Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Automatisering og elektronikkdesign, Bachelor i ingeniørfag	Vårsemesteret, 20.21 Åpen / Konfidensiell
Forfatter: Oliver Langvik Veland	Martin Vesterøy Hausken
Fagansvarlig: Morten Tengesdal Veileder(e): Morten Tengesdal	
Tittel på bachelorpppgaven: Kommunikasjon og videostrøm av fjernstyrt undervannsfartøy Engelsk tittel: Communication and video streaming of remotely operated underwater vehicle	
Studiepoeng: 2x20	
Emneord: ROV, Kommunikasjon, Kamera, Belysning, Kamerahus, SPI, Ethernet, Markedsføring	Sidetall: 135 + vedlegg/annet: …… Stavanger, 15.05.2021 dato/år

Sammendrag

Uis Subsea er en studentorganisasjon ved universitetet i Stavanger. Organisasjonen ble opprettet i 2013 og har hvert år et mål om å delta i den internasjonale undervannsfartøy konkurransen som blir arrangert av Marine Advanced Technology Education Center(MATE). Vi skal i den sammenheng bygge en ny ROV som vi skal bruke i konkurransen.

I denne rapporten tar vi for oss kamerasystemet, kommunikasjonen mellom datamaskinen og ROV-en samt belysning rundt ROV-en.

Kamerasystemet består av to IP-kamera. Vi har valgt å plassere et kamera framme på ROV-en som skal gi god oversikt over omgivelsene og manipulatoren. Vi har også plassert et kamera under ROV-en som skal gi oss oversikt over hva som skjer under ROV-en. Dette kameraet skal også brukes til oppgavene om autonom kjøring i MATE-konkurransen. For å feste dette kameraet til ROV-en har vi laget et kamerahus som vi fester til rammen til ROV-en. Kameraene blir så koblet til en svitsj som er plassert i elektronikkhuset til ROV-en. Videostrømmen blir så sendt til datamaskinen på overflaten hvor den blir behandlet av bildebehandlingsgruppen.

For kommunikasjonssystemet i år har vi bestemt oss for å gå for en ny løsning. Her har vi brukt en modul som gjør Ethernet-pakker om til SPI-signal. Modulen kobles til SPI-bussen til mikrokontrolleren i ROV-en. Modulen blir så koblet til svitsjen i elektronikkhuset. Fra svitsjen skal vi bruke fiber som vårt transmisjonsmedium opp til datamaskinen på land. Modulen fungerer som en HTTP-tjener hvor vi har en egen IP-adresse til SPI/Ethernet-modulen. Ved bruk av HTTP-forespørsler i JavaScript kan vi hente ut og endre verdiene til variablene i mikrokontrolleren. Her kan vi lese av verdiene fra sensorene og sende ned styringsdata fra XBOX-kontrolleren.

Til belysning på ROV-en har vi valgt å bruke LED-dioder som vi skal plassere rundt ROV-en hvor det er greit å ha lys. LED-diodene blir koblet til en LED-driver som er plassert i elektronikkhuset. Vi får 12V ut fra strømforsyningen, dette skal driveren gjøre om til 16V slik at lysene fungerer optimalt.

Vi har ennå ikke fått testet ROV-en i sin helhet, men har testet delene hver for seg. Ut ifra testene ser vi at programmet fungerer, men det kommer nok til å bli et par endringer i tiden frem mot konkurransen slik at ROV-en fungerer best mulig.

Forord

Dette har vært et tøft og spennende semester. Det har vært veldig krevende, men også veldig lærerikt. Vi har nå fått litt erfaring om hvordan det er å jobbe sammen på et større prosjekt.

Vi ønsker å takke Universitetet i Stavanger som har gjort det mulig å holde på med slike student-organisasjoner. Vi ønsker og å takke alle sponsorer som har hjulpet oss med utstyr og hjelp til prosjektet.

Til slutt vil vi takke Morten Tengesdal som har vært vår veileder og har alltid vært tilgjengelig til spørsmål og hjelp gjennom hele prosjektet.

Innhold

Nomenklatur

- ROV - Remotely operated vehicle. Fjernstyrt undervanns kjøretøy
- IP - Internet protokoll
- CAN bus - Controller Area Network. CAN-bus er et grensesnitt for kommunikasjon
- I^2C - Inter-Integrated Circuit. Dette er et grensesnitt som brukes i kommunikasjon.
- SPI - Serial Peripheral Interface. Dette er et kommunikasjonsgrensesnitt som blir brukt til overføring mellom korte avstander.
- SNTP - Simple Network Time Protocol. Nettverksprotokoll for synkronisering av klokker.
- DNS - Domain name system. Navnesystem til datamaskiner og andre enheter som er tilkoblet et nettverk.
- TFTP - Trivial File Transfer Protocol. Triviell filoverføringsprotokoll. Brukes til å sende filer mellom enheter over nettverket.
- FTP - File Transfer Protocol. Filoverføringsprotokoll. Brukes til å sende filer mellom enheter over nettverket.
- SNMP - Simple Network Management protocol. Internett-protokoll for å samle informasjon om enheter i nettverket.
- DHCP - Dynamic Host Configuration protocol. Nettverksadministrasjons protokoll.
- MQTT - Message Queuing Telemetri Transport. Nettverksprotokoll for sending av meldinger.
- HTTP - HyperText Transfer Protocol. Nettverksprotokoll for kommunikasjon over internett.
- HMS - Helse miljø og sikkerhet
- TCP - Transmission Control Protocol. Protokoll for overføringskontroll.
- SMTP - Simple Mail Transfer Protocol. Nettverksprotokoll for overføring av e-mail.
- PoE - Power over Ethernet. Strømtilførsel gjennom Ethernet.
- HEVC - Høyeffektiv videokoding
- HAVC - Avansert videokoding
- Topside - Overflate kontrollsytem til ROV

- USB - Universal Serial Bus. Protokoll for tilkobling av diverse utstyr.
- Baseline, mainline og highline - Produktkvalitet til kamera, der baseline er dårligest og highline er best
- FPS - Frames per second. Bilder som blir sendt per sekund
- Wireshark - programvare for og overåke ethernet pakker
- UART - Universal Asynchronous Reciever Transmitter, brukes for og sende meldinger mellom mikrokontroller og datamaskinen.
- MAC - Media Access Control. Identifikator for enheter som er koblet til internett.
- GPIO - General Purpose Input/Output. Dette er en pinne på mikrokontrolleren som kan brukes som inngang, utgang eller begge deler.
- GET - HTTP metode for og hente ut data fra en server eller nettside
- POST - HTTP metode for og endre eller sende data til en nettside eller server.
- HTML - HyperText Markup Languare - Programmeringsspråk som brukes til og vise frem innhold på en nettside.
- ASCII - American Standard Code for information Interchange. Standard for og sende tekst og tall mellom datamaskiner.
- JSON - JavaScript Object Notation. Er et tekst format som bruker leselig tekst for å lagre og overføre data.
- STM32CubeIDE - Programvare for å enkelt kunne programmere på mikrokontrolleren.
- PWM - Pulsbreddemodulasjon

Kapittel 1

Innledning

Innledningen er et felles kapittel for alle i UiS Subsea som er skrevet av sensorsystemgruppen.

Innledningsvis presenteres årets bacheloroppgaver og definerer hva en ROV er. Deretter introduseres UiS Subsea [1.2] og ROV-konkurransen [1.3] vi skal delta i.

I år er vi totalt 16 elektro- og maskin-studenter som er med på studentprosjektet til UiS Subsea. Målet er å bygge en velfungerende ROV som skal hevde seg i den internasjonale MATE ROV-konkurransen [1.3] som avholdes i USA om sommeren. Samtlige av årets studenter skal skrive bacheloroppgave[1] på prosjektet, derfor har vi valgt å dele oppgavene og gruppene som følgende:

- **Elektro**
  - **Bildegjenkjenning og autonom kjøring** - Bjørnar Wiik og Jens Trydal
  - **Kraftforsyningssystem** - Andrine Pedersen og Anniken Hjelm
  - **Kommunikasjonssystem** - Martin Hausken og Oliver Langvik Veland
  - **Mikro-ROV** - Geir Arne Solland Kindingstad og Mikael Rodal Helgesen
  - **Motorstyrings- og reguleringssystem** - Edmond Baloku og Markus Haldorsen
  - **Sensorsystem** - Daniel Vasshus og Espen Myrset
- **Maskin**
  - **Design og produksjon av ROV-manipulator** - Sindre Fjermedal og Joachim Merenyi Skjervik
  - **Design og montering av ROV-ramme, og ytelsesanalyse av motorer** - Alexander Falch Voerman og Sigvart Daniel Rodriguez Høien

[1] Oppgavebeskrivelsen for hver av oppgavene finnes i kapittel: 1.4

Til slutt vil alle oppgavene bli satt sammen til en mikro- og en vanlig ROV.

1.1 ROV

Et fjernstyrt ubemannet fartøy, også kjent som ROV[2], finnes i mange størrelser og former. De minste kan være på noen få kilo, mens de største[3] kan være på størrelsen av et lite hus og veie flere tonn.

De minste ROV-ene brukes ofte til vitenskaplig forskning i form av opplæring, overvåke plante- og dyreliv, prøvetaking og gi mulighet for å inspirere plasser hvor det er uforsvarlig å sende dykkere. Større ROV-er brukes til reparasjoner, vedlikehold og inspeksjoner av konstruksjoner under vann-overflaten. Ofte vil ROV-ene være utstyrt med kamera, lys, og manipulator men det er ikke uvanlig å utvide bruksområdet med å installere spesialdesignet tilleggsutstyr for gitte arbeidsoppgaver.

Man deler ROV-ene inn i følgende underkategorier:

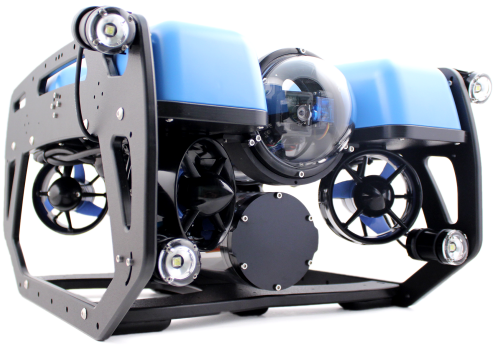
Frittsvømmende ROV med navlestreng[4]

Denne typen ROV er den mest vanlige og gjenkjennes med at den har en navlestreng mellom styrekonsoll og fartøy. BlueROV fra Bluerobotics som vist i figur 1.1 er en frittsvømmende ROV med navlestreng. Denne og tilsvarende fartøy blir ofte utstyrt med kamera, lys, manipulator eller spesialtilpassede verktøy for gitte arbeidsoppgaver.

