MEKATRONIKK - RAPPORT

HØGSKOLEN I ÅLESUND
Aalesund University College

TITTEL:
AQUAFARM INSPECTION - ROV

KANDIDATNUMMER(E):			
10008 **10007** **10006**			
DATO: 30.11.16	EMNEKODE: IP304814	EMNE: Innføring i Mekatronikk	DOKUMENT TILGANG:
STUDIUM:		ANT SIDER/VEDLEGG: /	BIBL. NR:

VEILEDER(E) :
Houxiang Zhang

SAMMENDRAG:

Denne rapporten omhandler prosessen gruppen vår har vært gjennom med å produsere en ROV fra grunn. Det har vært ett prosjekt med både Software,Hardware og design utfordringer som har satt kunnskapen til gruppen på prøve.

Med bruk av trådbaserte systemer i programmeringen har vi kunne løst flere utfordringene vi har kommet over i software løsningen.

Systemet tar i bruk både mikrokontroller, Mini-PC (Rasberry) og PC, som gjør at vi i programutviklingen har måtte forholde oss til flere språk. Det har også gjort at vi har benyttet flere kommunikasjonsmetoder for at systemet skal fungere i sin helhet.

Denne oppgaven er en eksamensbesvarelse utført av studenter ved Høgskolen i Ålesund.

Postadresse
Høgskolen i Ålesund
N-6025 Ålesund
Norway

Besøksadresse
Larsgårdsvegen 2
Internett
www.hials.no

Telefon
70 16 12 00
Epostadresse
postmottak@hials.no

Telefax
70 16 13 00

Bankkonto
7694 05 00636
Foretaksregisteret
NO 971 572 140

FORORD

Denne prosjektoppgaven er utført av tre studenter i faget Innføring i mekatronikk. Formålet med prosjektoppgaven er design og bygging av en to dimensjonal fjernstyrt undervannsfarkost. Denne farkosten skal på et senere tidspunkt integreres sammen med en flytende plattform som en annen gruppe jobber med. Oppdriften til farkosten blir styrt ved hjelp av en ramme på plattformen. Til sammen vil dette gjøre avanserte inspeksjoner på ulike havområder, som for eksempel oppdrettsnæringen der dette kan brukes for å finne hull i fiskenot.

Vårt arbeid i denne prosjektoppgaven er å bygge en ROV helt fra grunn av. Dette innebærer beregninger, analyser og matematiske beregninger for blant annet plassering av utstyr og fremdriftskrefter. Det skal programmeres et grafisk brukergrensesnitt der man styrer farkosten samt har sanntids sensorinformasjon og video.

Vi vil gjerne takke alle som har bidratt i dette prosjektet, og spesielt vil vi takke:

- Vår fagansvarlig Houxiang Zhang for veiledning gjennom hele prosjektet.
- Lab ingeniør Anders Sætersmoen for bistand ved innkjøp og utlån av deler og utstyr.
- Holstad Rørservice AS for gratis plastsveising og svært god bistand.

INNHOLD

SAMMENDRAG

Denne rapporten omhandler prosessen gruppen vår har vært gjennom med å produsere en ROV fra grunn. Det har vært ett prosjekt med både Software,Hardware og design utfordringer som har satt kunnskapen til gruppen på prøve.

Med bruk av trådbaserte systemer i programmeringen har vi kunne løst flere utfordringene vi har kommet over i software løsningen.

Systemet tar i bruk både mikrokontroller, Mini-PC (Rasberry) og PC, som gjør at vi i programutviklingen har måtte forholde oss til flere språk. Det har også gjort at vi har benyttet flere kommunikasjonsmetoder for at systemet skal fungere i sin helhet.

TERMINOLOGI

Symboler

P = Trykk

A = Areal

F = Kraft

Pa = Atmosfære trykk

g = Gravitasjonskonstant

H = Dybde

Fb = oppdrift

p = massetetthet

Vf = fortrengt Volum

R = Motstand

Vin = Spenning inn

$\Delta U = \text{Spenningsfall i Volt}$

$\Delta u = \text{Spenningsfall i prosent}$

$P = Lasteffekt\ (kW)$

$\rho = resitivitet\ \left(\frac{\Omega\ x\ mm^2}{m}\right), 0.018\ \left(\frac{\Omega\ x\ mm^2}{m}\right) for\ kopper.$

$l = Kabellengde\ (meter)$

$U_0 = Fasespenning$

$A = Ledertverrsnitt\ (mm^2)$

Forkortelser

ROV	Fjernstyrt undervannsfarkost
UDP	User Datagram Protocol
JSSC	Java Simple Serial Connector
OOP	Objekt orienter Programmering
DC	Direct Current
PWM	Pulsbreddemodulasjon
IP	Internett protokoll adresse
IMU	Treghets måler
UML	Unified Modeling Language
USB	Universal Serial Bus

1 INNLEDNING

1.1 Valg av oppgave

Vi har i dette prosjektet valgt en oppgave som går ut på å designe og utvikle en fungerende ROV som skal benyttes til feilsøking av note i en fiskemerd. ROV'en skal kunne navigeres av en bruker lokalisert på land/båt. Brukeren skal kunne motta live bildestømmning som gjennspeiler situasjonen der ROV befinner seg. ROV'en skal via en lang kabel, være festet til en plattform som ligger på overflaten. Denne plattformen er en prosjektoppgave utviklet av en annen gruppe, som da gjør at vi hele tiden må ha flytende kommunikasjon mellom gruppene for at de sepparerte oppgavene skal fungere sammen.

1.2 Krav til oppgave

Under har vi listet opp kravene som er gitt av studieveileder for oppgaven:

- ROV må inneholde kamera som kan inspisere note i en merd
- Skal ha kabel fra plattform som sender strøm, video og kontroll signal
- Må kunne kontrolleres i "YAW" aksen
- Design må minimalisere bevegelsene under vann
- Heving/senkning kontrolleres av vinsj på plattform
- Skulle kunne operere 10 meter under vann.

1.3 Problemstilling

«Hvordan kan vi utvikle en rimelig ROV som på enkles mulig måte kan gjenspeile ROV'ens omgivelser og detektere hull i note»

Ønsket med ROV er at den skal kunne gjenngi omgivelsene der den befinner seg på en mest informativ måte. For at dataen vi mottar skal kunne gi oss de ønskede skildringene, er sensorvalgene en viktig del av vår oppgave. Vi har i sammarbeid med faglig veileder utviklet en liste med sensorer vi tror kan gi oss de ønskede verdiene til dette. Med tanke på at dette er en total leveranse, er det også viktig at designet på selve ROV oppfører seg som ønsket i de aktuelle omgivelsene. Dette vil da også innebære utfordringer i designutviklingen i prosjektet.

1.4 Målsetting

Vi har i starten av denne prosjektperioden definert oss noen mål vi ønsker å oppnå. Vi har delt målene opp i 3 kategorier for å lettere se etter endt prosjekt om hvor vidt vi har oppnådd ønsket resultat.

1.4.1 Produktmål

Målsettingen med produktet vårt (ROV) er at den skal inneholde følgende egenskaper:

- Live bilde strøm skal sendes fra ROV direkte til bruker som kan detektere hull/skader i note
- Ha en oversiktlig GUI som kan overvåke all data sendt fra ROV
- Kunne manøvreres på en enkel måte
- Være designet slik det ikke blir store utfordringer med undervanns strømmer
- Inneholde nødvendige sensorer som kan gjengi ønsket omgivelse

1.4.2 Prosjektmål

Dette er ett prosjekt med svært avgrenset tidsperiode, samt kollisjon med flere samgående prosjekter i andre fag, så det er viktig for oss at vi arbeider strukturert og har ett godt samarbeid oss i mellom. For at vi på en best mulig måte skal få til dette har vi satt oss opp en prosjektplan med milepæler, som gjør det mulig for oss til en hver tid å se om vi ligger på/foran, eller etter tidsskjema. Studieveileder har også ønsket tilbakemelding hver uke om status på prosjektet, noe som har gjort at vi har oppdatert planen godt underveis.
Prosjektplanen er delt i to deler. Der den ene delen tar for seg hardware og design, og den andre Software.

Den endelige prosjektplanen vises i tabellen på neste side. Prosjektplanen har blitt modifisert flere ganger siden vi hadde første møte.

Nr	Milepæl	Dato
	Hardware og design	
1	**Finne og bestille nødvendige komponenter** *Få en oversikt over alle komponentene som skal brukes, og bestill de som mangler*	**uke 41**
2	**Utvikle konseptuelt design** *Produsere en 3D modell av prototype i software*	**uke 41**
3	**Utvikle prototype** *Printet ut og satt sammen nødvendige komponenter*	**uke 43**
4	**Teste tetthet i prototype** *Testet ROV i vann for å se om der er lekkasjer*	**uke 44**
5	**Implementere elektronikk** *Sette inn all elektronikk i tett ROV*	**uke 45**
6	**Teste ROV** *sending av sensorverdier til GUI*	**uke 47**
	Software	
7	**Opprette GUI** *Opprette både kode og grafisk grensesnitt til GUI*	**uke 41**
8	**Opprette program på Raspberry (ROV)** *Sette opp nødvendig kode som skal implementeres i raspberry.*	**uke 41**
9	**Sette opp kommunikasjon mellom ROV og GUI** *Sette opp kommunikasjon mellom GU og ROV.*	**uke 42**
10	**Klargjør kode på mikrokontroller** *Sette opp nødvendig kode som skal implementeres i mikrokontrolleren*	**uke 43**
11	**Sette opp kommunikasjon mellom Mikrokontroller og ROV** *Sette opp seriell kommunikasjon mellom ROV og mikrokontroller*	**uke 44**
12	**Teste fullt system** *Test all kode sammensatt*	**uke 45**
13	**Fin skrive kode til program** *Se over all kode og finskriv det som kan fin skrives*	**uke 46**
13	**Rapportskriving** *Bruk uken på rapportskriving. Rapporten skal fylles ut under hele prosjektet, så burde bare være sammendrag, resultat og konklusjon som gjenstår*	**uke 47**

Tabell 1.4.2 Fremdriftsplan

1.4.3 Prosessmål

Prosessmålet i dette prosjektet dreier seg i hovedsak om egen læring. Tilegne oss gode kunnskaper og nye erfaringer innenfor de forskjellige fagområdene vi vil berøre underveis i prosjektet. Vi er en gruppe bestående av tre medlemmer med veldig ulik bakgrunn, så her ser vi det absolutt mulig at alle vil kunne tilegne seg nye kvaliteter etter endt prosjektperiode.

Det er ett prosjekt som består av både konseptuell designutvikling, kobling av elektronikk, og utvikling av programkode, noe som alle tre i gruppen har ulik kjennskap til. Dette ser vi på som en spennende utfordring, og ser frem til dette.

2 TEORETISK GRUNNLAG

I dette underliggende kapittelet legger vi det teoretiske grunnlaget for oppgaven. Teori som vil bli presentert her, er knyttet mot vårt pensum, og er en viktig del av arbeidet for å komme til en løsning opp mot vår problemstilling.

2.1 Teori

2.1.1 Trykk

Innenfor fysikken er trykk definert som kraft fordelt på ett bestemt areal. Med P = trykket, A er areal og F er kraften har vi da formelen:

$$\boldsymbol{P = \frac{f}{A}}$$

SI enheten for Trykk er Pascal (PA), men blir også defineres i andre enheter i utgangspunkt i Pa. Som bland annet Bar. 1 Bar = 100kPa.

Det er flere områder hvor trykket har en påvirkningskraft. Dette er blant annet:

- *Væsketrykk*
- *Gasstrykk*
- *Lydtrykk*

Væsketrykket er kreftene som påvirker på ett objekt når det befinner seg omgitt av en type væske. Trykket øker jo dypere ned i væsken objektet befinner seg. Dette gir oss ifølge pascals lov formelen:

$$P = pa + pgh$$

Der pa er atmosfære trykket, p er væskens tetthet, g er gravitasjonskonstanten (9,81) og h er dybden objektet befinner seg på.

2.1.2 Temperatur

Temperatur i fysikken er definert som mengden termisk energi i ett system. Temperaturen øker/synker med endring i indre energi. I elektronikken vil også motstanden endre seg i takt med temperaturen.

Temperatur blir målt i flere forskjellige former, men SI-enheten er Kelvin (K). Denne starter på 0, hvor det absolutte nullpunktet befinner seg. (Temperatur,Wikipedia).

Andre enheter som blir benyttet til å måle temperatur er :

- Celcius (*C)
- Farenheit (*F)

2.1.3 Oppdrift

Oppdrift er en kraft som virker på ett delvis eller helt nedsenket objekt i en væske. Oppdriften tilsvarer den totale mengden væske objektet fortrenger når det er nedsenket.

Den matematiske formelen for oppdrift er: (Wikipedia, oppdrift)

$$Fb = -pVg$$

Der p =massetetther, V er det fortrengte volumet, og g er tyngdeakselerasjonen. Minustegnet beviser at kraften virker mot tyngdekraften. Altså oppdriften går opp mot overflaten.

2.1.4 ROV

ROV, står for «Remotely operated vehicle», eller på norsk – Fjernstyrt undervannsfarkost (wikipedia, 2015). Det er en robot som har klart definerte oppgaver som blir utført under vann, og blir som oftest kontrollert av en styringsenhet på land/båt. Arbeidsområdene/typene av ROV kan defineres i 3 hoved grupper:

2.1.4.1 Observasjonsfarkost

Denne farkosten/ROV har som oftest bare i oppgave å samle inn data som formidles videre til analyse. Type data som hentes inn kan være av «geografiske, kontroll og analytiske» data.

2.1.4.2 Arbeidsfarkost

En arbeidsfarkost er ofte litt mer avansert en observasjons farkost da denne ofte har mer fysiske oppgaver en observasjons farkost. Dette kan være vedlikehold, reparasjon eller utvikling av systemer som er lagt i havdypet.

2.1.4.3 Surveyfarkost

Er ganske lik en arbeidsfarkost, men har også ett kartleggingsutstyr som er implementert i roboten. Dette utstyret gjør det mulig for ROV å undersøke f.eks. miljøet i havbunnen før og etter en operasjon har blir utført for å se hvordan det menneskelige arbeidet har endret omgivelsene.

2.1.5 Merd

En merd er en fiskenot som blir brukt i oppdrettsbransjen. Det er en stor not som blir holdt utspent av ett rammeverk hvor den holder fiskebestanden «fanget» på innsiden. På denne måten kan fiske bevege seg i naturlige områder, men likevel under oppdrettsanlegget sin fulle kontroll.

2.1.6 Java

Java er i dag ett av de mest brukte objekt orienterte programmeringsspråkene som blir benyttet. En spesiell ting med Java er at det ikke kompileres i maskinkode, men plattformuavhengig bytekode som kjøres av ett underliggende lag programvare kalt Java Viritual Machine (JVM) (wikipedia, Java, 2015).

2.1.7 3D-Printing

3D-Printing er en metode som de siste årene har blitt veldig utbredt og blir benyttet til oppgaver som for eksempel bygging av prototyper. 3D-printing fungerer med at ett fast materiale blir varmet opp til løs form, og lagt lagvis oppover ut ifra en programmert 3D modell sendt fra brukeren. Når materialet er lagt på ønsket posisjon vil det synke til temperaturen hvor det går tilbake til fast form. På denne måten klarer vi å overføre en 3D tegning fra maskinen til en fysisk gjenstand.

2.1.8 Kommunikasjon

2.1.8.1 Seriell kommunikasjon

Seriell kommunikasjon betegner prosessen hvor data blir sendt sekvensielt over kabel en «bit» av gangen.

Nyere serielle grensesnitt har gjort at nesten all kommunikasjon mellom pc og periferiutstyr foregår serielt (SNL.no). Moderne grensesnitt som USB og IEEE er noen av standardene som er grunn i dette.

2.1.8.2 Protokoll

I datasammenheng kan det sies at en protokoll er ett sett med regler for hvordan sammenhengen mellom to endepunkter kan defineres. Dette kan være regler for:

- Hvilke datatyper som skal benyttes mellom de to endepunktene
- Hvilken kommunikasjonsform som skal benyttes
- Hvilken tilkobling som skal benyttes mellom endepunktene.

2.1.8.3 Port

En port kan sies å være applikasjonsadressen til ett program. Når en pakke blir sendt over nettverket, er det to adresser som blir sendt. Server adressen, som beskriver hvilken server pakken skal sendes til, og port adressen, som da beskriver hvilket program pakken skal sendes til på den aktuelle serveren.

Siden er server kan kjøre flere programmer samtidig, er det nødvendig at pakken får informasjon om hvilket program den skal sendes til for at pakken skal ankomme riktig destinasjon.

Portnumrene 1-1023 er reserverte adresser til system applikasjoner, men med unntak av de er det fritt frem for enhver applikasjon å benytte ett ledig nummer.

2.2 Viktige benevnelser i programmeringsbruk

Ved bruk av OOP er det noen gitte prinsipper som er viktig å følge for å opprettholde ett godt design. Dette er spesielt to prinsipper som «cohesian» og «coupling».

Om det oppstår problemer med å opprettholde slike prinsipper, er «design patterns» en utvei som kan være behjelpelig med å få god design på en kode.

2.2.1 Cohesian

Cohesian er sammenhengs-kraften til en modul i ett program. Mer eksakt omhandler det hvilke oppgaver hver klasse utfører. I et program er det ønskelig med høy sammenhengs-kraft (eng: high cohesion), dette betyr at det er klart definert og godt fokus i en klasse på hvilke oppgaver den har. En klasse skal kun ha funksjoner relatert til det som er intensjonen for klassen. (Michael Kolling,BlueJ, 2012)

2.2.2 Coupling

Coupling er også ett godt kjent begrep i OOP. Det legger vekt på avhengigheten mellom to klasser eller moduler. Når en utvikler systemer så ønsker en lav kobling (eng: loose coupling), dette vil i praksis si at om en gjør store endringer i en klasse, skal dette på minst mulig grad påvirke dette andre klasser Om et program har høg kobling (eng: high coupling) vil det være utfordrende å gjøre endringer, siden klassene er tett knyttet til hverandre. Da kan en liten endring i en klasse, gjør at en må endre hele systemet. (Michael Kolling,BlueJ, 2012)

2.2.3 Design patterns

Et «design pattern» er ett mønster som er utviklet for å kunne optimalisere et datasystem slik en opprettholder gode design prinsipper (ref 2.2.1 og 2.2.2), samtidig som en tilegner utvalgte egenskaper. Slike ønskede egenskaper kan variere mye, men noen eksempler er: ytelse, forutsigbarhet, pålitelighet, sikkerhet, gjenbrukbarhet og skalerbarhet.

Utfordringen ved å bruke designmønster er ofte å lete frem til det som passer til den aktuelle problemstillingen. Et designmønster har som regel en overordnet beskrivelse av problemet som hjelper å finne det rette. Videre så inneholder de som regel implementasjonsstrategier, oversikt over struktur, UML-diagrammer, konsekvenser av implementering og relaterte mønstre. (Eric Freeman, Head Frist Design Patterns, 2004)

2.3 Nødvendige egenskaper med software løsning

Vi har i kapittel 1 utredet hvilke egenskaper vi ønsker at vårt produkt skal inneholde. Vi vil videre gå inn på to teorier vi mener er viktige for vår oppgave. Med tanke på at vi skal designe ett produkt som skal sende direkte bildestrøm, og kunne navigeres i «real time» , mener vi at to teorier som er verd å utdype er :

- Sanntidssystemer (Utførelse av oppgaver i sanntid)
- Samtidighet (Kjøring av flere prosesser samtidig)

2.3.1 Sanntidssystemer (Real-time systems)

Sanntidssystemer (eng: Real-time system) er en benevnelse for ett datasystem som er avhengig av at systemets beregninger kommer innen en spesifisert tidsramme. Tidskravet kommer av at et sanntidssystem skal samhandle med et system i den virkelige verden som ofte er avhengig av tiden. Et sanntidssystem er ofte samtidige da de kan være en del av et større system, som skal observere og kontrollere modeller som er avhengig av parallelle handlinger.

Sanntidssystemer kan klassifiseres i forhold til hvile tidskrav de skal følge.

- Hard: Visst en beregning ankommer etter tidsfristen er ute, så har beregning ingen verdi. I slike systemer er en avhengig av at responsen kommer innen gitt tidsfrist. Det er ikke akseptert å ikke rekke en tidsfrist.
- Myk: I et slikt system er responstiden viktig, men systemet vil virke selv om tidsfristene ikke blir nådd en gang iblant.

Mange systemer kan ha både harde og myke sanntidsundersystemer. Videre kan slike systemer har flere egenskaper:

- Store og komplekse: Kan være et enkelt system med svært få kodelinjer til store og komplekse system over flere plattformer og mange ulike programmeringsspråk.
- Pålitelig og sikker: Kan ofte være systemer som styrer viktige deler av samfunnet vårt. Det er derfor viktig at slike system kan fortsette å kjøre, selv om noen feil oppstår. Om systemet ikke kan fortsette å kjøre så er det viktig at det går over i trygg tilstand før det stenger ned.

