Bacheloroppgave

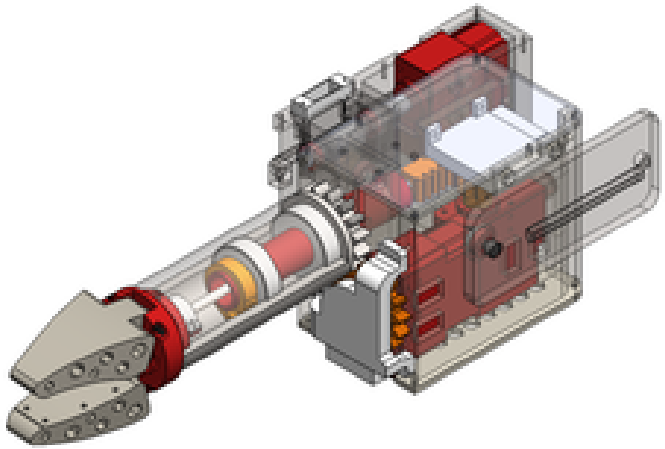
NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

Knut Ola Nøsen
Markus Johnstad
Joakim Fjeldkjøn
Tobias Slettebakken

3DOF Undervannsmanipulator

3DOF Subsea Manipulator

Bacheloroppgave i Automatisering og Robotikk
Veileder: Pål Holthe Mathisen
Mai 2024

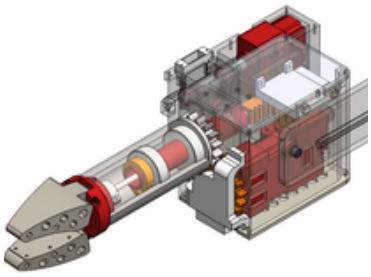


NTNU
Kunnskap for en bedre verden

Knut Ola Nøsen
Markus Johnstad
Joakim Fjeldkjøn
Tobias Slettebakken

3DOF Undervannsmanipulator

3DOF Subsea Manipulator



Bacheloroppgave i Automatisering og Robotikk
Veileder: Pål Holthe Mathisen
Mai 2024

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

NTNU
Kunnskap for en bedre verden

Forord

Denne bacheloroppgaven er utført ved Institutt for teknisk kybernetikk ved Norges teknisk naturvitenskapelige universitet (NTNU) som en del av IELET2920 - Bacheloroppgave i automatisering. Prosjektet ble gjennomført av studentene Markus Johnstad, Joakim Fjeldkjøn, Knut Ola Nøsen og Tobias Slettebakken, med veiledning av Pål Holthe Mathisen og Vortex representant Benjamin Roald Andersen.

Rapporten er tiltenkt for de som skal videreutvikle dette konseptet. Den er også rettet mot våre medstudenter og andre bachelorstudenter.

Prosjektet har vært en lærerik og utfordrende prosess som har gitt oss verdifulle erfaringer innen tverrfaglig ingeniørarbeid. Vi har fått dypere innsikt i mekanisk design, elektronikk og programvareutvikling. Vi har også erfart betydningen av samarbeid og kommunikasjon i et team. Vi ønsker å rette en stor takk til Vortex NTNU som har gitt oss muligheten til å bidra i deres prosjekt og all støtten de har gitt oss underveis.

Vi ønsker å uttrykke vår takknemlighet til veilederen vår Pål Holthe Mathisen, som har bidratt med gode tilbakemeldinger og råd underveis. Takk til Vortex representant Benjamin Roald Andersen for hans støtte og oppmuntring. Takk til mekanisk leder i Vortex, Sander August Hjortland, for gode tilbakemeldinger gjennom siste halvdel av prosjektet, og takk til Sebastian Opheim Hedel, som har gitt god innsikt i resten av det elektriske systemet. Til slutt vil vi takke våre familier og venner for deres forståelse og støtte gjennom denne perioden. Deres oppmuntring har motivert oss til å yte vårt beste og fullføre prosjektet.

Trondheim, Mai 2024

Knut Ola Nøsen,
Markus Johnstad,
Joakim Fjeldkjøn,
Tobias Slettebakken

Sammendrag

Denne bacheloroppgaven tar for seg utviklingen av en undervannsgripper for Vortex NTNU sin deltakelse i TAC Challenge 2024. Vortex er en studentorganisasjon ved NTNU som utvikler autonome undervannsfartøy. Formålet med dette prosjektet er å designe, produsere og montere en fungerende gripper som kan anvendes til å manipulere ventiler under vann. Oppgaven omfatter både de mekaniske, elektroniske og programvaremessige aspektene ved gripperen. Gripperen som er utviklet i dette prosjektet, er en essensiell komponent for å utføre disse oppgavene.

Gripperen er designet med CAD-programmet Solidworks og består hovedsakelig av 3D-printede deler i PETG, valgt for sin robusthet og vannmotstand. Designprosessen inkluderte flere iterasjoner med fokus på å oppnå god funksjonalitet og estetikk. Destruktive tester avdekket svakheter som ble dokumentert, og mekaniske forbedringer ble gjort.

Gripperens elektroniske system er bygget rundt servomotorer og magnetiske enkodere, som er beskyttet mot vanninntrengning ved hjelp av støping i epoxy. For å sikre at systemet er robust og pålitelig, er det implementert løsninger som gjør det mulig å overvåke og kontrollere strømforbruket til hver servomotor. Dette er spesielt viktig for å forhindre skader og sikre delvis funksjonalitet ved feil.

Programvarearkitekturen er basert på ROS2 (Robot Operating System 2), som gir en fleksibel plattform for kontroll og diagnostikk av gripperen. ROS2-pakkene utviklet for dette prosjektet inkluderer spesifikke moduler for bevegelsesstyring, diagnostikk og teleoperasjon. Disse modulene integrerer ulike sensordata og gir brukeren muligheten til å styre gripperen nøyaktig og pålitelig.

Den ferdige gripperen oppfyller alle krav, med enkelte justeringer basert på praktiske erfaringer. Prosjektet demonstrerer en vellykket integrering av en funksjonell undervannsgripper, klar til bruk i konkurranser og videre utvikling. Den er i stand til å rotere i flere akser, åpne og lukke seg, og fungerer pålitelig under vann. Fig 2 viser det endelige resultatet.

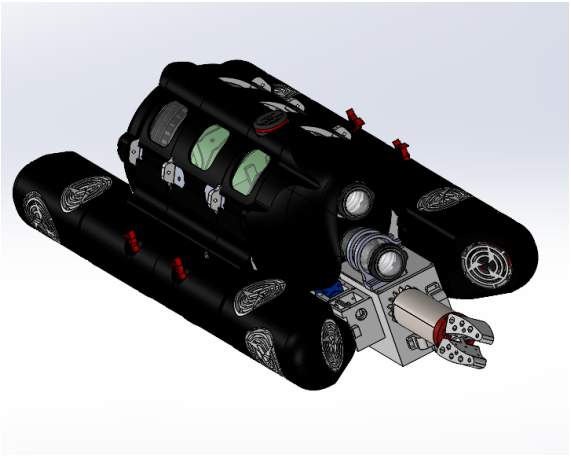


Figure 1: Hele dronen

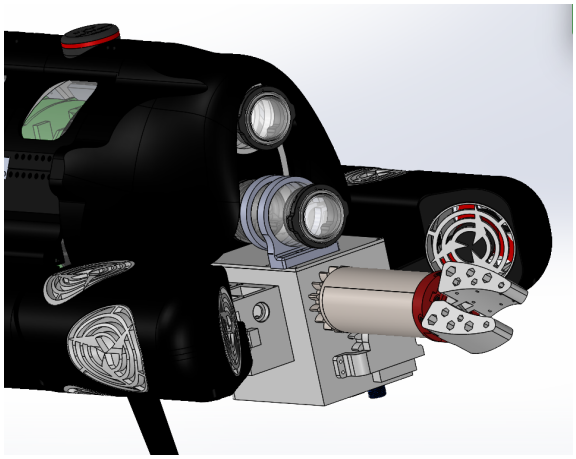


Figure 2: Hele dronen (gripper)

Summary

This bachelor thesis focuses on the development of an subsea manipulator for Vortex NTNU's participation in the TAC Challenge 2024. Vortex is a student organization at NTNU that develops autonomous underwater vehicles. The purpose of this project is to design, produce, and assemble a functional gripper that can be used to manipulate subsea valves. The project encompasses the mechanical, electronic, and software aspects of the manipulator. The system developed in this project is an essential component for performing the beforementioned.

The gripper is designed using the CAD software SolidWorks and primarily consists of 3D-printed parts made of PETG, chosen for its robustness and water resistance. The design process included several iterations focused on achieving the optimal balance between functionality and aesthetics. Destructive tests revealed weaknesses that were documented, and mechanical improvements were made.

The electronic system is built around servomotors and magnetic encoders, which are protected against water ingress by encapsulating them in epoxy. To ensure that the system is robust and reliable, solutions have been implemented to monitor and control the power consumption of each servomotor. This is especially important to prevent damage and ensure partial functionality in the event of failure.

The software architecture is based on ROS2 (Robot Operating System 2), which provides a flexible platform for controlling and debugging the gripper. The ROS2 packages developed for this project include specific modules for motion control, diagnostics, and teleoperation. These modules integrate various sensor data and allow the user to control the gripper accurately and reliably.

The end product meets all requirements, with some iterative adjustments. The project demonstrates a successful integration of a functional subsea manipulator, ready for use in competitions and further development. It is capable of rotating in multiple axes, opening and closing, and reliable submerged operation.

Innholdsfortegnelse

Figurliste

1 Innledning

Prosjektet ble utført som en del av bacheloroppgave i automatisering ved Institutt for teknisk kybernetikk ved NTNU. Rapporten presenterer en beskrivelse av designprosessen og implementeringen av verdens minste undervannsgripper med 3 frihetsgrader, og til slutt en evaluering av dens ytelse og funksjonalitet.

1.1 Bakgrunn

Autonome undervannsfartøy (AUV) har blitt en viktig teknologi i moderne marin forskning, industri og miljøovervåkning. Disse fartøyene er utstyrt med avanserte systemer for navigasjon, kommunikasjon og manipulasjon, og de kan operere i krevende undervannsmiljøer hvor menneskelig tilgang er begrenset.

Vortex NTNU er en studentorganisasjon som ble opprettet med formålet å bygge autonome undervannsfartøy som er i stand til både observere og manipulere objekter under vann.

Vortex deltar i flere konkurranser med undervannsdronen sin, ≪Orca 2024≫, og en av dem er "TAC Challenge 2024". Det er en konkurranse med mange oppgaver som strekker seg over en femdagers periode. En del av konkurransen går ut på å finne to ventiler i et stort basseng, for så å dreie disse ventilene 90 grader. Dette ønsker Vortex å løse ved å montere en gripper på dronen.

Denne rapporten skal ta for seg den fullstendige designprosessen til manipulatoren, og vil beskrive hvordan de fysiske delene ble designet, produsert og montert. Videre beskrives valg av motorer og sensorer, og hvordan det hele blir kontrollert ved hjelp av programvare.

Vortex prøvde tidligere å bestille en eksisterende manipulator til dronen som kan utføre de nødvendige oppgavene. I dag finnes det derimot ingen manipulatorer med 3 frihetsgrader, som er små nok til å få plass på dronen.

1.2 Rapportens oppbygning

Prosjektet inneholder kunnskap fra flere ingeniørfelt. Mekanisk har fokusert på å utvikle en robust og funksjonell gripper ved hjelp av 3D-printing og moderne CAD-verktøy. Elektronikk har tatt for seg valg og integrasjon av servomotorer og sensorer, samt beskyttelse mot vanninntrengning. Programvareutviklingen, basert på ROS2 (Robot Operating System 2) gir en plattform for nøyaktig kontroll, feilsøking i felten og trygge backup løsninger.

Siden rapporten tar for seg tre separate fagfelt som sjeldent overlapper, er det besluttet å benytte en alternativ struktur til å inndele rapporten i teori, metode, resultater og diskusjon. Dette gjør at teori kan introduseres rett før den er nyttig. Resultatet er en rapportstruktur som er mer oversiktlig. Dette gjør det blant annet enklere for leseren å huske detaljer når de er relevante. Innad i enkeltkapitler kommer fortsatt teoretisk informasjon før arbeids- og utviklingsmetoder, og etter disse presenteres resultater. Disse eksakte overskriftene er derimot ikke brukt, ettersom dette ville gjort teksten vanskeligere å følge. Det er ønskelig at rapporten skal kunne fungere som en dokumentasjon for videre utvikling, og strukturen er derfor valgt for å være enklere å navigere. Etter ønske fra Vortex er det også benyttet engelske fagbegrep for å gjøre dokumentet mer forståelig for fremtidige utviklere.

1.3 Krav til gripper

For gripperen ble det stilt helt spesifikke krav fra Vortex. Disse er tatt hensyn til fra start.

1.3.1 Opprinnelige krav

Dette er de orginale kravene stilt ved prosjektstart.

- Rotere mer enn 90 grader nedover om skulderleddet.
- Rotere opptil 90 grader oppover om skulderleddet.
- Rotere håndleddet minimum 360 grader.
- Total bredde der manipulator festes må være på 157mm for å få plass mellom aluminiumsprofiler.
- Skal kunne styres med ROS.
- Alle frihetsgraders posisjon skal være kjent via software.
- Gripperen skal kunne åpnes og lukkes.
- Skal være så billig og lett som mulig.
- Skal fungere under vann.

1.3.2 Endrede krav

Underveis er det kommet flere krav og spesifikasjoner fra oppdragsgiver.

- 3D-printede deler må være i PETG, med minimum 1,2mm tykke vegger og minimum 60% infill. Gir må ha 2 mm tykke vegger.
- Deler av metall skal være i rustfritt stål eller annet metall som er motstandsdyktig mot korrosjon.
- Gripperen skal være sentrert på dronen.
- Rotere over 45 grader oppover om skulderleddet.
- Utenfor elhuset er kan det maksimal brukes tre kabler, med maksimalt ti ledere i hver.
- Det bør maksimalt ta ti minutter å demontere selve armen fra hovedboksen.

2 Mekanisk

Den mekaniske delen tar for seg hvordan gripperen ble designet og modellert, hvilke utfordringer som er støtt på underveis og hva som ble det endelige resultatet.

2.1 Solidworks

For å designe delene er Solidworks blitt brukt. Solidworks er en CAD (Computer Aided Design) programvare for design og modellering av objekter som skal 3D-printes. Det er brukt NTNU sin studentlisens for tilgang på programmet. Ettersom Vortex bruker Solidworks til designet av dronen, Orca 2024, er det naturlig at gripperen blir designet i samme program. Dermed kan Vortex enkelt integrere 3D-modellene av gripperen sammen med resten av dronen.

2.1.1 Parts og assembly

I Solidworks kalles hver enkelt del som modelleres en "part". Hele grippperen er designet som enkeltdeler, som kan plasseres i en "assembly". Dette er en sammensetning av alle de ulike komponentene, og gjør det mulig å se hvordan disse fungerer sammen. [14]. Det er muligheter for "mating" av deler, som definerer og plasserer delene nøyaktig i forhold til hverandre. Deretter kan gripperens ledd og gir drives i programmet for å simulere de mekaniske komponentene. Dette er brukt som en nøyaktig måte å modellere og se etter kollisjoner og målefeil uten å måtte printe og teste deler.

2.2 3D-printing

I oppgaven er det valgt å bruke 3D-printing for å produsere de fleste delene. Dette brukes fordi 3D-printing gir en kort vei fra design til klart produkt. Det er også billig og raskt i forhold til å bestille deler. I tilegg har Vortex flere 3D-printere som er tilgjengelige.

2.2.1 Filament

Filament er materialet som brukes i en 3D-printer. Ofte er dette plast av ulike typer. Filamentet er formet som en tråd, som varmes opp og smeltes. Deretter legges det lag på lag til et 3D-objekt. Det er benyttet to typer filament av plast i dette prosjektet:

PLA PLA er et mindre kostbart filament, som også printes raskt. Dette er grunnene til at PLA er valgt til alle test-print. [20].

PETG PETG er mer fleksibelt og robust enn PLA, i tillegg til at det har bedre evne til å ikke trekke inn vann. Dette er positivt for både vekt og balanse på dronen. PETG tåler også bedre støt fordi det er mer fleksibelt. Derfor er PETG valgt som filament til det endelige produktet. [20].

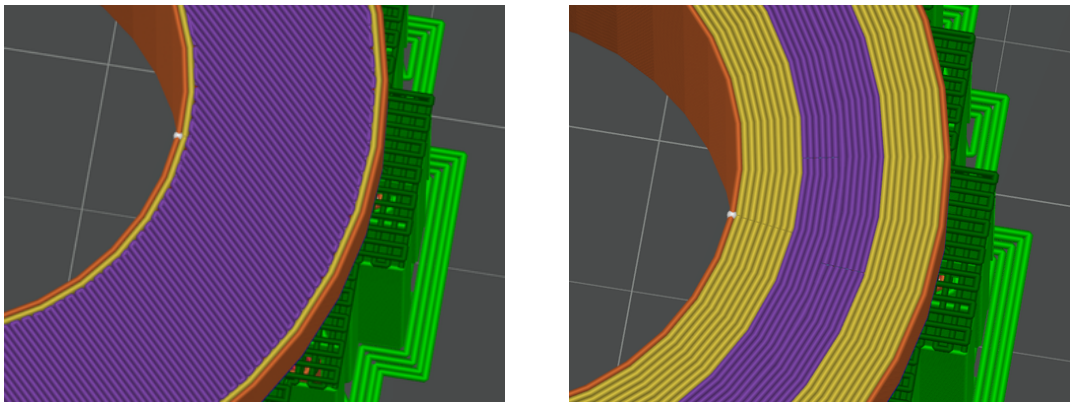
2.2.2 Slicing

Når design og modellering er ferdig i Solidworks gjenstår printingen av delene. Dette gjøres ved å lagre Solidworks filen som en STL fil. Et slicing-program brukes til å oversette STl-fil til en g-kode-fil som printeren kan forstå. En g-kode er en kommandoliste som printeren følger steg for steg. [33].

2.2.3 Valg av print-innstillinger

Print-innstillinger spiller en avgjørende rolle for resultatet av printet. Innstillingene påvirker materialbruk og kostnad, i tillegg til å påvirke styrke, vekt og overflate.

Innfil og veggtykkelse En 3D-printet komponent har ytter- og innervegger. Disse må ha en bestemt tykkelse, som er konfigurerbart i sliceren. Samtidig vil en 3D-printet del ofte inneholde mye tomt rom mellom veggene. Innfil innstillingene bestemmer hvor mye av dette tomrommet som skal fylles med filament, og hvilket mønster det skal ha. Siden dronen skal være autonom er det nødvendig å lage et system som er så balansert som mulig. Dette vil si at tettheten på alle deler må være mer eller mindre konstant. Ettersom det ikke er trivielt å gjøre 3D-print vanntett ned til 200m, er det av den grunn ønskelig å ha en del som ikke inneholder luft. For å oppnå dette er det benyttet 100% infill på alle deler, som betyr at alle hulrom i alle vegger fylles med plast. Vanligvis oppnås 100% infill ved å produsere et sikksakkmønster, men for delene i dette prosjektet ga det mer mening å øke veggtykkelsen til 100%. Dette vil også gi sterkere komponenter. Fig 3a og Fig 3b illustrerer respektivt forskjellen på 100% infill og høy veggtykkelse i en 3D slicer.



(a) 3D printet del med 100% infill

(b) 3D printet del med høy veggtykkelse

Figure 3: Infill/veggtykkelse

Supports og orientasjon FDM 3D-printing, som er benyttet i dette prosjektet, bygger deler lag på lag fra bunnen og opp. Noen ganger må det derimot printes deler med "overheng", hvor seksjoner av delen ikke har kontakt med byggeplaten. For å løse dette problemet genereres det automatisk supports, som vist i Fig 4. Selv om supports printes på smarte måter som gjør at de har en svak kontakt med selve produktet, etterlater de derimot ujevnheter på kontaktflaten som kan være problematisk for bevegende deler. I tilfelle av delen i Fig 4 må den orienteres på denne måten for at det tynne 2mm x 2mm sporet ikke skal bli ujevnt, ettersom enkoderkassettene skal kunne gli langs dette sporet uten motstand. Mer info om denne komponenten kommer senere.