[2] **ROV**: Fjernstyrt undervannsfartøy (Remotely operated vehicle

[3] "UT-1" en av verdens største er 7.8 meter lang, 7.8 meter bred, 5.6 meter høy og veier 60 tonn. Denne brukes til å installere kabler på havbunnen

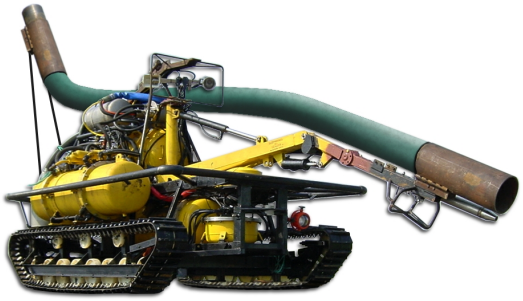
[4] På engelsk: Free-swimming tethered ROV



Figur 1.1: UiS har to stk frittsvømmende ROV-er med navlestreng av typen BlueROV2 fra BlueRobotics. Bilde hentet fra [39]

Krypende ROV[5]

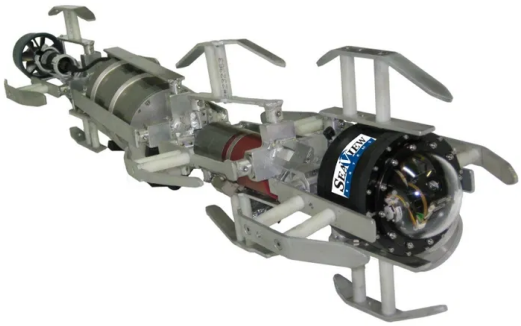
Figur 1.2 viser en krypende ROV. Denne brukes til å krype langs sjøbunnen eller gjennom rør. Ofte graver den ned rør og kabler i sjøbunnen, men kan også utføre inspeksjoner eller brukes til gruvedrift på havbunnen.



Figur 1.2: Krypende ROV av typen Seabed dredger fra Seascape. Hentet fra [40]

Strukturelt avhengig ROV[6]

En strukturelt avhengig ROV som vist i figur 1.3 er primært brukt for å inspisere og vaske konstruksjonen den er festet til. For å bevege seg langs konstruksjonen den er festet til, brukes blant annet hjul, kabel, skinner, taljer eller bruk av hydrauliske løsninger.



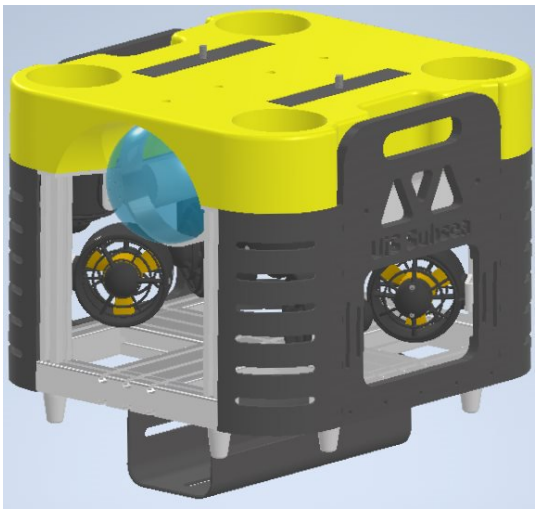
Figur 1.3: Strukturelt avhengig ROV av typen Serpent av Seaview. Hentet fra [41]

[5]På engelsk: Crawling ROV
[6]På engelsk: Structurally Reliant ROV

Hymir

I vårt prosjekt skal vi bygge en ROV av typen frittsvømmende ROV med navlestreng som blir styrt av en pilot ved kontrollpanelet. Denne skal brukes til å utføre diverse arbeidsoppgaver i basseng, men vi ønsker også å ha mulighet for å bruke den i ferskvann og sjø. I tillegg skal vi bygge en mini-rov som er festet til undersiden av ROV-en, denne skal kunne inspisere og hente ut objekter i rør helt ned til en størrelse på 6".

Hele kostnaden for prosjektet blir dekket av organisasjonen UiS Subsea.



Figur 1.4: Hymir

1.2 UiS Subsea

UiS Subsea er en studentorganisasjon ved Universitetet i Stavanger. Organisasjonen har siden 2013 hatt som mål å motivere studenter til å delta i større studentprosjekt, hvor man ønsker å skape et miljø for innovasjon, utvikle tekniske ferdigheter og vise kreativ tilnærming til problemstillinger som dukker opp. Samtidig er det svært viktig for UiS Subsea å styrke samarbeidet mellom universitetet og næringslivet. Tidligere år har det blitt utviklet både ROV [1.1] og AUV-er som bacheloroppgaver for data, elektro og maskin.

Figur 1.5: Logo UiS Subsea

Vi har i løpet av overtakelsen av UiS Subsea gjort store endringer på organisasjonsstrukturen for å gjøre det mer bærekraftig for fremtidig rekruttering og fordeling av arbeid.

I den sammenheng valgte vi å skille mellom styreoppgaver og prosjektoppgaver, slik at styret har ansvar for:

- Organisasjonen UiS Subsea
- Økonomi
- Sponsoravtaler
- Markedsføring
- Rekruttering av nye studenter
- HMS

Noen av oppgavene til prosjektledelsen er:

- Prosjektfremgang
- Finne en passende konkurranse laget skal delta i
- Passe på at prosjektet blir forsvarlig styrt
- Ansvar for at alle studentene er med og drar i riktig retning
- Påse at alle krav og retningslinjer følges

For å fordele ansvar og imøtekomme kravet til MATE ROV-konkurransen om entreprenørskap har vi fordelt rollene som følger:

- Styret

  - **Styreleder:** Daniel Vasshus
  - **Nestleder:** Geir Arne Solland Kindingstad
  - **Økonomiansvarlig:** Edmond Baloku
  - **Markedsføring:** Martin Hausken og Oliver Langvik Veland
  - **Sponsoransvarlig:** Sigvart Daniel Rodriguez og Bjørnar Wiik
  - **Styremedlemmer:** Espen Myrset og Mikal Rodal Helgesen

- Prosjektledelse

  - **Prosjektleder:** Jens Trydal
  - **Teknisk ansvarlig:** Sindre Fjermedal
  - **Lagleder elektro:** Markus Haldorsen
  - **Lagleder maskin:** Joachim Merenyi

1.3 MATE - Marine Advanced Technology Education

MATE Center er en nasjonal organisasjon i USA som samarbeider med skoler, forskningsinstitutt, myndigheteter, militær, og institutt for marinteknikk. Formålet til organisasjonen er å bruke marinteknologi til å inspirere og utfordre studenter ved å løse problemstillinger fra den virkelige verden. For å løse problemstillingene ønsker de at studentene skal ha en kreativ tilnærming til forskning, teknologi og ingeniørfag.

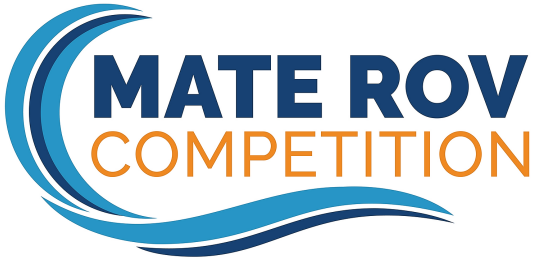
Figur 1.6: MATE logo. Hentet fra [37]

For å oppnå formålet til organisasjonen blir det årlig arrangert en internasjonal ROV konkurranse hvor samtlige lag skal løse en bestemt problemstilling. Konkurransen har fått navnet “MATE ROV Competition”

1.3.1 MATE ROV Competition

Hvert år presenterer konkurransen nye kunder med forskjellige problemstillinger som skal løses, og i årets utgave har vi "verdenssamfunnet" som kunde. Oppgaven som skal løses er å bygge en ROV som skal takle plastproblemet i sjøen, klimaendringens påvirkning på korallrev og konsekvensen med dårlig miljøpraksis på våre vannveier. I konkurransen skal vi ikke gå inn på hva vi må gjøre for å adressere den faktiske årsaken til problemstillingen, kun løse oppgavene som er gitt.

UiS Subsea stiller i utforsker[7] klassen og er den vanskeligste klassen i MATE ROV Competition.



Figur 1.7: Mate ROV-konkurranse logo. Hentet fra [38]

Konkurransen går ut på å løse tre forskjellige praktiske oppgaver, i tillegg gis det poeng på størrelse og vekt. Foruten om det får man poeng for dokumentasjon, presentasjon og sikkerhet. Mer om poengfordeling i delkappittel 1.3.1.4. Vi skal nå ta for oss de praktiske oppgavene:

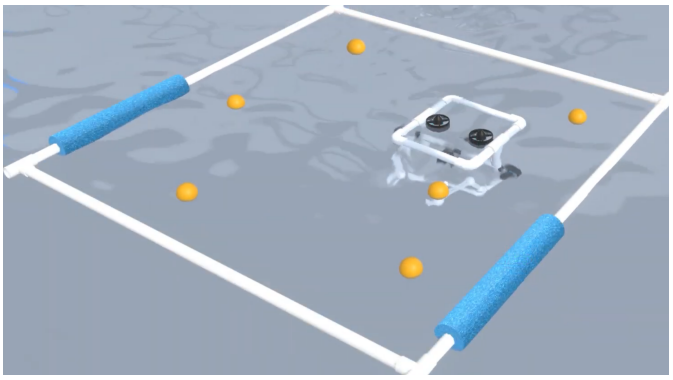
1.3.1.1 Oppgave 1: Problemet med plastforurensing (Totalt 90 poeng)

- **Søppelbøtte til sjøs - "Rydde opp havet, en havn om gangen"**
  - Koble fra gammel strømkontakt til en nyinstallert søppelbøtte - 5 poeng
  - Fjerne tidligere fangstpose av netting fra søppelbøtte - 10 poeng
  - Installere ny fangstpose av netting i søppelbøtta - 10 poeng
  - Koble til en ny strømkontakt[8] til den nyinstallerte søppelbøtta - 20 poeng