- Sanntids kontroll: Gir mulighet til å synkronisere metoder og handlinger med tiden. Det kan være viktig å kunne spesifisere når noen handlinger skal begynne, når de skal avslutte og hvordan systemet skal reagere om en bommer på tiden.
- Interaksjon mot hardware: Systemet skal ofte lese og skrive til utganger på et virkelig system. Derfor kan det være viktig at et signal inn fra en enhet kan avbryte en pågående prosess. Dette kan for eksempel være om en trykker på en stoppknapp, da burde systemet behandle denne interaksjonen med en gang. . (Lindsey,Javatech)(Concurrent and Real Time programming in Java)

I senere kapitler vil vi gå nærmere inn på hvilke teknikker og metoder vi har valgt for at systemet vi skal lage er i likhet med ett system beskrevet over.

2.3.2 Samtidighet (Concurrency)

Samtidighetsprogrammering (eng: concurrency) er et kjent begrep som er brukt for å uttrykke potensiell parallellitet i en applikasjon. Samtidighet omfatter teknikker som behandler kommunikasjon og synkronisering mellom parallelle enheter.

Samtidighet er viktig for å kunne utnytte kraften i en prosessor. Moderne prosessorer kjører i veldig stor hastighet i forhold til kravene til lesing og skriving av innganger og utganger. Ved å bruke samtidighet kan en foreta beregninger, presentasjon og lesing av innganger samtidig. Alternativet er å bruke sekvensielle program, da kan en risikere at presentasjonen av dataen låses, mens en regner ut de neste dataene som skal presenteres.

Visst en ønsker å bruke flere prosessorer for å løyse et problem så er en avhengig av å bruke samtidighet. Sekvensielle program kan bare utføres på en prosessor, et samtidighetsprogram kan utføres på flere prosessorer slik at en oppnår ekte parallellisme og raskere utføringstid.

Samtidighetsapplikasjoner skal ofte kunne styre og være grensesnitt mot virkelige enheter som roboter, biler, samleband eller lignende som har behov for parallelle prosesser.

Emnet samtidighet introduserer problemer som er ukjent for sekvensielle program. Aktiviteter og hendelser som skal jobbe sammen må være synkronisert og koordinert. Synkronisering av slike hendelser er svært viktig. Visst en mislykkes med dette kan en få mange feil og problemer. Noen kjente problemer er:

- Vranglås (eng: deadlock): Kan forekomme om flere samtidige aktiviteter venter på at andre skal utføre noe.

- Interferens (eng: interference): Kan forekomme om to eller flere tråder prøver å oppdatere det samme objektet.
- Sult (eng: starvation): Kan forekomme om en eller flere tråder blir nektet tilgang til en resurs på grunn av handlingen til andre tråder.

Det er viktig å nevne begrep som sikkerhet og «liveness» når en snakker om kvalitetene til en samtidighetsapplikasjon. Sikkerheten tar for seg at ingenting gale skal skje, som for eksempel at en får interferens slik at data kan bli korrupt. «Liveness» betyr at en skal sørge for at noe bra skjer, at systemet ikke går i vranglås eller lignende. (Concurrent and Real Time programming in Java) (Lindsey,Javatech)

3 MATERIALER OG METODE

I underliggende kapittel vil vi gå inn på:

- Hvilke komponenter vi har benyttet for å sette ROV sammen
- Hvilke metoder vi har benyttet for å få systemet til å fungere
- Hvilke programmer vi har benyttet gjennom prosjektperioden

3.1 Materialer

Her vil vi gå inn på hver enkel komponent som er benyttet for å sette sammen ROV. Det vil være informasjon om komponenten, og hvordan den er benyttet i vårt prosjekt.

3.1.1 Thrustere

En thruster (også på norsk kallet for en sidepropell), er delen som forårsaker fremdrift på ett skip. Det finnes flere typer thrustere: (Wikipedia)

- *Innfelt thruster* (Tunell thruster). Denne er innfelt i selve skroget på skipet.
- *Azimuth-thruster*. Denne er festet på en dreibar sokkel under skroget.
- *Compass-thruster*. En nedsenkbar propell.

Figur 3-1 Bilde av T200 Thruster

Vi bruker tre thrustere av typen T200 fra bluerobotics [1], se figur 3.1.3-1. De er av veldig god kvalitet og har gode egenskaper. Tabell 3-1 viser spesifikasjonene til T200 thrusterene. Dette er thrustere av typen «innfelt thruster».

Operasjonsspenning	Maks Effekt	Rotasjons hastighet	Diameter	Vekt i vann	Vekt i luft
6-20V	*350W*	*300-3800rev/min*	*100mm*	*156g*	*344g*

3.1.2 Hastighetskontroller

Hastighetskontroller som er benyttet opp mot thrusterene er 30A Electronics Speed Controller(ESC) fra bluerobotics [4], se figur 3-2 Thrusterene kan styres ved hjelp av en mikrokontroller som gir PWM signal til hastighetskontroller.

Figur 3-2 bilde av hastighetskontroller.

3.1.3 Trykk- og temperaturmåler

Trykk sensor som blir brukt er Bar30 fra bluerobotics [2], se figur 3-3. Denne sensoren har en høy oppløsning på trykkmåling og kan måle trykk helt opp til 30bar, noe som tilsvarer 300 meter dybde. Sensoren har også temperaturmåler med noe mindre oppløsning og nøyaktighet, men er helt grei til formålet vårt, med ±1°C oppløsning.
Sensor informasjonen kan enkelt hentes ut ved hjelp av en mikrokontroller. Den bruker inner-Integrated circuit (I^2C) protokoll.

Figur 3-3 bilde av trykk- og temperatur måler.
.

3.1.4 Signal- og strømkabel

Signal kabel er en CAT 5 Ethernet kabel fra bluerobotics [3] som er beregnet for å bære last. Oppgitt arbeidsstyrke er 35kg og en bruddstyrke på 155kg, se figur 3-4.

Figur 3-4 Bilde av signalkabel

Strømkabel som er brukt i dette prosjektet er en $4x4mm^2$ kabel, se figur 3-5. Denne ble valgt fordi det var et begrenset utvalg av kabler her på skolen. Den beste løsningen hadde vært en $2x4mm^2$kabel på grunn av tyngden og tykkelsen på kabelen.

Figur 3-5 bilde av strømkabel

3.1.5 Kamera

Vi hadde et Raspberry pi kamera som vi skulle benytte i dette prosjektet, men på grunn av manglende software som var tidkrevende hadde vi ikke tid til å sette oss inn i dette. Løsningen ble å bruke et HD-webkamera fra Creative siden bildebehandling biblioteket OpenCV støttet dette kameraet, se figur 3-6.

Figur 3-6 bilde av webkamera.
.

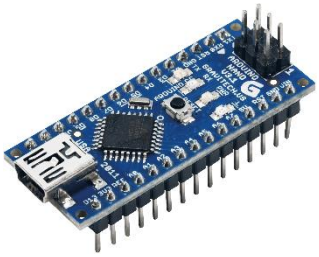
Figur 3-7 Bilde av Raspberry pi kamera.

Raspberry pi kameraet har veldig høy video kvalitet og 110° synsvinkel [5], noe som praktisk undervann, se figur 3-7.

3.1.6 Arduino Nano

«Arduino er en plattform for prototyping av elektronikk basert på program- og maskinvare med åpen kildekode» (Wikipedia, Arduino)

Vi bruker en Arduino nano mikrokontroller [7], se figur 3-8. Det er en 16bits mikrokontroller med 14 digitale IO, hvorav 6 av disse er PWM, og 8 analoge IO. Med en inngangsspenning fra 6-20VDC og utgangsspenning på 5V.



Figur 3-8 Bilde av Arduino nano.

3.1.7 Lys

Led lys fra bluerobotics [6], se figur 3-9. De er vanntette inn til 300 meter dypt og lys styrke kan styres ved PWM signal fra 3-48V. De bruker 15 watt på maksimal styrke, yter hele 1500 lumen og har en operasjonsspenning på 10-48VDC.

Figur 3-9 Bilde av Led lys.

3.1.8 Raspberry PI Modell 3B

Siden vi har behov for prosesserings ytelse bruker vi en Raspberry pi Model 3B, se figur 3-10. Dette fordi mikrokontrollere ikke er spesielt egnet til behandling av store mengder data, som i dette tilfellet blir bilde overføring og analyser av dette.

Figur 3-10 Bilde av Raspberry Pi Model 3B.

En Raspberry PI er en fullt fungerende datamaskin bygget på ett kretskort med mål på rundt 5.6cm * 8.5cm. Den første Raspberry PI ble utgitt 29.februar 2012, og har i senere tid kommet i to nye versjoner (PI 2 og PI 3). Den siste versjonen (PI 3) ble utgitt 29.februar 2016. Altså på dagen 4 år etter utgivelsen av første versjon.

I tabellen under er spesifikasjonene til PI 1 og PI 3 oppgitt. Den siste versjonen har også innebygd Wifi (802.11 b/g/n)

	PI 1	**PI 3**
Utgitt	29.feb.12	29.feb.16
Harddisk	WD PiDrive 314gb	SD-kort
Prosessor	Enkjernet ARM	Firekjernet ARM
Prosessor hastighet	700mHz	1200MHz
Minne	256MB	1GB
Operativsystem	Linux	Linux

Tabell 3.1.7 Raspberry spesifikasjoner.

3.1.9 Oksygenmåler

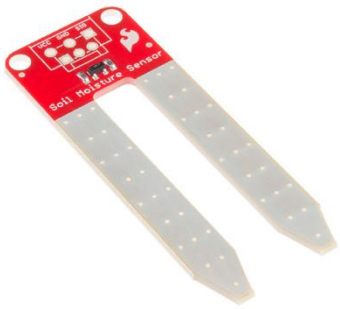
Siden dette prosjektet har som formål å observere fisk har vi benyttet av oss en oksygenmåler fra sparkfun [9], se figur 3-11. Sensoren kan kobles rett på en mikrokontroller og gir tilbake sensorverdi 0-20mg/L. Det er to måter denne kan brukes med en mikrokontroller. Den ene er ved seriellkobling og den andre er å koble den på I^2C bus. Vi

Figur 3-11 Bilde av oksygenmåler

har valgt seriellkobling i dette prosjektet på grunn av lite informasjon om hvordan den fungerer på I^2C bus.

3.1.10 Fuktmåler

Vi bruker en jordfuktighetssensor fra sparkfun [10], se figur 3-12. Denne sensoren kan kobles til en mikrokontroller og hente ut analoge sensorverdier. Sensorverdiene vil øke når sensoren utsettes for fuktighet. På denne måten kan vi få beskjed om det er fuktighet eller vann inne i ROVen.

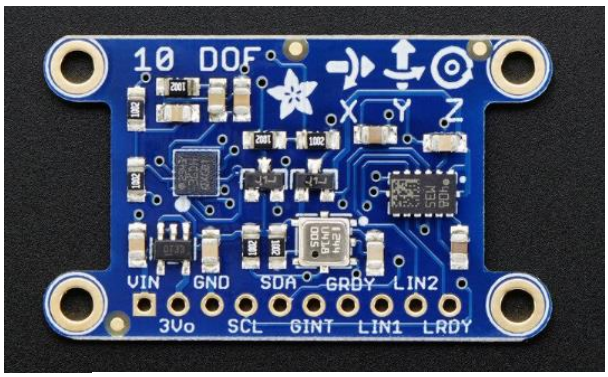


Figur 3-12 Bilde av fuktmåler

3.1.11 Treghet måleenhet – IMU

Vi bruker en IMU fra adafruit [11], se figur 3-13. Denne enheten har fire sensorer integrert og bruker I^2C bus som kan kobles til en mikrokontroller for å innhente dataene. Det finnes bibliotek som er laget for denne sensoren som er enkle å ta i bruk. Siden den også har temperatur måling er dette veldig nyttig da vi får data på hvor varmt det er inne i selve ROVen.

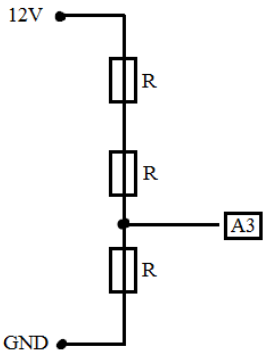
- 3 dimensjonalt gyroskop.
- 3 dimensjonalt kompass.
- 3 dimensjonalt akselerometer.
- Trykk og temperatur måler.



Figur 3-13 Adafruit 10 DOF IMU

3.1.12 Spenningsmåler

En spenningsmåler kan enkelt implementeres ved bruk av en mikrokontroller ved å bruke prinsippet for spenningsdeling, se figur 3-14. Siden vi har en mikrokontroller som kan lese spenninger opptil 5V må vi bruke spenningsdeling for å kunne lese reelle spenningsverdier. Bruker derfor 3 like resistanser som har høy verdi siden dette skaper et strømtrekk. Bruker formel for spenningsdeling:



Figur 3-14 Spenningsdeling

$$V_{A3} = \frac{R}{R+R+R}\, x\, V_{in} = \frac{R}{3R}\, x\, Vin = \frac{1}{3}\, V_{in}$$

Dette gir en tredje del av det opprinnelige spenningsområdet til forsyningen. På denne måten måler mikrokontrolleren 0 til 4V istedenfor 0 til 12V. Uten denne metoden ville ikke mikrokontroller kunne måle hele området, men nå kan den måle helt opp til 15V.

3.1.13 3D-print

Når komponenter skal 3D printes, eksporteres den CAD-genererte 3D filen (i dette tilfellet fra Autodesk Inventor) som .STL til *CURA.* Programpakken *CURA* levert av Ultimaker bruker deretter .STL filen for å generere en lagvis toolpath som 3D-printeren følger. I programvaren kan en definere utallige parameter for å kontrollere printeprosessen. I vårt tilfelle ble det brukt standardprofilen til Ultimaker for printene med 0.6 i dysestørrelse, og fast-profilen ved print med dysestørrelse 0.4. I begge tilfeller var infill satt til 50%, brim og støttestruktur aktivert.

3.2 Metoder

I metode kapittelet vil vi gjennomgå enkelt hvilke metoder vi har valgt å benytte oss av. Sluttresultatet vi har kommet frem til for de enkelte metodene vil vi se nærmere på i kapittel 4.Resultater.

3.2.1 Kommunikasjon

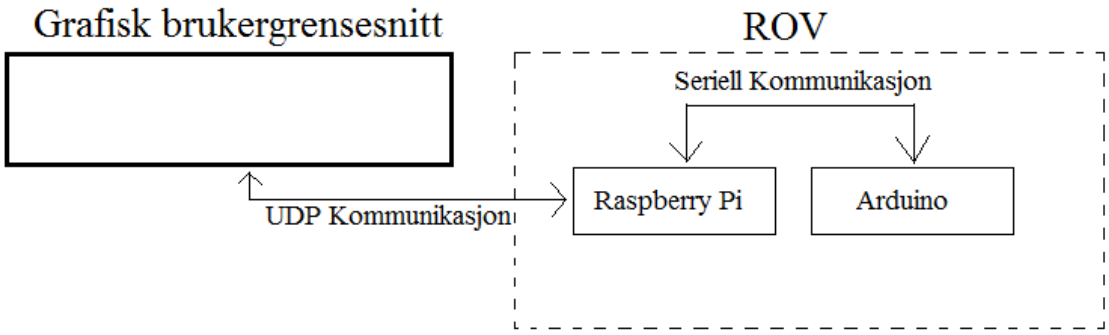
I dette prosjektet har vi flere systemer som må kommunisere sammen. Vi kan dele dette inn i to kommunikasjons kategorier.

- Selve ROVen som kommuniserer mot et grafisk brukergrensesnitt.
- Motsatt vei, altså det grafiske brukergrensesnittet som kommuniserer med ROVen.

Vi benytter to typer kommunikasjons protokoller i vårt prosjekt. Disse to er serielt- og UDP protokoll. I Java bruker vi et bibliotek som heter «JSSC», som er enkelt å ta i bruk for seriell kommunikasjon. UDP står for User Datagram Protokoll, og er en meldingsorientert nettverksprotokoll. Den har fordelen med at den er veldig rask, men ulempen at pakkene den sender ikke nødvendigvis kommer frem.

I selve ROVen ligger det både en mikrokontroller og en Raspberry pi. Disse to bruker toveiskommunikasjon ved hjelp av seriell protokoll for å innhente sensorinformasjon som mikrokontrolleren har og sender kommandoer hvilken styrke thrusterene skal kjøre.

Fra ROV til det grafiske brukergrensesnittet har vi endt opp med å benytte toveiskommunikasjon ved hjelp av UDP. Det grafiske brukergrensesnittet mottar video og sensorinformasjon, samtidig som den sender kommandoer hvilken styrke thrusterene skal kjøre. Figur 3-15 gjengir hvordan vi har valgt att kommunikasjonen skal foregå.



Figur 3-15 Kommunikasjon mellom GUI og ROV

3.2.2 Protokoll

Med tanke på at vi benytter to kommunikasjons metoder, har vi også valgt å sette oss opp to protokoller for å gjøre dataoverføringen på en mest mulig oversiktlig måte for oss. Måten vi har gjort dette er å sende ut ett byteArray på en fast størrelse. Med å gjøre dette har vi mulighet til å tildele en fast plass til hver enkelt verdi, som gjør at vi enkelt kan se egenskapene knyttet til de ulike sensorene.

3.2.3 Programmering

Vi har benyttet oss av Netbeans i utvikling av både GUI og programkode til Raspberry. Netbeans er en IDE til utvikling av programmer med støtte for mange ulike programmeringsspråk. Noen av de er f.eks.: Java, PHP, Python, C/C++, CSS, og flere.

3.2.3.1 GUI

På GUI siden (Styringsenheten) har vi benyttet oss av den integrerte GUI klassen i Netbeans for å utvikle brukergrensesnittet. Her blir det benyttet Swing biblioteket, som gjør det enkelt å flytte på komponentene våre dit vi ønsker, uten å gjøre dette ved kode. Her benyttes «drag-

and-dropp» prinsippet. Om vi ønsker å legge til flere komponenter, gjøres dette også via samme bruksmønster.

Vi ønsker å opprette en hjelpe klasse (DataHandler) som vil ha i oppgave å tildele alle kommandoer sendt fra GUI applikasjonen egne verdier og plassering i protokollen som skal sendes til Rasberryen. På denne måten vil GUI klassen bare inneholde metoder om selve kommandohåndteringen fra bruker, og datahandler tar seg av prosessen som skal til for å klargjøre data til sending over UDP.

3.2.3.2 Raspberry Kode

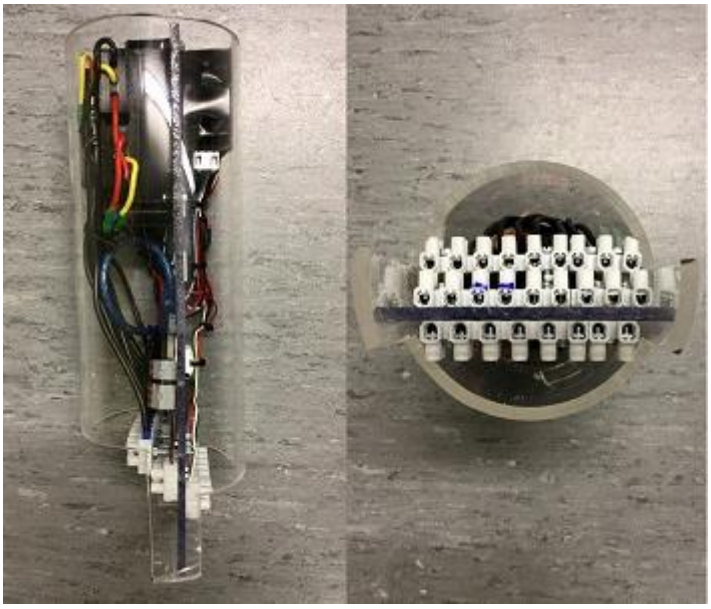
Dette blir bindeleddet mellom de tre systemene. Den vil ha funksjoner som sende og motta data både via seriell og UDP kommunikasjon. Vi vil også legge logikken inn i dette systemet. Dette for å unngå mest mulig logikk programmering i mikrokontrolleren. Mikrokontrolleren sin oppgave er å videresende data og motta data. På denne måten vil man kunne endre logikken dersom det er feil uten å måtte fysisk inn i selve ROVen og oppdatere arduino kode. Logikken ligger hos Raspberry pi siden denne fjern kjøres, se avsnitt 3.2.5 for dette. Raspberry pi koden har en DataHandler klasse står for det meste av datahåndteringen, men med en egen hjelpeklasse som kan ta seg av thruster utregningene.