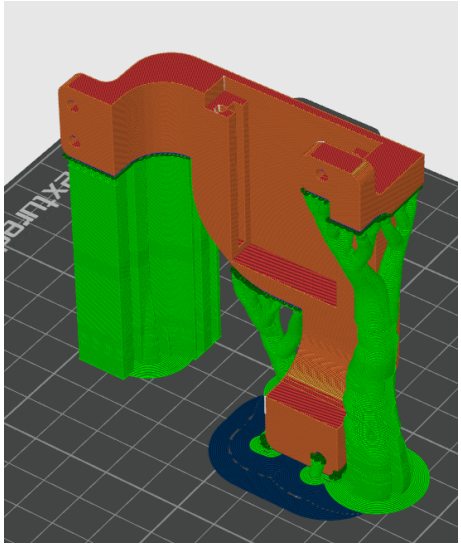


Figure 4: Supports generert av slicer

2.3 Transmisjoner

Forskjellige girtyper er brukt for de forskjellige leddene i gripperen. Dette på grunn av girtypenes unike egenskaper.

2.3.1 Wormgear

Et wormgear, illustrert av Fig 5a, gir en mekanisk utveksling med to spesielt bemerkelsesverdige egenskaper. Den flytter rotasjonsaksen 90 grader, og når utvekslingen er stasjonær vil krefter ikke overføres fra den drevne akselen til drivakselen. Det sies at utvekslingen ikke er "backdrivable". [6]. Dette var ønsket fra Vortex for å beskytte den dyre skulderservoen fra støt i forbindelse med kollisjon. Dette betyr også at skulderservoen ikke trenger å bruke strøm for å holde sin posisjon.

(a) Wormgear [15]

(b) Spurgear [4]

(c) Rack og pinion [7]

2.3.2 Spurgear

Et spurgear, illustrert av Fig 5b, er en type tannhjul kjent for å minimere dødrom mellom tennene [2], og blir brukt for å rotere håndleddet. Mekanismen gjør det mulig å plassere servoens drivaksel ved siden av armens rotasjonsakse, og er nødvendig for å kunne lese av posisjonen med enkoderen. Detaljer om enkoderene tas opp i kapittel 3.2.1.

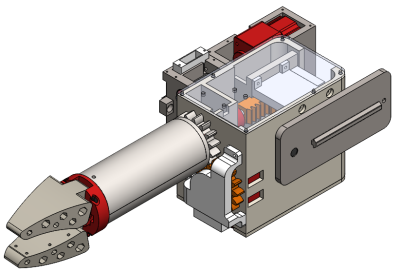
2.3.3 Rack og Pinion

En rack og pinion, illustrert av Fig 5c, er en utveksling som konverterer rotasjonsbevegelse til lineær bevegelse. Dette blir brukt for å aktuere kloen, og ble valgt fordi den er "backdrivable" som betyr at kloens gripekraft i fremtiden kan reguleres basert på servoenes strømtrekk.

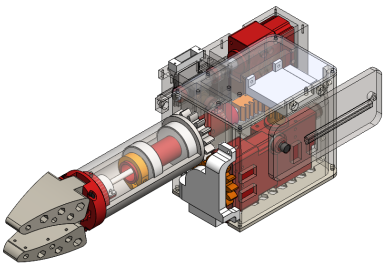
2.4 Mekaniske komponenter

Under utvikling av gripperen var det hensiktsmessig å dele den opp i flere komponenter for å gjøre prosessen raskere og enklere. Dette gjør det mulig å iterere på konseptet uten gjentatt produksjon av større komponenter. Dette legger også til rette for produksjon av reservedeler, noe som er essensielt i en konkurranse. Til slutt skaper dette muligheten for å printe hver enkelt komponent i en konfigurasjon som gir best mulig resultat.

Følgene avsnitt beskriver hver komponent i detalj. Fig 6 illustrerer det endelige resultatet av hele gripperen. Når det refereres til retninger, som venstre og høyre, brukes dronens perspektiv. For eksempel er venstre side av gripperen i Fig 6 på høyre side av bildet.



(a)

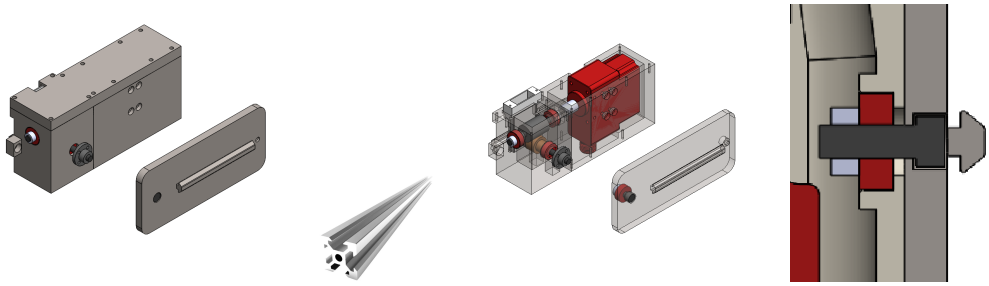


(b)

Figure 6: Hele gripperen

2.4.1 Aluminiumsprofilfester

Gripperen er festet mellom to aluminiumsprofiler som vist i Fig 7b. Det ble laget et spor langs sideplatene, som vist i Fig 7a og Fig 7c, som kunne skli inn i sporet på aluminiumsprofilene. Her ble det laget flere testversjoner av festemekanismen, for å gjøre den så nøyaktig som mulig. Konstruksjonen låses fast i aluminiumsprofilene med t-slots og bolter.



(a) (b) [8] (c) (d)

Figure 7: Sideplatene

2.4.2 Venstre skulderhengsel

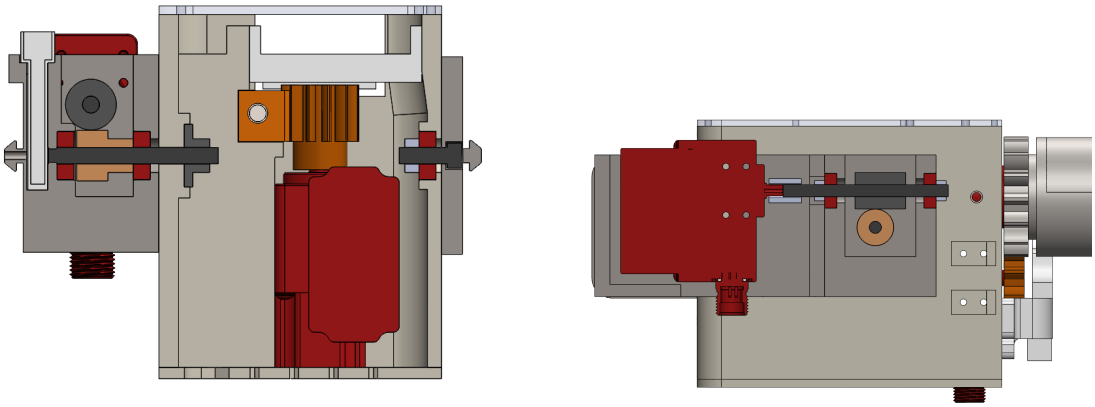
Venstre side av boksen festes til venstre sideplate gjennom rødt kulelager med en bolt som illustrert i Fig 7d.

2.4.3 Boks for ekstern skulderservo

Skulderbevegelsen skal drives av en servo som plasseres på utsiden av hovedboksen, som illustrert øverst i Fig 7c. Servoen er skrudd fast i en egen boks som er festet til aluminiumsprofilen.

2.4.4 Girkasse

Girkassen er festet til aluminuiumsprofilene, og inneholder wormgear utvekslingen og dens opplagring. Rotasjonskreftene fra servoen overføres til girene gjennom en stiv kobling. Akselen som går ut fra stivkoblingen gjennom wormgearet, går også gjennom et lager på hver side av girkassen. Dette illustreres gjennom tverrsnittet av servoboksen og girkassen i Fig 8b. Lagrene absorberer alle aksielle og radielle krefter, slik at servoens interne lager spares for påkjenningen. Girkassen er designet slik at disse lagrene monteres fra utsiden. Dette gjør at aksielle krefter føres gjennom hele girkassens konstruksjon, i stedet for kun en vegg. I likhet med wormen, går det en aksel gjennom wormwheelet, som også holdes på plass av et lager på hver side. Lagrene er integrert i girkassen og absorberer alle radielle krefter, som blant annet kommer av at vekten til hele armen hviler på dem. Dette illustreres av tverrsnittet til venstre i Fig 8a.



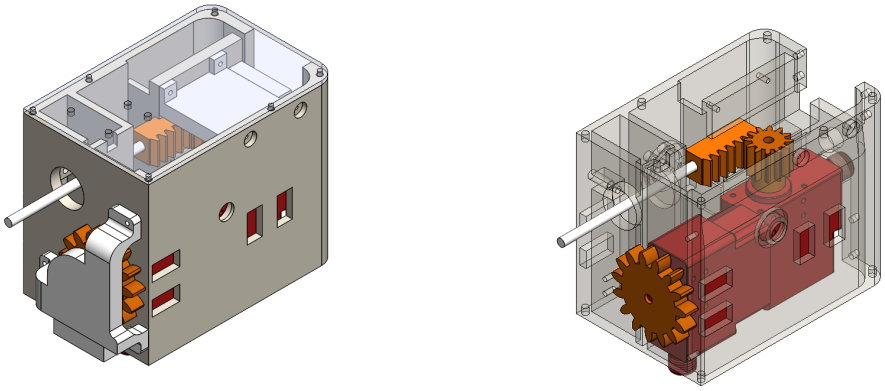
(a) Snitt Skulderaksel

(b) Snitt wormgear

Figure 8: Snitt av feste mellom sideplater og hovedboksen

2.4.5 Hovedboks

Hovedboksen er det sentrale festepunktet for mange av de mekaniske delene. Blant disse er armen, som er plassert asymmetrisk på hovedboksen for å stå midt imellom aluminiumsprofilene og dermed i senter av dronens bredderetning. Selve boksen står også asymmetrisk på dronen fordi girkassen ikke har lik bredde som venstre sideplate. Armen tres gjennom to hull i boksen og festes av en låsebolt fra høyre side. Fig 9a og Fig 9b illustrerer hovedboksen.

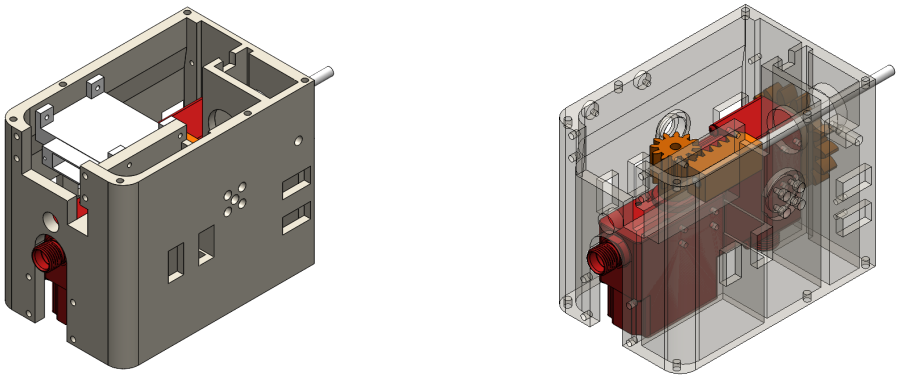


(a)

(b)

Figure 9: Hovedboks fremside

Servoene er plassert etter hverandre slik som vist i figur 9b fordi dette er eneste måten å få plass til armen i senter av aluminiumsprofilene.



(a) Hovedboks bak

(b) Arm gjennomsiktig

Figure 10: Hovedboks bakside

Boksen har firkantede hull på hver side for å gjøre plass til verktøy under montering. Dette illustreres av Fig 9a og Fig 10a.

Topplokket på hovedboksen er designet for å kunne montere et kamera, i tillegg til å hindre at fremmedlegemer kommer inn. Bunnlokket er designet i et nettingmønster for å enkelt kunne fylle boksen med vann, men forhindre uønskede elementer å trenge inn i boksen. Bunnlokket er illustrert i Figur 11. Begge lokkene festes til boksen med bolter. boltene festes i inserts, som er metallgjenger som smeltes inn i konstruksjonen med en loddebolt.

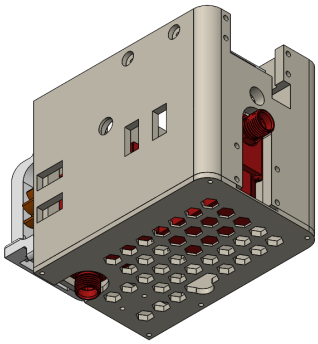
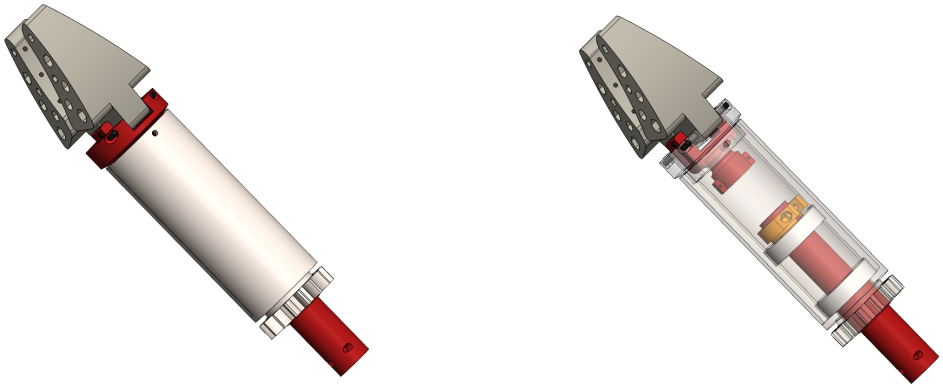


Figure 11: Underside hovedboks med lokk

2.4.6 Arm

Armen består at et ytterskall som er integrert med et tannhjul. Håndleddsrotasjonen til armen drives av dette tannhjulet.



(a) Arm (b) Arm gjennomsiktig

Figure 12: Selve armen

I ytterskallet til armen blir en sylinder holdt på plass av to lagre og et låsebånd. Gjennom denne sylinderen går en stang der den ene enden er festet til kloen. Den andre enden er festet til en rack inni hovedboksen, illustrert av Fig 9b, som drives av en servomotor. Denne servoen styrer altså klypefunksjonen til gripperen. Kloen består av to nesten identiske halvdeler som er skrudd fast til selve armen. De er også skrudd fast i en bevegelig adapter, som igjen er montert til stanga gjennom armen. Når servoen setter tannhjulklossen i bevegelse, vil altså stangen bevege seg frem og tilbake inni armen og sette adapteren i bevegelse. Armen illustreres i Fig 12b.

2.4.7 Klo

Kloen er formet slik at den er bredest ved armen og smalere på tupppen. Dette er for at den spisse delen skal være lett å styre inn i ventilene som skal åpnes, men samtidig få et godt og bredt grep når kloen er på plass inni ventilen. For å sørge for godt grep på ventilen når den skal skrus, er det integrert en gummimatte på kloen som gir en økt friksjon med kontaktflaten.

2.4.8 Enkoderfester

For å vite posisjonen til de ulike servoene brukes det magnetiske enkodere. Disse beskrives nærmere i 3.2.1 På hvert av de to tannhjulene i boksen er det laget en liten utskjæring til en magnet. Magneten er limt fast i et hull sentrert på tannhjulet. Chippen til enkoderene er sentrert på magneten, med 1-3mm avstand for å få en god avlesning.

Enkoderene tåler ikke vann og måtte derfor støpes i epoxy. Ettersom støpning av disse bare kan gjøres en gang, ble det besluttet å designe "kassetter" til enkoderene. På denne måten er det mulig å gjøre endringer på resten av gripperen uten at enkoderene må støpes på nytt. kassettene er identiske, men festepunktene er ulike for hver av de tre aksene.

Kassettene er designet slik at enkoderen akkurat får plass og kan skli nedi en liten boks. Boksen har ekstremt tynne vegger (1-2mm) for at den lett kan integreres på gripperen uten å ta for mye plass. På sidene av kassettene er det designet et spor på 2mm x 2mm som enkelt skal kunne skli inn i et tilhørende spor på brakettene. Øverst på kassettene er det tykkere vegger og større hulrom for å gi plass til enkoderkablene. Det er også designet to små skruehull på kassettene slik at de enkelt kan skrues fast. kassetten illustreres i Fig 13.

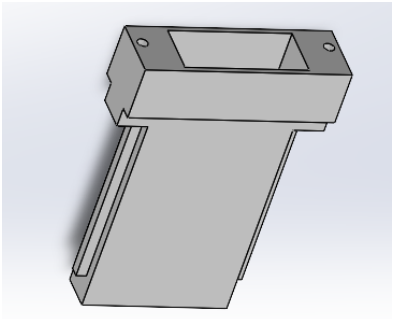


Figure 13: Enkoder Kassett

Skulder På girkassen er det designet et hull til kassetten. Hullet har samme fasong som kassetten, med en klaring på 0,2 - 0,5mm. Dette gjør at kassetten enkelt kan skli inn og ut uten friksjon. Komponenten illustreres i Fig 14.

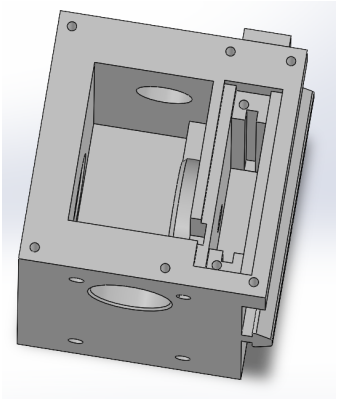
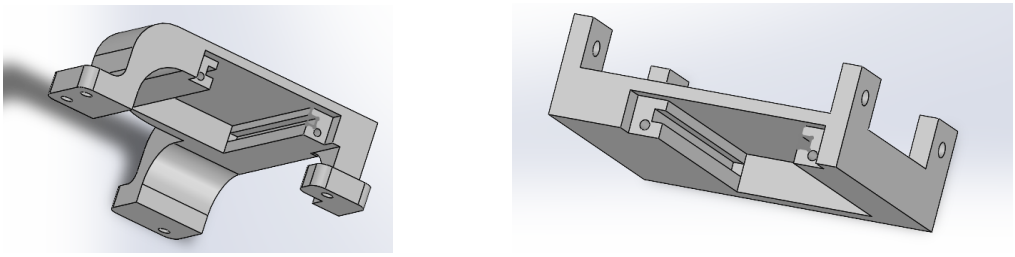


Figure 14: Girkasse med hull til kassett på høyre side

Håndledd For servoen som styrer rotasjonen til armens håndledd er det designet en egen brakett som kassetten kan skli inn i og skrues fast. Innsiden av braketten har nøyaktig samme mål som hullet på girkassen. Braketten skrus fast direkte på forsiden av hovedboksen, og fungerer som et deksel over tannhjulet. Komponenten illustreres i Fig 15a.



(a) Enkoderbrakett til håndleddsservo

(b) Enkoderbrakett til gripperservo

Figure 15: Enkoderbraketter

Gripper I likhet med braketten til håndleddsservo er det designet en egen brakett for å feste enkoderkassetten over tannhjulet som styrer gripefunksjonen. Braketten monteres på innsiden av hovedboksen, over tannhjulet og kassetten kan dermed skli inn og skrues fast. Komponenten illustreres i Fig 15b.

2.5 Støping av kabler og enkodere

For at det elektriske systemet skal fungere under vann må alle eksterne kabler og komponenter være vanntette. Begrenset antall plugger på elhuset krever også at flere eksterne kabler spleises sammen. Én kabel fører til alle tre enkoderne, og én kabel fører til to servoer. I tillegg måtte en av servokablene forlenges, noe som innebar spleising av en kabel til én servo. Totalt var det nødvendig å produsere tre vanntette spleiser.