[7] I konkurransen heter klassen "Explorer"
[8] Strømkontakten skal vi lage selv

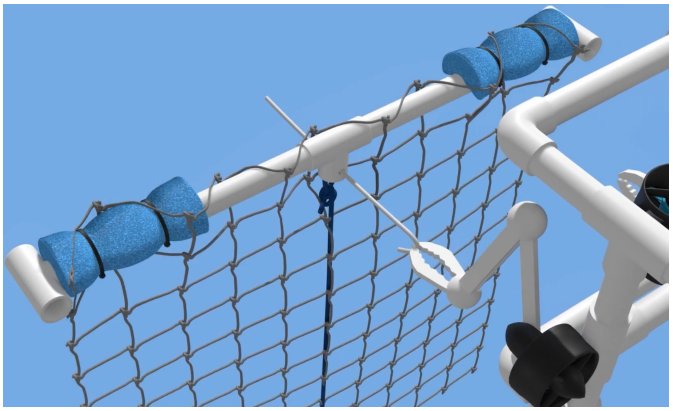
Figur 1.8: Ved at ROV-en fjerner støpselet fra søppelbøtta vil lyset slukkes. Videre i oppgaven fjerner man fangstposen med objekter i, for å så returnere med en ny fangstpose. Hentet fra [44]

- **Utbedring: Fjerne plastavfall fra topp til bunn**
  - Fjerne flytende plastavfall fra vannoverflaten
    - Fjerne alle 6 bitene av avfall - 15 poeng
    - Fjerne 3 til 5 av bitene av avfall - 10 poeng
    - Fjerne 1 til 2 biter av avfall - 5 poeng
    - Fjerne + biter av avfall - 0 poeng



Figur 1.9: Fjerne flytende plastavfall fra vannoverflaten. Hentet fra [44]

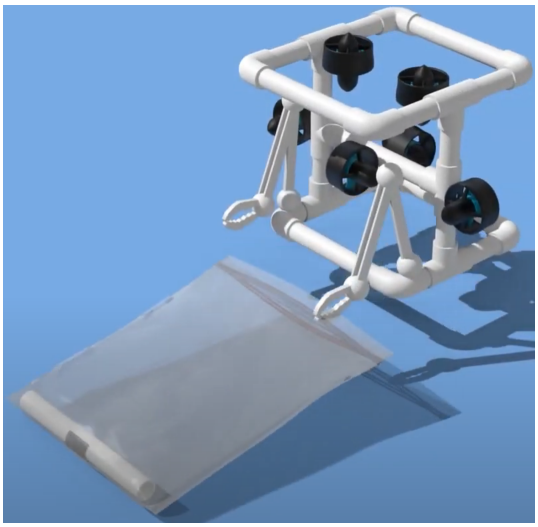
  - Fjerne et spøkelsesnett[9] fra vannet
    - Trekke ut en pinne for å simulere det å kutte spøkelsesnettet fritt - 10 poeng
    - Fjerne spøkelsesnettet fra vannet - 10 poeng



Figur 1.10: Fjerne spøkelsesnett fra vannet. Hentet fra [44]

[9]Spøkelsesnett er et fiskenett som er forlatt eller blitt mistet av fiskere.

– Fjerne plastrester fra bunnen av Marianergropen[10] - 5 poeng for hver, maks 10 poeng

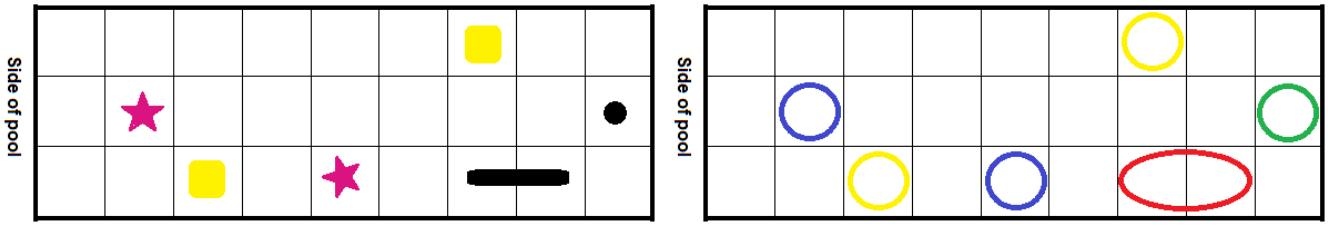


Figur 1.11: Fjerne plastrester fra havbunnen. Hentet fra [44]

1.3.1.2 Oppgave 2: Den katastrofale virkingen av klimaendringen på korallrev (Totalt 90 poeng)

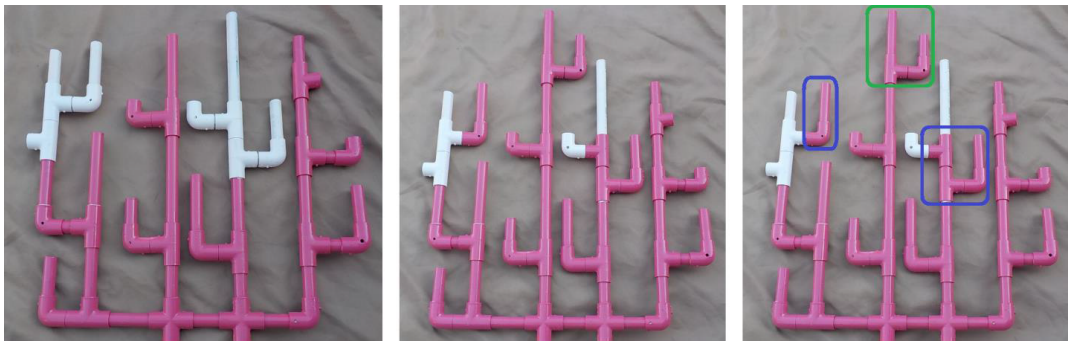
- **Kjøre i en transektlinje[11] over et korallrev og kartlegge interessepunkt**
  - Kjøre i en transektlinje over et korallrev
    - Kjøre transektlinjen autonomt - 15 poeng
    - Kjøre transektlinjen manuelt - 5 poeng
  - Kartlegge interessepunkt i korallrevet
    - Automatisk kartlegging på en dataskjerm - 10 poeng
    - Manuell kartlegging på en dataskjerm - 5 poeng

[10] Marianergropen er verdens dypeste havsgrop og ligger vest i Stillehavet og øst for Marianene
[11] En transektlinje er en linje på tvers av et habitat, eller deler av et habitat



Figur 1.12: Kjøre over korallrevet til venstre i figuren. Her er det to lokasjoner for å plante ut korall-fragmenter (gul firkant) i , to sjøstjerner (lilla stjerner), en korallkoloni (svart rektangel) en svamp (svart prikk). Høyre: Fargede sirkeloverlegg i riktige ruter for vise interessepunkt og organismene på revet. Hentet fra [42]

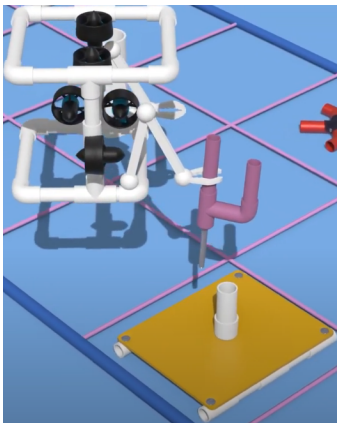
- **Bruke bildegjenkjenning for å bestemme helsen til en korallkoloni ved å sammenligne nåværende tilstand med tidligere data**
  - Bruke bildegjenkjenning for å bestemme helsen til korallkolonien
    - Alle endringer er identifisert - 20 poeng
    - Minst en, men ikke alle endringene er identifisert - 10 poeng
    - Ingen endringer er identifisert - 0 poeng
  - Bruke en håndbok for å bestemme helsen til korallkolonien - 5 poeng



Figur 1.13: Venstre: Bilde av korallkolonien for ett år siden. Senter: Kolonien slik den er i dag. Høyre: Korallkolonien med alle områder identifisert, et område med vekst og to området med gjenoppretting fra bleking/flekker. Hentet fra [42]

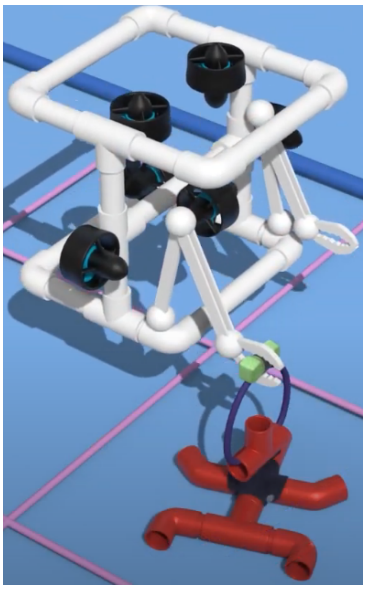
- **Gro koraller på rev**
  - Fjerne korall-fragmenter fra planteskolen[12] - 5 poeng for hver, maks 10 poeng
  - Plante ut korall-fragmenter på bestemte områder i revet - 5 poeng for hver, maks 10 poeng

[12]Gartneri der det dyrkes planter som siden plantes ut



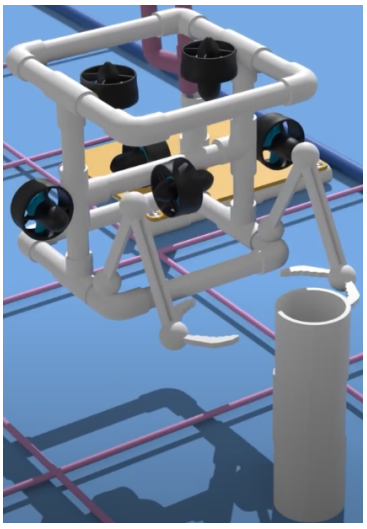
Figur 1.14: Gro koraller på rev. Hentet fra [44]

- **Begrense et utbrudd av tornekronesjøstjerne**[13] - 5 poeng for hver, maks 10 poeng



Figur 1.15: Begrense utbrudd av tornekronesjøstjerne. Hentet fra [44]

- **Samle prøver av svampearter for farmasøytisk forskning**
  - Hente en prøve fra en svamp - 10 poeng
  - Returnere prøven til overflaten - 5 poeng