3.2.3.3 Mikrokontroller kode

Mikrokontrolleren vi har benyttet er av type arduino. Her har vi utviklet koden i det tilhørende programmet «arduino IDE». Denne inneholder integrerte funksjoner som Serial.write(), som er en funksjon får å sende data til andre enheter via seriell kommunikasjon. (ref protokoll i 3.1). Det er her alle komponentene kobles direkte til. Av sensorene vil vi lese av dataen via den analoge lesefunksjonen som er integrert i Arduino biblioteket, «AnalogRead()». Det vil også legges inn metoder for thruster- og lysstyring. Med tanke på at vi har latt ansvaret for utregninger ligge i raspberryen, er det bare mapping av verdiene som trengs å gjøres her. Siden vi benytter byte[] som protokoll, og har tildelt en byte pr komponent, er det bare verdier mellom 0-255 som kan tildeles hver enkel komponent. Siden thruster og lysstyrken har ett arbeidsområde mellom 1100 – 1900 lar vi verdiene holdes mellom 0-255 fra raspberryen ned til mikrokontrolleren, og en mapping funksjon som tildeler riktig verdi innenfor gitt arbeidsområde. På denne måten skal det da bare være nødvendig med 1 byte pr komponent (lys/thruster), og ikke to.

3.2.4 Elektriske komponenter og beregninger

For å få et velfungerende elektrisk system har vi laget en boks som inneholder alle de elektriske komponentene vi trengte til dette prosjektet. Med en slik boks er det bare å koble utstyret, som f.eks. thrustere og sensorer, rett på rekkeklemmene, se figur 3-16.



Figur 3-16 , Bilde av elektronikk boks

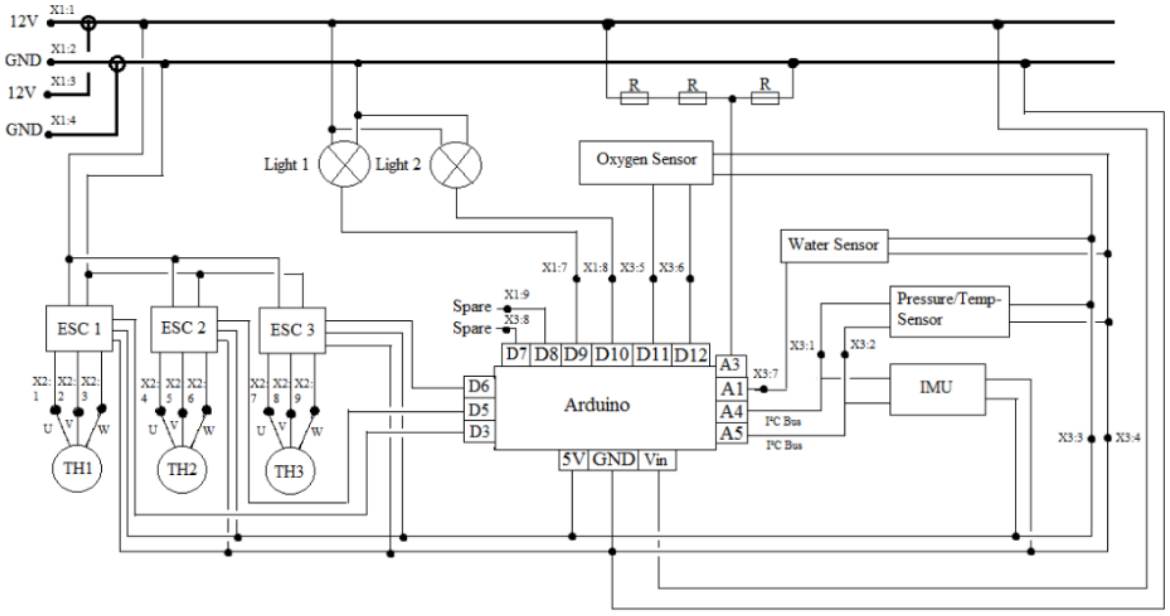
Eksterne utstyr som er medberegnet i dette prosjektet er:

- Oksygen sensor.
- Trykk- og temperatur sensor.
- 3 stk Thrustere.
- 2 stk Lys.
- Vann sensor.
- IMU.
- Egen temperatur sensor for måling av innvendig temperatur.

Vi har tenkt på videreutvikling av prosjektet, slik at det finnes to digitale reserverte utganger på rekkeklemmene for slikt. Disse er X1:9 og X3:7. Det er også mulig å bruke oksygen sensoren på I^2C bus som er utganger X3:1 og X3:2. Dette fordi sensoren både kan bruke seriell og I^2C bus, men grunnet dårlig informasjon på nettet har vi valgt å bruke serielt. Om dette gjøres har man fire utganger tilgjengelig, se figur 3-17.

Rekkeklemmer	1	2	3	4	5	6	7	8	9
X1:	Strømkabel 12VDC	Strømkabel 12VDC	Strømkabel GND	Strømkabel GND	Lys 12VDC	Lys GND	Lys nr.1 D9	Lys nr.2 D10	Spare (D7)
X2:	Thruster 1: U	Thruster 1: V	Thruster 1: W	Thruster 2: U	Thruster 2: V	Thruster 2: W	Thruster 3: U	Thruster 3: V	Thruster 3: W
X3:	Trykksensor A4 (SDA)	Trykksensor A5 (SCL)	5VDC	GND	Oksygen-sensor D11	Oksygen-Sensor D12	Spare (A1)	Spare (D8)	

Figur 3-17 Rekkeklemme tabell.



Figur 3-18 Koblingsskjema for styring- og informasjonskrets.

Det meste av utstyret vi bruker i dette prosjektet bruker 12V DC utenom noen av sensorene som bruker 5V DC. Siden mye av utstyret har relativt høyt strømforbruk er det nødvendig å gjøre kalkulasjoner for å beregne størrelse på strømkabel.

Utstyret med maksimalt oppgitt strømforbruk er som følger:

- 3 stk T200 thrustere, hver enkelt har oppgitt i dokumentasjonen at den bruker 25A maksimalt [105], men i deres dokumentasjon har de også kurver over hvilket strømbruk thrusteren har ved forskjellige spenninger. Ved 12V har den et maksimalt strømtrekk på 15.4A. Dette blir da totalt: $15.4A\ x\ 3 = 46.2A$
- 2 stk Led lys, hver enkelt har en effekt på maksimalt 15W [103], dette blir $P = U\ x\ I \rightarrow I = \frac{P}{U} = \frac{15W}{12V} = 1.25A$. Dette blir totalt $2\ x\ 1.25A = 2.5A$.
- Trykk- og temperatur sensor, 1.25mA [102].
- Oksygen sensor, 18.3mA [100].

- Raspberry pi Model 3B, 700-1000mA [101].
- Arduino Nano, oppgitt i dokumentasjon at den bruker 40mA per I/O pin [104]. Vi bruker 14 utganger, dette blir $40mA\ x\ 14 = 560mA$. Med dette er det ikke tatt hensyn til hva selve arduino nano trekker av strøm, men dette er det lite dokumentasjon på internett. Det er også så lite i helheten at det har lite betydning.

Dette blir totalt $46.2A + 2.5A + \frac{1.25mA}{1000} + \frac{1000mA}{1000} + \frac{560mA}{1000} = \mathbf{50.26A}$

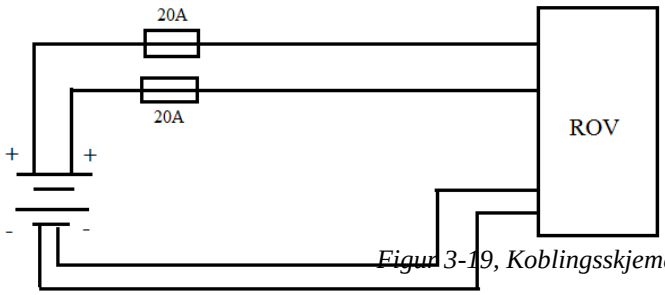
Siden thrusterene aldri kjøres med maksimal styrke hele tiden må vi beregne noe mindre strømtrekk. Dersom vi skulle tatt høyde for maksimalt strømtrekk hadde vi måtte bruke en kabel med høyt tverrsnitt, noe som er upraktisk på en ROV.

Vi hadde få valg når det gjaldt strømkabel og endte derfor opp med en strømkabel som er $4x4mm^2$. Dette blir parallellkoblet slik at det blir i praksis en $2x8mm^2$kabel. Ifølge Montørhandboken Nek 400:2014 s.201-202 er det krav til at parallellkobling av ledere har samme lengde og at disse vernes hver for seg.

Siden det er krav om å verne hver enkelt kurs bruker vi 20A sikringer, se figur 3-19.

Ifølge Prof. Houxiang Zhang ved NTNU Ålesund kan vi belaste kabel med 6A pr kvadratmillimeter.

Vi bruker $8mm^2$ leder. Denne kan belastes med $8\ x\ 6A = 48A$.



Figur 3-19, Koblingsskjema hovedstrøm

Dette blir $P = U\ x\ I \rightarrow 50.26A\ x\ 12V = 603.12W$.

Spenningsfall

Bruker Montørhandboken Nek 400:2014 s.197 for beregning av spenningsfall

$$\Delta U = \frac{P\ x\ l\ x\ 2\ x\ \rho}{U_0\ x\ A} = \frac{0.6kW\ x\ 15m\ x\ 2\ x\ 0.018}{12V\ x\ 8mm^2} = 0.003375V$$

Dette blir $\Delta u = \frac{\Delta U\ x\ 100}{U} = \frac{0.003375V\ x\ 100}{12V} = 0.028\%$

ΔU = Spenningsfall i Volt	ρ = esitivitet $\left(\frac{\Omega\ x\ mm^2}{m}\right)$, 0.018 $\left(\frac{\Omega\ x\ mm^2}{m}\right)$ for kopper.
Δu = Spenningsfall i prosent	$l = Kabellengde\ (meter)$
$P = Lasteffekt\ (kW)$	$A = Ledertverrsnitt\ (mm^2)$
$U_0 = Fasespenning$	

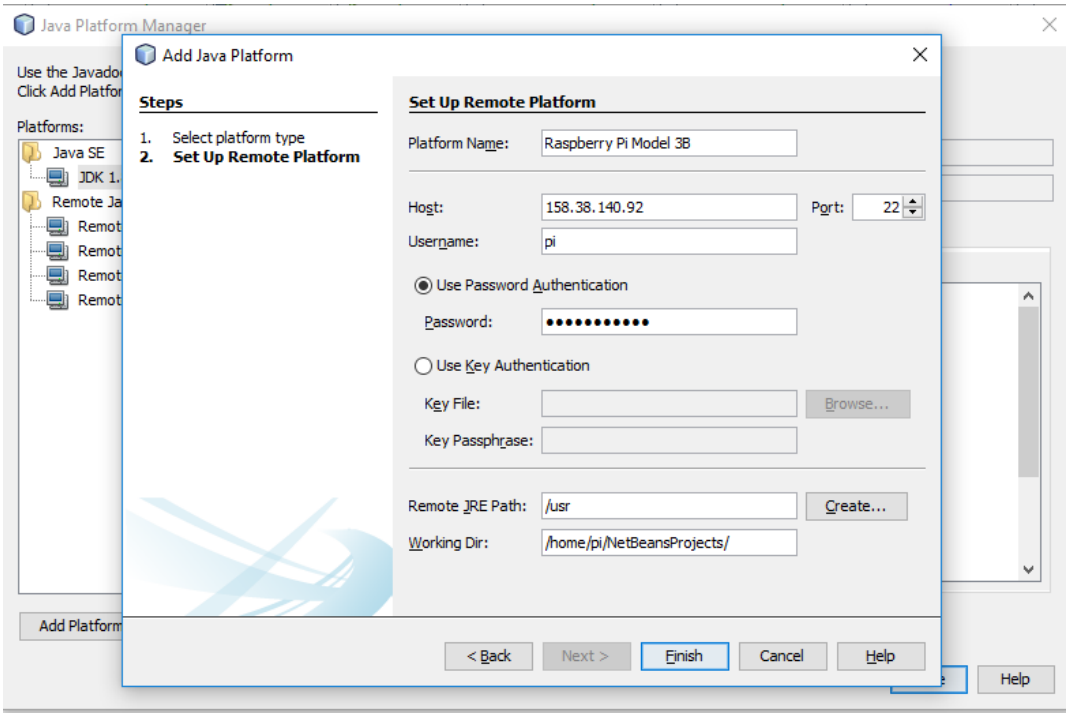
3.2.5 Fjernkjøring av Java program

Siden Raspberry Pi ligger inne i selve ROVen, var det nødvendig for oss å finne en metode for å kjøre Java programmet vårt utenifra. Dette fordi det er lite praktisk å måtte ta ut selve elektronikken for å kunne starte opp programmet. Dette løste vi i Netbeans, da det finnes muligheter for å opprette eksterne plattformer. Med dette kan man sitte på hvilket som helst nettverk å koble seg på, for så å kjøre programmet.

Dette finner man under $Tools \rightarrow Java\ Platform \rightarrow Add\ Platform$ for så å velge Remote Java Standard Edition. Her inne må man fylle ut informasjon om plattformen man skal koble seg på, som f.eks. IP adresse, brukernavn, passord og plassering av Java filer, se figur 3.2.5.

Brukernavnet og passord tilhørende Raspberry Pi som er brukt i prosjektet er:

- Brukernavn: pi
- Passord: vegardrogne



Figur 3-20 Oppretting av ekstern plattform Netbeans

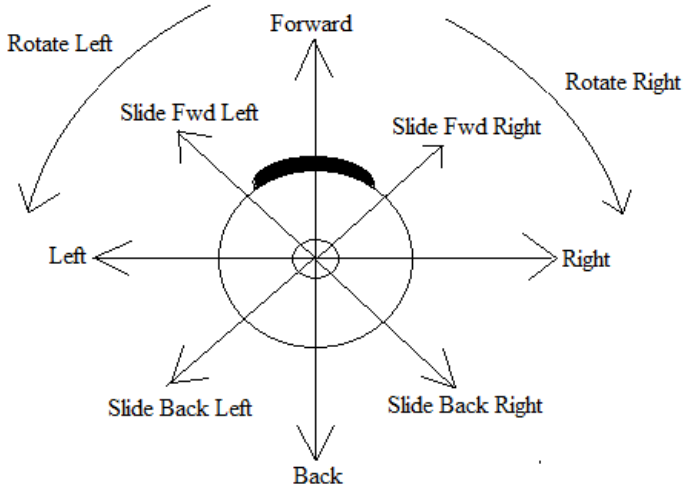
Et annet problem er at vi trenger en fast IP adresse. Dette kan løses ved at vi bruker en ruter for så gå inn på konfigurasjonene til ruteren. Her kan man endre IP adressen, slik at uansett hvilket nettverk man kobler seg på kan man kommunisere med Raspberry Pi.

3.2.6 Thruster kalkulasjoner

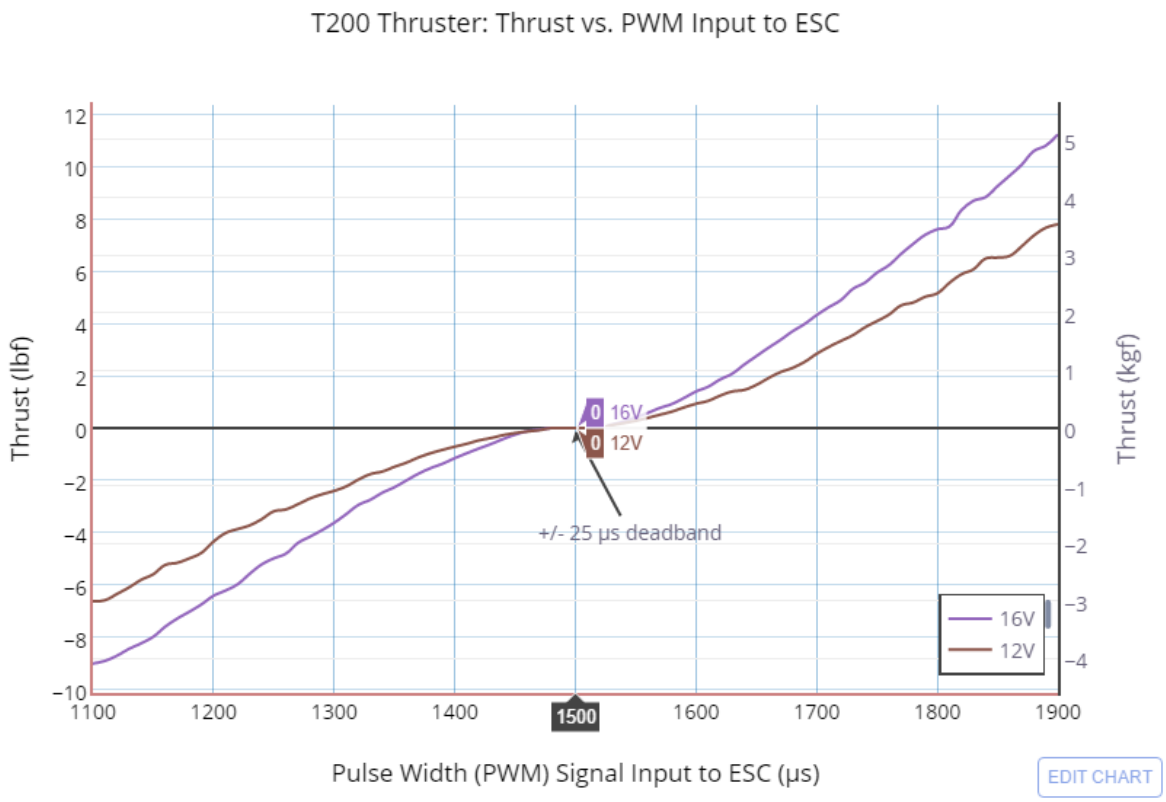
Vi har i vårt prosjekt valgt å bruke tre thrustere. Disse er plassert 120° i forhold til hverandre. Dette gjorde vi for å kunne kjøre ROVen i alle retninger. Når vi startet prosjektet hadde vi to ønsker. Dette var å kunne kjøre ROVen i alle retninger uten at selve ROVen roterte, og den andre er at den skal rotere. Se vedlegg for detaljert thruster kalkulasjoner for hver enkel retning.

De ulike retningene vi ønsker er, se figur 3-21.

- Fremover.
- Fremover høyre uten rotasjon.
- Fremover venstre uten rotasjon.
- Høyre.
- Venstre.
- Bakover.
- Bakover høyre uten rotasjon.
- Bakover venstre uten rotasjon.
- Rotasjon til venstre.
- Rotasjon til høyre.



Figur 3-21 , Bilde av thruster med de ulike retningene.



Figur 3-22 Kurve av inn signal og vannforskyvning av thruster

For dette trenger vi å regne ut vektorene for de ulike thrusterene, slik at thruster kreftene til hver enkelt thruster er bestemt ut i fra hvilken retning vi ønsker. Siden thrusteren gir noen lunde lineær graf for vannforskyvning for hvilken input de får, se figur 3-22, vil kalkulasjonene være noe teoretiske. Man ser at vannforskyvning til en thruster varierer hvilken vei den kjører, det vil si at den er mer vannforskyvning av å kjøre fremover enn bakover.

Dersom dette gir et stort avvik når vi får testet ROVen, vil vi kompensere dette i programmeringen. Dette blir gjort ved kalkulasjoner i software, slik at vannforskyvningen til thrusterene er kompensert ut ifra hvilket inn signal de har.