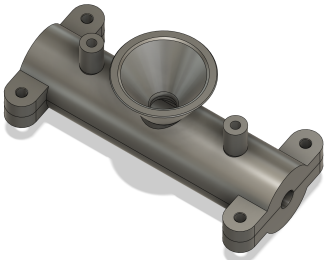
For støping av kabler benyttes "Scotchcast 2131", en flytende masse som binder seg til kabelkappene og fullstendig størkner etter 72 timer. For å holde massen på plass ble det designet spesifikke støpeformer for hver spleis. Fig 16a, Fig 16b, og Fig 16c viser de tre forskjellige støpeformene med luftehylser og trakt montert. Fig 16d og Fig 16e illustrerer innsiden av formene, mens Fig 16f viser et tverrsnitt av enkeltformen. skruehullene på sidene gjør det mulig å stramme støpeformene rundt kablene, noe som forhindrer sammenfall og sikrer at lederne ikke kommer i kontakt med ytterveggene, som ville resultert i at spleisene ikke ble vanntette.

Scotchcast 2131 avgir giftige gasser under støping, og dette arbeidet ble derfor utført med heldekkende drakter og pustemasker i et godt ventilert rom. Endene av kabelkappene ble skrapt med sandpapir og vasket med isopropanol for å sikre bedre sammenføyning med massen. Minst 5mm av kabelkappen var inne i støpeformene for å sikre at innsiden også inneholdt noe av massen. Dette gir bedre beskyttelse for elektronikken dersom kabelkappen skulle bli skadet.

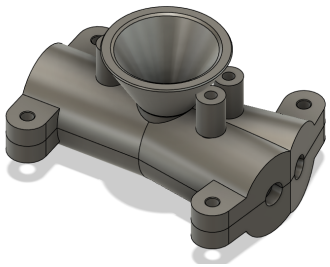
Fig 16g, Fig 16h, og Fig 16i viser lederne integrert i støpeformen. Fig 16j viser de fikserte formene med tildekte kabler for å hindre søl, mens Fig 16k viser en form fylt med masse.

For å gjøre enkoderne vanntette, ble de støpt i epoxy. Det var viktig at denne epoxyen ikke

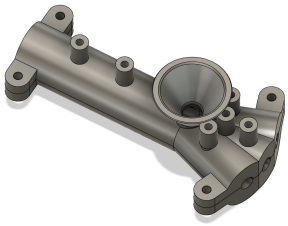
inneholdt sølv, ettersom dette ville gjort den elektrisk ledende. Fig 16l viser en enkoder og enkoderkabelen støpt i kassetten.



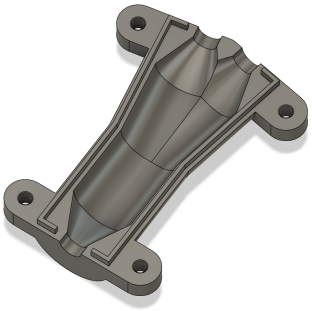
(a) Helhetlig bilde av enkel støpeform



(b) Helhetlig bilde av dobbel støpeform

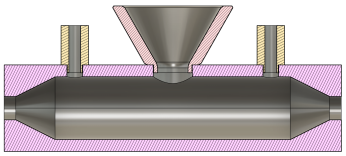


(c) Helhetlig bilde av trippel støpeform

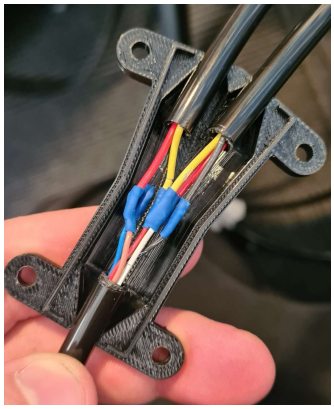


(d) Nederste halvdel av dobbel støpeform

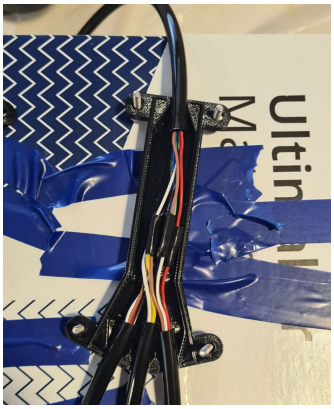
(e) Øverste halvdel av trippel støpeform



(f) Tverrsnitt av enkel støpeform



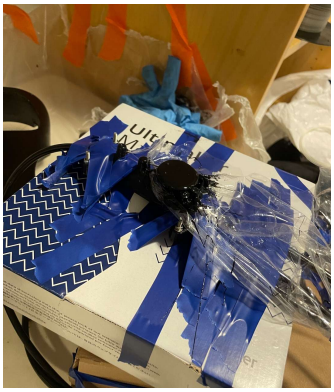
(g) Kabler i dobbel støpeform



(h) Kabler i trippel støpeform

(i) Dobbel støpeform sett fra utsiden

(j) Alle støpeformer fiksert og tildekt



(k) Dobbel støpeform fylt med masse

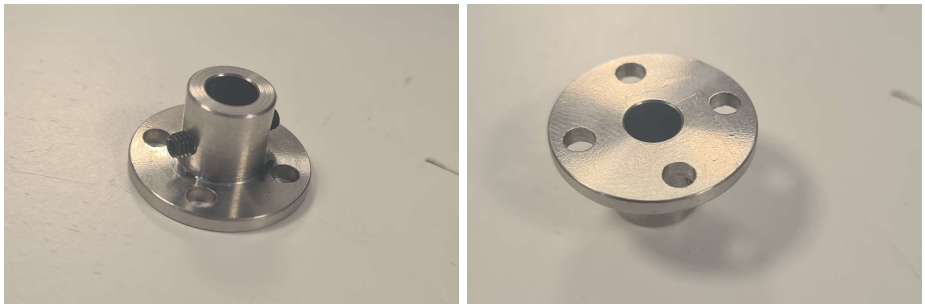
(l) Enkoder støpt i kassett

2.6 Maskinering

Skulderaksen opplever betydelig større krefter enn de andre. Av denne grunn måtte det maskineres deler i rustfritt stål for å gjøre den sterkere. Følgende avsnitt beskriver hvilke deler som måtte produseres på mekanisk verksted.

2.6.1 Skulderaksel

Kraften generert av girkassen overføres til hovedboksen gjennom en flenskobling som tidligere vist i Fig 8. Siden wormwheelet, drivakselen og flensen skal rotere sammen, må de festes i hverandre. For å løse dette benyttes en gjennomgående M3 bolt, både for flens og wormwheel. Fig 17a og Fig 17b viser flensen.



(a) Skulder flens topp (b) Skulder flens bunn

Figure 17: Skulder flens

2.6.2 Enkodermagnet i skulderaksel

Skulderaksens posisjon måles av en magnetisk enkoder. For å gjøre akselens posisjon lesbar integreres en magnet i enden som peker mot girkassen. Dette er oppnådd ved å bore et 1mm dypt hull med et 3mm bor, sette magneten på høykant i hullet, og lime fast med hurtigtørkende metallim som vist på Fig 18. Detaljer om enkoderene tas opp i kapittel 3.2.1.

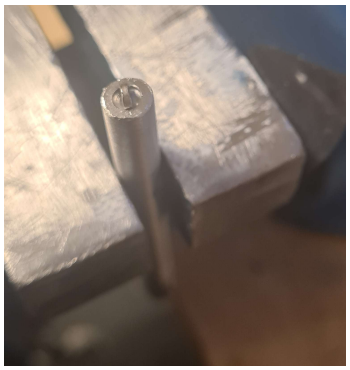


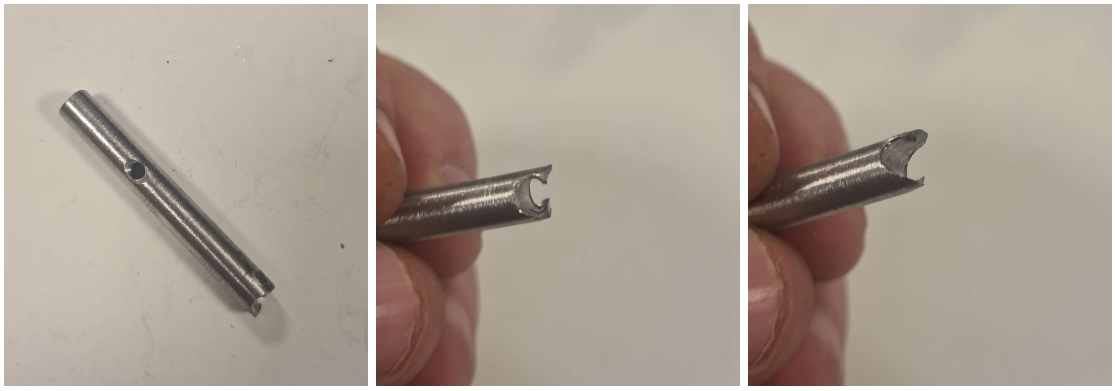
Figure 18: Magnet integrert i skulderaksel

2.7 Testing og forbedringer

Utviklingen av systemet har vert en iterativ prosess, og flere designvalg er basert på testresultater. Følgende avsnitt beskriver de mest bemerkelsesverdige problemstillingene og testresultatene som dukket opp underveis.

2.7.1 Destruktiv test

Etter en destruktiv test bekreftes det at det svakeste punktet på konstruksjonen er skulderakselen som kobler wormwheelet til hoved boksen. Det kommer frem av testen at akselen tåler opp til 5,5kg belasting ved armens ytterste flens, også kjent som klypens hengselpunkt, og knekker mellom 5,5kg og 6kg. Presisjonen på 0,5kg kommer av at det under testing ble brukt kjettinger på 0,5kg. Det svake punktet er midt på de gjennomgående låseboltene som vist på Fig 19a, Fig 19b og Fig 19c. Dette tallet representerer hvor mye vekt som kan belastes på gripperen uten at den knekker.



(a) (b) (c)

Figure 19: Knekt skulderaksel

2.7.2 Maks løftekraft

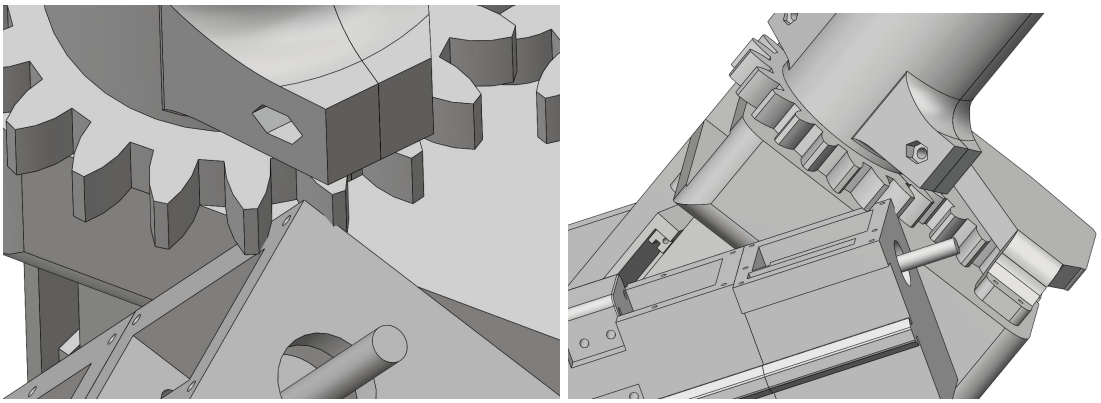
Skulderaksens løftekraft ble testet med en mekanisk belastning opptil 1,5kg ved armens ytterste flens. Med andre ord tåler armen en belastning på 5,5kg, men kan kun løfte et objekt på 1,5kg. Dette tallet er derimot trolig høyere, siden strømforsyningen var en begrensende faktor. Det ble ikke gjennomført en ny test med en alternativ strømforsyning, siden løftekraften anses å være god nok. Det vurderes også om smøring av wormgear og wormwheel kan redusere tapet i utvekslingen, og derfor øke maks løftekraft. Denne hypotesen kan derimot ikke testes før det kommer klarhet i henhold til TACC, som krever at dronen ikke skal avgi annet enn luft og vann. Girkassen er med hensyn til dette designet for å være så tett som mulig, for å gi muligheten til å fylle den med marine grease om dette skulle vise seg å være tillatt.

2.7.3 Mekanisk dødgang i skulderakse

På grunn av begrensede ressurser relatert til mekanisk maskinering har hullet for låseboltene i skulderakselen en diameter litt over 3mm. En standard M3 bolt er også litt mindre enn 3mm. Resultatet er en klaring på 0,1mm, som fører til en dødgang i armen på ca. ± 15 grader. Halvparten av denne dødgangen kommer fra låsebolten i wormwheelet, og den andre halvparten kommer fra låsebolten i flenskoblingen inni boksen. Fordi skuldermagneten er integrert i selve akselen, betyr dette at kun halvparten av slarken kan merkes av enkoderen. Dette fordi kun den delen av slarken som resulterer i en akselbevegelse. Med andre ord blir det en umålbar bevegelse på ± 7.5 grader i skulderaksen hvis tyngderetningen på armen endrer seg. Dette kan for eksempel skje hvis armen peker rett ned. Dette må løses ved å skaffe et nytt wormwheel med en ny festemekanisme, og en ny flenskobling for hovedboksen. Oppdragsgiver har derimot besluttet at de er fornøyde med den eksisterende løsningen, og avventer alternative løsninger til etter TACC.

2.7.4 Forenkling av hovedboks

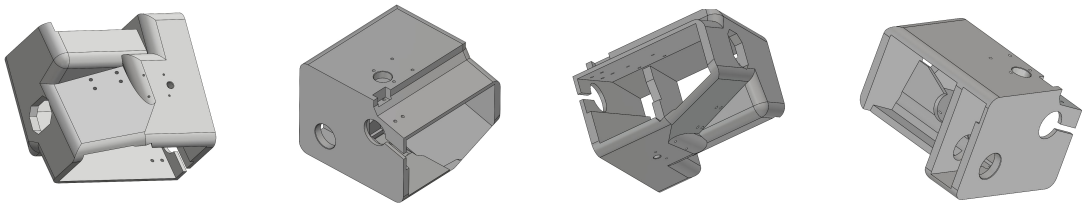
Det ble i de opprinnelige kravene fra Vortex, se 1.3.1 ønsket et så lite design som mulig . Det ble derfor et sterkt fokus på å få servoene i hovedboksen tett opp til hverandre. Måten det ble gjort på var å vinkle servoene, for å fremdeles bruke samme arm og gir. Dette utkastet av hovedboks fungerte fint, men har en del svakheter og begrensninger. Blandt annet stakk håndleddets tannhjul utenfor boksen, som skapte kollisjon med sideplatene. Dette illustreres i Fig 20.



(a) (b)

Figure 20: Tannhjul krasj på første hovedboks

Tidlig i prosessen ble det, som illustrert i Fig 21, utforsket med ulike former på hovedboksen, der veggene ble snevret inn for å lage boksen så liten som mulig. Det viste seg å ikke gjøre gripperen noe bedre egnet til sitt formål, og derfor ble det besluttet å heller ha en større og firkantet boks, der man kan bruke tomrommet i boksen til å gjøre eventuelle fremtidige endringer.



(a) (b) (c) (d)

Figure 21: Første versjon av hovedboksen

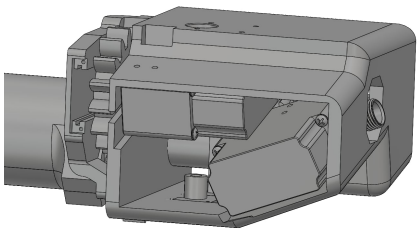
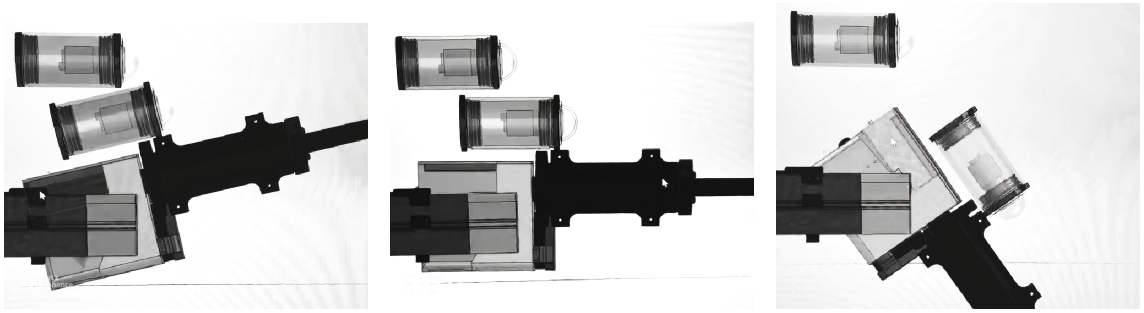


Figure 22: Første versjon av hovedboks med komponenter

På dette utkastet av hovedboksen er det, som illustrert i Fig 22, minimalt med klaring mellom komponentene, og man har ikke mye rom for forbedringer og forflytninger. Kamera som skulle festes på toppen av boksen var betraktelig større enn forventet, og gjorde det vanskelig å rotere skulderen. Som illustrert i Fig 23a kolliderer kameraene ved en liten rotasjon oppover. Det var kun 10mm fra toppen av kameraet festet på gripperen til undersiden av kameraet festet til ROV. Dette resulterte i en ekstremt begrenset rotasjon både opp og ned.



(a) (b) (c)

Figure 23: Kamera festet i drone for nær kamera på gripper

Det ble undersøkt å flytte festene til akslene opp på hovedboksen slik at det ville blitt mer plass mellom kamera på gripper og øvre kamera. Ved for eksempel å flytte festet 10mm, ville forbedringen kun tilsvart noen grader ekstra rotasjon. Dette var ikke tilstrekkelig for å nå kravet om over 45 grader rotasjon oppover. Derfor ble det laget en helt ny hovedboks som tok i betraktning

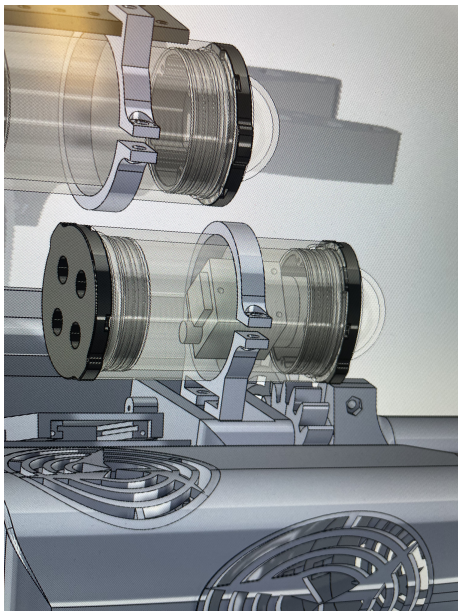


Figure 24: Kamera krasj på ROV

oppdragsgivers nye ønsker og det som ble lært fra den første boksen, hvor dybden på boksen ikke trengte å være så kort som tidligere. Dermed var det plass til å plassere servoene på rad i lengderetning. I tillegg til å få plass til to litt mindre tannhjul innenfor boksen slik at ingenting stikker utenfor.

Først ble servoene plassert inntil boksens yttervegg. Dermed kunne boltene skrus rett inn fra yttersiden. På siste hovedboks ble det heller laget hull i boksen hvor det var plass til både verktøy og bolter. Vegger til servofestet ble plassert inni boksen for en mer fleksibel servoplassering, og et generelt enklere design.

2.8 Fremtidige forbedringer

Basert på testing er det flere aspekter ved prosjektet som kan forbedres, men som ikke er implementert på grunn av tids- og ressursbegrensninger. Noen av disse beskrives i følgende avsnitt.