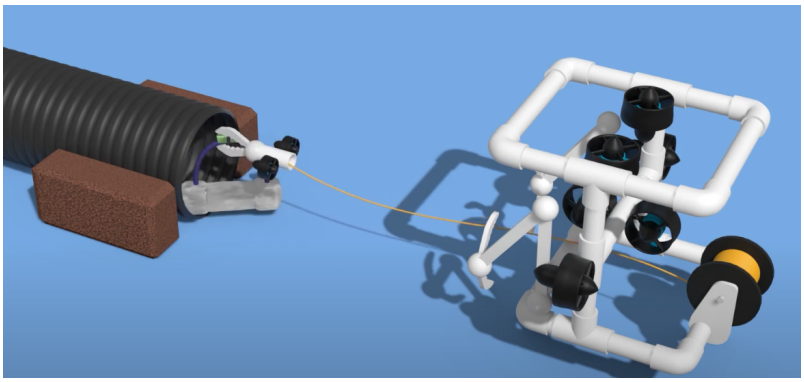
Figur 1.16: Hente prøve fra svampearter. Hentet fra [44]

[13]Tornekronesjøstjerne er en av verdens største sjøstjerner og er beryktet for å skade koraller i tropiske farvann

1.3.1.3 Oppgave 3: Vedlikeholde sunne vannveier. Del II: Delawarebukten og elva (Totalt 90 poeng)

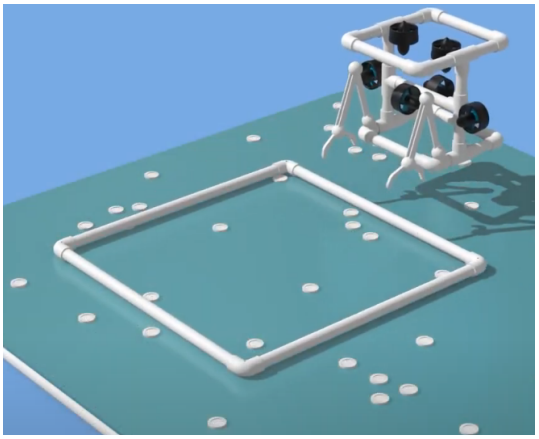
Delarwarebukten er en dyp bukt og er en forlengelse av elven Delaware på USAs nordøstlige kyst. Ferskvannet renner ut i Atlanterhavet.

- **Hente ut en sedimentprøve fra innsiden av et avløpsrør for analyse av forurensing**
  - Utplassere en enhet i røret for å hente sedimentprøven - 25 poeng
  - Returnere sedimentprøven til overflaten - 10 poeng
  - Bestemme hvilken type forurensing(er) som er tilstede i sedimentprøven - 5 poeng



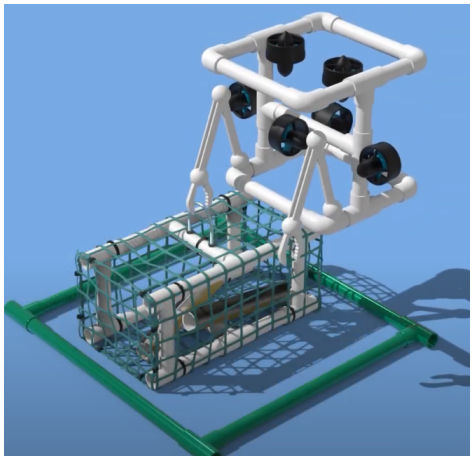
Figur 1.17: Hente ut en sedimentprøve fra et 6rør. Hentet fra [44]

- **Estimere antall blåskjell i en blåskjell-klynge**
  - Utplassere en kvadrant og telle antall blåskjell i kvadranten - 5 poeng
  - Estimere antall blåskjell og hvor mye vann som blir filtrert av blåskjellene - 5 poeng



Figur 1.18: Estimere antall blåskjell

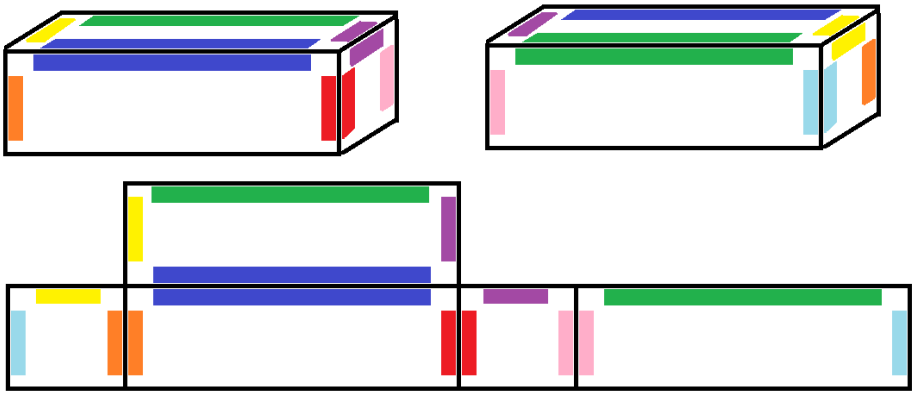
- **Restaurere ålebestanden**
  - Fjerne en full åleteine fra et bestemt område - 10 poeng
  - Plassere en tom åleteine i et bestemt område - 10 poeng



Figur 1.19: Fjerne og plassere ut åleteine. Hentet fra [44]

- **Lage fotomosaikk[14] av en nedsenket togvogn for å skape et kunstig rev**
  - Autonomt - 20 poeng
  - Manuelt - 10 poeng

[14]Fotomosaikk er en samling av bilder satt sammen til ett bilde



Figur 1.20: Topp: Diagram med fargesammensetting på togvognen. Bunn: Fotomosaikk av togvognen som vist på dataskjermen. Hentet fra [42]

1.3.1.4 Poengfordeling

For å kåre en vinner av konkurransen, vil samtlige poeng man har samlet, lagt sammen, og laget med flest poeng blir kåret som vinner.

I listen nedenfor har vi en oversikt på alle delene vi kan hente poeng på.

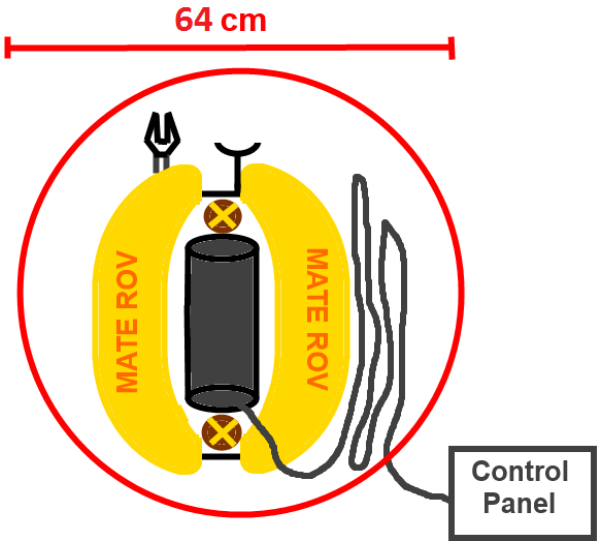
Produktdemonstrasjon	
Produktdemonstasjon (oppgaver)	270 + tidsbonus
Størrelse- og vektrestriksjoner	20
Produktdemonstrasjon av organisasjonseffektivitet	10
Prosjektering og kommunikasjon	
Teknisk dokumentasjon	100
Produktpresentasjon	100
Markedsføring	50
Selskapets spesifikasjonsark	10
Bedriftsansvar	20
Sikkerhet	
Sikkerhet og dokumentasjon gjennomgang	20
Sikkerhetsinspeskjon	30
Sikker jobb analyse (SJA)	10
Sum poeng:	**650 + tidsbonus**

1.3.1.5 Tekniske krav til ROV-en

Konkurransen [1.3] stiller strenge krav til ROV-ene som skal delta. Disse kravene setter føring på hvordan vi ønsker å løse våre prosjektoppgaver. I produktmanualen til konkurransen [42] er det en full oversikt over krav man må følge. Her vil vi gi en kort oversikt på de aller viktigste punktene. Spesifikke krav som angår enkelte grupper blir belyst i bacheloroppgavene hvor det er relevant.

Fysiske krav

I årets utgave er den fysiske grensa for å kunne delta satt til å være 92 cm i diameter og kan ikke veie mer enn 35 kg. Hele ROV-en, med tilleggsutstyr og navlestreng skal passe innenfor en ring som vist i figur 1.21



Figur 1.21: ROV med verktøy og navlesteng kveilet opp ved siden av ROV-en. Hentet fra [42]

Videre får man ekstrapoeng for å være innenfor bestemte størrelse og vektklasser, som vist i tabellen under.

Størrelse	**Poeng**	**Vekt (på land)**	**Poeng**
<64 cm diameter	+ 10 poeng	<20 kg	+ 10 poeng
65.1 til 75 cm diameter	+ 5 poeng	20.01 kg til 28 kg	+ 5 poeng
75.1 til 92 cm diameter	0 poeng	28.01 til 35 kg	+ 0 poeng