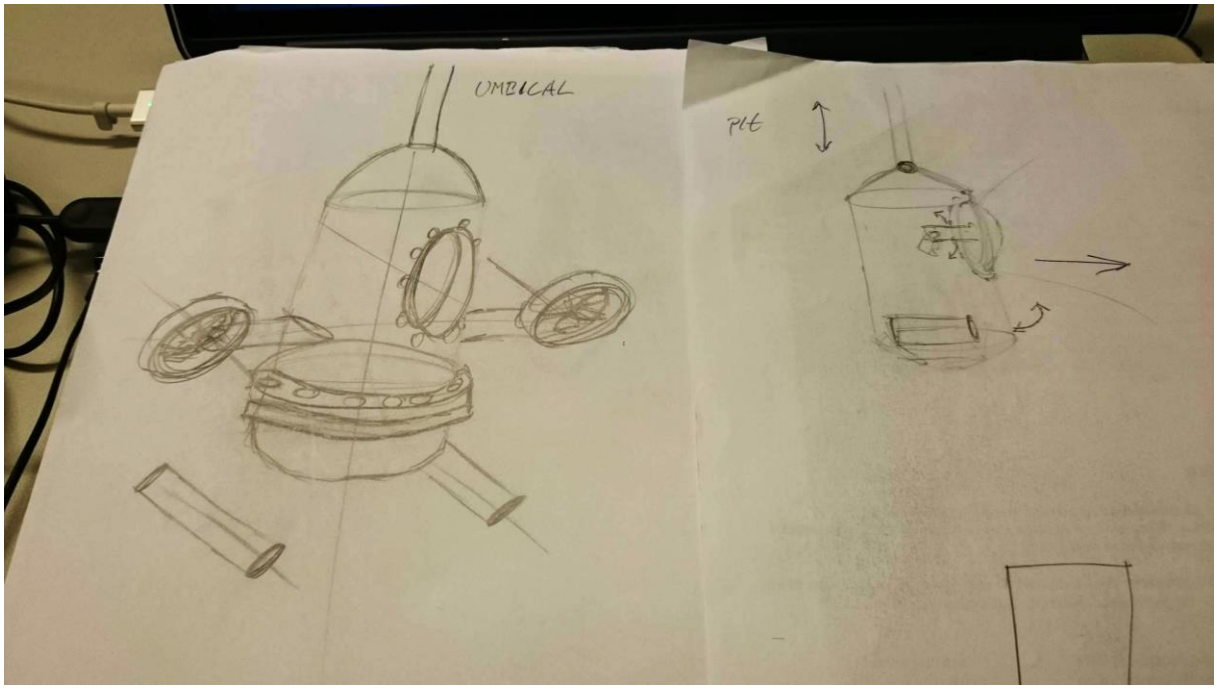
3.2.7 Utvikling av design ROV

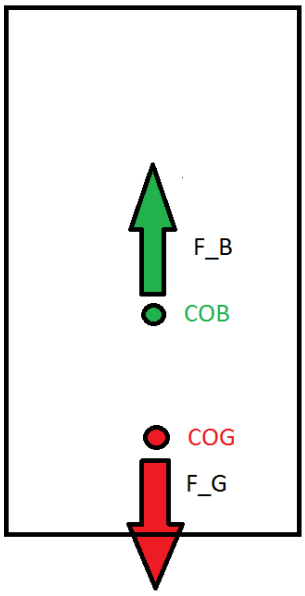
Grunnkonsept

- ROV skroget bygges i lett tilgjengelige og kostnadseffektive PE deler brukt innen VA(Vann og Avløp). Komponentene sammenføyes ved sveising etter gjeldende industristandarder ved bruk av varmespeil og ekstrudersveis.
- Alle VA-deler brukt i sammenstillingen har en trykkgodkjenning minimum PN16, som godkjenner arbeidstrykk på 16 [bar] (ca. 160 [m] vannsøyle). Dette er betydelig mer enn trykket som ROV utsettes for ved designdybde på 10 [m] (ca. 1 [bar]). Dette forenkler designarbeidet, da en allerede kan garantere for integriteten til skroget.
- Bruke mest mulig standardiserte deler for å presse ned kostnader og holde høg effektivitet i byggeprosessen.

Konsept Mk 0

- Etablerer grunnprinsippene for videre iterasjoner.
- Skrogdesign som kan fremstilles enkelt og kostnadseffektivt.
- Utnytter dykkerklokkeprinsippet med et indre mottrykk som begrenser eventuelle lekkasjer og gir i kombinasjon med en fuktsensor i bunn av hovedrør mulighet til å stoppe nedsenking før utstyr tar skade av vanninntrenging. (Den stigende vannmassen vil komprimere luften på innsiden ved en lekkasje når trykket på utsiden stiger)



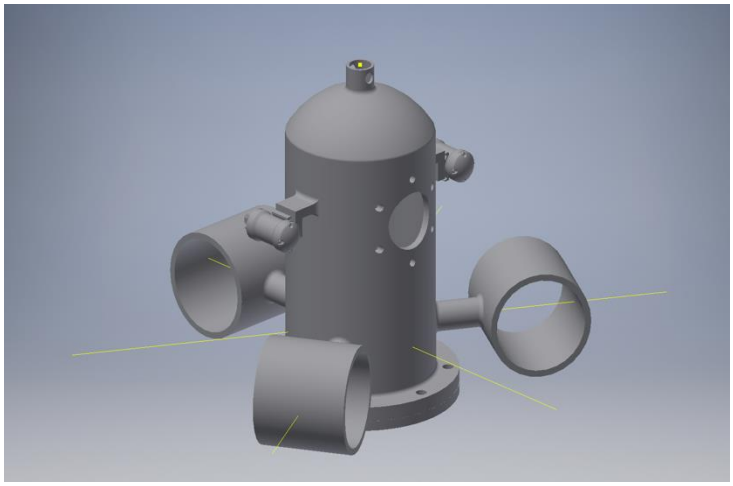


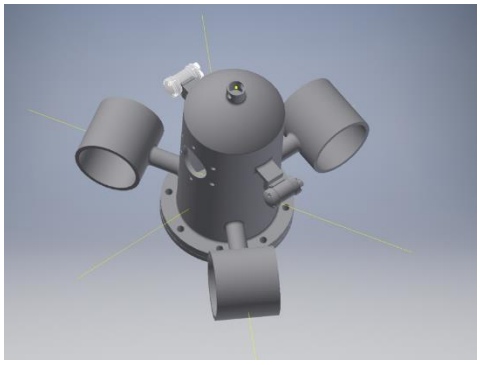
Illustrasjonen til venstre viser hovedprinsippet som skal sikre stabiliteten til fartøyet. Så lenge COB (Center Of Boyancy) er over COG (Center Of Gravity), og F_G (Force_Gravity) er større enn F_B (Force_Boyancy) vil fartøyet ha negativ oppdrift (henge i umbilical) og holde seg i rett orientering i vannet. Grunnet den designmessige symmetrien til fartøyet befinner COB og COG seg langs z-aksen i senter av fartøyet, og forhindrer derfor utfordringer med videre balansering.

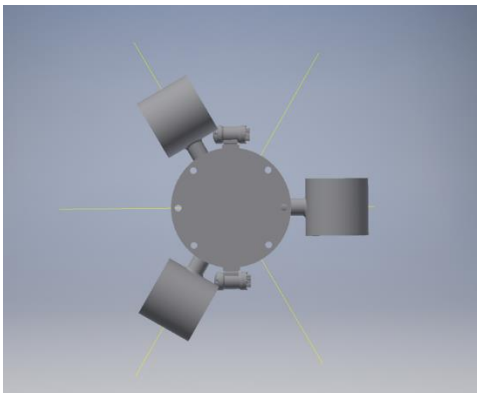
Hovedutfordringen blir å balansere massen og oppdriften rett, for å få en ROV som henger i umbilical, men ikke har så liten oppdrift i at det blir umulig for thrusterene å manøvrere den. Vekt blir senere en forholdsvis viktig designfaktor når ROV skal ut av vannet, og henge i plattformen.

Konsept Mk 1

- Tar grunnprinsippene fra Mk 0. Skroget bygges i sin helhet i PE inkludert monteringsrør for thrustere.
- Vinkel på thrustere skråstilles for å kunne utnytte vektorbasert propulsjon og bedre kunne kompensere for avdrift grunnet strømninger i sjøen. (Se vedlegg 9.2 for thrusterkalkulasjoner)

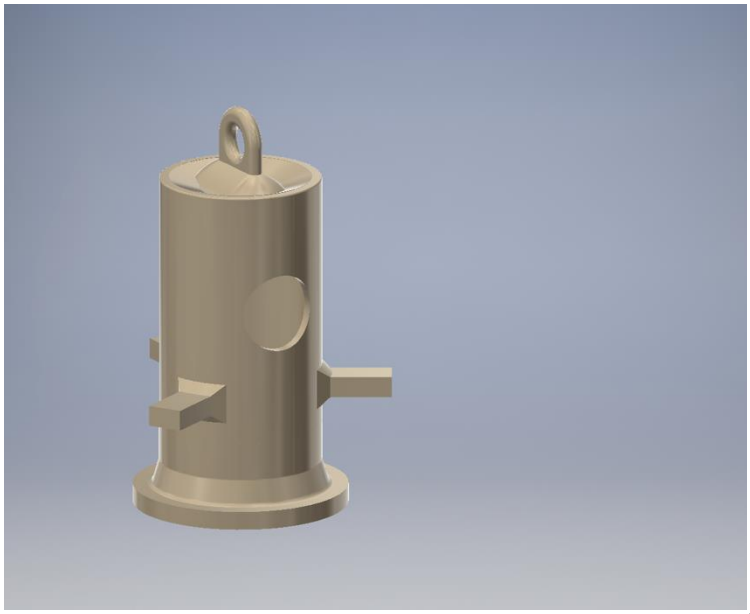




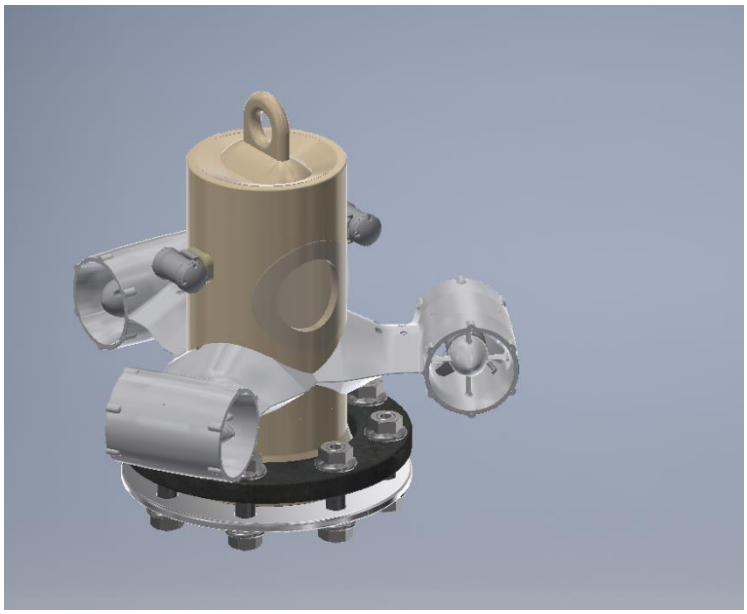


Konsept Mk II

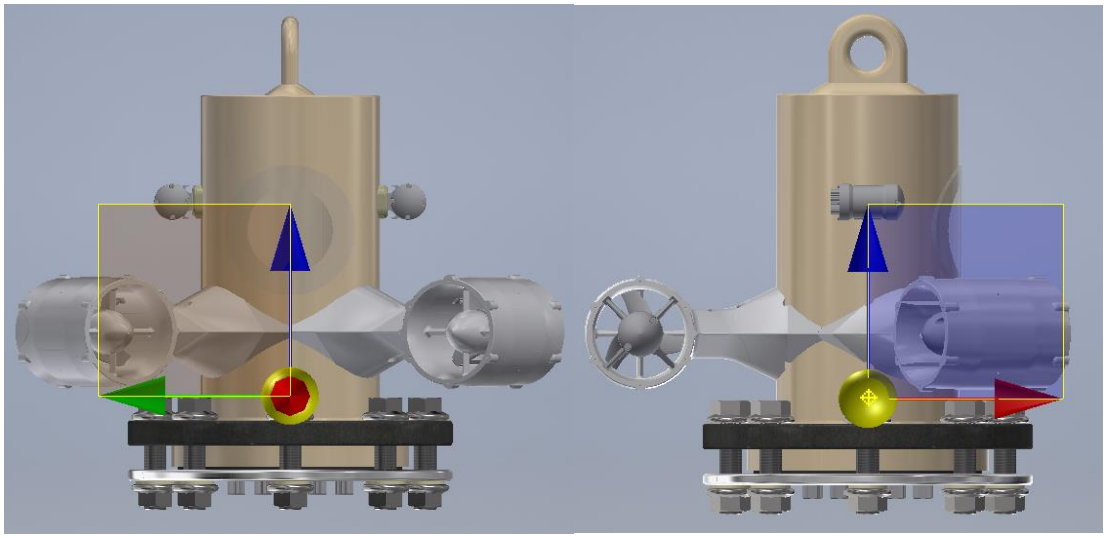
- Bygger videre på Mk I, men tar konseptet denne gangen til produksjonsklar prototype.
- Alle komponenter i kontakt med sjø er utført i plast, aluminiumslegeringer eller rustfritt stål for å motstå korrosjon.
- Bunnflens i aluminium senker COG ytterligere og fungerer også som heatsink for den innvendige elektronikken for utveksling av spillvarme til sjøvannet. Hull for gjennomføring av kabler. Masseproduseres ved hjelp av vannskjæring for å minimere etterarbeid.
- Eksterne komponenter (Thruster casing, Festebraketter for lys) 3D-printes i PLA i prototypefase, men kan med enkle modifikasjoner sprøytestøpes i ABS for senere masseproduksjon.



Hovedrøret(Hull) med festepunkt for thrustere og forankringspunkt på topp, alt i PE



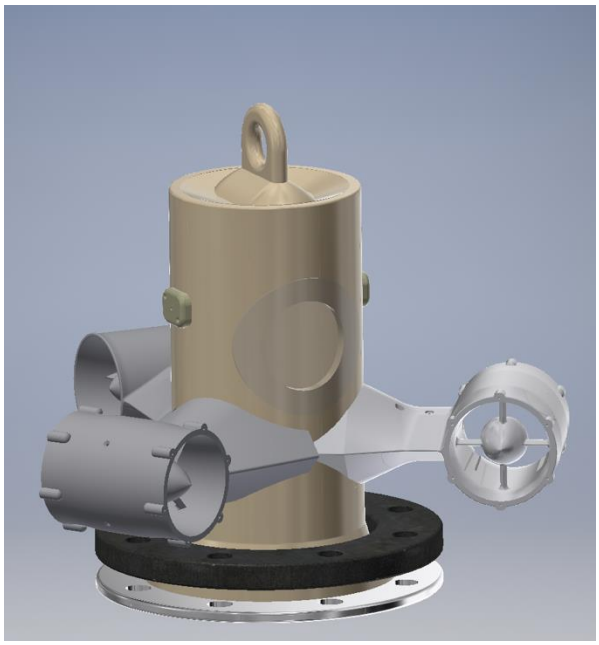
Prototype-sammenstilling.

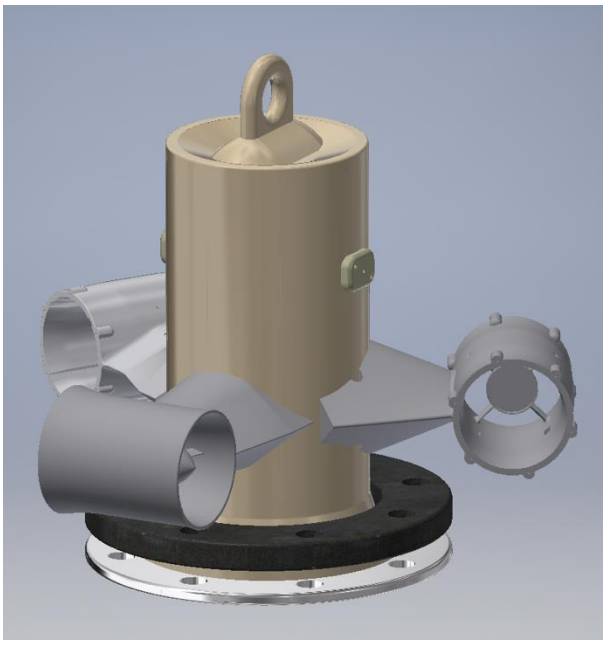


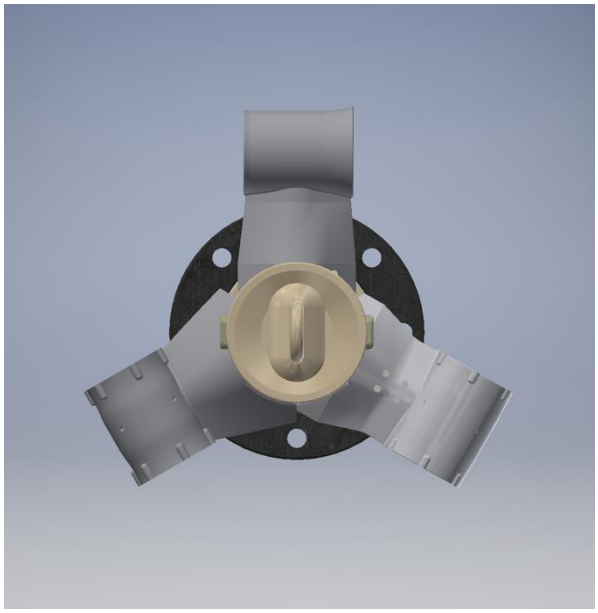
COG til sammenstillingen, COB vil befinne seg i senter av hulrommet definert av innsiden av hovedrøret, litt over thustere.

Thrustercasing

Thrustercasingen gikk igjennom 3 iterasjoner før den endelige versjonen var klar.

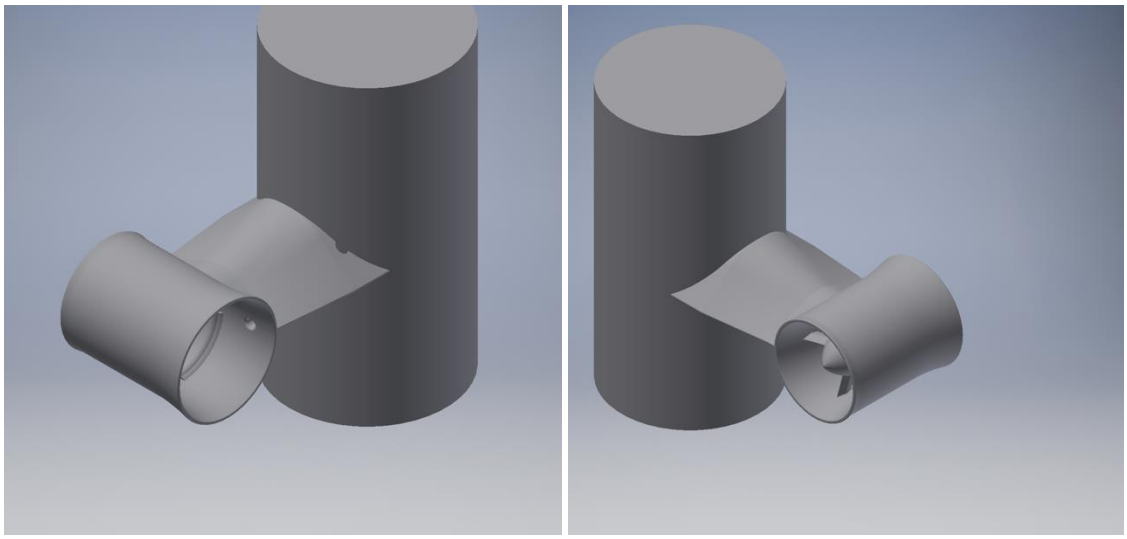




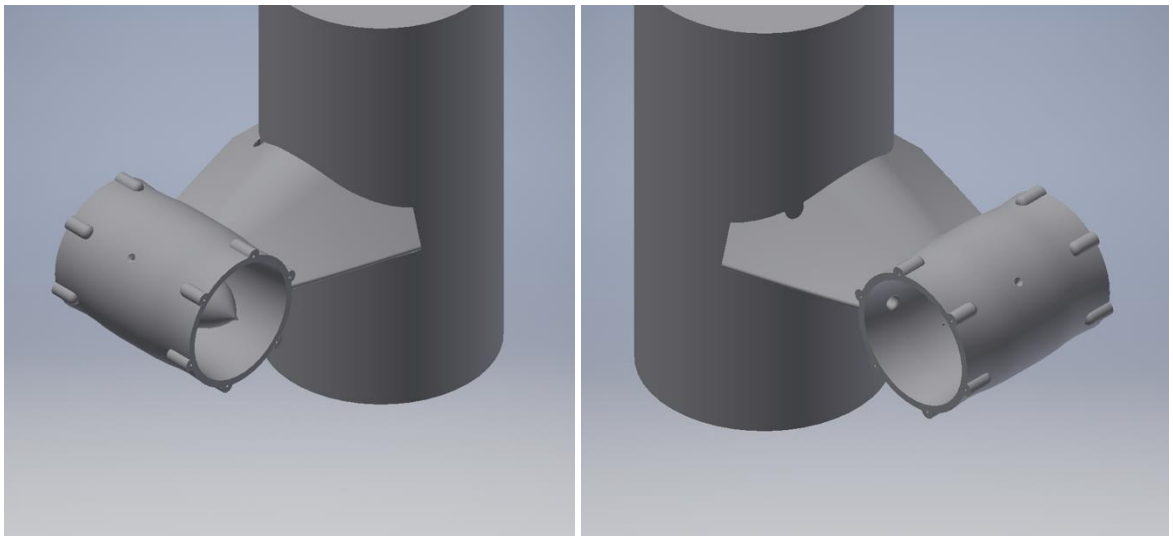


Fra nede til høgre med klokka:
Mk 3, Mk 1 og Mk 2.

Mk 1 ble litt for grovbygd og var ikke symmetrisk og vil derfor gi ulik drag-kraft avhengig av retning. Tidlig designstadie; ikke fullført og vraket til fordel for Mk 2.