2.8.1 Planetgir i skulderakse

Hovedboksen er nå designet for å gi plass til flere komponenter som kan være nødvendig for forbedring. Et eksempel på en slik forbedring er et planetgir plassert i hovedboksen slik som vist på Fig 25. Dette ville kommet i tilegg til wormgearet, og kunne gitt en enda større utveksling. Dette er ikke lagt til etter oppdragsgivers forespørsel, ettersom de ønsket en mindre kompleks del, med mye intern plass for andre fremtidige utvidelser.

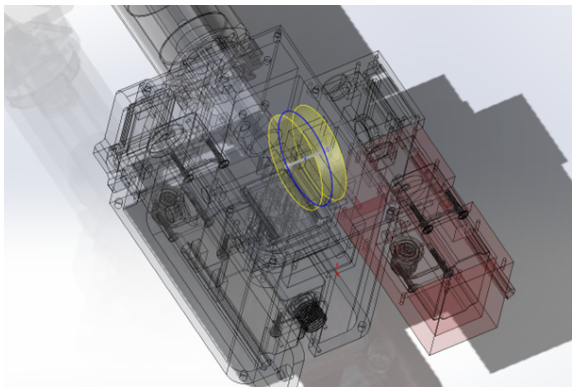


Figure 25: Område tilgjengelig for planetgir

2.8.2 Reduksjon av armlengde

En måte å øke styrken på armen på er å forkorte lengden. En halvering av armlengden ville ført til dobling i knekkpunkt og løftekraft, siden dreiemoment er proporsjonalt til lengden på armen som utsettes for påkjenningen.

Nå med en armlengde på 300mm, tåler den opp mot 5,5kg. En halvering i armlengde ville resultert i en styrke på 11kg. Maks løftekreft basert på motor ville også blitt dobblet fra 1,5kg til 3kg. Armens lengde er ikke redusert etter vortex sine ønsker, for at kloen skal være tilstrekkelig langt ut fra resten av dronen. Den er fremdeles mer enn sterk nok til å utføre oppgavene sine. Fig 26 gir et inntrykk av forholdet mellom lengden på dronen, thrusterene og selve armen.

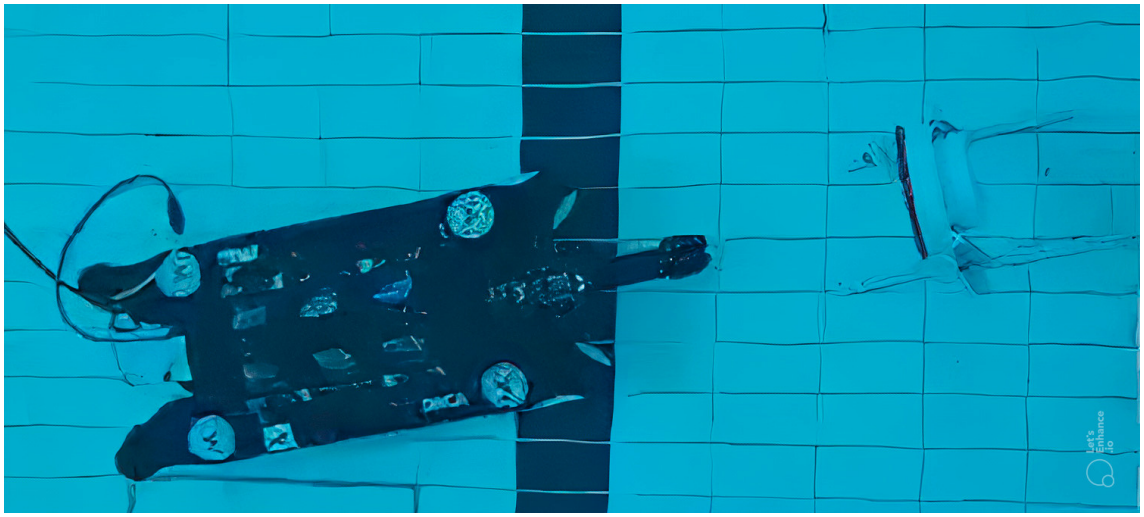
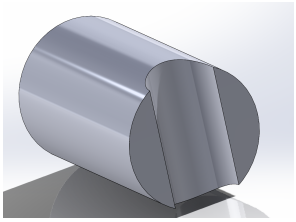


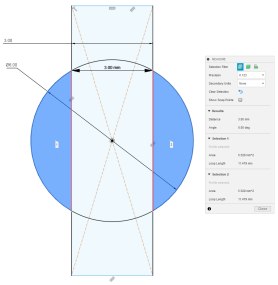
Figure 26: Drone ovenifra

2.8.3 Svake punkt

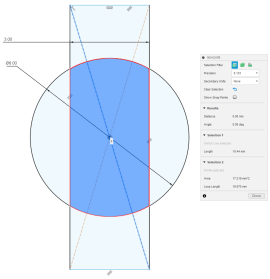
Det svakeste punktet i konstruksjonen, bekreftet av destruktiv test, er skulderakselen. Som vist i Fig 27a kommer dette av innfestingsmekanismen mellom wormwheelet og akselen, og flensen og akselen. Arealet i dette området er $11mm^2$, som vist i Fig 27b. Innfestingen utgjør som vist i Fig 27c et tap på $17mm^2$, som er en 61% reduksjon i styrke.



(a) Tverrsnitt av svakeste punkt i konstruksjonen



(b) Areal i svakeste punkt



(c) Tapt areal i svakeste punkt

Systemet vil bli mer robust av en økning i skulderfestets akseldiameter. Nå brukes en 6mm aksel, men styrken kan økes til nærmest det dobbelt ved å øke akselens diameter til 8mm. Dette fordi styrken i en aksel vil øke proporsjonalt med arealet, som øker med kvadratet av radiusen. Som vist i Formel 1 og Formel 2 gir dette en økning på 1,78. Dette vil derimot kreve en erstatning av det eksisterende wormwheelet, siden det ikke har muligheter for alternativ innfesting.

$$Areal(6mm) = 3^2\pi = 9\pi \tag{1}$$

$$Areal(8mm) = 4^2\pi = 16\pi \tag{2}$$

3 Elektronikk

Hovedfokuset ved design av elektronikken var stabilitet og et robust system, som ved feil i systemet ikke ødelegger komponenter. Det var ønskelig at systemet delvis vil fungere om det er feil på et annet sted i systemet. For å løse dette trengs det kontroll over strømforbruk på servomotorer og mulighet til å skru av forsyningstrømmen til hver enkelt servo. Det trengs også posisjonsmåling på de forskjellige aksene, for å unngå selvkollisjon og for å ha oversikt over posisjonen til manipulatoren. Dette blir svært viktig i fremtiden, ettersom Vortex ønsker å utvikle manipulatoren til å bli autonom.

3.1 Servomotorer

En av de viktigste komponentene for å kontrollere manipulatoren er servomotoren. Moterene måtte være sterke nok til til å kunne bevege armen, samt rotere en ventil under vann. Etter undersøkelser kunne Blue Trail Engineering levere undervannsservoer som er vanntette ned til 200 meter [11]. Se Fig 28. Ved å velge en ferdig løsning på servoene er man sikret at de er vanntette. I motsetning ville det å designe et vanntett skall selv, krevd omfattende testing under vann.

Figure 28: Underwater Servo SER-2020 [11]

Det er valgt servoer av typen "Underwater Servo SER-2020" som har et dreiemoment på 34 kgf-cm, som i SI-enheter tilsvarer 3,3 Nm [11]. Dette er nok dreiemoment til alle aksene, men det ble i tilegg økt ved bruk av gir. Servoene er reprogrammerbare, og kan konfigureres for enten posisjons eller hastighetsstyring. Siden servoenes interne enkodere er inkrementelle vil servoen derimot ikke huske sin posisjon etter strømbrudd/utkobling av releer. Av denne grunn må det uansett brukes eksterne enkodere, og derfor er servoene konfigurert for hastighetsstyring.

3.2 Sensorer

Siden dronen skal operere under vann, og derfor ikke være synlig for operatøren, er det nødvendig å kunne bestemme dens fysiske tilstand gjennom software. Følgende avsnitt beskriver sensorene som brukes for å måle disse verdiene.

3.2.1 Enkodere

Enkodere måler vinkelen rundt en rotasjonsakse. Det finnes to typer enkodere, inkrementelle og absolutte. En inkrementell enkoder måler endring i posisjon, mens en absolutt enkoder måler posisjonen direkte. Ved å bruke absolutte enkodere, har man kontroll på posisjonen selv etter strømtap. [3]. I dette prosjektet var det viktig å vite posisjonen til leddene etter strømtap, fordi det ville skapt mye ekstraarbeid å måtte kalibrere posisjonen etter oppstart. Det er derfor blitt brukt absolutte enkodere som vist i Fig 29.

Disse magnetiske enkoderne bruker hall-effekten , og måler endringer i retningen til magnetfeltet. De har høy presisjon, med en 12-bits oppløsning, som betyr 4096 posisjoner på en full rotasjon. Sensoren er kontaktløs, og egner seg derfor godt til bruk under vann [10].

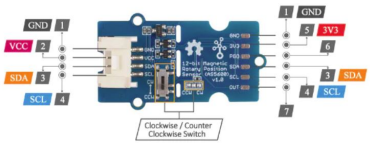
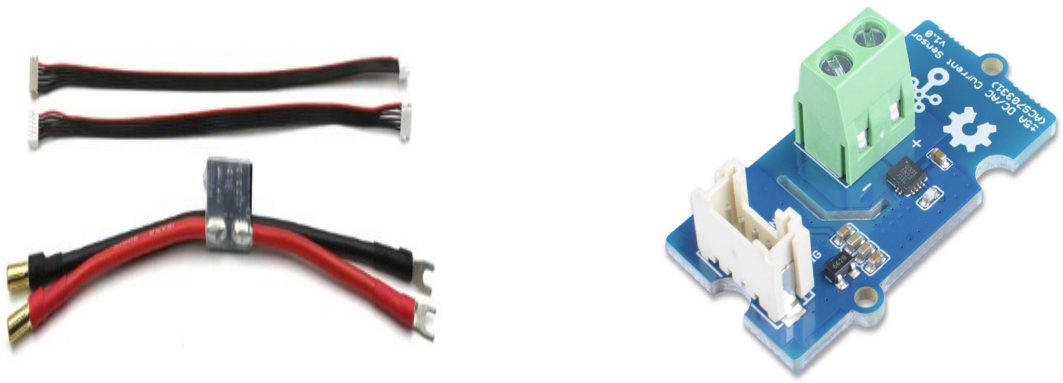


Figure 29: Magnetic Rotary Encoder. [10]

3.2.2 Strømsensorer

Det er blitt brukt sensorer for å kontrollere om strømmen holder seg innenfor normalen. Det er en strømsensor per servo, som gjør at man kan regne ut hvor mye kraft hver enkelt servo gir ut. Dette kan igjen brukes til å oppdage om armen er under mekanisk belastning. Om strømforbruket går over 2 Ampere på en servo i over 2 sekunder, kobles forsyningsstrømmen ut.

Vi benytter to typer strømsensorer, PSM-ASM-R2-RP [22] og ACS70331 [31]. Disse kan ses i Fig 30a og Fig 30b. Sensorene ble valgt på grunn av at Vortex hadde de tilgjengelig, og ønsket ikke gå til innkjøp av flere sensorer om disse kunne brukes. Begge sensorene passet til måleområdet, som var 0 - 2,7A ved 0 - 7,4V. De har relativt lik virkemåte og baserer seg på å måle endringer i magnetfeltet når det flyter en strøm gjennom.



(a) PSM-ASM-R2-RP. [22]

(b) ACS70331. [31]

Figure 30: Strømsensorer

3.3 Rele

Et 4-kanals relé brukes for å kutte hovedstrømmen til servoene ved feil. Som nevnt over kuttes strømmen om servoene trekker for mye strøm over tid. Releet fungerer ved å gi et lavt inngangsignal på den tilhørende inngang for å gi strøm til en servo [32]. Det er blitt brukt "Normally Open" utgangen, som betyr at det ikke kan gå strøm hvis styrestrømmen ikke er koblet til. Se Fig 31.

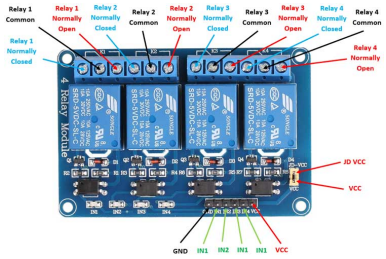


Figure 31: 4Ch-Relay. [32]

3.4 Mikrokontroller

All elektronikk overvåkes og styres ved bruk av Raspberry Pi Pico. Pico er en mikrokontroller som er veldig lett tilgjengelig. Den har nok prosessorkraft og minne til formålet, og har også nok inn- og utganger til å styre og overvåke all elektronikken til manipulatoren. Pico støtter Inter-Integrated Circuit (I2C) og Pulse Width Modulation (PWM) som brukes til å sende data melllom enheter.

3.5 Styresignal til servoer

For å kontrollere servoene sendes det ett PWM signal til hver servo. Hastigheten til den enkelte servo justeres etter bredden på pulsen. Dette ble etter testing funnet ut at servoene reagerer på en pulsbredde mellom 1300μs og 1700μs. Ved PWM-signal med en pulsbredde 1500μs står servoen helt i ro. Ved pulbredde 1700μs kjører den i maks fart med klokken, og ved 1300μs i maks fart mot klokken.

3.6 I2C

I2C er en kommunikasjonsprotokoll som kan brukes til å koble sammen mange komponenter. Den består av to kabler, klokke (SCL) og data (SDA). Klokken gir frekvensen dataen sendes med og selve dataen blir sendt via SDA kabelen. [30]. Dette er valgt fordi det krever få kabler i forhold til andre protokoller (SPI), og fordi man enkelt kan utvide tilkobling til flere enheter. Vortex bruker allerede I2C kommunikasjon til andre deler av dronen, så integrasjon og feilsøking blir lettere for dem.

3.7 I2C Oppsett

3.7.1 Raspberry Pi til Raspberry Pi Pico

Vortex bruker en Raspberry Pi til å kontrollere og kommunisere med resten av dronen. Det var derfor naturlig at den ble brukt til kommunikasjon til Raspberry Pi Pico. I2C er satt opp på Pico, som en slave med adresse ”30”.

3.7.2 Pico til enkodere

For å lese sensordata fra enkoderne, brukes også I2C kommunikasjon. I2C adressene på enkoderne var satt fra fabrikk, og disse var like til alle enkoderne. Dette kunne blitt løst ved bruk av multipleksing, men ble ikke brukt fordi da måtte Pico byttet på å være master og slave. Dette betyr at sensordata ikke kunne vært sendt kontinuerlig, noe som er ønskelig. Derfor ble dette løst ved å bruke et eget I2C nettverk for enkoderne. Dette er satt opp i programvaren ved bruk av biblioteket SoftI2C [29], som gjør at man kan bruke andre pins enn hardware I2C for kommunikasjon. Det er satt opp en SDA pin til hver av enkoderne, mens SCL er felles for alle tre enkoderne.

4 Software

For å kunne styre elektronikken på en trygg og intuitiv måte er det implementert en omfattende software stack. Dette inkluderer firmware, som snakker direkte med elektronikken, og en standardisert ROS2 stack for både manuell og automatisk styring. All kode ligger i GitHub repoet til prosjektet [19].

4.1 Hvorfor ROS

ROS er en industristandard for automatiserte systemer, og har vært i utvikling siden 2007. Det er et rammeverk som har blitt utviklet av mange leverandører og et stort økosystem. Dette resulterer i mange ferdige løsninger som kan brukes til å bygge komplekse systemer raskt og effektivt. Ved å bruke ROS sikres en arkitektur som er godt kjent og enkel å vedlikeholde og videreutvikle.

ROS2 har mange forskjellige distribusjoner. Under dette prosjektet sier den offisielle ROS2 dokumentasjonen at "Iron Irwini" er den nyeste, men at den har EOL (End of Life) i November 2024 [23]. Derfor benyttes i stede "Humble Hawksbill", som har EOL i Mai 2027.

4.2 Sentrale ROS2 konsepter

4.2.1 Nodes

En ROS2 node er et lite program som kommuniserer med andre ROS2 noder gjennom Topics, Services og andre ROS2 rammeverk. Formålet med noder er å isolere konkrete funksjoner fra resten av systemet å gjøre koden mer gjenbrukbar og enklere å teste [25]. En node kan subscribe til og publishe flere topics, snakke med og hoste flere services, og har nøyaktig en parameterserver.

4.2.2 Topics

ROS2 noder kommuniserer mellom hverandre i hovedsak gjennom topics. Et topic er en navngitt kanal hvor meldinger av en konkret form sendes av en eller flere noder og mottas av en eller flere andre noder. Topics egner seg til kommunikasjon som oppdateres ofte, for eksempel hastighetskommandoer, eller som skal deles ut til mange subscribers som for eksempel systemtilstander. Topics følger en push modell, hvor noder som subscriber til et topic venter på at en annen node publisher en melding. Dette illustreres i Fig 32. Det er ikke mulig for en subscriber å etterspørre informasjon. Dette er en av grunnene til at ROS2 også har services [28].

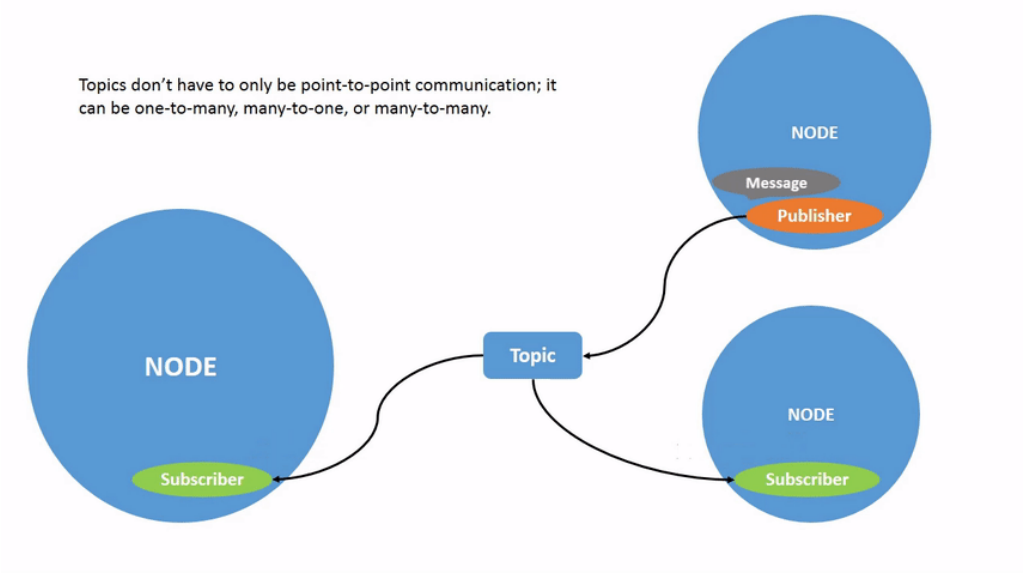


Figure 32: Illustrasjon av Topics fra den offisielle ROS2 dokumentasjonen [28]

4.2.3 Services

ROS2 Services er en annen måte å kommunisere med ROS2 noder på. Topics er ment til å overføre hyppige meldinger, men gir ingen tilbakemelding på resultatet. En tjeneste er ment til mer sjelden bruk, og gir en tilbakemelding på en handling. I forbindelse med manipulatoren blir tjenester brukt til å aktivere og deaktivere releene på hver enkelt servo, og for å kvittere alarmtilstander på hardware. En ROS2 node kan ha flere tjenester, samtidig som den også har andre noder. Tjenester følger en pull modell, hvor en node ved behov kan etterspørre informasjon eller en handling fra en annen node [27]. Dette illustreres i Fig 33.