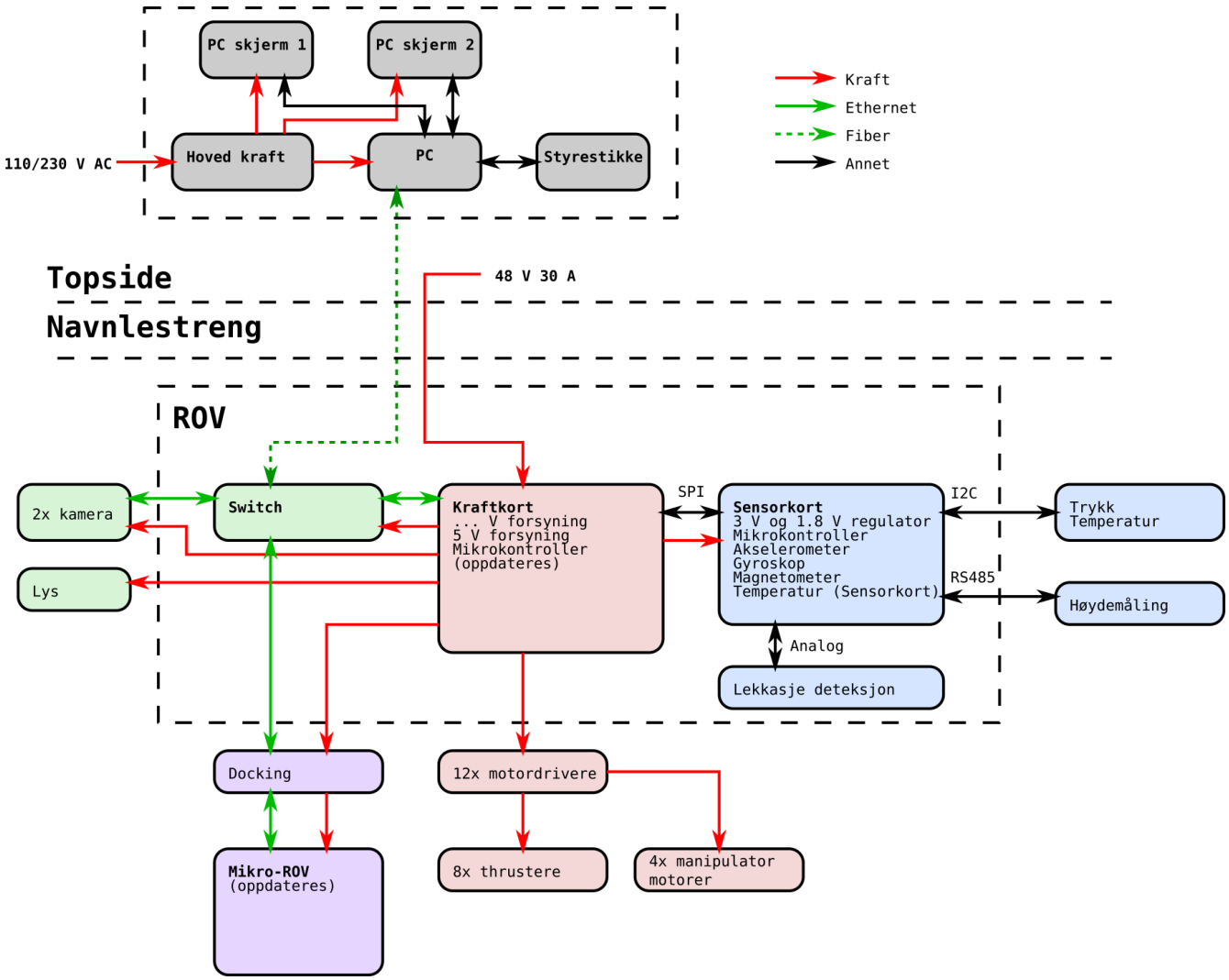
Tekniske løsninger

Det er kun lov å bygge en ROV til å utføre oppgavene i bassenget, men i oppgaven hvor man skal hente ut sedimentprøven1.3.1.3 har man lov å bygge en mikro-ROV.

Videre skal ROV-en kunne kjøre i et rent klor-basseng med en temperatur mellom 15 og 30 °C. Dybden på bassenget er maks 7 meter og samtlige oppgaver foregår innenfor 10 meter fra bassengkanten. Styrekonsollen blir plassert nær bassengkanten (maksimalt 3 meter), navlestrengen må være lang nok til å utføre alle oppgavene.

MATE ROV Competition vil ha tilgjengelig 48 v og 30 A DC ved styrekonsollen som skal forsyne ROV-en. Det er først lov å endre på spenningsnivået på innsiden av ROV-en. Konkurransen stiller ekstra strenge krav til pneumatikk, hydraulikk og laser hvis man velger å bruke dette. Tas dette i bruk må man følge spesifikasjonene og dokumentere utstyret i henhold til konkurransemanualen [42].

1.4 Overordnet blokkskjema



Figur 1.22: Blokkskjema av ROV

1.4.0.1 Bildegjenkjenning og autonom kjøring

Oppgaven består av to hoveddeler, bildebehandling og brukergrensesnitt. Den første går ut på å lage algoritmer som løser oppgavene om bildebehandling- og autonom kjøring, fra MATE ROV-konkurransen. Den andre delen går ut på å lage et nettbasert styringsprogram/brukergrensesnitt til ROV-en. I tillegg til disse to oppgavene inneholder bacheloren litt om prosjektledelse.

1.4.0.2 Kraftforsyningssystem

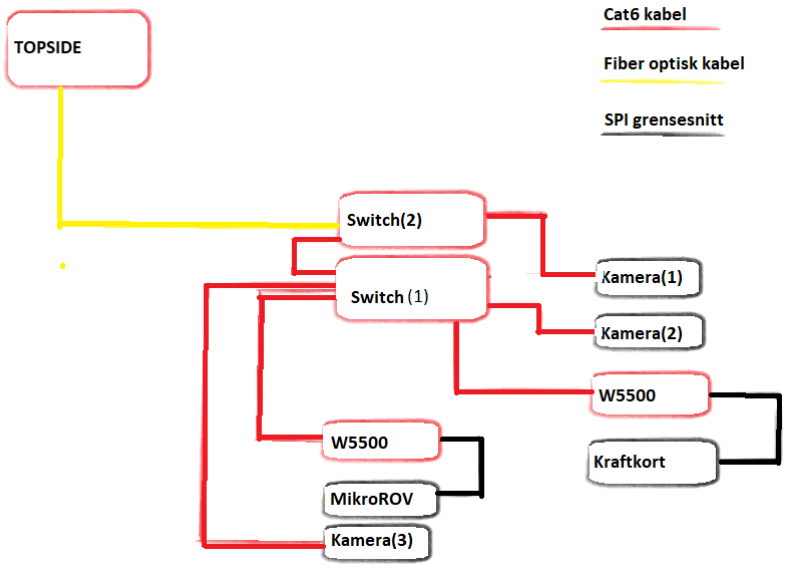
Oppgaven består av å utvikle og konstruere et kraftfordelingskort for ROV, og en navlestreng for krafttilførsel fra overflatesystemet til ROV-en. Kraftfordelingskortet skal kunne forsynes med en spenning på 48 - 56 V DC med på maksimalt 30 A strøm.

Tilførsel av kraft skjer ved hjelp av navlestrengen, som også vil bli brukt til kommunikasjonsoverføring (fiberkabel). De ulike komponentene i ROV-en forsynes med ulik spenning, derfor vil det være behov for spenningsregulatorer for 12 V, 5 V og 3,3 V på kraftfordelingskortet inni ROV-en. ROV-ens elektronikkhus er et lukket system, ved bruk av kjøleribber og vifter vil en få kjøling og sirkulasjon av luften ved de varmeste delene i elektronikkhuset

1.4.0.3 Kommunikasjonssystem

I kommunikasjonssystemet i årets ROV skal vi bruke en SPI/Ethernet omformer som gjør at vi kan kommunisere med mikrokontrolleren i ROVen over ethernet. Vi skal her lage et program som gjør at vi kan lese av informasjon om ROV-en og sende ned styrings data fra XBOX-kontrollen og reguleringen. Mikrokontrolleren blir så koblet til en ethernet svitsj med flere porter som sender ethernet pakkene til topside. Kommunikasjonen fra ROV til topside skal være over en fiberoptiskkabel.

Vi har også ansvar for å sette opp kommunikasjon til kamera og lys. Til kamera skal vi bruke IP-kamera som sender videostrøm over ethernet. Vi skal ha et kamera inne i kuppelen som viser hva som skjer foran ROVen og som gir oss oversikt over manipulatoren. I tillegg til dette har vi et kamera under som skal filme havbunnen. Til belysning på ROVen skal vi bruke noen små LED dioder som blir montert rundt omkring etter behov. LED diodene blir så koblet i serie opp mot en driver som sender ut den spenningen vi må ha for å drive lysene. Her må vi bruke en loddebolt til å feste ledningene på LED diodene. Vi må også finne en måte å gjøre dette vanntett. Figur 1.23 viser et utdrag av hvordan kommunikasjons oppsettet er planlagt. Videre i rapporten skal vi se nærmere på hvordan kommunikasjonen og video overføringen samt belysningen blir realisert. Vi går først gjennom behov og valg av kamera og lys før vi til slutt går igjennom hvordan kommunikasjonen blir valgt og konfigurert.



Figur 1.23: Blokkskjema over kommunikasjonssystemet

Her ser vi hvordan vi hvordan koblingen av kommunikasjonssystemet blir og hvilke transmisjonsmedium som blir brukt. Vi skal bruke fiber fra datamaskinen på land ned til en svitsj hvor vi går videre med Ethernet-kabler til SPI/Ethernet-modulen og kameraene.

1.4.0.4 Motorstyring- og reguleringssystem

ROV-en består av 12 motorer, som skal sørge for et robust og driftssikkert motorstyrings- og reguleringssystem for fremdrift, samt et robust og driftssikkert styringssystem for manipulator. ROV'en skal bli styrt av en pilot ved hjelp av et brukergrensesnitt som videre kommuniserer med ROV-en. ROV-en skal også reguleres i frihetsgradene rull, stamp og hiv.

1.4.0.5 Sensorsystem

Oppgaven består av å utvikle og konstruere sensorsystem og elektronikkhus til ROV-en. Sensorsystemet består av flere forskjellige sensorer som skal informere om statusen til ROV-en i vannet, og målingene som mottas kalibreres og brukes til utregninger før det kan brukes av de andre delsystemene. Elektronikkhuset inneholder alt av elektronikk, og har konnektorer for å koble sammen utstyr på utsiden med elektronikken på innsiden. Videre i kapittel 2 og utover skal vi fokusere på planlegging, utvikling og produksjon av sensorsystemet og elektronikkhuset.

1.4.0.6 Design og kontroll av ROV-manipulatoren

Oppgaven består av å utvikle og konstruere en manipulator samt klype for ROV-en. Manipulatoren skal være i stand til å utføre gitte oppgaver i MATE konkurransen, samt være modulær. Det skal også lages ulike simuleringer av manipulatoren for å øke forståelsen av de ønskede bevegelser og laster.

1.4.0.7 Design og montering av ROV-ramme, og ytelsesanalyse av motorer

Rammegruppen konstruerer et rammedesign basert på *Design for Assembly* konseptet. Det er modulært tilpasset slik at andre komponenter kan plasseres på ROV-en. ROV-en skal ha et lavt *Center of mass* og høyt *Center of buoyancy* for å ha stabil og ha høy manøverabilitet. Alle rammekomponenter styrkeanalyseres og det foretas en thrusteranalyse for plassering av thrustere for å få geometrisk optimalisert design basert på vektklasse og størrelse. Samtidig blir Autodesk benyttet til å beregne ROV-ens faktisk bevegelighet i vann og vektoranalyser i alle seks aksene ROV-en skal bevege seg i.

1.4.0.8 Mikro-ROV

For å hente ut et objekt fra et 6 tommers rør skal vi utvikle en Mikro-ROV som skal dokkes til Hoved-ROV-en som igjen forsyner den med strøm og signaler via en navlestreng. Oppgaven består av maskinvare, kretskort, programvare, design, fysikk og mekanisk arbeid.