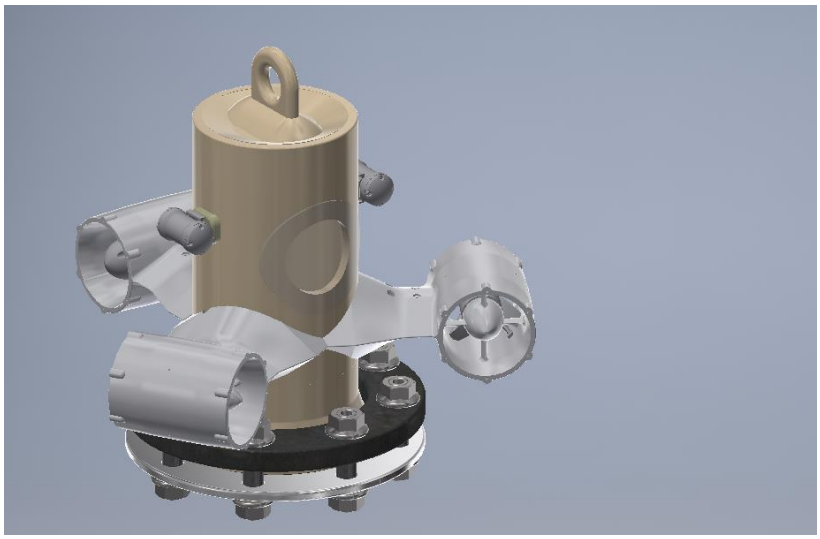


Mk 2 ble designet symmetrisk for å bedre kunne brukes rundt hele ROV'en og fortsatt gi mest mulig like drag-krefter som blir forårsaket av strømningene i sjøen. Den var fortsatt litt for stor, og tok dermed lang tid å printe. (51 timer) og brukte mye fillament.(72 m) ved 50% infill



Mk 3 (*Mk IV* på ferdig modell)

Er den ferdige modellen som blir brukt på ROV. Denne har et symmetrisk design som gir mest mulig lik drag-påvirkning uavhengig av retning. Denne versjonen er også modifisert for å redusere printetid og fillamentforbruk (48 timer, 68 m), samt bedre de hydrodynamiske egenskapene ved å gjøre den slankere.



3.2.8 Simulering

Fysikksimulering ble gjennomført på Konseptet Mk 1 i *AgX* fysikkmotoren med grafisk output i *Jmonkey* for å verifisere ønsket oppførsel i vannet.

3.2.9 Testing av vanninntrenging

Det ble gjennomført 4 tester for å verifisere og kunne garantere at ROV er vanntett før elektronikken ble satt inn. Se vedlegg for testdokumentasjon.

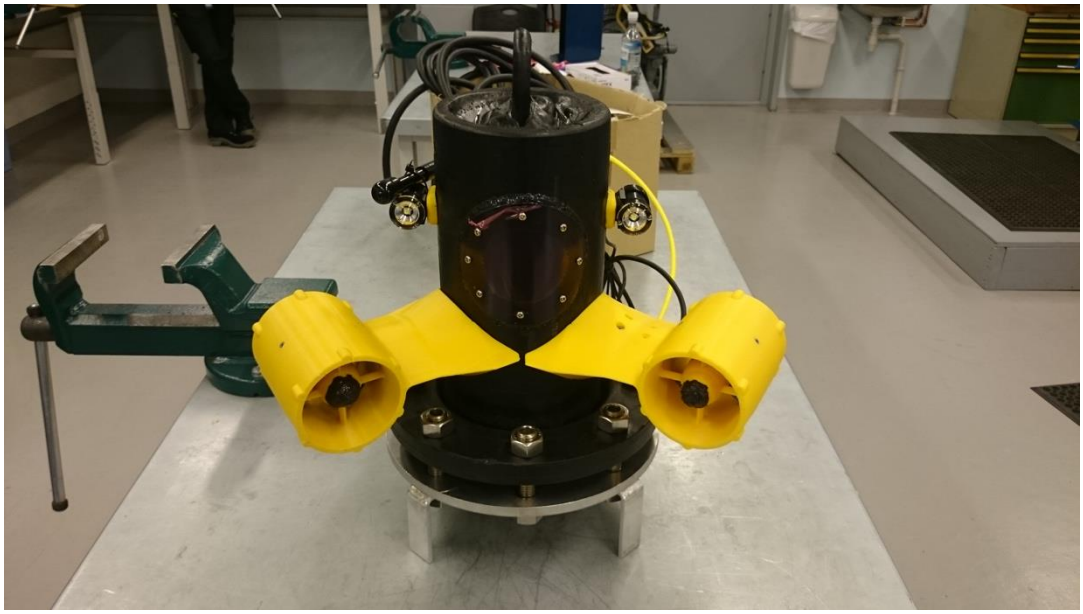
4 RESULTATER

I dette kapittelet vil vi gi ett godt innblikk i alle resultatene og løsningene vi har endt opp med i slutten av prosjektet vårt. Alle resultater er kommet frem til med anvendte metoder beskrevet i kapittel 3.

4.1 Design utvikling

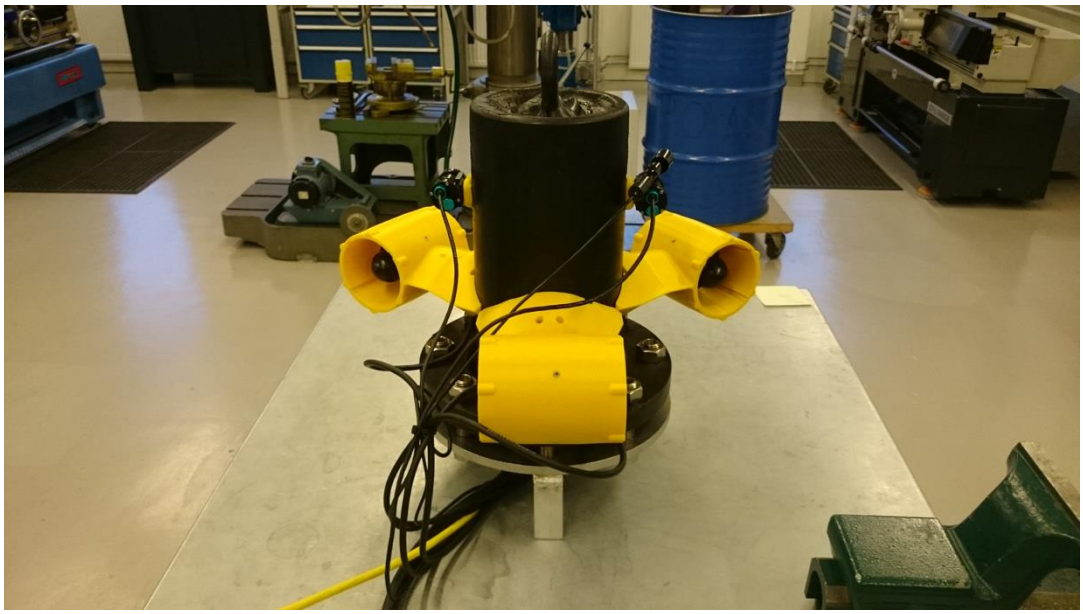
4.1.1 Design

Den ferdige ROV'en

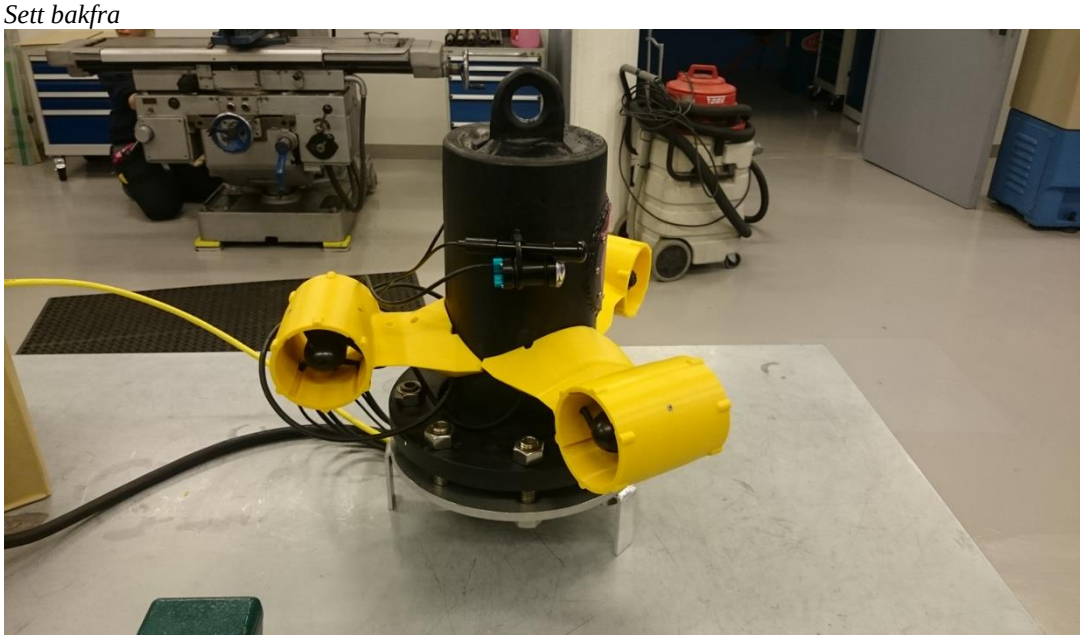


Sett forfra

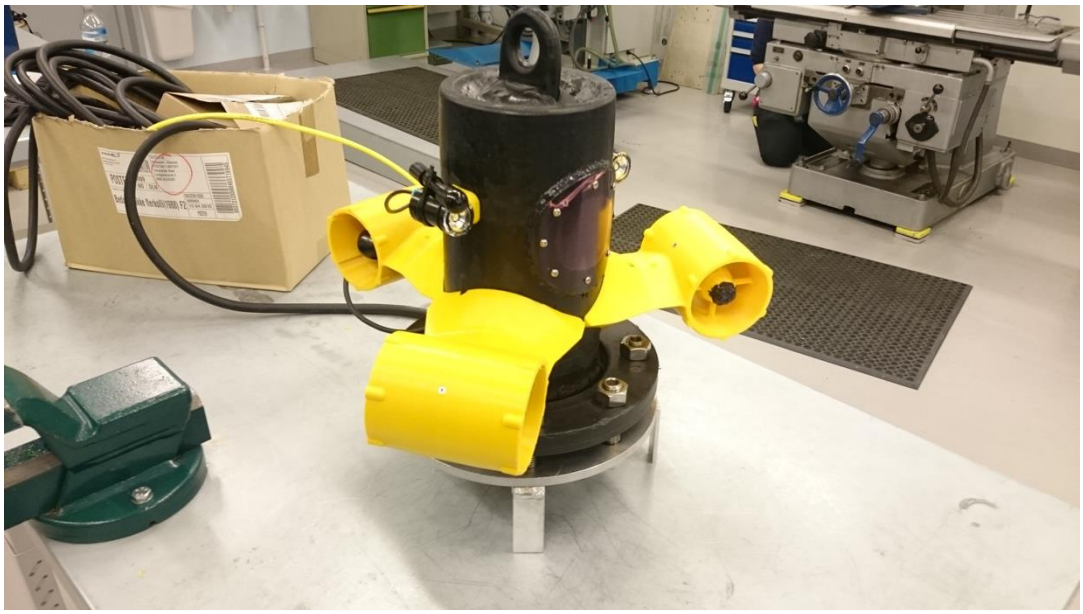
Sett fra høyre



Sett bakfra



Sett fra venstre

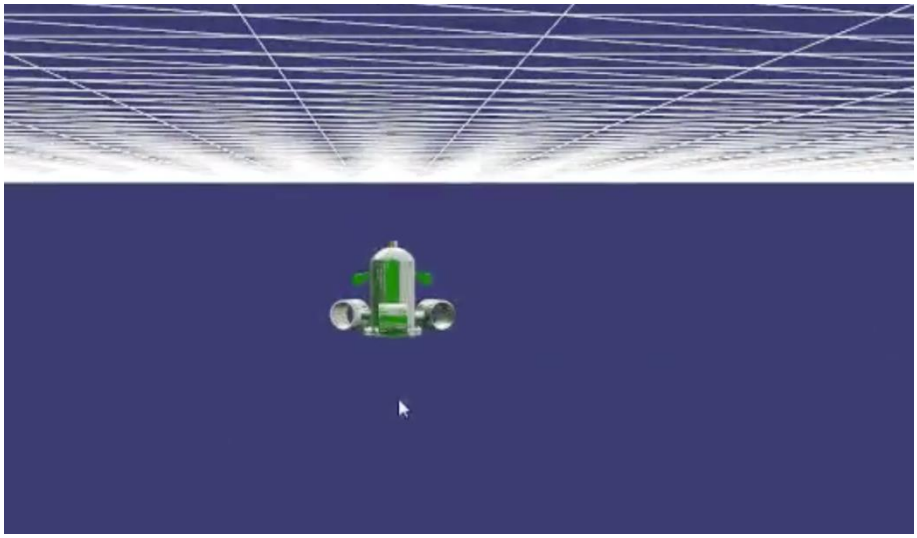


Perspektiv

Den ferdige ROV'en påmontert alle utvendige komponenter.
Som forventet ble ROV noe tung å håndtere når den er ute av vanndomenet, men i vanndomenet viser den fine karakteristikker og har omlag 1 [kg] (10 [N]) i negativ oppdrift, alt utstyr inkludert. Om ROV skulle vert gjort lettere i tørrvekt må det fortrengte volumet reduseres(ROV gjøres mindre), eller massen på ROV gjøres regulerbar (Ballast) for å beholde samme karakteristikk i vannet. Den litt større massen gir også ROV en viss massetreghet, som virker både til sin fordel (motstand mot å bli forflyttet av strøm) og en bakdel (thrustere må jobbe hardere for å flytte farkosten).

4.1.2 Simulering

ROV viste egenskaper som forventet og holdt seg vertikal i vannet, ved påvirking av ytre krefter rettet den seg opp igjen til vertikal stilling. Dette verifiserer designkonseptet som holdbart for å få den oppførsel i vanndomenet som vi ønsker.



4.1.3 Testing

Som verifiseringsmetode for kritiske punkt i designet ble det utnyttet FEA for å kontrollere at designet i teorien skal tåle den påkjenningen som blir påført. Videre ble det benyttet real-life testing av festebolter etter modifikasjon, og test av vanntettheten til skroget.

4.2 Software utvikling

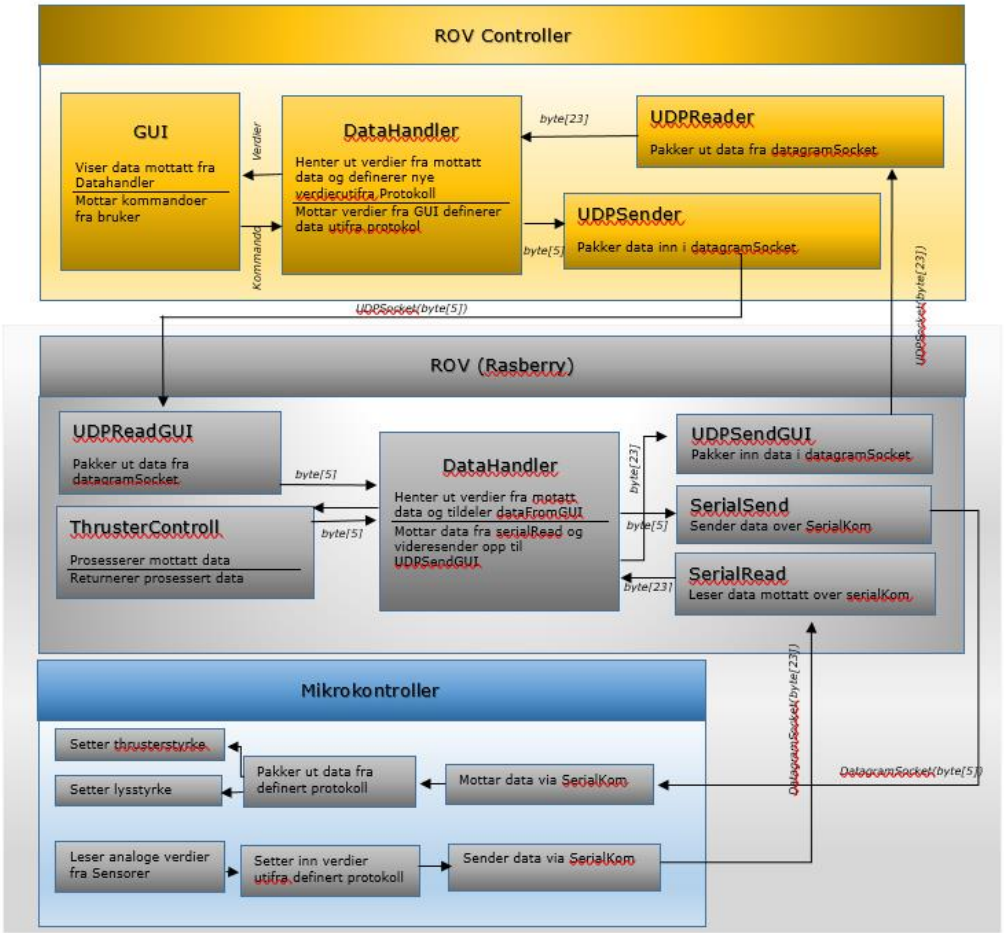
I følgende kapittel vil vi gjennomgå den endelige software løsningen vi endte opp med i vårt system. Vi vil først gjennomgå hvilke løsninger vi endte opp med i 4 hovedkategorier:

- Systemets oppbygging
- Kommunikasjon
- GUI
- Logikk

4.2.1 Systemets oppbygging

Med tanke på at vi har tre systemer som skal kommunisere og fungere opp mot hverandre, er det mye dataflyt som foregår på tvers av systemene. I flytdiagrammet i figur 4-1, viser vi en overordnet oversikt over hvordan dataflyten mellom alle de tre undersystemene er satt sammen, og fungerer som ett sammensatt system. De tre systemene er :

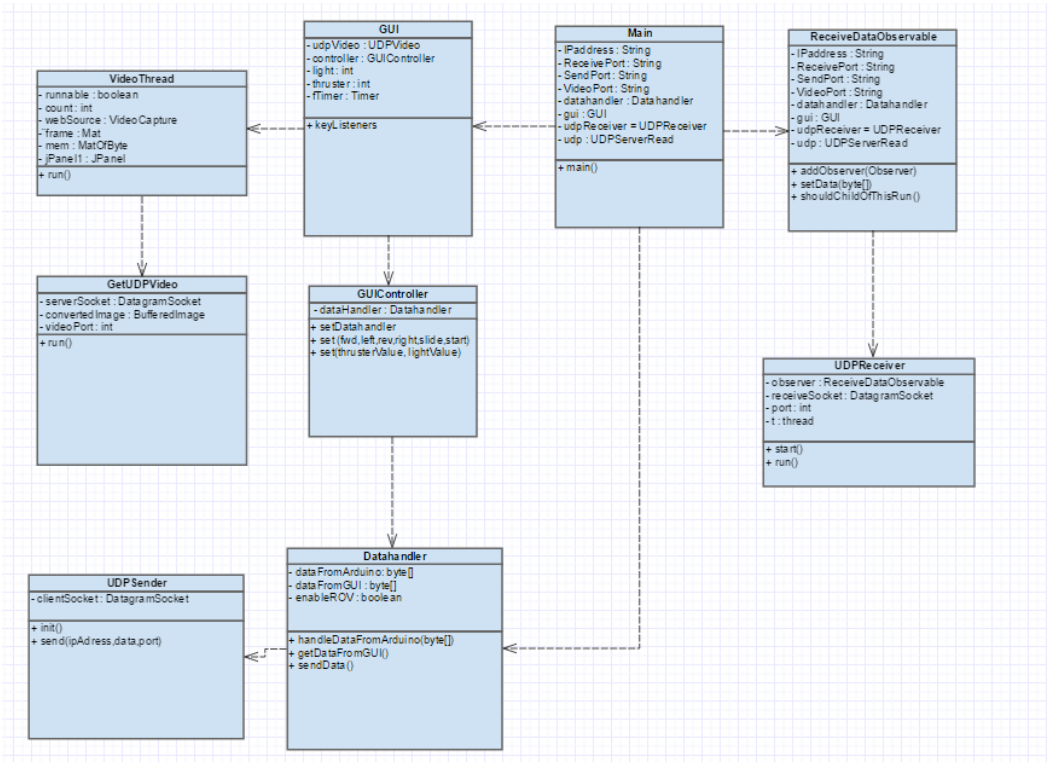
- ROV Controller (GUI)
- ROV (Raspberry siden)
- Mikrokontroller



Figur 4-1 Flytdiagram over dataflyt på tvers av de tre systemene.

4.2.1.1 ROV Controller

Den ferdige oppbyggingen av dette systemet vises i UML diagrammet i figur 4-2. Her vises alle klassene applikasjonen inneholder, og alle funksjoner de ulike klassene innehar.