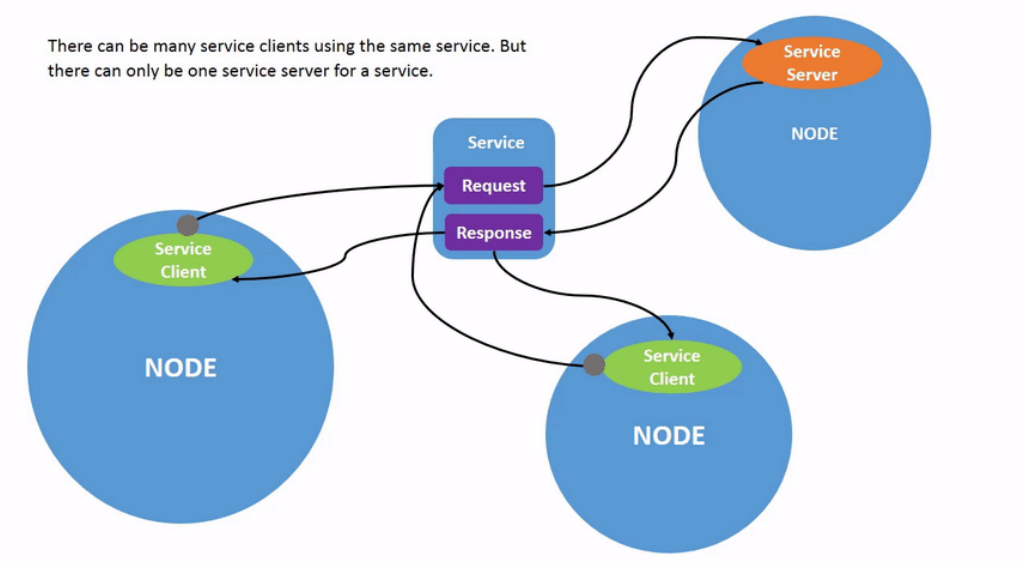


Figure 33: Illustrasjon av Services fra den offisielle ROS2 dokumentasjonen [27]

4.2.4 Parameter Server

ROS2 har en standardisert parameter server service innebygd i alle noder, som kan brukes til å redigere parametre i sanntid. Dette er spesielt nyttig til for eksempel PID kontrollere, hvor det er ønskelig å kunne justere på parametre uten å måtte starte systemet på nytt [26]. Alle parametre kan til slutt lagres i konfigurasjonsfiler og legges ved i den ferdige ROS2 pakken.

4.3 Software arkitektur

Systemet består av en grensesnittdatamaskin som kjører visualiserings og joystick programmvare, en Raspberry PI som kjører selve dronens kontrollsystem, og en Raspberry Pi Pico, som fungerer som et grensesnitt mellom dronens Raspberry Pi og all hardware i gripperen. Fig 34 illustrerer hvordan informasjon flyter gjennom systemet. De to blokkene i Fig 34 som heter "ROS2 Nodes" refererer til et nettverk av mindre programsnutter som snakker med hverandre ved å utveksle informasjon gjennom topics. Noen av disse konseptene er kort oppsummert i andre kapitler, men for en dypere forståelse bør den offisielle ROS2 dokumentasjonen benyttes [24] [17] [16]. Det er laget to forskjellige slike systemer til dette prosjektet. Det er en ROS2 Control stack som inneholder kinematikk, baneplanlegging og regulatorer, og en diagnostikk stack som kun sender enkle kommandoer direkte til hardware. Diagnostikk noden fungerer som en backup i tilfelle noen av komponentene som kreves for regulatorbasert posisjonering blir skadet under operasjon, eller som en mer innsiktsgivende og gjennomsiktig plattform for feilsøking som ikke angår ROS2. For å gjøre diagrammer som angår konkrete software funksjoner mer oversiktlige vil noe informasjon fra Fig 34 utelates.

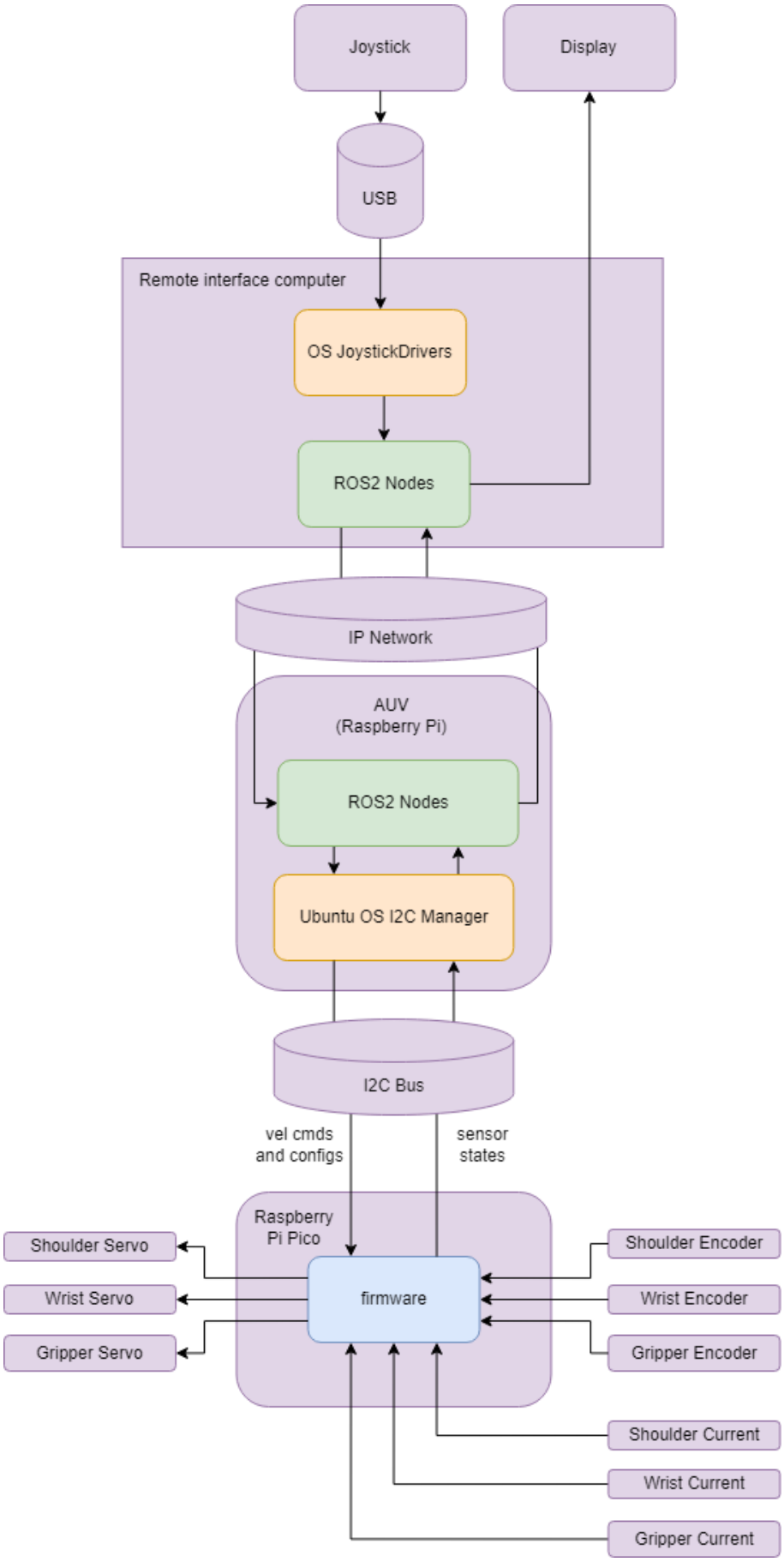


Figure 34: Diagram over software arkitektur

4.3.1 ROS2 Control stack

Fig 35 illustrerer hvordan informasjon flyter gjennom ROS2 Control stacken. Grønne bokser representerer standardiserte ROS2 Noder, rød representerer standardiserte ROS2 Control plugins, blå representerer unik C++ kode skrevet for dette prosjektet og lilla representerer fysisk hardware.

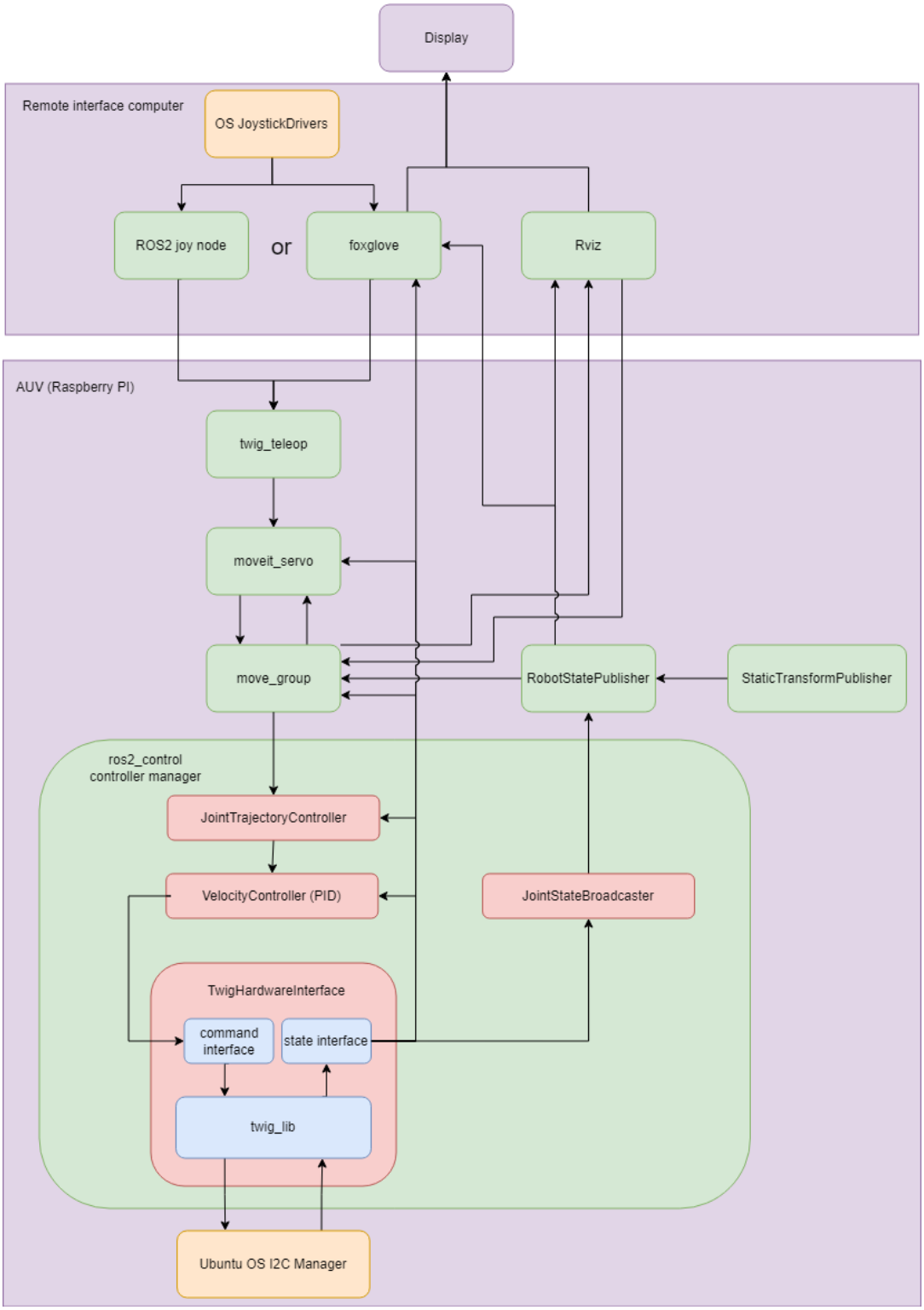


Figure 35: Diagram over ROS2 Control stack

4.3.2 Diagnostikk stack

Fig 36 illustrerer hvordan informasjon flyter gjennom diagnostikk stacken. Fargekodingen er den samme som for kapittel 4.3.1.

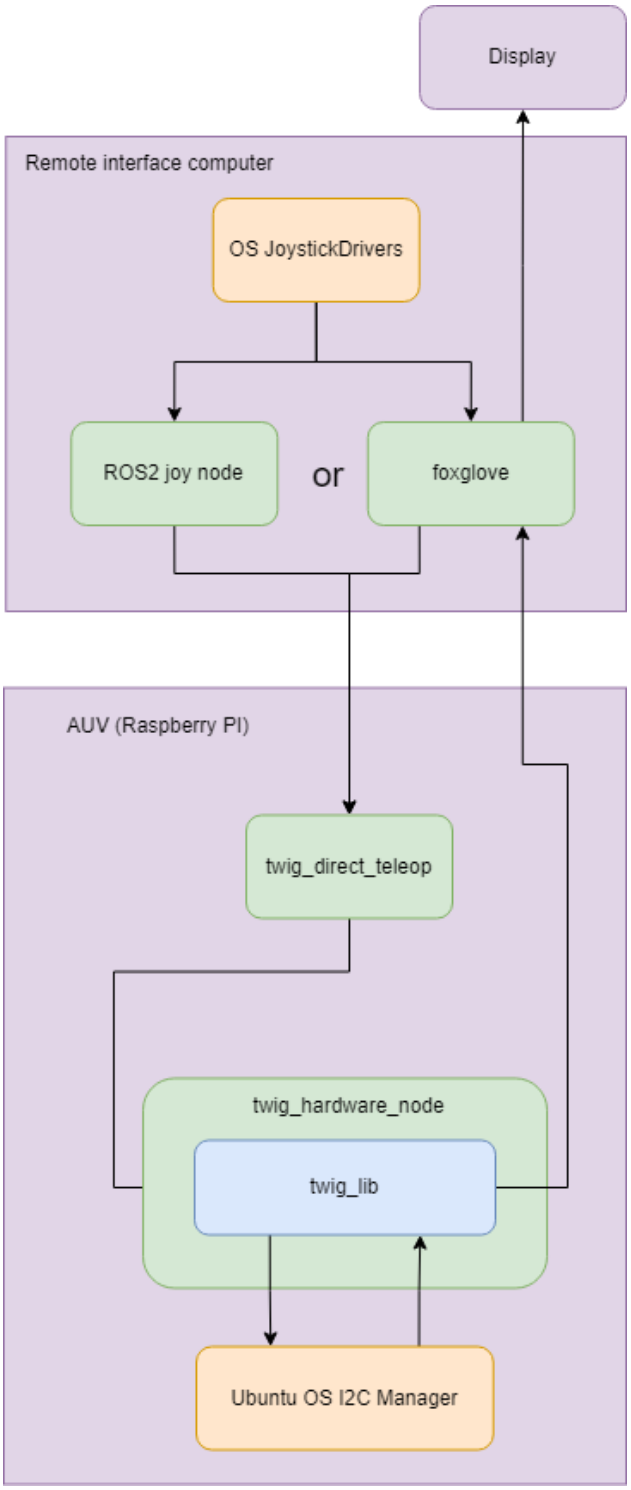


Figure 36: Diagram over diagnostikk stack

4.4 Standardiserte ROS2 pakker brukt i prosjektet

ROS2 økosystemet inneholder en rekke standardiserte pakker som løser problem som dukker opp i de fleste robotikkapplikasjoner. Nedenfor beskrives pakkene som ble tatt i bruk i dette prosjektet.

4.4.1 ROS2 Control

ROS2 Control er en ROS2 pakke som håndterer hardware drivere og kontrollere. Denne pakken er designet for å være en standardisert måte å kommunisere med hardware på. Dette oppnås ved å isolere all kode som snakker med fysisk hardware i "Hardware Interfaces".De tre mest vanlige grensesnittene: Actuator, Sensor og System. PID kontrollere og andre kontrollere kan deretter implementeres i "Controller Interfaces", som sammens med "Hardware Interfaces" kombineres basert på standardiserte konfigurasjonsfiler.

4.4.2 Mocked Components

ROS2 Control handler i hovedsak om å isolere ROS2 utvikling fra hardware. For å kunne simulere eller teste ROS2 stacken uten å ha en fysisk robot tilgjengelig, inneholder ROS2 Control pakken "Mocked Components". Disse er generiske implementasjoner av actuator og sensor hardware interfaces. Dette muliggjør grundig utvikling og testing av for eksempel PID kontrollere og avanserte navigasjonssystemer i Docker og Gazebo, uten at det fysiske systemet må være involvert. Mocked Components vil i praksis erstatte TwigHardwareInterface i Fig 35.

4.4.3 Robot State Publisher

Robot State Publisher (RSP) er en node som bruker robot beskrivelsen (URDF) fra twig_description til å beregne fremoverkinematikken til roboten basert på /joint_states topicet som publiseres av "joint_state_broadcaster/JointStateBroadcaster" fra ROS2 Control som er konfigurert i twig_moveit_config. Dette er en viktig node for Moveit2, som trenger å vite robotens posisjon og orientasjon for å kunne planlegge baner. TF2 kan bruke denne informasjonen til å beregne relative transformasjoner mellom forskjellige koordinatsystemer.

4.4.4 Tf2

TF2, eller Transform 2, brukes til å regne ut transformasjonene mellom forskjellige koordinatrammer. Dette blir nyttig for oppdragsgiver i senere tid, når systemet skal bli helautomatisk, og manipulatoren skal flyttes til en posisjon basert på ekkolokasjon eller video.

4.4.5 Moveit2

Moveit2 brukes for å generere baner for roboten. Dette blir ekstra viktig for oppdragsgiver i senere tid når dronen skal automatiseres fullstendig, og derfor må kunne planlegge alle bevegelser på egenhånd.

4.4.6 Moveit Servo

Moveit Servo er en ROS2 pakke som genererer posisjonskommandoer til Moveit2 basert på en ønsket leddhastighet. På denne måten kan en operatør manuellt styre roboten gjennom det samme systemet som ved helautomatisk operasjon. Fordelen ved å "jogge" individuelle ledd gjennom dette systemet fremfor en enkel hjemmesnekret ROS2 node er at Moveit Servo også inkluderer automatisk singularitets, endepunkts og objekt unngåelse. Dette skjer ved å eksponensielt redusere leddhastighetene når roboten nærmer seg en uønsket tilstand. Moveit Servo gjør systemet derfor betydelig mer operatørvennlig på en måte som ellers hadde tatt ekstremt mye tid å implementere på egenhånd.

4.4.7 Rviz2

Rviz2 benyttes for å visualisere og teste robotens Moveit2 og Moveit Servo stack. Grunnet et simplistisk design og mangel på Windows støtte er dette verktøyet derimot ikke godt egnet under normal operasjon. Fig 37 illustrerer hvordan Rviz2 kan brukes for å visualisere robottilstanden og planlegge baner med Moveit2.

