frame = self.cap.read(camera_port)
39 
40         if ret == True:
41             if self.calibrate:
42                 self.gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
43                 # Find the chess board corners
44                 ret2, corners = cv2.findChessboardCorners(self.gray, ...
                       (7,9), None)
45                 # If found, add object points, image points (after ...
                       refining them)
46                 if ret2 == True:
47                     self.objpoints.append(self.objp)
48                     corners2 = cv2.cornerSubPix(self.gray,corners, ...
                           (11,11), (-1,-1), self.criteria)
49                     self.imgpoints.append(corners2)
50                     # Draw and display the corners
51                     cv2.drawChessboardCorners(frame, (7,9), corners2, ret)
52                     ret, self.mtx, self.dist, rvecs, tvecs = ...
                           cv2.calibrateCamera(self.objpoints, ...
                           self.imgpoints, self.gray.shape[::-1], None, None)
53 
54             mtx = np.array(((426.17685011,   0,          320.38487224),
55              (0,           429.76884458, 225.27715136),
56              (0,            0,           1          ),))
57             dist = np.array(((-0.35653578,  0.11731714,  0.01052246,  ...
                   0.00376304, -0.01392377),))
58 
59             img = frame.copy()
60             h, w = frame.shape[:2]
61             newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, ...
                   (w,h), 1, (w,h))
62             mapx, mapy = cv2.initUndistortRectifyMap(mtx, dist, None, ...
                   newcameramtx, (w,h), 5)
63             dst1 = cv2.remap(img, mapx, mapy, cv2.INTER_LINEAR)
64 
65             # crop the image
66             x, y, w, h = roi
67             dst = dst1[y:y+h, x:x+w]
68             height = max(img.shape[0], dst.shape[0])
69             width_img = int(img.shape[1] * (height / img.shape[0]))
70             width_dst = int(dst.shape[1] * (height / dst.shape[0]))
71             img_resized = cv2.resize(img, (width_img, height))
72             dst_resized = cv2.resize(dst, (width_dst, height))
73 
74             return dst_resized
75 
76     def timer_callback(self):
77 
78             dst_image = self.impliment_matrix(0)
79             self.publisher.publish(
80                 self.cv_bridge.cv2_to_imgmsg(dst_image,"bgr8"))
81 
82             if self.record:
83                 # Write frame to video file
84                 if self.video_writer is None:
```

```
85                      # Define the codec and create VideoWriter object
86                      fourcc = cv2.VideoWriter_fourcc(*'mp4v')
87                      self.video_writer = cv2.VideoWriter('output.mp4', ...
                            fourcc, 20.0, (dst_resized.shape[1], ...
                            dst_resized.shape[0]))
88                  # Write the frame
89                  self.video_writer.write(dst_image)
90
91
92  def main(args=None):
93      rclpy.init(args=args)
94      Camera =  USB_Camera()
95      signal.signal(signal.SIGINT, lambda sig, frame: Camera.custom_cleanup())
96      rclpy.spin(Camera)
97      Camera.custom_cleanup()  # Ensure cleanup is called if exit wasn't ...
            due to SIGINT
98      rclpy.shutdown()
99      Camera.destroy_node()
100
101 if __name__ == '__main__':
102     main()
```

Vedlegg K

kode for verdenen i simulator

Kode K.1: Koden fra bench.world

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <world name="default">
  <gravity>0 0 0</gravity>

      <include>
        <uri>model://sun</uri>
      </include>



      <include>
        <uri>model://background</uri>
        <name>water2</name>
      </include>

      <include>
        <uri>model://valve</uri>
          <pose>3 0 0 1.57 0 0</pose>
      <name>valve</name>
      </include>

      <include>
        <uri>model://saruco0</uri>
          <pose>2.352033 -0.903928 0.772462 0 0 0</pose>
      <name>saruco0</name>
      </include>

      <include>
        <uri>model://saruco1</uri>
          <pose>3.590819 -0.903928 1.232259 0 0 0</pose>
      <name>saruco1</name>
      </include>

      <include>
        <uri>model://saruco2</uri>
```

```
37            <pose>3.643968 -0.283789 0.982318 0 0 0</pose>
38        <name>saruco2</name>
39        </include>
40
41        <include>
42          <uri>model://saruco3</uri>
43            <pose>2.882864 0 0.916771 0 1.57 0</pose>
44        <name>saruco3</name>
45        </include>
46
47          <plugin name="gazebo_ros_state" filename="libgazebo_ros_state.so">
48            <ros>
49              <namespace>/demo</namespace>
50              <argument>model_states:=model_states_demo</argument>
51            </ros>
52            <update_rate>1.0</update_rate>
53          </plugin>
54
55    </world>
56  </sdf>
```

Kode K.2: Koden fra pipeline.world

```
1  <?xml version="1.0" ?>
2  <sdf version="1.5">
3    <world name="default">
4    <gravity>0 0 0</gravity>
5
6        <include>
7          <uri>model://sun</uri>
8        </include>
9
10       <include>
11         <uri>model://background</uri>
12       </include>
13
14       <include>
15       <uri>model://yellow_pipe</uri>
16       <pose>3 0 0 1.5708 0 1.5708</pose>
17       <name>yellow_pipe_1</name>
18       </include>
19
20       <include>
21       <uri>model://aruco0</uri>
22       <pose>3 0 0.25 3.1415 1.5708 0</pose>
23       <name>aruco_0</name>
24       </include>
25
26       <include>
27       <uri>model://corner</uri>
28         <pose>6 0 0 0 0 0</pose>
29       <name>corner_1</name>
30       </include>
31
32       <include>
33       <uri>model://yellow_pipe</uri>
34         <pose>6 3 0 1.5708 0 0</pose>
```

```
35        <name>yellow_pipe_2</name>
36        </include>
37
38        <include>
39        <uri>model://aruco1</uri>
40        <pose>6 3 0.25 1.5708 1.5708 0</pose>
41        <name>aruco_1</name>
42        </include>
43
44        <include>
45        <uri>model://corner</uri>
46          <pose>6 6 0 0 0 0</pose>
47        <name>corner_2</name>
48        </include>
49
50        <include>
51        <uri>model://yellow_pipe</uri>
52          <pose>9 6 0 1.5708 0 1.5708</pose>
53        <name>yellow_pipe_3</name>
54        </include>
55
56        <include>
57        <uri>model://aruco2</uri>
58        <pose>8.1 6.25 0.25 1.5708 1.5708 0</pose>
59        <name>aruco_2</name>
60        </include>
61
62        <include>
63        <uri>model://corner</uri>
64          <pose>12 6 0 0 0 0</pose>
65        <name>corner_3</name>
66        </include>
67
68        <include>
69        <uri>model://yellow_pipe</uri>
70          <pose>14.1 8.09 0 1.5708 0 2.3562</pose>
71        <name>yellow_pipe_4</name>
72        </include>
73
74        <include>
75        <uri>model://aruco4</uri>
76        <pose>13.34 7.32 0.25 1.5708 1.5708 2.3562</pose>
77        <name>aruco_5</name>
78        </include>
79
80          <plugin name="gazebo_ros_state" filename="libgazebo_ros_state.so">
81            <ros>
82              <namespace>/demo</namespace>
83              <argument>model_states:=model_states_demo</argument>
84            </ros>
85            <update_rate>1.0</update_rate>
86          </plugin>
87
88    </world>
89  </sdf>
90  5
```