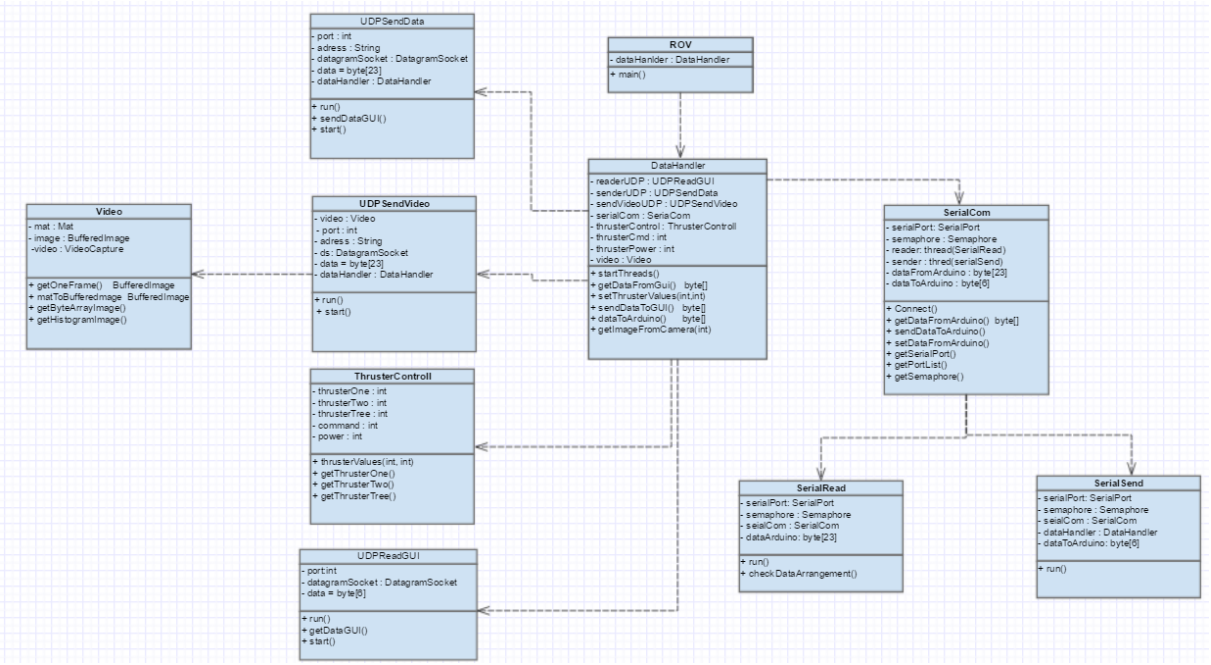


Figur 4-2 Klassediagram over Controller system

Vi har valgt å ha en Main klasse som oppretter de nødvendige klassene, og kjører programmet direkte fra denne. Main oppretter datahandler, GUI og ReceiveDataObservable, som videre oppretter sine underliggende klasser. På denne måten har vi klart å følge cohesian og coupling prinsippene på en god måte. Dette gjør at det er lite kjennskap mellom alle klassene. Hver enkelt klasse har på det meste bare kjennskap til to andre klasser, og holder informasjonen de ikke har nødvendigheter til å ha kjennskap om utenfor. En viktig funksjon i GUI klassen var at den må både kunne sende og motta data fra GUI applikasjonen. Den skal kunne sende kommandoer fra bruker, men også kunne automatisk motta data registrert i ROV (mikrokontroller). Samtidig som sensorverdier blir mottatt fra Mikrokontroller, mottar den en live bildestrøm fra VideoThread. Det vil si at det er tre kommunikasjonsprosesser som skal kunne foregå samtidig. Dette har vi løst med å opprette Videostrømming, UDPSend og Read som tre ulike tråder. Det vil da ikke være noen forstyrrelser mellom prosessene, og det vil heller ikke bli noe påvirkning på de andre trådene om den ene prosessen låser seg.

4.2.1.2 ROV

Hvordan vi har satt sammen systemet på ROV enheten, er vist i klasediagrammet i figur 4-3. Her har vi latt dataHandler være bindeleddet mellom alle de forskjellige kommunikasjonsplattformene. Datahandleren mottar protokollen sendt fra GUI og sender den videre til ThrusterControll for logikk beregninger. Verdiene her blir da sendt tilbake og videre ned til Mikrokontroller via Seriell kommunikasjon. Utfordringen her var at det ikke bare skulle sendes og mottas data mellom Raspberry og GUI, men det skulle også være en sende/motta kommunikasjon mellom Raspberry og Mikrokontroller. Dette har vi løst på samme måte som i GUI med Tråder, men også inkludert en Semaphore for å være sikker på at de ulike trådene får kjøre.



Figur 4-3 Klassediagram over ROV systemet

4.2.1.3 Mikrokontroller

Siden vi har valgt å legge all logikk i Raspberryen, er det lite prosessering som gjenstår til mikrokontrolleren. Mikrokontrolleren tar seg av styringsfunksjoner til thrustere og lys, og innhenting av sensor data. Styringsfunksjonen til thrusterene baseres på verdier mellom -100 til 100 som mottas i protokollen fra Raspberryen. Disse verdiene blir mappet innenfor thrusterene sine arbeidsområder, og setter thruster hastigheten til dette, se figur 4-4.

```
void thrusterControl()
{
  int value1 = map((int)inputDataArranged[2], -100, 100, 1100, 1900);
  int value2 = map((int)inputDataArranged[3], -100, 100, 1100, 1900);
  int value3 = map((int)inputDataArranged[4], -100, 100, 1100, 1900);
  thrusterOne.writeMicroseconds(value1);
  thrusterTwo.writeMicroseconds(value2);
  thrusterThree.writeMicroseconds(value3);

}
```

Figur 4-4 Kode for å kontrollere thrustere i mikrokontrolleren

Vi ser her at hver enkel thruster får et signal fra Raspberry. Dette signalet har arbeidsområdet -100 til 100. Denne mappes så til verdier som er i hastighetskontrollerens arbeidsområde, altså 1100-1900.

4.2.2 Kommunikasjon

Som nevnt i kapittel 3 metoder, har vi benyttet oss av to typer kommunikasjon i vårt system. UDP og seriell kommunikasjon, og satt opp en dataoverførings protokoll tilegnet hver av kommunikasjonsprotokollene. Den endelige protokollen vi endte opp med fungerte med ønsket resultat. Vi satt opp tre protokoller for sending. Der den ene tok for seg sending fra GUI til Raspberry, den andre for sending av data fra Raspberry til mikrokontroller, og den siste fra mikrokontroller opp til GUI. De to første protokollene som tar for seg sending fra GUI ned til mikrokontrolleren har vi endt opp med å dele opp i to. Siden vi har valgt å gjøre all logikk og databehandling på Raspberryen, vil den utføre gitte kommandoer og utregninger i Raspberryen, før den sender resultatene videre i en ny protokoll. På denne måten slipper vi å gjøre noe utregning i Mikrokontrolleren, bare tilegne ferdig utregnet verdier til forskjellige komponenter.

4.2.2.1 Protokoll Fra Styringsenhet til Raspberry

I tabellen under viser vi til hvordan vår kommunikasjonsprotokoll fra styringsenheten (GUI) til Raspberry er satt opp. Dette er verdier som blir satt av styringssystemet, og som skal sendes til prosessering, før de skal sendes videre ned til mikrokontrolleren.

Protokollen er satt opp som et byte array med størrelse på 5 bytes. På denne måten kan vi enkelt ha oversikt over kommandoene som blir sendt fra styringssystemet vårt.

Tabellen under viser hvilke kommandoer som er tilknyttet hver bit, se figur 4.2.2.1.

Byte	0	1	2	3	4
bit[0]	Stopp				
bit[1]	FWD				
bit[2]	REV				
bit[3]	LEFT			Start ROV	
bit[4]	RIGHT				
bit[5]	glideLeft				
bit[6]	glideRight				
bit[7]					
Kommand:	command	Thruster Power	Light Power		Reserved

Tabell 4.2.2.1 Protokoll fra styringsenhet til Raspberry

Eksempel på dette: Om vi sender kommandoen FWD fra styringssystemet vårt, vil bit 1 i første byte (byte[0]) bli satt til høy. Dette gir en verdi på 1*2^1 = 2. Tallverdien 2 blir altså satt i byte[0]. Dette gir en verdi på {2,0,0,0,0} som blir sendt til mikrokontrolleren. Om ROV får kommando start og FWD vil det sende protokollen {2,0,0,8,0}

4.2.2.2 Protokoll fra Raspberry til Mikrokontroller

Denne protokollen er en videresending av dataen mottatt fra GUI etter prosessering. Byte[1] har fått tildelt samme verdi som byte[4] fra GUI (Start bit), og byte[5] er tildelt samme verdi som byte[4] fra arduino (Lysstyrken). Det eneste som egentlig er gjort her, er å sende thrusterPower inn til ThrusterControll klassen, og omgjort ThrusterPower (0-100) til en verdi tildelt hver av de tre thrusterene (-100 til 100). På denne måten kan vi sette thrusterverdien direkte i mikrokontroller uten noe som helt prosessering annet en mapping (fra -100 til 100, til nytt område 1100 til 1900).

Byte	0	1	2	3	4	5
Kommand:	Flag byte	Start byte	Thruster 1 power	Thruster 2 Power	Thruster 3 power	Light Power

Tabell 4.2.2.2 Protokoll fra Raspberry til Mikrokontroller

4.2.2.3 Protokoll Fra Mikrokontroller til Styringsenhet

For å sette opp protokollen som sendes fra mikrokontrolleren har vi sett på hvilke sensorer vi har, og hvilke verdier de gir oss. Med tanke på at de fleste sensorene gir en float verdi, har vi valgt å tildele hver aktuell sensor to byte. Der det ene bytet holder på verdien før komma, og

det andre bytet holder på de to desimalene etter komma. F.eks. float verdien 34,56 vil altså bli tildelt byte[0] = 34 og byte[1] = 56. Videre divideres byte 2 på 100 i logikk delen når det kommer til GUI, slik det er tilbake til reell verdi 34,56.

Den endelige protokollen vi har endt opp med er gjengitt i tabellen under:

	Sensor	Info	Value
byte[0]	-128	setting the startbyte value	-128
byte[1]	Temperatur	The water temperature	-128 to 127
byte[2]	Descimal Temperatur	Descimal value	0-99
byte[3]	Pressure	The pressure outside the ROV	-128 to 127
byte[4]	Descimal Pressure	Descimal value	0-99
byte[5]	Depth	The depth of the ROV	-128 to 127
byte[6]	Descimal depth	Descimal value	0-99
byte[7]	Oxygen	Oxygen level in the water	-128 to 127
byte[8]	Descimal Oxygen	Descimal value	0-99
byte[9]	WaterLevel	waterlevel inside the ROV	-128 to 127
byte[10]	Roll	The Roll value of the ROV	-128 to 127
byte[11]	Descimal Roll	Descimal value	0-99
byte[12]	Pitch	The Pitch value of the ROV	-128 to 127
byte[13]	Descimal Pitch	Descimal value	0-99
byte[14]	Heading	The direction of the ROV	-128 to 127
byte[15]	TempInROV	Temperature inside the ROV	-128 to 127
byte[16]	Descimal TempInROV	Descimal value	0-99
byte[17]	xAccmeter	Speed in X direction	-128 to 127
byte[18]	Descimal xAccmeter	Descimal value	0-99
byte[19]	yAccemeter	Speed in Y direction	-128 to 127
byte[20]	Descimal yAccmeter	Descimal value	0-99
byte[21]	zAccmeter	Speed in Z direction	-128 to 127
byte[22]	Descimal zAccmeter	Descimal value	0-99
byte[23]	voltageBattery	Voltage Battery	-128 to 127
byte[24]	Descimal voltageBattery	Descimal value	0-99

Tabell 4.2.2.3Protokoll fra mikrokontroller til styringsenhet

4.2.2.4 Seriell kommunikasjon mellom Raspberry og Mikrokontroller

Som tidligere forklart endte vi opp med å benytte seriell kommunikasjon via USB for kommunikasjonen mellom Raspberry og Mikrokontroller. Vi benyttet oss av JSSC biblioteket for seriell kommunikasjon i Java. I denne klassen setter vi opp porten kommunikasjonen skal foregå, bufferraten og en semaphore. Med denne informasjonen opprettes to tråder. Den ene er for lesing av data (SerialReader), den andre for sending (SerialSend). Med å kalle på connect() metoden i SerialCom, aktiveres de trådene som vist i figur under.

```
public void connect() {
    try {
        if(!serialPort.isOpened()){
        serialPort.openPort();
        getSerialPort().setParams(19200, 8, 1, 0);
        reader = new Thread(new SerialRead(this,semaPhore,serialPort));
        sender = new Thread(new SerialSend(this,semaPhore,serialPort));
        sender.start();
        reader.start();
        }
    } catch (SerialPortException e) {
        System.out.println("No Port Found On: " + System.getProperty("os.name"));
    }
}
```

Figur 4-5 connect() metode i serialCom klasse

SerialReader klassen står for kommunikasjonsbehandlingen der den lytter på inngående port, og leser inngående data. Dataen som blir mottatt kjøres inn i en checkDataArrangementTest(data) som har i oppgave å forsikre om at data kommer i riktig rekkefølge. Vi opplevde noen ganger at dataen som ble sendt fra Mikrokontrolleren var forskjøvet. Altså at det som skulle være på plass byte[0] kom inn på byte[1], og det som skulle være på byte[1] kom på byte[2] osv. Med å kjøre det gjennom denne funksjonen sikret vi oss for at dette ikke skulle skje. Når vi har forsikret oss om at dataen er lagt i riktig rekkefølge, skrives dataen opp til dataFromArduino variabelen som ligger i SerialCom klassen. Denne blir da tilgjengelig for dataHandler i neste steg. Run metoden for lesing av data fra mikrokontroller blir gjengitt i figur 4-6 under.

```
public void run() {
    try {
        while (true) {
            semaphore.acquire();
            byte[] data = serialPort.readBytes(23);
            if(data.length > 0){
            byte[] arrangedData = checkDataArrangementTest(data);
            serialCom.dataFromArduino = arrangedData;
            serialCom.setDataFromArduino(arrangedData);
            }

            semaphore.release();
        }
    } catch (SerialPortException ex) {
        System.out.println("SerialPortException i SerialRead");
    } catch (InterruptedException e) {
        System.out.println("InterruptedException i SerialRead");
    }
}
```

Figur 4-6 Kode for lesing av seriell kommunikasjon på Raspberry fra mikrokontroller

SerialSend har i oppgave å overføre data mottatt fra DataHandler videre til mikrokontroller. Dataen fra dataHandler blir satt til variabelen sendDataToArduino i serialCom klassen, og blir hentet fra denne. Koden for run metoden i denne klassen vises i figur 4-7 under:

```
public void run() {
    try {
        while (true) {
            semaphore.acquire();
            byte[] dataToArduino = new byte[6];
            dataToArduino = serialCom.sendDataToArduino();
            System.out.println("SEND Serial " + Arrays.toString(dataToArduino));
            serialPort.writeBytes(dataToArduino);
            semaphore.release();
        }
    } catch (SerialPortException ex) {
        System.out.println("SerialPortException i SerialSend");
    } catch (InterruptedException e) {
        System.out.println("InterruptedException i SerialSend");
    }

}
}
```

Figur 4-7 Kode for sending av data via seriell kommunikasjon fra Raspberry til mikrokontroller.

4.2.2.5 UDP kommunikasjon mellom GUI og Raspberry

Med tanke på at det skal sendes både data og video fra Raspberry til GUI, valgte vi å gjøre dette i to forskjellige klasser, disse er UDPSendData og UDPSendVideo. I GUI er det da to klasser for å motta det som blir sendt fra Raspberry, og i tillegg er det en egen klasse som sender kommandoer for thrustere og lysene. Alt dette er ved hjelp av UDP kommunikasjon.

Hvordan vi har satt opp overføringen av data fra GUI klienten vises i figur 4-8 under:

```
public void run() {
    while (true) {
        try {

            DatagramSocket ds = new DatagramSocket();
            InetAddress ia = InetAddress.getByName(this.address);
            DatagramPacket packet1 = new DatagramPacket(this.dataHandler.sendDataToGUI(),
                    this.dataHandler.sendDataToGUI().length, ia, port);
            ds.send(packet1);
            Thread.sleep(500);
```

Figur 4-8 Kode for sending av data via UDP kommunikasjon fra Raspberry til GUI.

Hvordan vi har satt opp siden for lesing av data på GUI siden vises i figur 4-9 under:

```
public void run() {
    while(true){
    try {
        DatagramPacket packet = new DatagramPacket(data, data.length);
        datagramSocket.receive(packet);
        this.data = packet.getData();
        System.out.println("Read UDP: " + Arrays.toString(data));
    }
```

Figur 4-9 Kode for mottak av data via UDP kommunikasjon fra GUI til Raspberry.

4.2.3 Logikk

Datahandler i Raspberry videresender data mottatt fra GUI til klassen ThrusterControll. Her har vi tatt for oss utregningene som setter verdiene til de forskjellige thrusterene. Dette er eneste logikken vi har definerte utregninger for i denne klassen. Lysstyrke settes en verdi på mellom 0-100 i GUI, som settes rett inn i egen byte i protokollen, og får beholde samme verdi helt ned til mikrokontrolleren. Der mappes den til en verdi innenfor sitt arbeidsområde. (mapper 0-100 til en verdi mellom 1100-1900).

GUI sender thrusterpower på samme måte som lysPower ned til Raspberryen. Men thrusterene er også avhengig av kommandoene sendt fra bruker. Altså hvor ROV skal navigere seg. Det er dette thrusterControll klassen tar seg av. Både kommandoen og thrusterPower verdien blir sendt inn i klassen, og funksjonen thrusterValues(int command,int power) omgjør da de verdiene til en egen verdi tilknyttet hver av de tre thrusterene. Det ene bytet for thrusterPower, blir da omgjort til tre nye byte i ny protokoll som inneholder en verdi tilknyttet hver truster i egen byte.

Koden vi har lagt i klassen vies i figur 4-10 Dette er et utsnitt av koden som viser hvilke verdier som blir tilegnet ved de forskjellige retningene. Vi ser her at funksjonen har to inn parametere, disse er command og power, som er hvilken retning og hvor fort den skal kjøre.

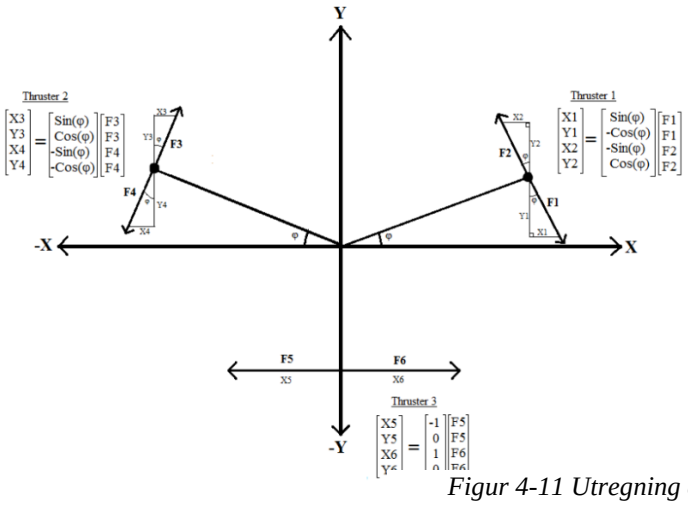
```
public void thrusterValues(int command, int power) {

    if (command == 2) { // FWD
        thrusterOne = power;
        thrusterTwo = -(power);
    } else if (command == 18) { // Slide FWD Right
        int calculationVar = power * 58; // ThrusterOne and ThrusterTwo need 58% of ThrusterThree's Power.
        thrusterOne = calculationVar/(100);
        thrusterTwo = calculationVar/(100);
        thrusterThree = power;
    } else if (command == 16) { // Right
        thrusterOne = -(power);
        thrusterTwo = -(power);
        thrusterThree = power;
    } else if (command == 20) { // Slide Back Right
        int calculationVar = power * 58; // ThrusterOne and ThrusterTwo need 58% of ThrusterThree's Power.
        thrusterOne = -(calculationVar/(100));
        thrusterTwo = calculationVar/(100);
        thrusterThree = power;
    } else if (command == 4) { // Back
        thrusterOne = -power;
        thrusterTwo = power;
```

Figur 4-10 Kode for kalkulasjon av thrusterstyrke i ThrusterControll klassen.

Ut ifra de to variablene sendt fra GUI, command og power, blir thrustene kalkulert og gitt en verdi som blir sendt videre til mikrokontrolleren. Se avsnitt 3.2.6 eller vedlegg for thruster kalkulasjoner for dette.

Figur 4-11 viser vektor tabellene som ble brukt for å kalkulere hver enkel thruster.