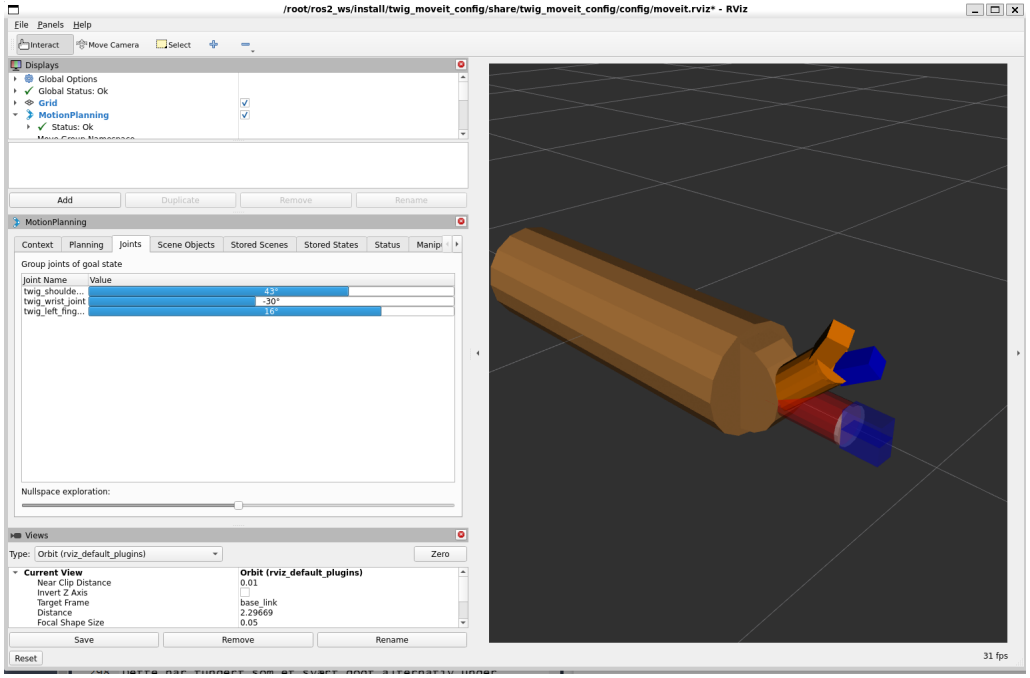


Figure 37: Rviz2 som kjører Moveit2 plugins

4.4.8 Foxglove

Foxglove er et verktøy for å lage dashboards for blant annet ROS2. Dette fungerer som et mer brukervennlig alternativ til Rviz2, og egner seg bedre til å designe operatørpanel. Til gripperen brukes Foxglove i hovedsak til å publisere joystick data fra spillkontrollere til ROS2. Dette har fungert som et svært godt alternativ under utvikling i Docker, siden det ofte er vanskelig å få tilgang til USB enheter i Docker gjennom WSL, som diskuteres i 4.8.1 og 4.8.3. Foxglove kan også brukes til å visualisere data fra ROS2, og for å sette ROS2 parametre under testing. Fig 38 illustrerer hvordan Foxglove kan brukes for å publisere joystick meldinger og overvåke robotens fysiske tilstand og statuslogger.

Figure 38: Foxglove dashboard

4.5 ROS2 pakker utviklet for dette prosjektet

I dette prosjektet er det ønskelig å følge industristandarder i høyest mulig grad. Derfor består prosjektet hovedsakelig av konfigurasjoner som benytter eksisterende løsninger, med unntak av controller mappings i twig_teleop og en egen diagnostikk og feilsøkings node i twig_hardware.

4.5.1 twig

Denne pakken inneholder launch filer for å starte systemet i forskjellige konfigurasjoner. Pakken fungerer som et felles grensesnitt for oppstart av manipulator stacken, og brukere av manipulatoren skal utenom utviklingssammenhenger ikke trenge å forholde seg til de andre pakkene direkte. Under er en liste av de forskjellige launch filene i twig pakken.

Normal operasjon

robot_moveit.launch.py Starter alle pakker relatert til hardware og moveit. Med andre ord alle andre pakker enn de som er relatert til GUI og user input. Denne noden må kjøres på selve roboten.

controller_manager.launch.py Starter en ROS2 Control Controller Manager node konfigurert av twig_moveit_config pakken. Denne noden må kjøres på selve roboten. Launchfilen er en del av robot_moveit.launch.py og trenger ikke å kjøres manuelt.

servo_teleop.launch.py Starter Moveit Servo og teleop noden som trengs for å konvertere joystick input til leddhastigheter. Denne noden kan kjøres hvor som helst.

foxglove.launch.py Starter en normal Foxglove bridge, som trengs for at Foxglove applikasjonen skal kunne snakke med ROS2 systemet. Den er gjort tilgjengelig gjennom twig pakken for enkel installasjon og tilgang. Denne noden kan kjøres hvor som helst.

joy.launch.py Starter en normal joy node konfigurert for å publisere joystick meldinger på /twig_joy topicet. Denne noden må kjøres på samme PC som joysticken er tilkoblet.

rviz.launch.py Starter en konfigurert Rviz node. Denne kan være nyttig for å visualisere den interne tilstanden til roboten. Noden vil kun fungere med ROS2 Control stacken, siden diagnostikk stacken ikke publiserer de nødvendige transformasjonene og sensortilstandene. Noden må kjøres på en PC med skjerm.

Diagnostikk

diagnostics.launch.py Starter en diagnostikk node som kommuniserer direkte med hardware, og en teleop node for joystick kontroll. Den kan ikke kjøres parallelt med andre launchfiler som bruker hardware, og er derfor kun tilgjengelig for diagnostikk og feilsøking. Formålet med denne noden er å gjøre tilgjengelig all informasjon til feilsøking, som ellers ville vært overflødig for normal operasjon. Noden fungerer også som en manuell backup løsning i tilfelle software feil skulle skapt problemer i det større systemet. Noden må kjøres på selve roboten, og krever at en annen node publiserer joystick data på twig_joy topicet.

Utvikling uten hardware

moveit_demo.launch.py Starter en ren software demo node med Foxglove, Rviz, Moveit, Moveit Servo og en teleop node. Formålet med denne launch filen er å gi en enkel måte å teste ROS2 stacken uten hardware.

4.5.2 twig_description

Denne pakken inneholder en URDF fil, som beskriver manipulatorens fysiske form, joint typer og joint limits. Denne beskrivelsen benyttes av blant annet Moveit2 og Tf2 for å regne ut transformasjoner og baner for roboten.

4.5.3 twig_moveit_config

Denne pakken inneholder en konfigurasjon generert av Moveit Setup Assistant, og brukes til å starte Moveit stacken. Den inneholder bland annet flere konfigurasjoner for baneplanlegging, en kollisjonsmesh basert på robotbeskrivelsen i twig_description, TF2 noder, kinematikk osv.

4.5.4 twig_hardware

Denne pakken inneholder hardware drivere, "Hardware Interfaces" for ROS2 Control og en ROS2 node for diagnostikk og feilsøking. I tillegg inneholder den også firmware og alle konfigurasjoner relatert til firmware, drivere og kommunikasjon mellom dem. Firmware inngår ikke i ROS2 Stacken, men er gruppert under denne noden siden firmware og drivere skrives sammen.

4.5.5 twig_servo

Denne pakken inneholder konfigurasjon for Moveit Servo noden som brukes i prosjektet.

4.5.6 twig_teleop

Denne pakken inneholder noder, som basert på joystickmeldinger sender kommandoer til twig_hardware nodene gjennom topics og services.

4.6 Firmware

Firmware inneholder kode som kommuniserer direkte med fysiske komponenter. Dette gjelder å justere servo hastighet, aktivere og deaktivere releene, lese av strømsensorer og lese av posisjon gjennom enkoderne. Firmware er skrevet i C++ og kjører i dette tilfellet på en Raspberry Pi Pico. Den er skrevet for å være så enkel, robust, generisk og konfigurerbar som mulig, for å redusere antall endringer som trengs under testing. For å isolere kode som angår de forskjellige komponentene, brukes objektorientert programmering. Fig 39 illustrerer firmware arkitekturen.

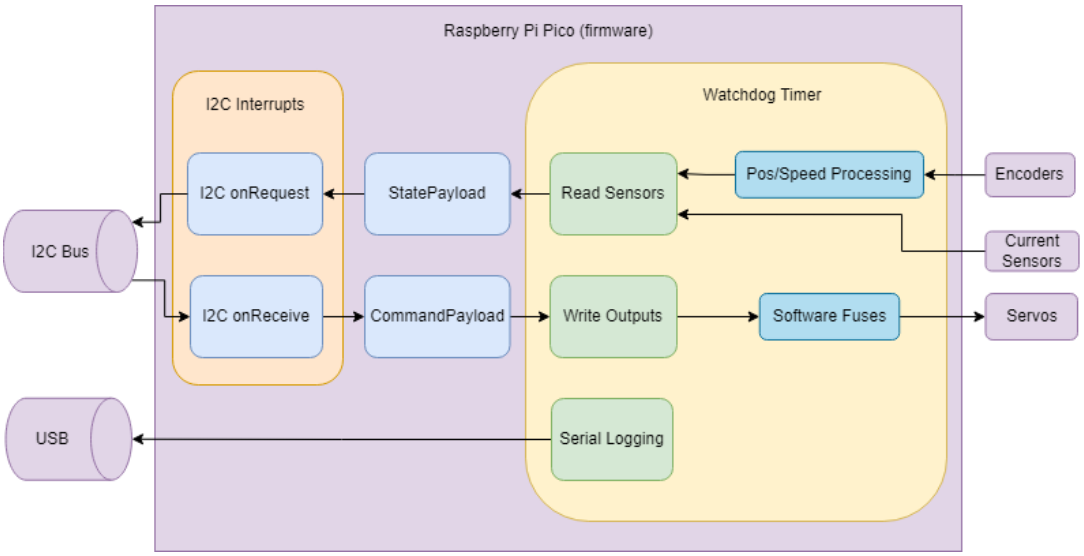


Figure 39: Firmware arkitektur diagram

4.6.1 Watchdog timer

Siden firmware skal kjøre på en undervannsdrone er den å anse som en kritisk prosess. For å forhindre at systemet blir ødelagt om firmware av hvilken som helst grunn fryser, benyttes en watchdog timer som fullstendig restarter mikrokontrolleren etter en bestemt tidsperiode. Dette er en enkel og effektiv måte å sikre mot feiltilstander som kan oppstå ved kommunikasjonsfeil eller andre uforutsette hendelser. Når watchdogtimeren slår inn vil firmware umiddelbart motta nye kommandoer fra ROS2, og basert på en konfigurerbar oppførsel, enten varsle brukeren gjennom loggmeldinger og fortsette der den slapp, eller deaktivere alle utganger frem til brukeren kvitterer alarmen ved å deaktivere servoreleene. ROS2 detekterer omstarten ved at mikrokontrolleren genererer en ny session id ved oppstart.

4.6.2 Session Id

Ved noen feil kan driver eller firmware starte på nytt. For å detektere denne feiltilstanden benyttes en session id, ett tilfeldig tall mellom 1 og 10 000, som genereres av firmware ved oppstart og legges ved sensor data. Fig 40 og Fig 41 illustrerer dette systemet. Driveren mottar denne id'en og sender den tilbake med alle kommandopakker. Hvis en session id ikke stemmer vil firmware ignorere kommandoen. Hvis en kommando med riktig session id ikke mottas innen en spesifisert tidsperiode vil firmware anta at driveren har krasjet og deaktivere servoene. Dette er en enkel måte å sikre mot feiltilstander, som kan oppstå ved kommunikasjonsfeil eller andre uforutsette hendelser. Kvittering av denne feiltilstanden skjer gjennom en Trigger Service på hardware noden. Ved bruk at twig_teleop noden trigges denne tjenesten ved å trykke på deaktiveringsknappen for releene. Alternativt kan autokvittering av slike tilstander konfigureres i twig_hardware noden.

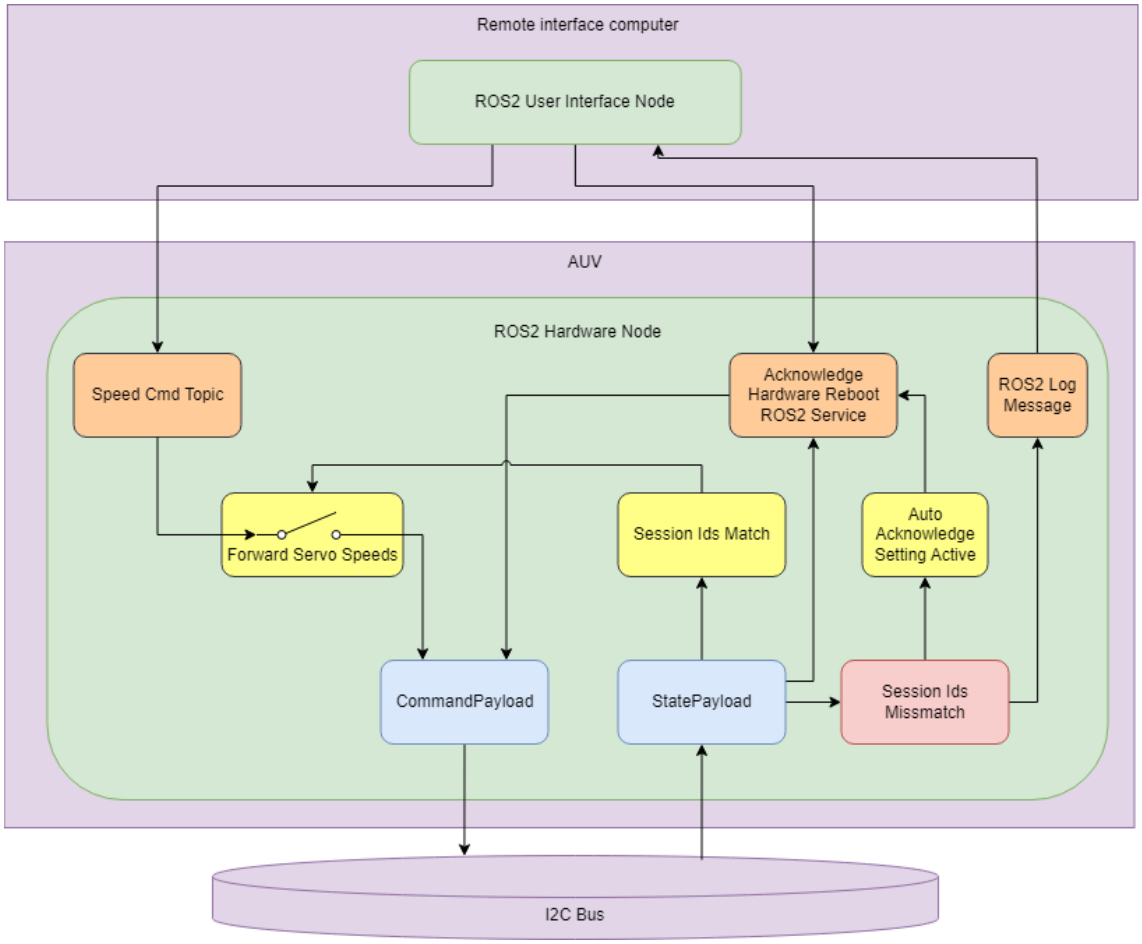


Figure 40: Øvre del av diagram over Session Id system

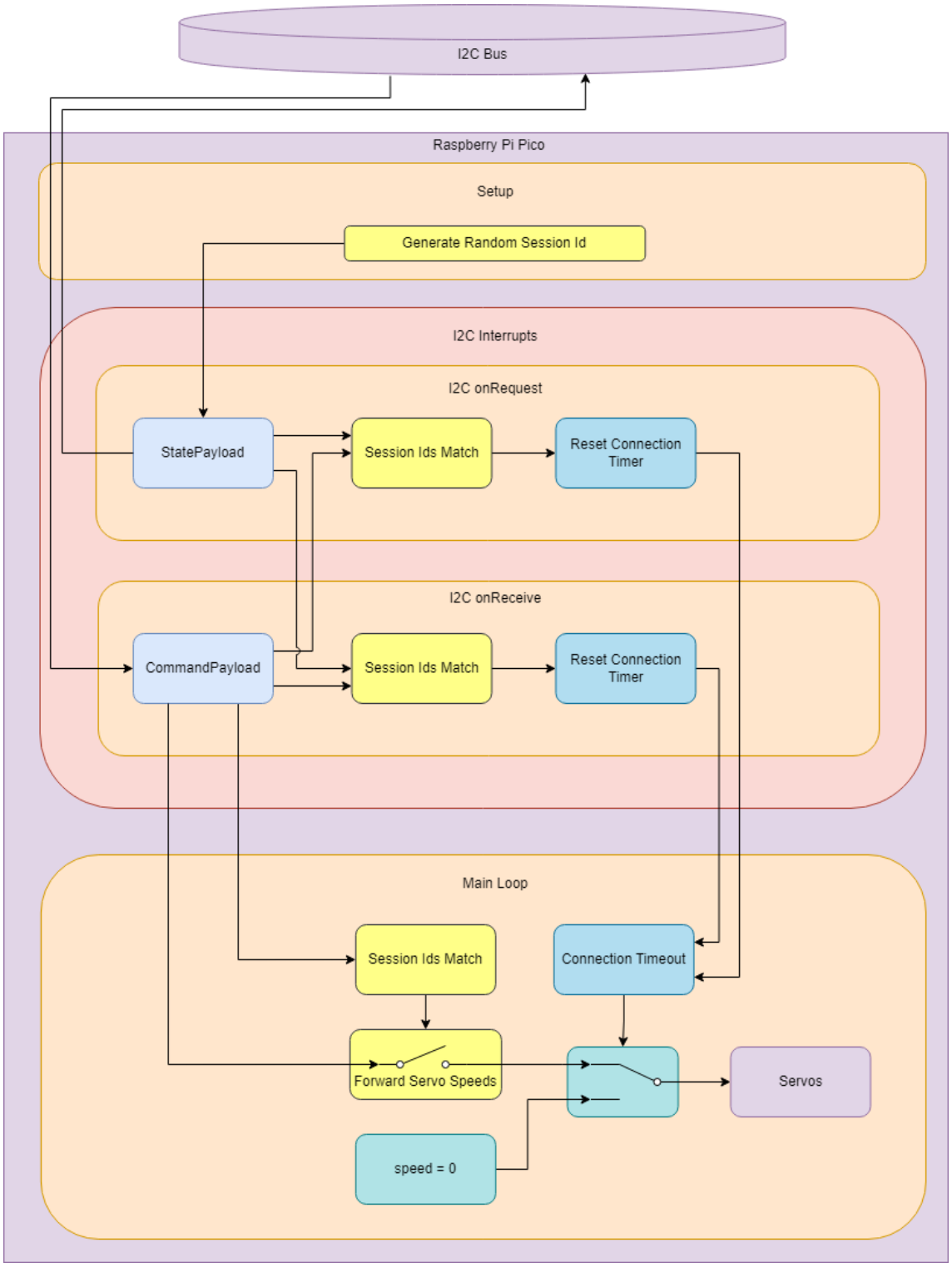


Figure 41: Nedre del av diagram for Session Id system

4.6.3 Remote Configuration

Firmware tar tid å flashe, og det er derfor upraktisk å justere parametre som maks servo strøm og utkoblingstider ved å kompilere og laste opp ny kode. For å effektivisere denne prosessen sendes alle

parametre over I2C koblingen. På denne måten kan også firmware konfigureres gjennom driverne, uten å oppdatere selve koden. Disse innstillingene ligger under twig_hardware ros noden i en yaml fil.

4.6.4 Max og Min Joint Limits

For å forhindre at manipulatoren kjører leddene mot mekaniske stopp er det satt opp grenser i software for hvor langt leddene kan bevege seg. Denne begrensningen er på plass for å sikre mot operatørfeil. Av den grunn er den implementert i selve driveren, siden det uansett er utkoblings-og overstrøms sikringer i firmware fra før av, for å sikre mot systemfeil. Dette holder kompleksiteten i firmware så lav som mulig, slik at det er enklere å feilsøke og vedlikeholde. Posisjonen måles av 360 graders magnetiske enkodere, og for å forenkle implementasjon sentreres posisjonsdata mellom + og -180 grader. Maks og minimumsvinkler bestemmes basert på denne skalaen, og driveren sender et stoppsignal om en grense overskrides med en hastighetskommando i grenseretningen.

4.6.5 Connection Timeout

Hvis firmware ikke blir lest fra eller skrevet til innen et spesifisert tidsintervall, vil releene kobles ut og servoene settes i stasjonær tilstand. På denne måten sikres systemet i tilfeller hvor Raspberry Pi fryser på grunn av annen programvare, krasjer, mister forbindelsen eller andre uforutsette hendelser. Dette er del av Session Id systemet illustrert i Fig 41.

4.6.6 Enkodere

De magnetiske enkoderene plukker opp mye høyfrekvent støy. For å beregne hastighet ut ifra dette må signalet filtreres. Derfor er det benyttet et andregrads butterworth lavpassfilter med en knekkfrekvens på 2Hz. Fig 42 illustrerer dataflyten fra enkoderene, og grafene i Fig 43 og Fig 44 sammenligner det filtrerte og ufiltrerte signalet. Merk at den filtrerte farten er derivatet av den filtrerte posisjonen. Koden for lavpassfilteret er hovedsakelig en ren kopi av koden til Curio Res [21], som lager opplæringsvideoer og biblioteker for signalbehandling i Arduino rammeverket.