Kapittel 2

Kamera

Innhold

2.1 Kamerabehov

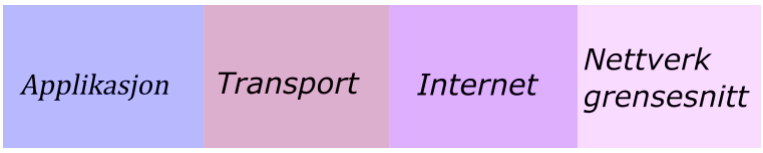
Når vi skal velge kamera er det flere ting å ta hensyn til. I ROV-en bør vi ha et kamera som gjør at vi får god kvalitet på bildene under vann slik at piloten klarer å kjøre uten problemer. Vi må også tenke på forsinkelser ved overføring og komprimering av bildene, synsvinkelen til linsen og støy fra andre elektriske komponenter. Kameraets størrelse er også av betydning, vi ønsker at kameraet er mest mulig kompakt og veier minst mulig.

2.1.1 Kamera alternativ

Etter våre behov og ønsker er det mange forskjellige typer kamera å velge mellom, de mest brukte til undervannsfartøy er Analog og IP(Internet Protocol) vi har derfor bestemt oss for å gå litt mer innpå forskjellene i disse kameratypene.

2.1.1.1 IP-Kamera

IP-kamera er et digitalt videokamera som er koblet til internett. IP-kamera er som oftest brukt til videoovervåkning og sikkerhet, men kan også brukes til andre ting. Slike kamera sender bilder digitalt ved bruk av overføring og sikkerhetsfunksjonen til TCP/IP(Transmission Control Protocol/Internet Protocol) protokollen. TCP/IP protokollen gjør at vi enkelt kan koble sammen flere typer system. Protokollen er bygget opp i 4 forskjellige lag som tar for seg forskjellige oppgaver. Figur 2.1 viser hvordan de 4 lagene i en TCP/IP-protokoll er delt opp. Det første laget er Applikasjon her får vi tak i grensesnitt og protokoller som vi får bruk for. De mest brukte protokollene er HTTP(Hypertext Transfer Protocol), FTP(File Transfer Protocol) og SMTP(Simple Mail Transfer Protocol). Neste lag er transportlaget, her blir dataen vi får fra applikasjonslaget delt opp i segmenter slik at vi får en sekvens. Transportlaget bestemmer også hvor dataen skal leveres og hvor fort dette skal gå. Etter transport har vi Internettlaget, hovedoppgaven her er å sende pakker med data til den destinasjonen pakkene har. Det siste laget i en TCP/IP-modell er et nettverk grensesnitt. Nettverks grensesnitt laget blir delt opp i to deler. Disse delene er datalinklaget og det fysiske laget. Her bestemmes det hvordan dataen fysisk skal sendes gjennom nettverket. Her velger vi også hvilket nettverksmedium vi skal bruke, f.eks. fiber optisk, catx kabel eller koaksial kabel. Dette laget er også ansvarlig for overføring av data mellom enhetene på nettverket. [1] Fordelene



Figur 2.1: TCP/IP-modell

med et slikt kamera er at vi også kan bruke PoE(Power over Ethernet) som gjør at vi kun trenger en Ethernet-kabel til både strømforsyning og videooverføring. Dette gjør at vi kan overføre over større lengder og vi slipper en ekstern strømforsyning til kamera. Vi har også muligheten til og enkelt endre oppløsningen og komprimeringen til kamera i programvaren. Det er selvsagt noen ulemper også med IP-kamera. Noen av ulempene er at forsinkelsen i systemet blir høyere jo bedre oppløsning vi har på bildene. IP-kamera er ofte også litt dyrere enn analoge.

2.1.1.2 Analoge kamera

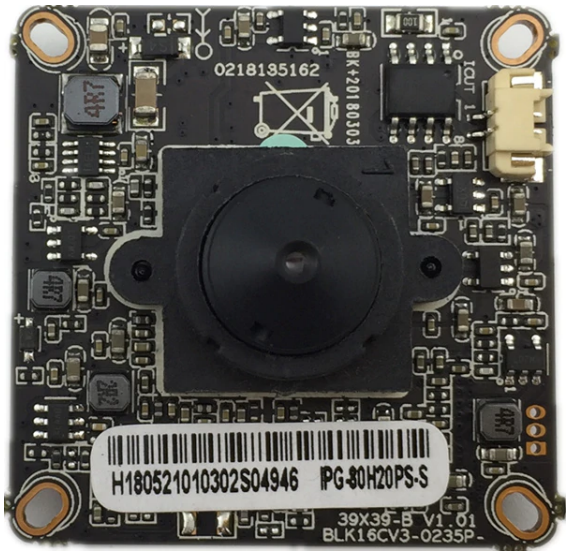
Analoge kamera sender analoge signaler direkte gjennom en kabel opp til topside hvor vi kobler oss opp til PC-en vi skal styre ROV-en med. Her blir det analoge signalet omgjort til et digitalt signal som gjør at vi kan se bildene på skjermen. Analoge kamera krever ikke like mye båndbredde som et IP-kamera og er enklere å sette opp. Her går kabelen direkte opp uten noe tilkobling via en svitsj slik vi må i IP-kamera. Analoge kamera blir ofte brukt i slike systemer på grunn av vi får veldig liten forsinkelse ved bruk av analoge kamera. De er også litt billigere enn andre alternativ. Det er også noen ulemper med analoge kamera. Den største ulempen er støy fra andre elektriske ledninger som kan påvirke signalet. Dette gjør at vi kan få en dårligere videokvalitet enn vi ønsker. Dette gjør at det blir vanskelig for piloten å navigere og skille objekter under vann.[2] Et annet problem som vi må ta hensyn til er vekten på kabelen, ved bruk av analoge kamera må vi ha en kabel til hvert kamera som går fra ROV-en og opp til topside. Dette gjør at vi får 3 ekstra kabler i navlestrengen, noe som gjør at vi får en navlestreng som er tyngre enn det vi ønsker.

2.1.2 Valg av kamera

Etter å ha sett på ulike kameraalternativ og fordeler og ulemper, har vi bestemt oss for å bruke IP-kamera i vår ROV. Dette passer best til vårt behov. Vi får da et kamera med litt høyere forsinkelse, men til gjengjeld får vi bedre videokvalitet. Vi har i år bestemt oss for å bruke to kamera på vår ROV, et kamera fremme i ROV-en som viser hva som skjer foran ROV-en og et kamera som er plassert under. I konkurransen går den ene oppgaven ut på å ta målinger ved hjelp av bildebehandling, vi må av den grunn ha et kamera som filmer havbunnen. Vi har blitt enige om at det beste blir å ha et kamera inne i selve kuppelen og et kamera under som blir plassert i et vanntett kamerahus. Etter litt søking og lesing om forskjellige kamera endte vi opp på et kamera fra Grandado.no som heter VOLDRELI. I figur 2.2 ser vi hvordan kamera modulen ser ut. [3] Dette er kun en kameramodul. Det vil si kun innmaten i kamera, vi må altså lage selve kamerahuset selv.

2.1.2.1 Spesifikasjoner til kamera

Dette kamera har en oppløsning på 1080p(1920x1080 piksler) noe som gjør at vi får god nok kvalitet på bildene. Vi må også tenke på hvordan vi skal komprimere bildene slik at vi får minst mulig



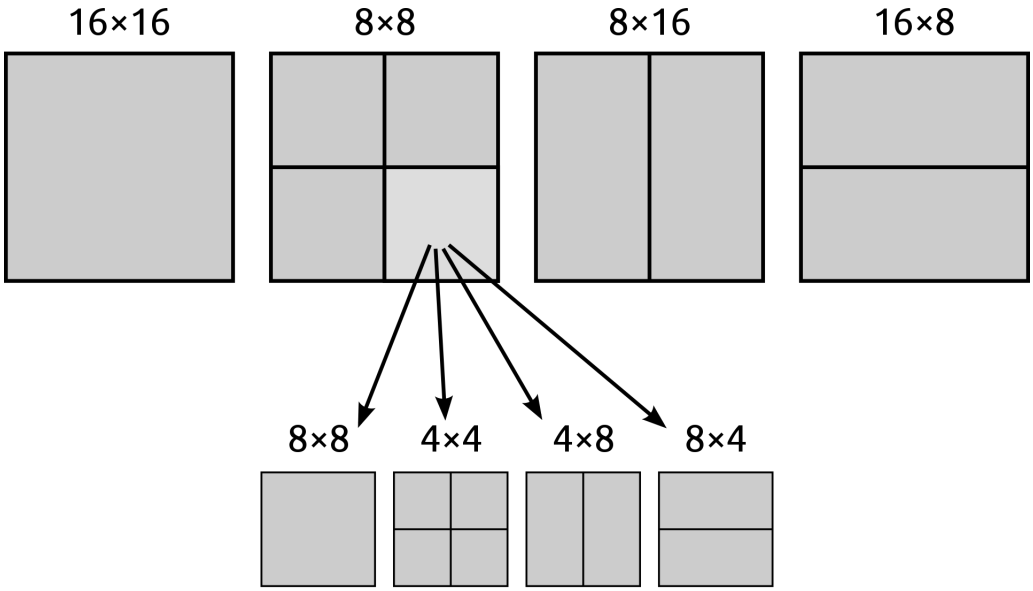
Figur 2.2: VOLDRELI kamera [3]

forsinkelse og best mulig kvalitet. Med dette kamera har vi valget mellom å bruke H.264(HAVC) eller H.265(HEVC) til å komprimere bildene. Noen av forskjellene på H.264 og H.265 er:

- HEVC gir omtrent dobbelt så bra komprimering som HAVC.
- HEVC trenger mindre båndbredde enn HAVC.
- HEVC kan støtte oppløsning opptil 8192x4320 piksler, men HAVC kan bare støtte opptil 4096x2304
- HAVC krever mindre datakraft og mindre strøm enn HEVC

Felles for begge er metoden som blir brukt. Bildet blir delt opp i blokker som inneholder samme farger. Det blir så delt opp i mindre og mindre blokker for å prøve å forutsi hvordan bilder kommer til å bli slik at vi får minst mulig endring mellom hvert bilde. Dette gjør at vi kun trenger å overføre data fra de blokkene som endrer seg. H.264 bruker blokker opptil 16x16 piksler, mens H.265 bruker blokker opptil 64x64 piksler. I figur 2.3 ser vi hvordan blokkene blir delte opp. [5] [6] Altså HEVC gir bedre kvalitet, bedre komprimering og trenger mindre båndbredde, men trenger altså en del mer datakraft og krever mer strøm. Mer om dette kommer i testfasen.
Noen av de andre spesifikasjoner til kamera er:

- Kamera har et strømforbruk på under 5W
- Vi kan velge mellom 12V DC strømforsyning eller 48V PoE(Power over Ethernet) strømforsyning.