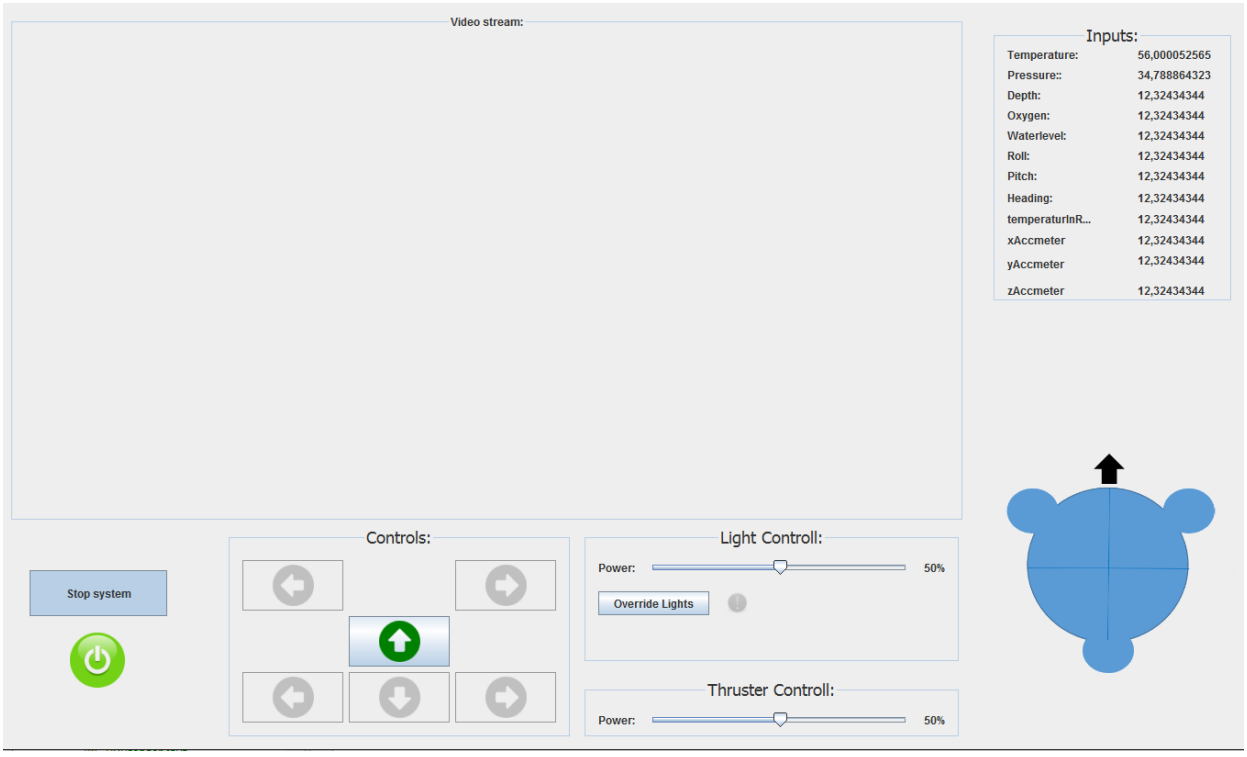


Figur 4-11 Utregning av thruster krefter.

4.2.4 GUI

Den endelige GUI vi endte opp med blir vist i figur 4-12 under.

Hoveddelen av GUI er live bildestrøm fra kameraet vi har montert på ROV.



Figur 4-12 Bilde av endelig GUI.

Denne gir oss oversikt over hva den observerer direkte til vår GUI. På høgre siden har vi lagt til en ramme som gir oss alle verdiene som de forskjellige sensorene detekterer. På denne måten har vi full kontroll over omgivelsene rundt ROV farkosten.

Controls:

Under bildestrømmen har vi lagt til noen knapper. Vi har valgt å ikke legge noe funksjon i knappene (altså om vi trykker på knappene med musen, vil det ikke skje noe). Dette grunnet at vi ikke vet hvor systemet skal implementeres. Vi har valgt å heller legge med knappene som en oversikt over hvilke kommandoer som er aktivert. Om Styringssystemet sender kommandoen FWD til mikrokontroller (ROV), ønsker vi at pilen som peker frem skal lyse.

Vi har også lagt inn til slidere, disse er for styring av lys og thruster styrken. Verdien som blir satt av sliderene er den prosentivse verdien av maksimal verdi. F.eks. en verdi på 50 tilsvarer 50% av maksverdi.

Inputs:

Her blir alle sensorverdiene som blir registrert i ROV presentert.

OverView

Her ar vi satt inn en enkel fremstilling som viser hvilken retning ROV skal bevege seg når de aktiverte kommandoene er iverksatt.

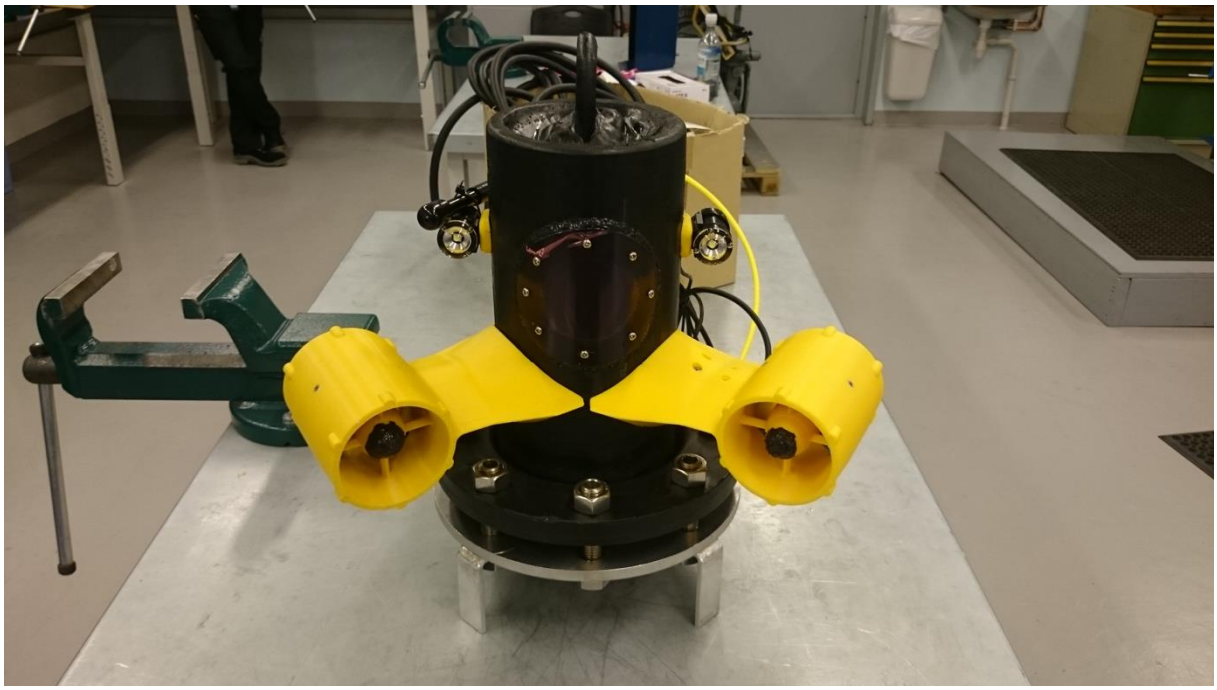
4.3 Endelig resultat av ROV

Designet av ROV er ferdig utviklet, og vi har fått et vanntett objekt. Vi har også testet software, og konkludert med att all kommunikasjon mellom de tre systemene fungerer.

Alle komponentene (sensorer og thruster/lys) har blitt testet hver for seg og samlet med positivt resultat.

Med den gitte tidsrammen har vi i enden av prosjektet fokusert på å få utviklet ferdig rapporten til innleveringen, før vi igjen leger fokuset på ferdigstilling av selve ROVen.

Det neste steget som gjenstår blir å sette all elektronikk inn i ROV for å få testet den som en helhetlig ROV. Med tanke på at både elektronikken, software og tettheten på ROV er testet med positivt resultat, håper vi på en positiv test på første forsøk.



5 DRØFTING

I endt prosjektperiode, sitter vi igjen med flere inntrykk av prosjektet. Det har vært ett prosjekt vi skulle ønske vi hadde fått litt mer tid på, men selv mener vi har fått utviklet et produkt som tilfredsstiller de ønskede kravene. Med litt utvidet tidsramme kunne vi blant annet fått utvidet med litt flere sensorer.

Vi satt oss i starten opp 3 mål, og vil ut ifra våre erfaringer gjennom prosjekttiden svare på de målene, men i tillegg legge til noen erfaringer knyttet til prosjektet.

5.1 Resultater fra test:

5.1.1 Software

Vi har under hele prosjektperioden utført testing på hvordan systemet fungerer som helhet og hver for seg (de tre under systemene). Vi kom tidlig i mål med å få en fullstendig utviklet GUI og kommunikasjon mellom alle enhetene. Vårt fokus har vært å utvikle alle kommunikasjonsprotokollene så tidlig som mulig, da kommunikasjonsbiten ved prosjektet er svært tidkrevende. Etter dette ble all logikk som skulle være i programmet ferdigutviklet. Deretter ble alle komponentene involvert i prosjektet koblet opp og testet med vårt program. Dette fungerer som det skal, men vi vet fortsatt ikke om de utregningene som ble gjort for thrusterene fungerer, siden vi ikke har kommet i mål med å få utstyret montert på ROVen og testet systemet i vann.

5.1.2 Hardware

Alle komponentene i prosjektet har vært opp til våres forventninger, utenom Raspberry Pi. Siden vi har mye bildeprosessering og kalkulasjoner merker vi dette i video fremvisningen i GUI. En mulig forbedring hadde vært å bruke en ODROID-XU4 [12], da denne har bedre prosessering ytelse enn Raspberry Pi.

5.2 Prosessmål

Med tanke på utfordringer på flere ukjente former for alle i oppgaven, sitter vi igjen med ett inntrykk at det har vært ett prosjekt med en bratt læringskurve. Vi har alle bistått med å få ett endelig produkt, og sitter nå igjen med ny kompetanse i forhold til hva vi gikk inn i prosjektperioden med.

5.3 Prosjektmål

Det har blitt jobbet strukturert gjennom hele prosjektet. Med tanke på at vi har delt gruppen delvis i to, der en har tatt seg av det visuelle, og de to andre har tatt seg av kobling av elektronikk og programmeringen, har også kommunikasjonen mellom oss vært bra. Dette har gjort at vi nå sitter igjen med ett produkt vi kan se oss fornøyde med. Det har blitt noe avvik i forhold til prosjektplanen på milepælene, men klart å holde oss godt innenfor på de fleste.

5.4 Produktmål

Vi sitter igjen med litt blandede følelse i denne fasen av prosjektet. Vi skulle ønske vi hadde ett helt komplett produkt vi kunne levere, ferdig testet. Men med tanke på at rapporten skal leveres 9 dager før endelig prosjekt slutt, har vi tatt oss tid til rapportskrivingen fremfor ferdigstillelse av det gjenstående på produktet. Ferdigstillelse av produktet er noe vi vil benytte tiden videre på å få gjort ferdig. Vi er sikre på at det endelige produktet vi får levere, tilfredsstiller det vi satt oss i starten som mål.

Vi kunne også ønsket vi hadde litt mer tid slik vi kunne tilpasset prosjektet litt mer i forhold til flere sensorer og ekstra bildebehandling, men med de gitte tidsrammene har vi måtte tatt noen beslutninger som har gått på bekostning på dette.

6 KONKLUSJON

På en kommende slutt av prosjektperioden sitter vi igjen med mange inntrykk fra det vi har gjennomført. Dette er et samarbeidsprosjekt med tverrfagligbakgrunn, som gjenspeiler seg gjennom klare og definerte arbeidsoppgaver for hver enkelt som er involvert i prosjektet. Fordelingen av arbeidsoppgavene har gjort at prosjektet har blitt gjennomført på en effektiv måte, selv om det fortsatt gjenstår arbeid. Dette regner vi med kommer i mål i løpet av de kommende dagene. Det har vært en spennende oppgave med mange utfordringer, slik at vi sitter igjen med god læring, nyttig kunnskap og mye inntrykk.

7 FIGURER OG TABELLER

8 REFERANSER

[1] https://www.bluerobotics.com/store/thrusters/t200-thruster/

[2] https://www.bluerobotics.com/store/electronics/bar30-sensor-r1/

[3] https://www.bluerobotics.com/store/cables/fathom-tether-nb-4p-26awg-r1/

[4] https://www.bluerobotics.com/store/thrusters/besc-30-r1/

[5] https://www.bluerobotics.com/store/electronics/cam-rpi-wide-r1/

[6] https://www.bluerobotics.com/store/electronics/lumen-light-r1/

[7] https://www.arduino.cc/en/Main/ArduinoBoardNano

[8] https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

[9] https://www.sparkfun.com/products/11194

[10] https://www.sparkfun.com/products/13322

[11] https://www.adafruit.com/product/1604

[12] http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825

wikipedia. (2015, 12 28). Fjernstyrt undervannsfarkost. Hentet fra wikipedia: https://no.wikipedia.org/wiki/Fjernstyrt_undervannsfarkost

wikipedia. (2015, 8 15). wikipedia. Hentet fra Java (programmeringsspråk): https://no.wikipedia.org/wiki/Java_(programmeringsspr%C3%A5k)

Netbeans. Netbeans. u.d. https://netbeans.org/about/.

OpenCV. www.Opencv.org. u.d. www.Opencv.org.

OpenCV.org. OpenCV. u.d.

Lindsey, T. L. (u.d.). *JavaTech.*

SNL.no. Seriell Kommunikajson - IT. u.d. https://snl.no/seriell_kommunikasjon%2FIT.

Temperatur. Wikipedia. u.d. https://no.wikipedia.org/wiki/Temperatur.

Wikipedia. Arduino. u.d.

wikipedia. Fjernstyrt undervannsfarkost. 28 12 2015. https://no.wikipedia.org/wiki/Fjernstyrt_undervannsfarkost.

Wikipedia. oppdrift. u.d. https://no.wikipedia.org/wiki/Oppdrift.

—. Thruster. u.d. https://no.wikipedia.org/wiki/Thruster.

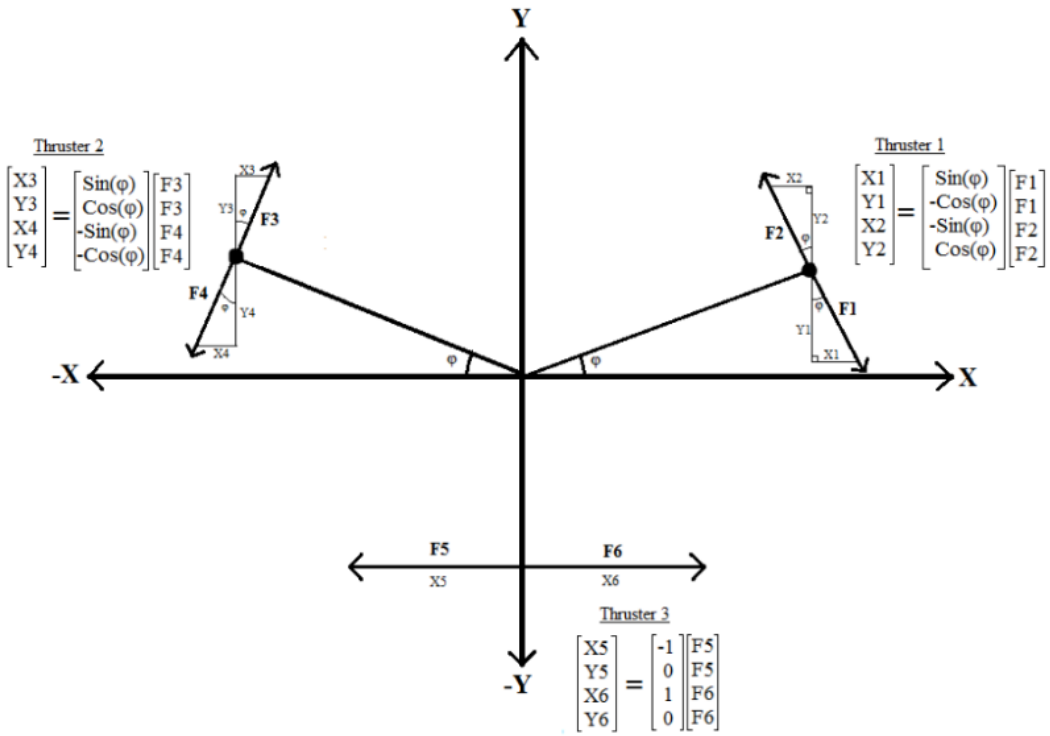
wikipedia. wikipedia. 15 8 2015.
https://no.wikipedia.org/wiki/Java_(programmeringsspr%C3%A5k).

9 VEDLEGG

9.1 Kildekoder

Kildekodene til applikasjonene for GUI, Raspberry PI og mikrokontroller er levert på fronter.

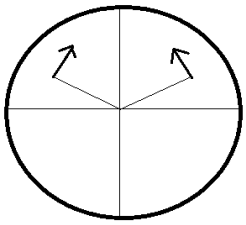
9.2 Thruster kalkulasjoner



Figur 8.2.1, Bilde av vektorkrefter til hver enkelt thruster.

Kraften F tilsvarer 0-100% av hva kraften thrusteren kan gi, dette blir da vektorer. Vektorene er avhengig av hvilken retning de peker i forhold til koordinatsystemet, se figur 8.2.1. Kalkulasjonene for hver retning blir som følger:

Fremover



Figur 8.2.2, Illustrasjon av thruster pådrag for fremover.

Thruster 1 får negativ x- og positiv y komponent.

Thruster 2 får positiv x- og y komponent., se figur 8.2.2.

$\boldsymbol{X2} = -F2\ x \sin\left(\frac{\pi}{6}\right), \qquad \boldsymbol{X3} = F3\ x \sin\left(\frac{\pi}{6}\right)$

$\boldsymbol{Y2} = F1\ x\ Cos\left(\frac{\pi}{6}\right), \qquad \boldsymbol{Y3} = F3\ x\ Cos(\frac{\pi}{6})$

Figur 8.2.2, Illustrasjon av thruster pådrag for fremover.

$\boldsymbol{X} = X2 + X3 = -F2\ x \sin\left(\frac{\pi}{6}\right) + F3\ x\ \mathrm{Sin}\left(\frac{\pi}{6}\right) =$

Figur 8.2.2, Illustrasjon av thruster pådrag for fremover.

$(F3 - F1)Sin(\frac{\pi}{6})$. Ser her at dersom F1=F3 vil $\boldsymbol{X} = \mathbf{0}$.

Figur 8.2.2, Illustrasjon av thruster pådrag for fremover.

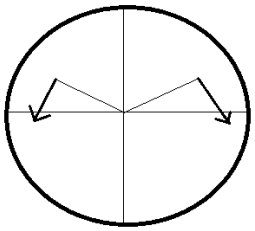
$Y = Y1 + Y3 = F1\ x\ Cos\left(\frac{\pi}{6}\right), +F3\ x\ Cos\left(\frac{\pi}{6}\right) =$

$(F1 + F3)Cos\left(\frac{\pi}{6}\right)$.

Siden F1 = F3, vil man kunne få med maksimal kraft på begge to:

$Y = (100 + 100)Cos\left(\frac{\pi}{6}\right) = 173.2$. Dette vil si at man kan få 173% av hva kraften til en thruster er.

Bakover



Figur 8.2.3, Illustrasjon av thruster pådrag for bakover.

Thruster 1 får positiv x- og negativ y komponent.

Thruster 2 får negativ x- og y negativ, se figur 8.2.3.

$\boldsymbol{X1} = F1\ x \sin\left(\frac{\pi}{6}\right), \qquad \boldsymbol{X4} = -F4\ x \sin\left(\frac{\pi}{6}\right)$

$\boldsymbol{Y1} = -F1\ x\ Cos\left(\frac{\pi}{6}\right), \qquad \boldsymbol{Y4} = -F4\ x\ Cos(\frac{\pi}{6})$

Figur 8.2.3, Illustrasjon av thruster pådrag for bakover.

$\boldsymbol{X} = X1 + X4 = F1\ x \sin\left(\frac{\pi}{6}\right) + F4x\ \mathrm{Sin}\left(\frac{\pi}{6}\right) = (F4 -$ F1=F4

Figur 8.2.3, Illustrasjon av thruster pådrag for bakover.

vil $\boldsymbol{X} = \mathbf{0}$.

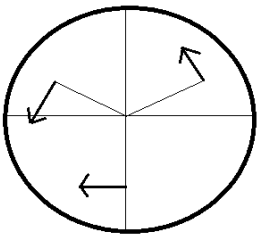
$Y = -Y1 + (-Y4) = -F1\ x\ Cos\left(\frac{\pi}{6}\right), -F4\ x\ Cos\left(\frac{\pi}{6}\right)$

Figur 8.2.3, Illustrasjon av thruster pådrag for bakover.

Siden F1 = F4, vil man kunne få med maksimal kraft på begge to:

$Y = (-100 - 100)Cos\left(\frac{\pi}{6}\right) = -173.2$. Dette vil si at man kan få 173% av hva kraften til en thruster er.

Venstre



Thruster 1 får negativ x- og positiv y komponent.

Thruster 2 får negativ x- og y komponent.

Thruster 3 får negativ x komponent og ingen påvirkning i y-verdi, da vektorkraften er parallell med x-aksen, se figur 8.2.4.

Figur 8.2.4, Illustrasjon av thruster pådrag for venstre.

Figur 8.2.4, Illustrasjon av thruster pådrag for venstre.

Figur 8.2.4, Illustrasjon av thruster pådrag for venstre.