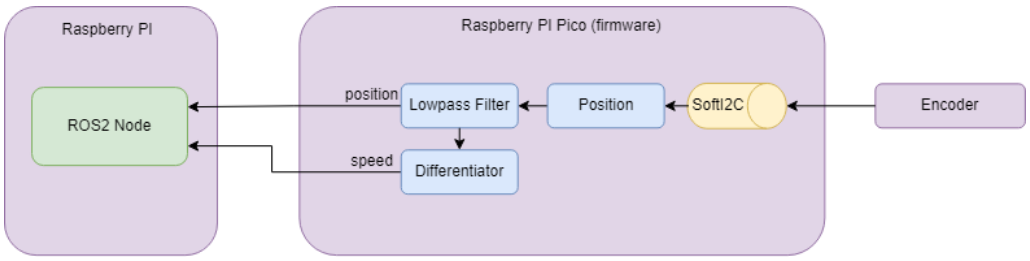


Figure 42: Diagram for dataflyt fra enkodere i firmware

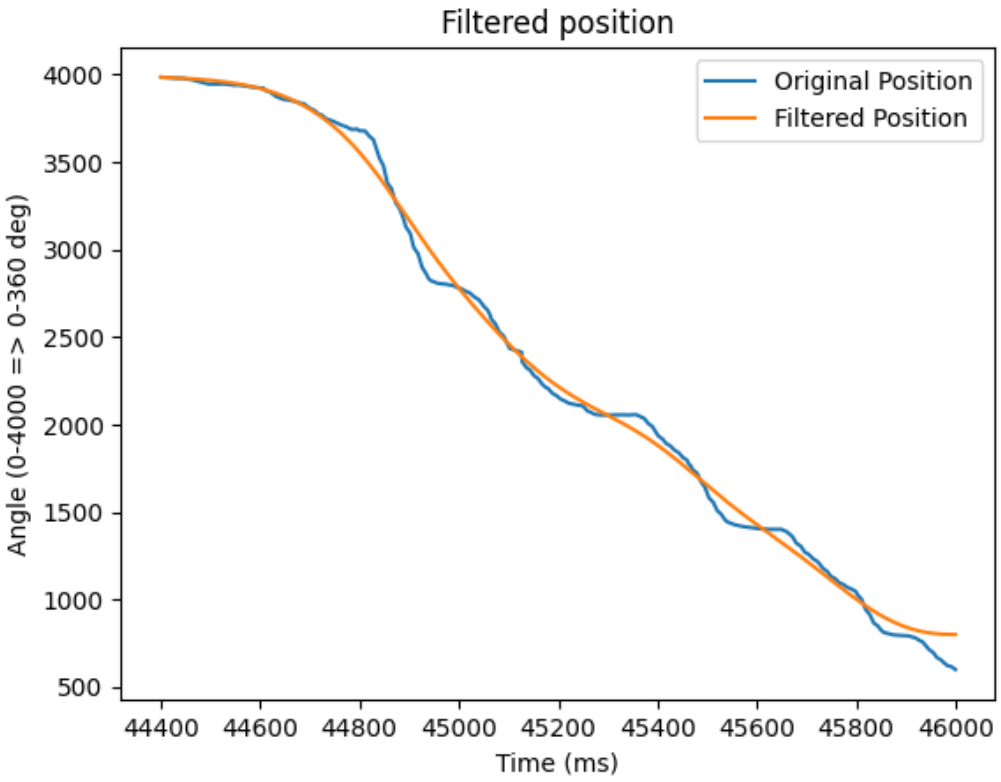


Figure 43: Filtrert og ufiltrert posisjon

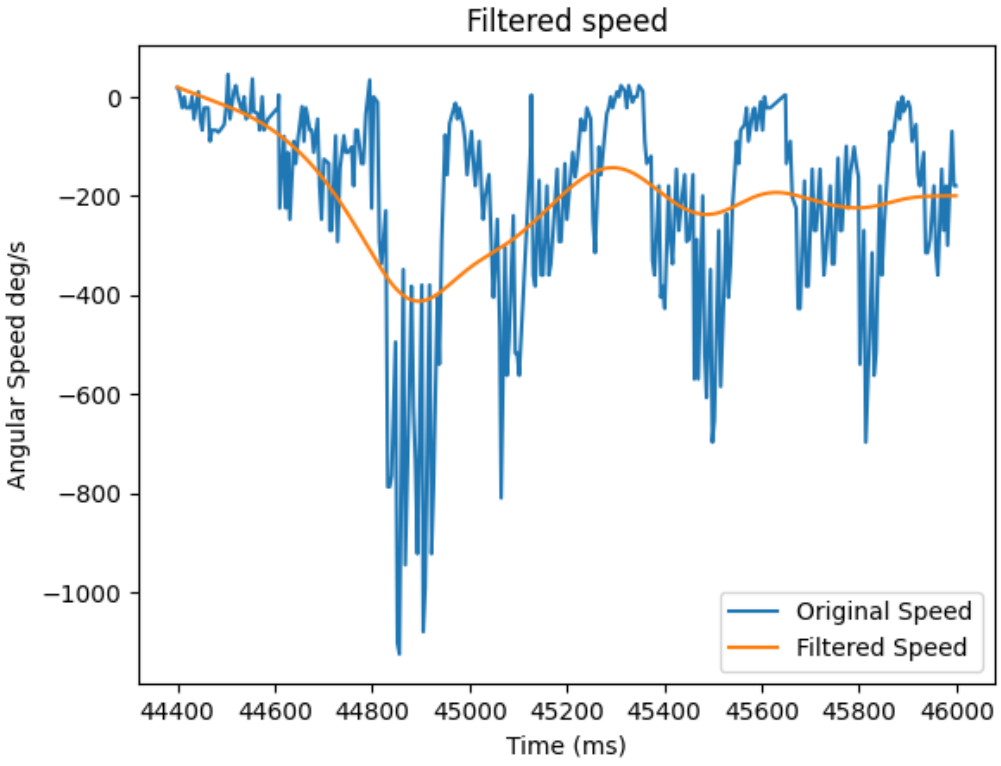


Figure 44: Enkoder fart basert på filtrert og ufiltrert posisjon

4.6.7 Software Fuse

Oppdragsgiver ønsker at servoene skal kobles ut via releer i en kort tidsperiode hvis de bruker mer enn 2 Ampere i mer enn 2 Sekunder. Dette er for å forhindre overbelastning av servoene, som kan føre til at de blir ødelagte. Systemet er illustrert av diagrammet i Fig 45.

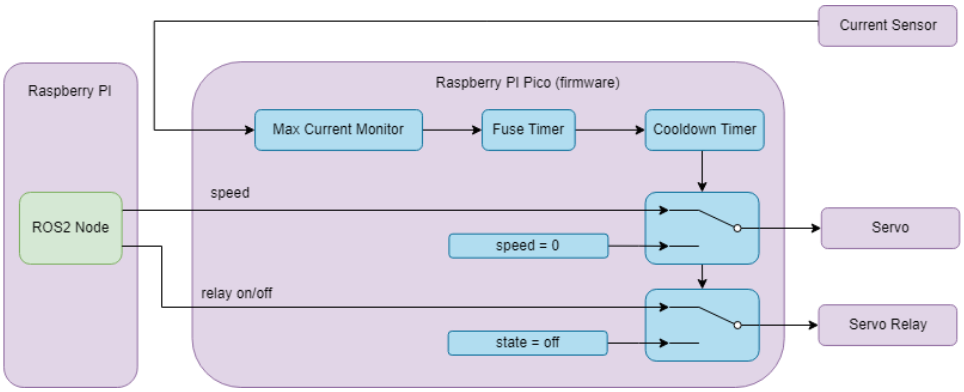


Figure 45: Diagram for Software Fuse

4.7 Raspberry Pi

ROS2 stacken kjører på en Raspberry Pi. Relatert til den gir følgende avsnitt informasjon om teknologier som er spesielt relevante for utvikling av dette prosjektet.

4.7.1 mDNS

Multicast DNS (mDNS) er en protokoll som lar enheter på et lokalt nettverk oppdage hverandre uten en DNS server. Dette er spesielt nyttig for å oppdage enheter som ikke har en fast IP adresse, som for eksempel en Raspberry Pi basert på et "hostname". Denne tjenesten er som standard installert på de fleste Raspberry PI OS distribusjoner.

4.7.2 Ubuntu

Ubuntu er operativsystemet som ROS2 er designet for. ROS2 er også tilgjengelig for andre OS, men ofte krever dette mye ekstra arbeid. Ubuntu versjonen brukt i prosjektet er "Ubuntu Linux - Jammy Jellyfish (22.04)", som også er den anbefalte versjonen for ROS2 Humble.

4.7.3 Pi Tunnel

pitunnel.com er en tjeneste som kan kobles til en Raspberry Pi for å gjøre en terminal tilgjengelig over internett. Dette gjør feilsøking enklere om det skulle være problemer med å oppdage den over mDNS.

4.8 Workspace

For å kunne utvikle ROS2 prosjekter kreves en omfattende installasjonsprossess av Ubuntu, en rekke toolchains og et utviklingsmiljø (IDE). For å unngå at denne prosessen må gjentas for alle som skal jobbe på prosjektet, er det lagt inn mye innsats i å designe et ROS2 workspace som er enkelt i bruk og tilpasset til akkurat dette prosjektet. Flere av konfigurasjonene er sterkt inspirert av prosjektet "vscode_ros2_workspace" av Allison Thackston [5]. Andre deler av konfigurasjon kommer fra den offisielle dokumentasjonen til Docker [9] og VSCode [18].

4.8.1 Docker

Docker er et system som gjør det mulig å pakke en applikasjon og alle dens avhengigheter i en container. Denne containeren kan baseres på en spesifikk distribusjon av Linux, og kjøres på hvilken som helst maskin som støtter Docker. Dette forenkler drastisk oppsettet for å kjøre prosjektet på forskjellige maskiner, siden Docker er uavhengig av operativsystemet som kjører det. Dette er spesielt nyttig for utvikling, siden prosjektet kan utvikles på platformen som er nærere det faktiske systemet det skal kjøre på. I dette tilfellet er Docker hovedsakelig brukt for å standardisere utviklingsmiljøet.

4.8.2 VSCode Devcontainer

ROS2 krever ikke bare en spesifikk versjon av Ubuntu, men også flere andre programmer og konfigurasjoner for å fungere. For å redusere arbeidsmengden som kreves for å installere ROS2 utviklingsmiljøet på en ny PC benyttes devcontainers. I ".devcontainer" mappen beskrives med konfigurasjonsfiler en Ubuntu container som er ferdig konfigurert med VSCode innstillinger og extensions, ferdig installert ROS2, Python og C++ toolchain. I tillegg er devontaineren konfigurert til å viderekoble filsystemmapper fra WSL2 som gir devcontaineren tilgang til GPU og display drivere, USB-enheter, delt minne for Intra Process Communication (IPC), nettverkskonfigurasjon for å skjule container barrieren under utvikling og selve workspace mappen som brukes under utvikling. [18].

4.8.3 WSL2

Windows Subsystem for Linux 2 (WSL2) er en funksjon i Windows som gjør det mulig å kjøre Linux distribusjoner på Windows. Dette trengs i tillegg til Docker, for å gi Docker tilgang til USB enheter, grafikkort og grafikk drivere. Dette gjør det mulig å kjøre GUI applikasjoner som Gazebo og Rviz direkte i Docker på windows uten å installere annet enn Docker og WSL2. [1].

4.8.4 VSCode Tasks

ROS2 inneholder mange lange kommandoer som er tidkrevende å skrive manuelt. For å effektivisere arbeidsflyten er det derfor laget en rekke Tasks i VSCode for å utføre disse kommandoene. Disse taskene ligger under ".vscode/tasks.json" og kan kjøres ved å trykke "Ctrl + Shift + P" og skrive "Run Task". Dette vil gi en liste over tilgjengelige tasks som kan kjøres. For å gjøre det enda enklere kan taskene også kjøres ved å trykke "Ctrl + Shift + B" og velge ønsket task. Dette vil kjøre den siste kjørte tasken, eller gi en liste over tilgjengelige tasks om ingen har blitt kjørt enda. For videre forenkling er det også forhåndsinstallert en VSCode Extension som heter "Task Buttons", som også gjør de mest hyppige kommandoene: "ROS2 Install Dependencies", "ROS2 Build" og "ROS2 Purge" tilgjengelige som knapper i VSCode nederst på statuslinjen, som illustrert i Fig 46.

Figure 46: VSCode Task Buttons

4.8.5 Github

GitHub brukes til å lagre kildekoden i prosjektet. Dette gjør det enkelt å samarbeide, dele kode og holde oversikt over endringer. [19].

–skip-worktree I dette prosjektet gir det mening å inkludere en default konfigurasjon for IDE oppsettet. Disse konfigurasjonene er derimot ofte personaliserte til hver enkelt bruker. For å unngå at endringer på disse konfigurasjonene blir pushet til remote, benyttes kommandoen "git update-index –skip-worktree filename" for å ignorere lokale endringer på disse filene. I workspace repo ligger det to scripts for å gjøre denne oppgaven automatisk. Disse ligger under "scripts/git/" og heter "ignore_local_config_file_changes.bash" og "un_ignore_local_config_file_changes.bash".

4.8.6 Vcstool

Vcstool er et standard verktøy i ROS2 for å håndtere flere git repositoryer. Dette verktøyet ble brukt for å kunne legge pakkene inn i egne git repositoryer, og deretter importere dem inn i workspace repoet. På denne måten dekobles pakker som ikke er relevante for hverandre, noe som gjør det enklere å skaffe overblikk og vedlikeholde prosjektet.

4.9 Arbeidsflyt

Etter utviklingen av et godt konfigurert workspace er installasjons- og utviklingsprosessen av prosjektet betraktelig forenklet. Følgende avsnitt gir en oversikt over arbeidsprosedyrer som er relevante for de som vil bruke og videreutvikle prosjektet.

4.9.1 Installere prosjektet

Den følgende installasjonen tar mindre enn 10 minutter manuelt arbeid, og mellom ett minutt og en time med hands-off automatisk bygging avhengig av hvor rask datamaskinen er.

- Installer VSCode
- Installer Git
- Installer Docker
- Installer Foxglove fra foxglove.dev
- Kjør `git clone https://github.com/nosknut/bachelor-thesis.git`
- Kjør `code bachelor-thesis` for å åpne repo i VSCode
- Kjør `bash ./scripts/git/ignore_local_config_file_changes.bash` for å ignorere lokale endringer på konfigurasjonsfiler.
- Følg installasjonsinstruksene for WSL2 og Docker i prosjektets README.md fil for GUI support gjennom Docker og WSL2.
- Åpne Command Palette og kjør kommandoen: "Dev Containers: Reopen in Container"
  - Dette vil bygge en Docker container basert på konfigurasjonen i ".devcontainer" mappen.
- VSCode vil nå bygge en Docker Container basert på konfigurasjonen i ".devcontainer" mappen.
- Når byggingen er ferdig vil VSCode åpne på nytt i et fullverdig og ferdig konfigurert ROS2 utviklingsmiljø.

Dette vil også fungere for linux. For linux brukere som ikke ønsker Docker kan det meste automatisk installeres ved å kjøre `bash ./scripts/ubuntu/install-ros2.bash`. Automatisk sourcing av ROS2 workspace kan konfigureres ved å kjøre `bash ./scripts/configure-workspace.bash`.

4.9.2 Bygge Prosjektet

Installasjon av alle dependencies Trykk på "ROS2 Install Dependencies" knappen nederst i VSCode. Dette kan også oppnås ved å kjøre `bash ./scripts/ros2/install-dependencies.bash` i terminalen.

Bygging av prosjektet Trykk på "ROS2 Build" knappen nederst i VSCode. Dette kan også oppnås ved å kjøre `bash ./scripts/ros2/build-packages.bash` i terminalen. Merk at denne kommandoen kjører med `--symlink-install` som betyr at endring av python og konfigurasjonsfiler ikke krever en ny bygging av prosjektet for å tre i kraft.

4.9.3 Starte Stacken

Kjøring På roboten, kjør `ros2 launch twig robot_moveit.launch.py` og `ros2 launch twig servo_teleop.launch.py`. På grensesnitt maskinen, kjør `ros2 launch twig foxglove.launch.py` og eventuekt `ros2 launch twig joy.launch.py` om det ikke er ønskelig å bruke Foxglove til å publisere joystick data. Til slutt, kjør `ros2 launch twig rviz.launch.py` på en PC med skjerm for å visualisere de interne sensortilstandene til roboten. Dette er derimot ikke nødvendig ettersom Foxglove også kan brukes for å visualisere posisjonen til hver enkelt joint.

Utvikling For å kjøre en demo av hele ROS2 stacken uten hardware visualisert i Rviz2, kjør `ros2 launch twig moveit_demo.launch.py`. GUI applikasjonen Rviz2 vil åpne seg og vise en 3D modell av manipulatoren. For å kjøre en demo med hardware, kjør `ros2 launch twig robot_moveit.launch.py`. Denne kommandoen vil starte alle ROS2 noder som er nødvendige for å kjøre manipulatoren, inkludert hardware noder.

For å kontrollere modellen, åpne Foxglove og publiser gamepad data til "/twig_joy" gjennom gamepad extension. Hold inne dødmannsknappen "RT" på kontrolleren og bruk venstre og høyre joystick for å styre aksene.

Denne utviklingsstacken er også tilgjengelig som en ren Docker Container. For å kjøre denne containeren uten å åpne VSCode i en devcontainer, åpne repoet i en WSL2 terminal og kjør "docker compose up twig".

4.9.4 Konfigurere Prosjektet

Nesten alle pakker i “src” mappen inneholder en “config” mappe med .yaml formarterte konfigurasjonsfiler. Disse config filene importeres av “.launch.py” filene i de respektive “launch” mappene.

Noen pekere på relevante konfigurasjoner: - firmware og drivere ligger under “twig_hardware/config”. - Keybinds for spillkontrollere ligger under “twig_teleop/config”. - Moveit Servo konfigurasjoner med collision avoidance ligger under “twig_servo/config”. - Inverskinematikk og baneplanlegging ligger under “twig_moveit_config/config”. - URDF beskrivelsen ligger under “twig_description/config”. - ROS2 Control konfigurasjoner ligger under “twig_hardware/config” og “twig_moveit_config/config”.

4.9.5 Importere Prosjektet

Prosjektet kan enten startes manuelt, eller legges til som dependency og startes av launchfiler i et helhetlig prosjekt. Siden all kode utenom firmware er implementert gjennom standardiserte ROS2 noder, kan også enkelte deler av prosjektet importeres separat. Dette vil være nyttig for å integrere “twig” stacken med resten av AUV når systemet til slutt skal gjøres autonomt. Se “twig” pakken som eksempel på hvordan dette gjennomføres.

4.9.6 Utvikle i Prosjektet

Utvikling kan gjennomføres som alle andre ROS2 prosjekter. Om det er ønskelig kan workspacet brukes. Devcontaineren er bygget lokalt på hver enkelt utviklings PC og kan dermed tilpasses den enkeltes smak og behag. Filene i “.vscode” og “.devcontainer” vil i så fall være av relevanse. Ved bruk av devcontaineren vil alt være ferdig konfigurert, og den normale terminalen kan brukes til å kjøre normale ROS2 kommandoer og ROS2 GUI applikasjoner som Rviz2 og RQT. Devcontaineren kan også brukes til å kjøre Rviz og andre GUI applikasjoner oppimot en fysisk robot på samme nettverk som i en vanlig ROS2 stack. For å unngå mye skriving av kommandoer kan knappene “ROS2 Build”, “ROS2 Install Dependencies” og “ROS2 Purge” på statuslinjen i VSCode brukes under utvikling. Formålet med “ROS2 Purge” er å fjerne alle bygde filer og konfigurasjoner for å starte på nytt om det skulle være nødvendig med en fersk start. Ved større problemer kan devcontaineren bygges på nytt ved å åpne repoet lokalt (ikke i devcontaineren) og kjøre “Dev Containers: Rebuild and Reopen Container” fra Command Palette. Selve repoet vil bli værende, men alle andre konfigurasjoner internt i devcontaineren (inkludert VSCode extensions) vil bli nullstilt, for å fjerne eventuelle problemer som har oppstått på grunn av dette. For å beholde andre konfigurasjoner kan enkeltfiler mountes til det fysiske filsystemet ved å legge de til under “services.ros2.volumes” i “.devcontainer/Docker-compose.yml” filen. Merk at disse filbanene refererer til plasseringen i WSL2, ikke på Windows. For de som bruker Ubuntu i stede for Windows refererer filbanene til det fysiske filsystemet.