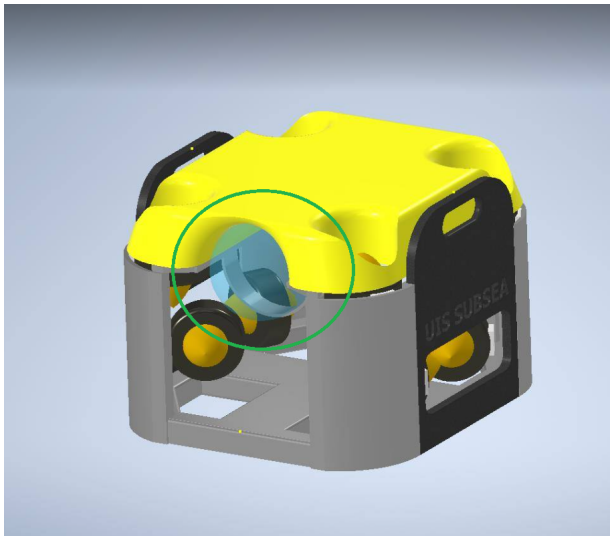
Figur 2.3: H.264 oppdeling [4]

- Kamera har en synsvinkel på 90°slik at vi får god oversikt.

Alt i alt virker dette som et kamera som funker bra til vårt behov, men vi må også gjennomgå en del testing før vi kan si oss helt sikre.

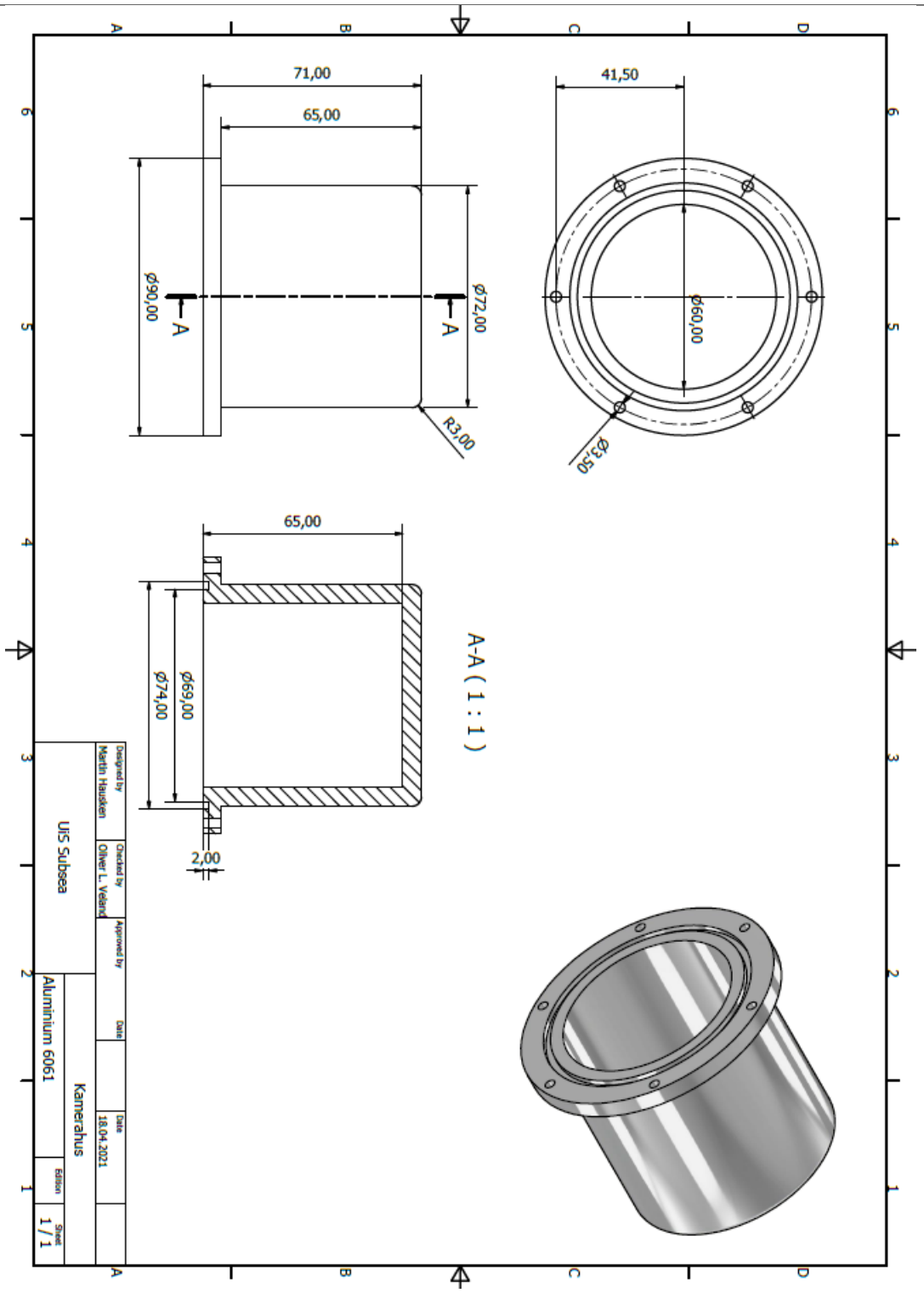
2.1.3 Plassering av kamera

Vi må plassere kameraene slik av vi kan utføre MATE- konkurransen best mulig. I forhold til bildebehandling er vi nødt til å ha minst 2 kamera, ett foran og ett under. Det som er viktig for bildebehandlingen er at de får klare bilder som ikke blir blokkert av noe. Det som vi også trenger kamera til er ved bruk av manipulatoren. Den må vi ha full sikt over når den skal brukes. Vi plasserer derfor et kamera i fronten av ROV-en. Når vi skal utføre bildebehandling i front skal vi ta av manipulatoren. Med tanke på at vi kan fjerne manipulatoren under konkurransen så trenger vi kun å bruke 2 kamera. På figur 2.4 ser vi en skisse over hvor kamera skal plasseres.

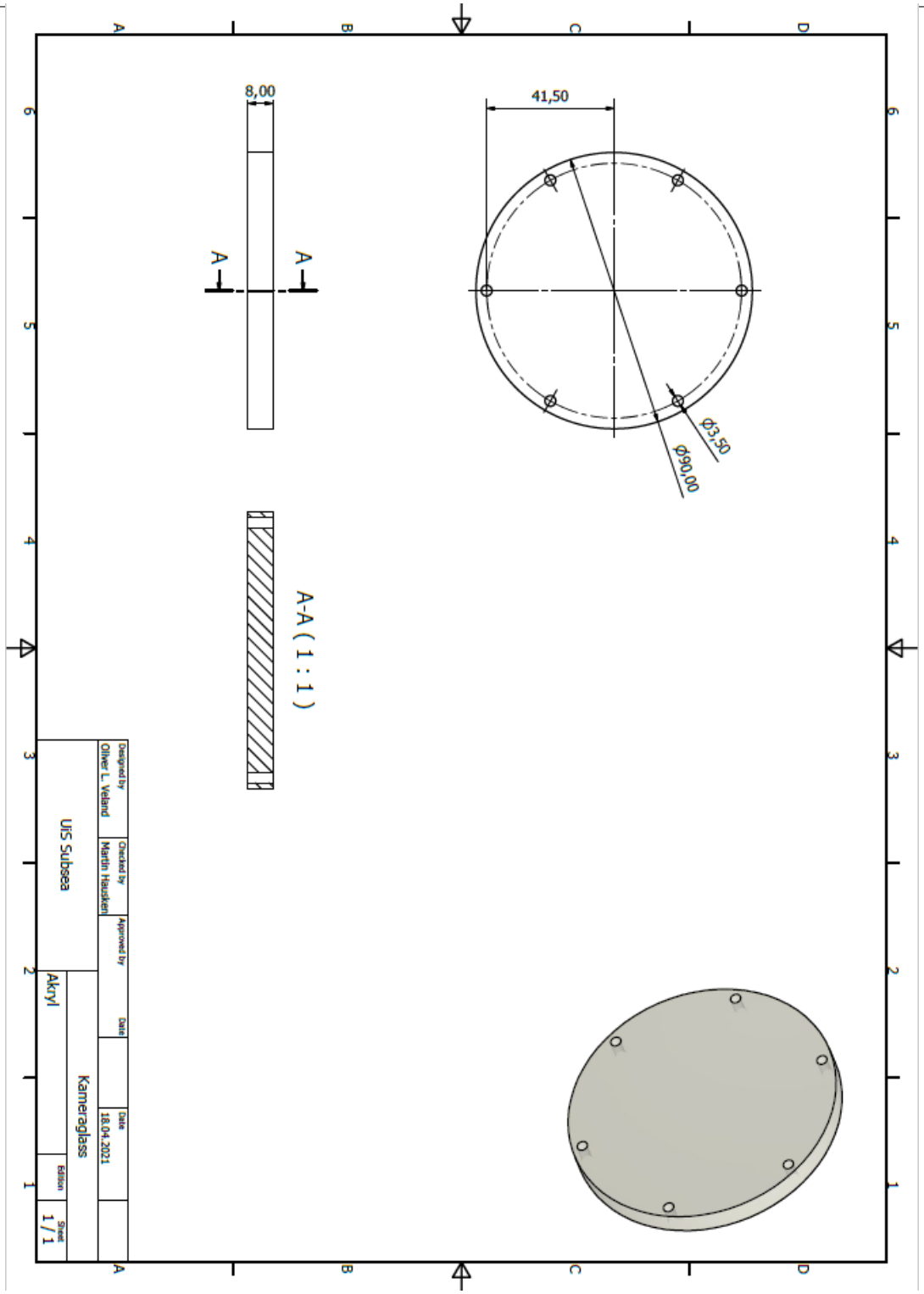


Figur 2.4: Plassering av kamera framme på ROV-en

Til kamera under må vi ha et hus som er vanntett som kamera skal plasseres i. Her har vi snakket sammen med maskiningeniørene og fått laget et kamerahus som vi skal bruke på ROV-en. Kamerahuset skal være i aluminium. Her har vi vært på verksted og maskinert ut en gjennomsiktig plate i akryl som skal brukes til vindu i kamerahuset. I bakenden av kamerahuset skal vi ha en plugg som gjør at vi kan føre kabelen inn i kamerahuset og holder dette tett. Figur 2.5 viser tegningen over kamerahuset. Her har vi og med dimensjonene til huset. Huset er 71mm langt og har en diameter på 90mm. Figur 2.6 viser tegningen av akrylglasset som skal plasseres på kamerahuset.