Figur 8.2.4, Illustrasjon av thruster pådrag for venstre.

$$X2 = -F2\ x\ Sin\left(\frac{\pi}{6}\right), \quad X4 = -F4\ x\ Sin\left(\frac{\pi}{6}\right), X5 = -1\ x\ F5$$

$$Y2 = F2\ x\ Cos\left(\frac{\pi}{6}\right), \quad Y4 = -F4\ x\ Cos\left(\frac{\pi}{6}\right),$$

$$Y5 = 0\ x\ F5.$$

$$X = X2 + X4 + X5$$

$$Y = Y2 + Y4 + Y5$$

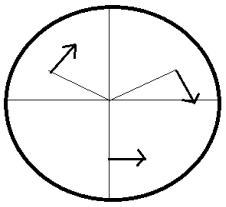
$$X = -F2\ x\ Sin\left(\frac{\pi}{6}\right) - F4\ x\ Sin\left(\frac{\pi}{6}\right) - F5 = (-F2 - F4)Sin\left(\frac{\pi}{6}\right) - F5$$

$$Y = F2\ x\ Cos\left(\frac{\pi}{6}\right) - F4\ x\ Cos\left(\frac{\pi}{6}\right) + 0\ x\ F5 \rightarrow F2 = F4 \rightarrow Y = (F2 - F4)Cos\left(\frac{\pi}{6}\right) = 0.$$

Maksimal styrke i x retning er:

$X = (-100 - 100)Sin\left(\frac{\pi}{6}\right) - 100 = -100 - 100 = -200$. Dette vil si at man kan få 200% av hva kraften til en thruster er.

Høyre



Thruster 1 får positiv x- og negativ y komponent.

Thruster 2 får positiv x- og y komponent.

Thruster 3 får positiv x komponent og ingen påvirkning i y-verdi, da vektorkraften er parallell med x-aksen, se figur 8.2.5.

Figur 8.2.5, Illustrasjon av thruster pådrag for høyre.

Figur 8.2.9-1, Illustrasjon av thruster pådrag for rotasjon venstre.Figur 8.2.5, Illustrasjon av thruster pådrag for høyre.

Figur 8.2.9-2, Illustrasjon av thruster pådrag for rotasjon venstre.

Figur 8.2.9-3, Illustrasjon av thruster pådrag for rotasjon høyre.Figur 8.2.9-4, Illustrasjon av thruster pådrag for rotasjon venstre.Figur 8.2.5, Illustrasjon av thruster pådrag for høyre.

Figur 8.2.9-5, Illustrasjon av thruster pådrag for rotasjon venstre.Figur 8.2.5, Illustrasjon av thruster pådrag for høyre.

$$X1 = F1\ x\ Sin\left(\frac{\pi}{6}\right), \qquad X3 = F3\ x\ Sin\left(\frac{\pi}{6}\right), \qquad X6 = 1\ x\ F6$$

$$Y1 = -F1\ x\ Cos\left(\frac{\pi}{6}\right), \qquad Y3 = F3\ x\ Cos\left(\frac{\pi}{6}\right),$$

$$Y6 = 0\ x\ F6.$$

$$X = X1 + X3 + X6$$

$$Y = Y1 + Y3 + Y6$$

$$X = -F1\ x\ Sin\left(\frac{\pi}{6}\right) - F3\ x\ Sin\left(\frac{\pi}{6}\right) - F6$$

$$= (-F1 - F3)Sin\left(\frac{\pi}{6}\right) - F6$$

$$Y = -F1\ x\ Cos\left(\frac{\pi}{6}\right) + F3\ x\ Cos\left(\frac{\pi}{6}\right) + 0\ x\ F6 \rightarrow F1 = F3 \rightarrow Y =$$

$$(F3 - F1)Cos\left(\frac{\pi}{6}\right) = 0.$$

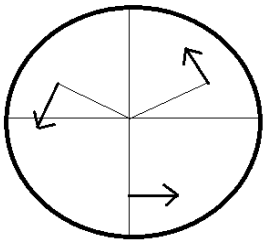
Maksimal styrke i x retning er:

$X = (100 + 100)Sin\left(\frac{\pi}{6}\right) + 100 = 100 + 100 = 200$. Dette vil si at man kan få 200% av hva kraften til en thruster er.

Rotasjon venstre

Thruster 1 får negativ x- og positiv y komponent.

Thruster 2 får negativ x- og y komponent.



Thruster 3 får positiv x komponent og ingen påvirkning i y-verdi, da vektorkraften er parallell med x-aksen, se figur 8.2.6.

$$X2 = -F2\ x\ Sin\left(\frac{\pi}{6}\right),\ \ X4 = -F4\ x\ Sin\left(\frac{\pi}{6}\right),\ \ X6 = 1\ x\ F6$$

$$Y2 = F2\ x\ Cos\left(\frac{\pi}{6}\right),\ \ Y4 = -F4\ x\ Cos\left(\frac{\pi}{6}\right),\ \ \ Y6 = 0\ x\ F6$$

$$X = X2 + X4 + X6$$

$$Y = Y2 + Y4 + Y6$$

$$X = \ -F2\ x\ Sin\left(\frac{\pi}{6}\right) - F4\ x\ Sin\left(\frac{\pi}{6}\right) + F6$$

$$= (-F2 - F4)Sin\left(\frac{\pi}{6}\right) + F6$$

Siden $F2 = F4 = F6$, det vil si at alle thrusterene kjører med samme hastighet, vil $X = 0$.

$$Y = F2\ x\ Cos\left(\frac{\pi}{6}\right) - F4\ x\ Cos\left(\frac{\pi}{6}\right) = (F2 - F4)Cos\left(\frac{\pi}{6}\right)$$

$$\rightarrow F2 = F4 \rightarrow Y = 0$$

Her ser man at dersom alle thrusterene kjører med samme hastighet, vil dette rotere ROVen til venstre uten at dette skaper vektorkraft i noen retning.

Figur 8.2.9-6, Illustrasjon av thruster pådrag for rotasjon venstre.

Figur 8.2.9-7, Illustrasjon av thruster pådrag for rotasjon høyre.Figur 8.2.9-8, Illustrasjon av thruster pådrag for rotasjon venstre.

Figur 8.2.9-9, Illustrasjon av thruster pådrag for rotasjon høyre.

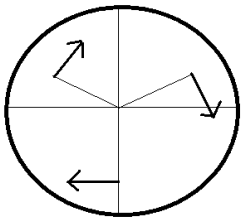
Figur 8.2.9-10, Illustrasjon av thruster pådrag for fremover høyre.Figur 8.2.9-11, Illustrasjon av thruster pådrag for rotasjon høyre.Figur 8.2.9-12, Illustrasjon av thruster pådrag for rotasjon venstre.

Figur 8.2.9-13, Illustrasjon av thruster pådrag for rotasjon høyre.Figur 8.2.9-14, Illustrasjon av thruster pådrag for rotasjon venstre.

Rotasjon høyre

Thruster 1 får positiv x- og negativ y komponent.

Thruster 2 får positiv x- og y komponent.



Thruster 3 får negativ x komponent og ingen påvirkning i y-verdi, da vektorkraften er parallell med x-aksen, se figur 8.2.7.

$$X1 = F1\ x\ Sin\left(\frac{\pi}{6}\right),\ \ X3 = F3\ x\ Sin\left(\frac{\pi}{6}\right),\ \ X5 = -1\ x\ F5$$

$$Y1 = -F1\ x\ Cos\left(\frac{\pi}{6}\right),\ \ Y3 = F3\ x\ Cos\left(\frac{\pi}{6}\right),\ \ \ Y5 = 0\ x\ F5$$

$$X = X1 + X3 + X5$$

$$Y = Y1 + Y3 + Y5$$

$$X = \ F1\ x\ Sin\left(\frac{\pi}{6}\right) + F3\ x\ Sin\left(\frac{\pi}{6}\right) - F5$$

$$= (F1 + F3)Sin\left(\frac{\pi}{6}\right) - F5$$

Siden $F1 = F3 = F5$, det vil si at alle thrusterene kjører med samme hastighet, vil $X = 0$.

$$Y = F1\ x\ Cos\left(\frac{\pi}{6}\right) - F3\ x\ Cos\left(\frac{\pi}{6}\right) = (F1 - F3)Cos\left(\frac{\pi}{6}\right) \rightarrow F1$$

$$= F3 \rightarrow Y = 0$$

Her ser man at dersom alle thrusterene kjører med samme hastighet høyre uten at dette skaper vektorkraft i noen retning.

Figur 8.2.9-15, Illustrasjon av thruster pådrag for rotasjon høyre.

Figur 8.2.9-16, Illustrasjon av thruster pådrag for fremover høyre.Figur 8.2.9-17, Illustrasjon av thruster pådrag for rotasjon høyre.

Figur 8.2.9-18, Illustrasjon av thruster pådrag for fremover høyre.

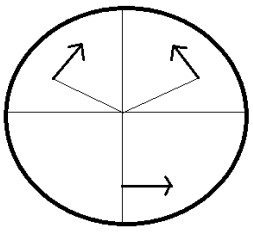
Figur 8.2.9-19, Illustrasjon av thruster pådrag for fremover venstre.Figur 8.2.9-20, Illustrasjon av thruster pådrag for fremover høyre.Figur 8.2.9-21, Illustrasjon av thruster pådrag for rotasjon høyre.

Figur 8.2.9-22, Illustrasjon av thruster pådrag for fremover høyre.Figur 8.2.9-23, Illustrasjon av thruster pådrag for rotasjon høyre.

Fremover høyre uten rotasjon

Thruster 1 får negativ x- og positiv y komponent.

Thruster 2 får positiv x- og y komponent.



Thruster 3 får positiv x komponent og ingen påvirkning i y-verdi, da vektorkraften er parallell med x-aksen, se figur 8.2.8.

Figur 8.2.9-24, Illustrasjon av thruster pådrag for fremover høyre.

Figur 8.2.9-25, Illustrasjon av thruster pådrag for fremover venstre.Figur 8.2.9-26, Illustrasjon av thruster pådrag for fremover høyre.

Figur 8.2.9-27, Illustrasjon av thruster pådrag for fremover venstre.

Figur 8.2.9-28, Illustrasjon av thruster pådrag for bakover høyre.Figur 8.2.9-29, Illustrasjon av thruster pådrag for fremover venstre.Figur 8.2.9-30, Illustrasjon av thruster pådrag for fremover høyre.

Figur 8.2.9-31, Illustrasjon av thruster pådrag for fremover venstre.Figur 8.2.9-32, Illustrasjon av thruster pådrag for fremover høyre.

$$X2 = -F2\ x\ Sin\left(\frac{\pi}{6}\right),\ \ X3 = F3\ x\ Sin\left(\frac{\pi}{6}\right),\ \ X6 = 1\ x\ F6$$

$$Y2 = F2\ x\ Cos\left(\frac{\pi}{6}\right),\ \ \ Y3 = F3\ x\ Cos\left(\frac{\pi}{6}\right),\ \ \ \ Y6 = 0\ x\ F6$$

$$X = X2 + X3 + X6 = \ -F2\ x\ Sin\left(\frac{\pi}{6}\right) + F3\ x\ Sin\left(\frac{\pi}{6}\right) + F6$$

$$= (F3 - F2)Sin\left(\frac{\pi}{6}\right) + F6$$

$$Y = Y2 + Y3 + Y6 = F2\ x\ Cos\left(\frac{\pi}{6}\right) + F3\ x\ Cos\left(\frac{\pi}{6}\right) + 0$$

$$= (F2 + F3)Cos(\frac{\pi}{6})$$

For å kunne kjøre fremover til høyre uten rotasjon, må vektorkreftene til X og Y være like, altså X = Y.

$$(F3 - F2)Sin\left(\frac{\pi}{6}\right) + F6 = (F2 + F3)Cos(\frac{\pi}{6})$$

For å unngå rotasjon er det nødvendig å sette F2 = F3, da står vi igjen med følgende:

$$F6 = (F2 + F3)Cos(\frac{\pi}{6})$$

F2 og F3 skaleres ned til 58% av F6.

Tar eksempel på dette:

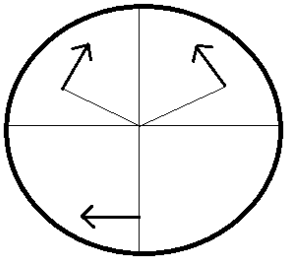
$$F6 = 100 \rightarrow F6 = (F2 + F3)Cos\left(\frac{\pi}{6}\right) \rightarrow F6 = (58 + 58)Cos\left(\frac{\pi}{6}\right) = 100.45$$

Dette gir $100\ \approx 100.45$, som oppfyller X = Y.

<u>Fremover venstre uten rotasjon</u>

Thruster 1 får negativ x- og positiv y komponent.

Thruster 2 får positiv x- og y komponent.



Thruster 3 får negativ x komponent og ingen påvirkning i y-verdi, da vektorkraften er parallell med x-aksen, se figur 8.2.9.

$$X2 = -F2\ x\ Sin\left(\frac{\pi}{6}\right),\ \ X3 = F3\ x\ Sin\left(\frac{\pi}{6}\right),\ \ X5 = -1\ x\ F5$$

$$Y2 = F2\ x\ Cos\left(\frac{\pi}{6}\right),\ \ \ Y3 = F3\ x\ Cos\left(\frac{\pi}{6}\right),\ \ \ \ Y5 = 0\ x\ F5$$

$$X = X2 + X3 + X5 = \ -F2\ x\ Sin\left(\frac{\pi}{6}\right) + F3\ x\ Sin\left(\frac{\pi}{6}\right) - F5$$

$$= (F3 - F2)Sin - F5$$

$$Y = Y2 + Y3 + Y5 = F2\ x\ Cos\left(\frac{\pi}{6}\right) + F3\ x\ Cos\left(\frac{\pi}{6}\right) + 0$$

$$= (F2 + F3)Cos(\frac{\pi}{6})$$

Figur 8.2.9-33, Illustrasjon av thruster pådrag for fremover venstre.

Figur 8.2.9-34, Illustrasjon av thruster pådrag for bakover høyre.Figur 8.2.9-35, Illustrasjon av thruster pådrag for fremover venstre.

Figur 8.2.9-36, Illustrasjon av thruster pådrag for bakover høyre.

Figur 8.2.9-37, Illustrasjon av thruster pådrag for bakover venstre.Figur 8.2.9-38, Illustrasjon av thruster pådrag for bakover høyre.Figur 8.2.9-39, Illustrasjon av thruster pådrag for fremover venstre.

Figur 8.2.9-40, Illustrasjon av thruster pådrag for bakover høyre.Figur 8.2.9-41, Illustrasjon av thruster pådrag for fremover venstre.

For å kunne kjøre fremover til venstre uten rotasjon, må vektorkreftene til -X og Y være like, altså -X = Y.

$$(F3 - F2)Sin\left(\frac{\pi}{6}\right) - F5 = (F2 + F3)Cos(\frac{\pi}{6})$$

For å unngå rotasjon er det nødvendig å sette F2 = F3, da står vi igjen med følgende:

$$-F5 = (F2 + F3)Cos(\frac{\pi}{6})$$

F2 og F3 skaleres ned til 58% av F6.

Tar eksempel på dette:

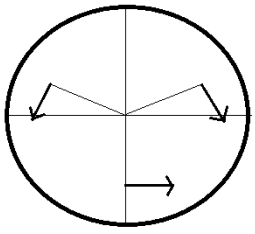
$$-F5 = 100 \rightarrow -F5 = (F2 + F3)Cos\left(\frac{\pi}{6}\right) \rightarrow -F5 = (58 + 58)Cos\left(\frac{\pi}{6}\right) = 100.45$$

Dette gir $-100\ \approx 100.45$, som oppfyller -X = Y.

<u>Bakover høyre uten rotasjon</u>

Thruster 1 får positiv x- og negativ y komponent.

Thruster 2 får negativ x- og y komponent.



Thruster 3 får positiv x komponent og ingen påvirkning i y-verdi, da vektorkraften er parallell med x-aksen, se figur 8.2.10.

Figur 8.2.9-42, Illustrasjon av thruster pådrag for bakover høyre.

$$X1 = F1\ x\ Sin\left(\frac{\pi}{6}\right),\ \ X4 = -F4\ x\ Sin\left(\frac{\pi}{6}\right),\ \ X6 = 1\ x\ F6$$

Figur 8.2.9-43, Illustrasjon av thruster pådrag for bakover venstre.Figur 8.2.9-44, Illustrasjon av thruster pådrag for bakover høyre.

$$Y1 = -F1\ x\ Cos\left(\frac{\pi}{6}\right),\ \ Y4 = -F4\ x\ Cos\left(\frac{\pi}{6}\right),\ \ \ Y6 = 0\ x\ F6$$

Figur 8.2.9-45, Illustrasjon av thruster pådrag for bakover venstre.

$$X = X1 + X4 + X6 = F1\ x\ Sin\left(\frac{\pi}{6}\right) - F4\ x\ Sin\left(\frac{\pi}{6}\right) + F6$$

$$= (F1 - F4)Sin + F6$$

Figur 9.1.9-46 Bilde av signalkabelFigur 8.2.9-47, Illustrasjon av thruster pådrag for bakover venstre.Figur 8.2.9-48, Illustrasjon av thruster pådrag for bakover høyre.

$$Y = Y1 + Y4 + Y6 = -F1\ x\ Cos\left(\frac{\pi}{6}\right) - F4\ x\ Cos\left(\frac{\pi}{6}\right) + 0$$

$$= (-F1 - F4)Cos(\frac{\pi}{6})$$

Figur 8.2.9-49, Illustrasjon av thruster pådrag for bakover venstre.Figur 8.2.9-50, Illustrasjon av thruster pådrag for bakover høyre.

For å kunne kjøre bakover til høyre uten rotasjon, må vektorkreftene til X og -Y være like, altså X = -Y.

$$(F1 - F4)Sin\left(\frac{\pi}{6}\right) + F6 = (-F1 - F4)Cos(\frac{\pi}{6})$$

For å unngå rotasjon er det nødvendig å sette F1 = F4, da står vi igjen med følgende:

$$F6 = (-F1 - F4)Cos(\frac{\pi}{6})$$

F1 og F4 skaleres ned til 58% av F6.

Tar eksempel på dette:

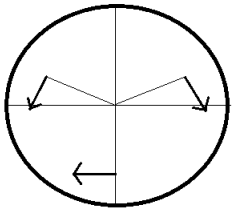
$$F6 = 100 \rightarrow Fy = (-58 - 58)Cos\left(\frac{\pi}{6}\right) \rightarrow F6 = (-58 - 58)Cos\left(\frac{\pi}{6}\right) = -100.45$$

Dette gir $100 \approx -100.45$, som oppfyller X = -Y.

<u>Bakover venstre uten rotasjon</u>

Thruster 1 får positiv x- og negativ y komponent.

Thruster 2 får negativ x- og y komponent.



Thruster 3 får negativ x komponent og ingen påvirkning i y-verdi, da vektorkraften er parallell med x-aksen, se figur 8.2.11.

$$X1 = F1\ x\ Sin\left(\frac{\pi}{6}\right),\ \ X4 = -F4\ x\ Sin\left(\frac{\pi}{6}\right),\ \ X5 = -1\ x\ F5$$

$$Y1 = -F1\ x\ Cos\left(\frac{\pi}{6}\right),\ \ \ Y4 = -F4\ x\ Cos\left(\frac{\pi}{6}\right),\ \ \ \ Y5 = 0\ x\ F5$$

$$X = X1 + X4 + X6 = \ F1\ x\ Sin\left(\frac{\pi}{6}\right) - F4\ x\ Sin + F5$$

$$= (F1 - F4)Sin + F5$$

$$Y = Y1 + Y4 + Y6 = -F1\ x\ Cos\left(\frac{\pi}{6}\right) - F4\ x\ Cos\left(\frac{\pi}{6}\right) + 0$$

$$= (-F1 - F4)Cos(\frac{\pi}{6})$$

Figur 8.2.9-51, Illustrasjon av thruster pådrag for bakover venstre.

Figur 9.1.9-52 Bilde av signalkabelFigur 8.2.9-53, Illustrasjon av thruster pådrag for bakover venstre.

Figur 9-54 Oppretting av ekstern plattform Netbeans*Figur 8.2.9-55, Illustrasjon av thruster pådrag for bakover venstre.*

Figur 9.1.9-56 Bilde av signalkabelFigur 8.2.9-57, Illustrasjon av thruster pådrag for bakover venstre.

For å kunne kjøre bakover til venstre uten rotasjon, må vektorkreftene til -X og -Y være like, altså -X = -Y.

$$(F1 - F4)Sin\left(\frac{\pi}{6}\right) - F5 = (-F1 - F4)Cos(\frac{\pi}{6})$$

For å unngå rotasjon er det nødvendig å sette F1 = F4, da står vi igjen med følgende:

$$-F5 = (-F1 - F4)Cos(\frac{\pi}{6})$$

F1 og F4 skaleres ned til 58% av F5.

Tar eksempel på dette:

$$-F5 = 100 \rightarrow Fy = (-58 - 58)Cos\left(\frac{\pi}{6}\right) \rightarrow F6 = (-58 - 58)Cos\left(\frac{\pi}{6}\right) = -100.45$$

Dette gir $-100\ \approx -100.45$, som oppfyller -X = -Y.