4.10 Teknologier som ikke ble brukt

Følgende teknologier ble vurdert og fungerte, men ble ikke tatt i bruk i det endelige prosjektet fordi det var enklere eller mer effektivt å bruke andre teknologier. De er derimot inkludert i følgende avsnitt siden de har egenskaper som kan gjøre de nyttige for fremtidig utvikling.

4.10.1 Gazebo

Gazebo er en 3D simulator med fysikkmotor som ofte blir brukt under utvikling av roboter. Originalt var tanken å benytte Gazebo for å produsere sensor feedback. Det viste seg derimot at Mocked Components fra ROS2 Control og Rviz2 var en bedre løsning. Gazebo har fortsatt en relevanse for mer kompleks systemtesting, men dette ble ikke prioritert.

4.10.2 Usbipd

Usbipd er en tjeneste som lar deg dele USB enheter over IP nettverk. Dette verktøyet ble vurdert for å dele usb spillkontroller fra windows operativsystemet inn i WSL, som derfra kunne deles til en Docker container. Dette fungerte fint, men krevde litt for mye arbeid for å være praktisk i arbeidsflyten. Dette verktøyet ble erstattet med Foxglove, som kan publisere en generisk spillkontroller direkte til ROS2.

5 FNs bærekraftsmål

Prosjektet har tatt hensyn som støtter FNs bærkraftsmål, og har vært bakgrunn for forskjellige valg underveis. To av disse er relevant å trekke frem for prosjektet.

Figure 47: FNs Bærekraftsmål 12 [12]

- Sikre bærekraftig forbruks- og produksjonsmønstre. [12]
Ved hjelp av simuleringer av ledd og gir i Solidworks, i tillegg til nøye og planlagt modellering har det vært mulig å printe færrest mulig testversjoner. Dette er gjort for å begrense ressursbruk.
Girboksene er designet for å ikke kreve olje og smøring. Tanken er å redusere miljøskadelige utslipp i hav eller basseng. PETG er valgt som materiale for gripperen, og er et ugiftig og resirkulerbart materiale.
Alle komponenter og materialer er valgt med hensyn på levetid og materialbruk. Et eksempel er kulelagre som er valgt i rustfritt stål slik at det ikke stadig må byttes. Selv om utskiftning ville vært en mindre kostbar løsning i dette prosjektet.

Figure 48: FNs Bærekraftsmål 14 [13]

- Bevare og bruke havet og de marine ressursene på en måte som fremmer bærekraftig utvikling. [13]
Prosjektet bidrar til FN's bærekraftmål nr 14, "Livet i havet", og da særlig delmål 14.a som sier: "Styrke vitenskapelig kunnskap, bygge opp forskningskapasitet og overføre marin teknologi..." [13]. Selvom denne manipulatoren er designet til en studentkonkurranse, kan den også benyttes til oppgaver som den marine industrien trenger, og dermed også bidra til bærekraftig bruk av havene.

6 Konklusjon

Denne bacheloroppgaven har dokumentert utviklingen av en undervannsgripper for Vortex NTNU, spesifikt designet for å møte kravene i TAC Challenge 2024. Gjennom en tverrfaglig tilnærming har prosjektet integrert mekanisk design, elektronikk og programvareutvikling for å skape en funksjonell og pålitelig gripper som kan operere i krevende undervannsmiljøer.

Gripperen oppfyller de opprinnelige kravene, inkludert evnen til å rotere i flere akser, og åpne og lukke kloen. Skulderrotasjonen hadde opprinnelig et krav på en rotasjon 90 grader oppover , men dette kravet er justert underveis i prosessen i samarbeid med Vortex. Manipulatoren innfrir det nye kravet, på 45 graders rotasjon oppover. Armens håndledd skulle også ha mulighet til å rotere minst 360 grader. Servoene er nå hastighetsstyrte og kan rotere ubegrenset.

Gripperen kan brukes under vann ved at elektriske komponenter er støpt i materialer som hindrer at vann kommer i kontakt med elektronikken. Alle 3D-printede deler er laget av PETG, og komponentene blitt printet med 100% infill for å minske vanninntak. Skruer og lagre er av rustfritt stål for å sikre en pålitelig drift under vann.

Det hele kan styres ved bruk av ROS, og har sikkerhetsmekanismer for å beskytte servoene mot overbelastning. Enkoderne måler posisjon og hastighet med tilfredstillende nøyaktighet.

Prosjektet har også identifisert potensielle forbedringer, som implementering av planetgir i skulderaksen for økt løftekraft og økning av akseldiameteren for å forbedre den mekaniske styrken. Disse forbedringene kan implementeres i fremtiden for økt ytelse og pålitelighet.

Gripperen er nå klar til bruk i TAC Challenge 2024, og vil med god sannsynlighet bidra til Vortex NTNU sin suksess i konkurransen. Samtidig legger prosjektet et sterkt grunnlag for å nå Vortex sitt fremtidige mål om et fullstendig autonomt system.

Figure 49: Drone bilde fra unveiling

Referanser

[1] craigloewen-msft & mattwojo & mrienstra & Seldaek & Aaron-Junker & benmcmorran & v-alje. *Connect USB devices*. URL: https://learn.microsoft.com/en-us/windows/wsl/connect-usb (visited on 16/05/2024).

[2] IQS Directory 2024. *Spur Gears*. URL: https://www.iqsdirectory.com/articles/gear/spur-gears.html (visited on 20/05/2024).

[3] Elteco AS. *Enkodere*. URL: https://www.elteco.no/enkodere-1 (visited on 14/05/2024).

[4] RS Components AS. *RS PRO Steel 22 Teeth Spur Gear, 1.5 Module, 10mm Bore Diam, 33mm Pitch Diam, 25mm Hub Diam*. URL: https://no.rs-online.com/web/p/spur-gears/1827836 (visited on 20/05/2024).

[5] athackst. $vscode_ros2_workspace$. URL: https://github.com/athackst/vscode_ros2_workspace (visited on 16/05/2024).

[6] Aaron Black. *Worm Gears Explained*. URL: https://www.machinerylubrication.com/Read/1080/worm-gears (visited on 20/05/2024).

[7] MILES BUDIMIR. *Rack and Pinion Drive System: What Is It?* URL: https://www.motioncontroltips.com/what-is-a-rack-and-pinion/?fbclid=IwAR078lSrtuaR9d-HqviwZJlpn0SlnjuWgfbpDG6-8SH1L3FAgS5YiALKrJk (visited on 20/05/2024).

[8] Kjell Company. *Ratrig Aluminiumsprofil 20 x 20 mm 50 cm*. URL: https://www.kjell.com/no/produkter/elektro-og-verktoy/elektronikk/aluminiumprofiler/ratrig-aluminiumsprofil-20-x-20-mm-50-cm-p88150 (visited on 19/05/2024).

[9] Docker. *Docker Docs*. URL: https://docs.docker.com/ (visited on 20/05/2024).

[10] Mouser Electronics. *Seeed Studio Grove AS5600 Magnetic Rotary Position Encoder*. URL: https://no.mouser.com/new/seeed-studio/seeed-studio-grove-as5600-encoder/ (visited on 14/05/2024).

[11] Blue Trail Engineering. *Underwater Servo SER-20XX*. URL: https://www.bluetrailengineering.com/product-page/underwater-servo-ser-20xx (visited on 14/05/2024).

[12] FN. *Ansvarlig forbruk og produksjon*. URL: https://fn.no/om-fn/fns-baerekraftsmaal/ansvarlig-forbruk-og-produksjon (visited on 18/05/2024).

[13] FN. *Livet i havet*. URL: https://fn.no/om-fn/fns-baerekraftsmaal/livet-i-havet (visited on 18/05/2024).

[14] Matthew Fontana. *How to Create a Part in SOLIDWORKS*. URL: https://www.swyftsol.com/blog/how-to-create-a-part-in-solidworks (visited on 20/05/2024).

[15] norelem LTD. *Worm gears, right-hand centre distance 31 mm*. URL: https://norelem.co.uk/en/Product-overview/Systems-and-components-for-machine-and-plant-construction/22000/Worm-screws-worm-wheels/Worm-gears-right-hand-centre-distance-31-mm/p/agid.19970 (visited on 20/05/2024).

[16] Steven Macenski et al. 'Impact of ROS 2 Node Composition in Robotic Systems'. In: *IEEE Robotics and Autonomous Letters (RA-L)* (2023). DOI: 10.48550/arXiv.2305.09933. URL: https://arxiv.org/abs/2305.09933.

[17] Steven Macenski et al. 'Robot Operating System 2: Design, architecture, and uses in the wild'. In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[18] Microsoft. *Developing inside a Container*. URL: https://code.visualstudio.com/docs/devcontainers/containers (visited on 20/05/2024).

[19] Knut Ola Nøsen. *Project GitHub Repository*. URL: https://github.com/nosknut/bachelor-thesis/tree/submitted-with-report (visited on 20/05/2024).

[20] Jackson O'Connell. *PETG vs PLA Filament: The Main Differences*. URL: https://all3dp.com/2/petg-vs-pla-3d-printing-filaments-compared/ (visited on 20/05/2024).

[21] Curio res. *Curio Res Tutorials*. URL: https://curiores.com/ (visited on 16/05/2024).

[22] Blue Robotics. *Power Sense Module*. URL: https://bluerobotics.com/store/comm-control-power/control/psm-asm-r2-rp/ (visited on 14/05/2024).

[23] Open Robotics. *Distributions*. URL: https://docs.ros.org/en/rolling/Releases.html (visited on 16/05/2024).

[24] Open Robotics. *ROS2 Dokumentasjon*. URL: https://docs.ros.org/en/humble/index.html (visited on 16/05/2024).

[25] Open Robotics. *Understanding Nodes*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html (visited on 16/05/2024).

[26] Open Robotics. *Understanding Parameters*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Parameters/Understanding-ROS2-Parameters.html (visited on 16/05/2024).

[27] Open Robotics. *Understanding services*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html (visited on 16/05/2024).

[28] Open Robotics. *Understanding Topics*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html (visited on 16/05/2024).

[29] Yasir Shahzad. *SoftI2C*. URL: https://github.com/yasir-shahzad/SoftI2C (visited on 15/05/2024).

[30] SparkFun. *I2C*. URL: https://learn.sparkfun.com/tutorials/i2c/all (visited on 15/05/2024).

[31] Seeed Studio. *Grove - ±5A DC/AC Current Sensor (ACS70331)*. URL: https://wiki.seeedstudio.com/Grove-5A_DC_AC_Current_Sensor-ACS70331 (visited on 14/05/2024).

[32] Handson Technology. *4-Channel 5V Optical Isolated Relay Module*. URL: https://handsontec.com/dataspecs/module/4Ch-relay.pdf (visited on 14/05/2024).

[33] UltiMaker. *UltiMaker Cura*. URL: https://ultimaker.com/software/ultimaker-cura/ (visited on 20/05/2024).

Appendix

A Poster

Undervannsgripper

Forfattere: Markus Johnstad, Joakim Fjeldkjøn, Knut Ola Nøsen, Tobias Slettebakken
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk
Dato: 16.05.2024

Introduksjon

Denne oppgaven dokumenterer prosessen med design, utvikling og implementering av verdens minste undervannsgripper med 3 frihetsgrader, for Vortex NTNU sin deltakelse i TAC Challenge 2024. Gripperens evne til å gripe og rotere en ventil under vann er vikitg for å lykkes i denne konkurransen.

Mekanisk

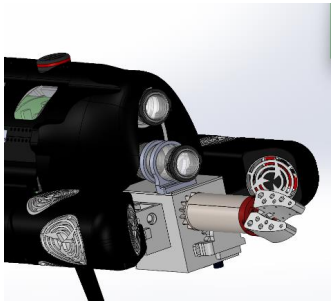
Gripperen er designet i Solidworks og består hovedsakelig av 3D-printede deler i PETG, valgt for sin robusthet og vannmotstand. Designprosessen inkluderte flere iterasjoner med fokus på å oppnå god funksjonalitet og estetikk. En viktig mekanisk komponent er wormgearet, som gir både nødvendig rotasjon og stabilitet til gripperens skulderledd.

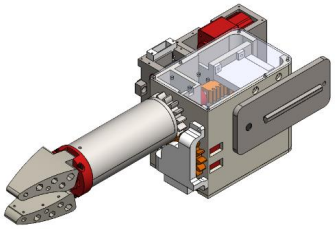
Elektrisk

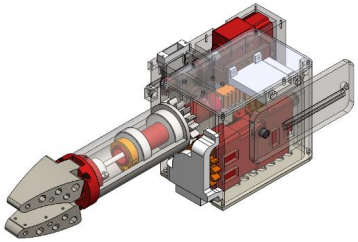
Gripperens elektroniske system er bygget rundt servomotorer og magnetiske encodere, beskyttet mot vanninntrengning ved hjelp av støping i epoxy. For å sikre systemets robusthet og pålitelighet, er det implementert løsninger som gjør det mulig å overvåke og kontrollere strømforbruket til hver enkelt komponent. Dette er spesielt viktig for å forhindre skader og kunne skru av servoer ved overbelastning.

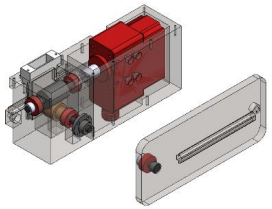
Software

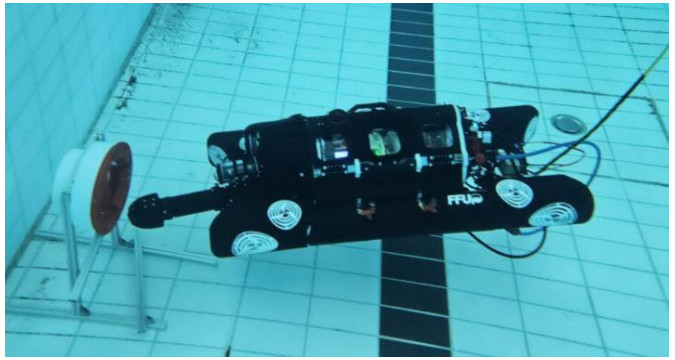
Programvarearkitekturen er basert på ROS2 (Robot Operating System 2), som gir en fleksibel plattform for kontroll og diagnostikk av gripperen. ROS2-pakkene utviklet for dette prosjektet inkluderer spesifikke moduler for bevegelsesstyring, diagnostikk og teleoperasjon. Disse modulene integrerer ulike sensordata og gir brukeren muligheten til å styre gripperen nøyaktig og pålitelig.











Bakgrunn

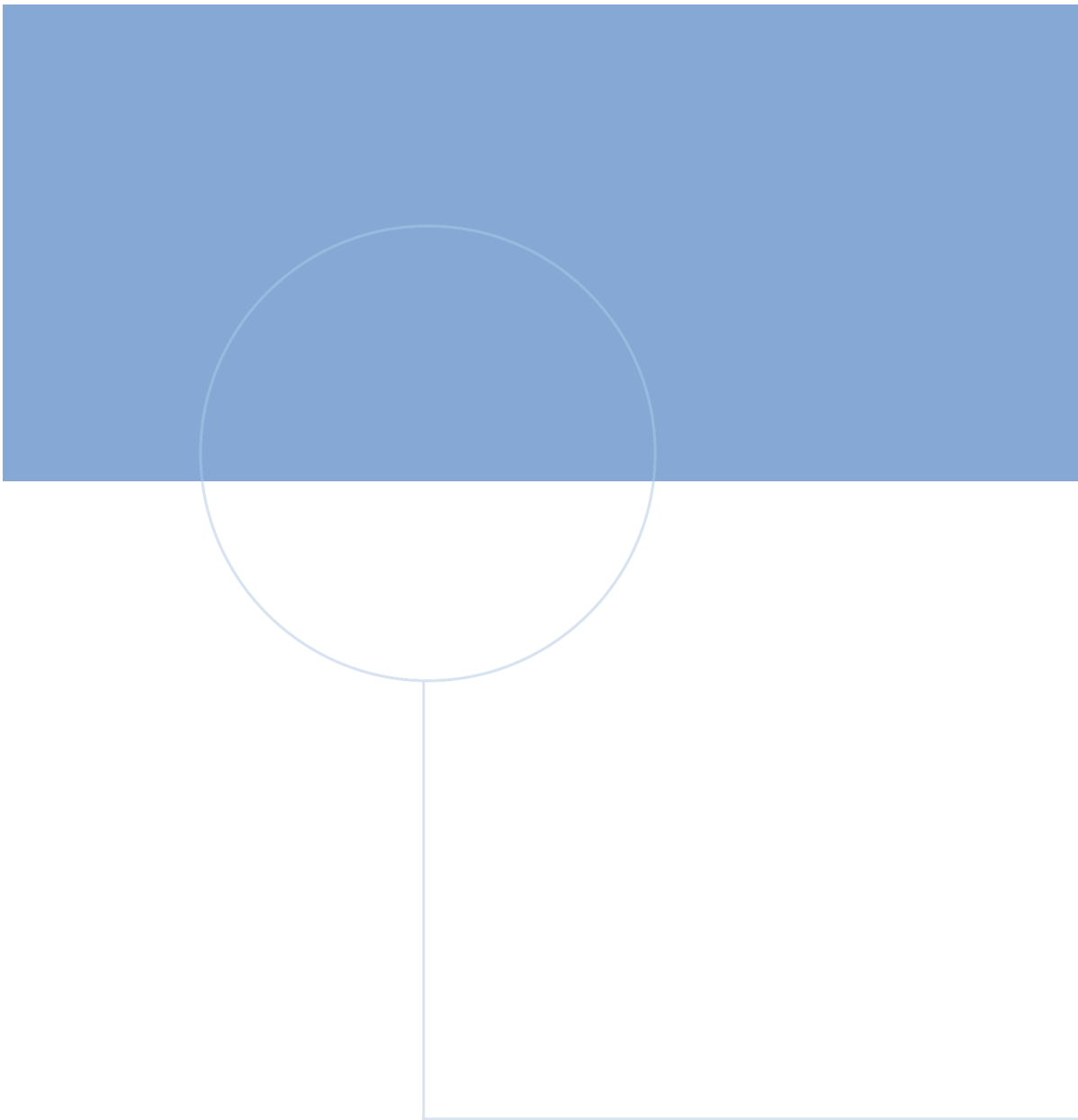
Vortex NTNU, en studentorganisasjon ved NTNU, utvikler autonome undervannsfartøy som deltar i internasjonale konkurranser. En viktig del av disse konkurransene inkluderer oppgaver som krever presis manipulasjon under vann, slik som å skru ventiler. Gripperen som er utviklet i dette prosjektet, er en essensiell komponent for å utføre disse oppgavene.

Resultat

Denne oppgaven demonstrerer en vellykket integrering av mekanisk design, elektronikk og programvare for å utvikle en funksjonell undervannsgripper, klar til bruk i konkurranser og videre utvikling.

Konklusjon

Den endelige gripperen oppfyller alle de opprinnelige kravene, med unntak av enkelte justeringer gjort i henhold til praktiske erfaringer under utviklingsprosessen. Den er i stand til å rotere i flere akser, åpne og lukke seg, og fungerer pålitelig under vann. Videre potensielle forbedringer inkluderer implementering av et planetgir i skulderaksen og økning av akseldiameteren for å øke systemets robusthet.



NTNU

Kunnskap for en bedre verden