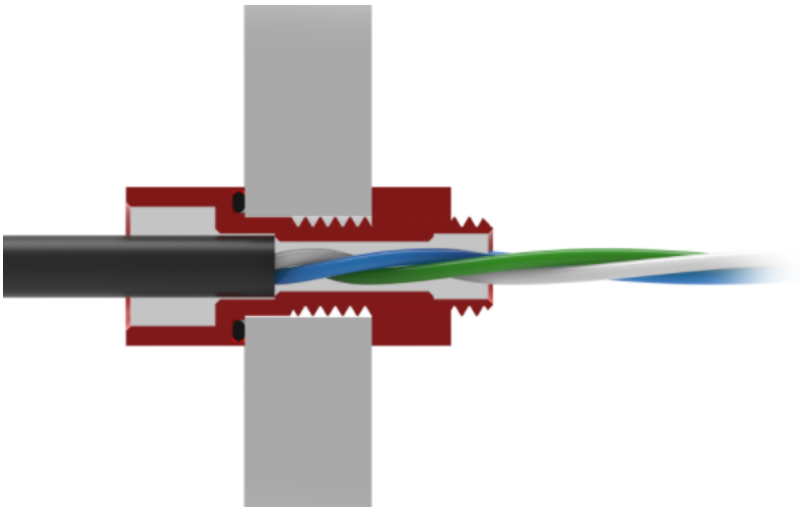
Figur 2.5: Kamerahus tegning



Figur 2.6: Tegning av akrylglasset som skal brukes til vindu i kamerahuset

For og få kabelen inn i kamerahuset blir det brukt en plugg fra BlueRobotics.com. [34] Med denne pluggen kan vi enkelt føre kabelen inn i kamerahuset. Her må vi lage et hull i baksiden av huset for å feste pluggen.

Figur 2.7: Pluggen vi skal bruke fra BlueRobotics [34]



Figur 2.8: Kabelinnføring i pluggen [34]

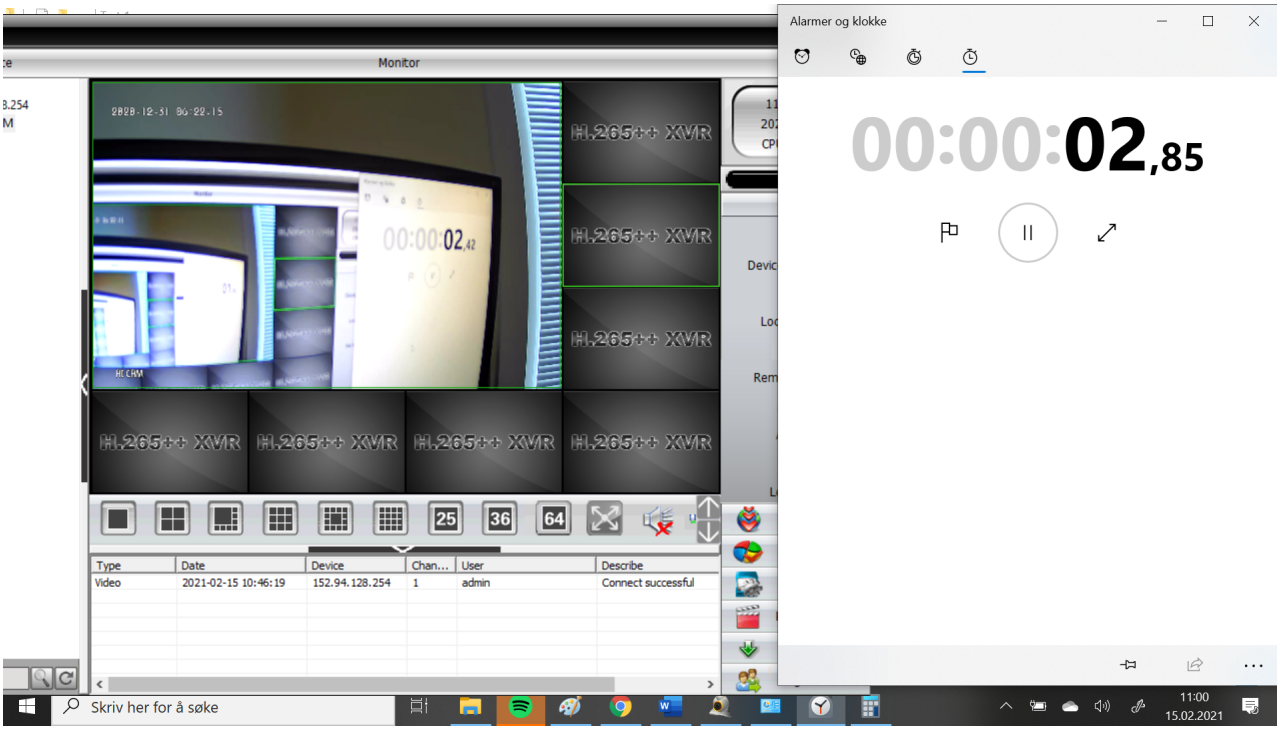
I figur 2.7 ser vi hvordan pluggen ser ut. Pluggen er veldig liten og billig slik at den passer bra til å brukes i kamerahuset. Figur 2.8 viser hvordan kabelen blir ført inn i kamerahuset. Etter kabelen er ført inn i huset blir denne koblingen gjort vanntett ved bruk av epoxy som vi fyller i åpningen.

2.1.4 Grensesnitt CMS

CMS er ett video overvåkingsprogram vi bruker til testing av kamera vårt. Dette er programmet vi ble anbefalt å bruke av de vi kjøpte kamera av. Her kobler vi til kamera vårt ved å koble det til en svitsj sammen med Pc-en. På dette programmet har vi mange forskjellige muligheter for å endre innstillinger på kamera. CMS har også en app så du kan koble til å se kamera via telefonen. Dette programmet er bra for bruk av overvåknings kamera. Du kan se opp til 64 kamera samtidig på programmet. Det finnes også opptaksmuligheter med CMS, så om vi skal teste ROV-en tidlig så er det mulig å ta det opp med CMS. CMS vil kun bli brukt til testing. Til MATE-konkurransen skal vi mest sannsynlig bruke Python som grensesnitt. CMS var et gratis og veldig enkelt program å bruke.

2.2 Testing av kamera

Nå skal vi teste forsinkelsen vi har fra PC-en vår til IP kameraet. Vi skal gjøre dette ved å ha en stoppeklokke på PC-en og filme med kameraet på PC-en som vist på bilde.

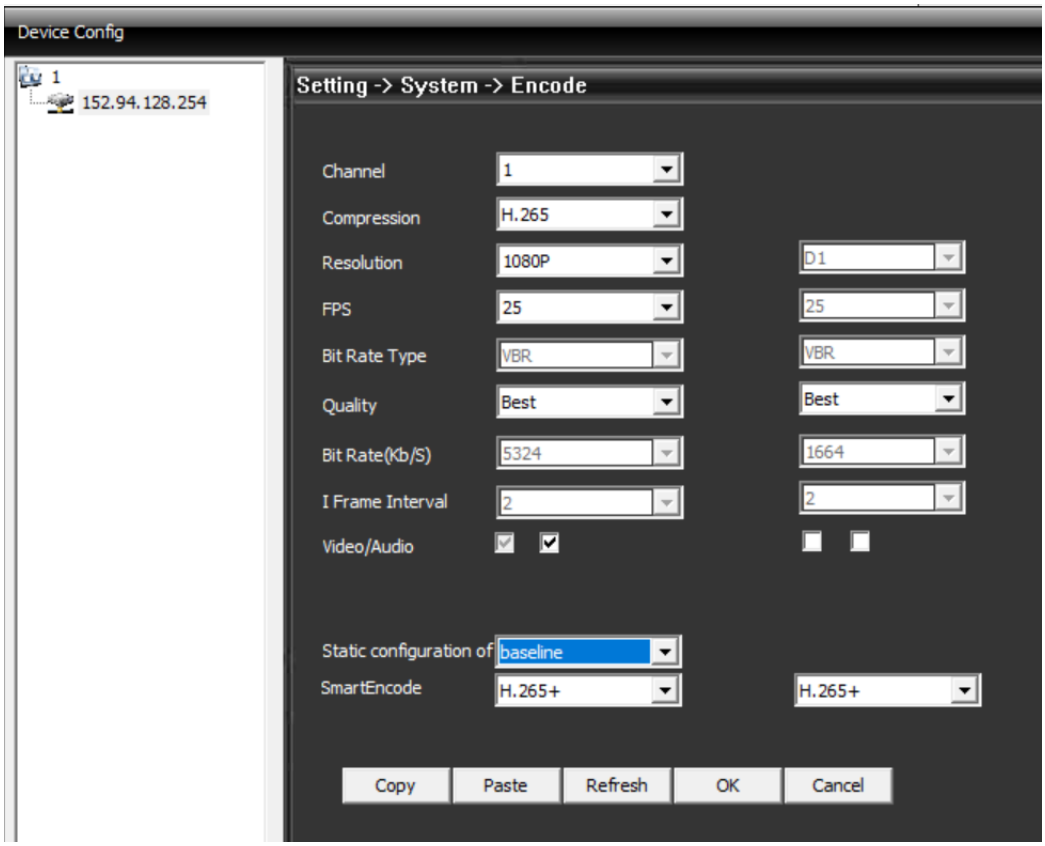


Figur 2.9: Kameratest

Vi koblet til kamera til internett og sjekket forsinkelsen ved hjelp av en stoppeklokke som vist på figur 2.9. Vi bruker en stoppeklokke på PC-en for å måle etterslepet. Vi filmer med ett kamera på PC skjermen. Her tar vi så ett skjermbilde av skjermen på PC-en, så får vi stoppeklokken på kamera samtidig som vi får stoppeklokken på PC-en. Vi tar da 2.85 sekund minus 2.42 sekund og får da et etterslep på 430 millisekund. Testen blir gjort slik som vist i figur 2.12. Vi gjør 5 tester for å sjekke hvor stort avvik vi kan ha. På kamera har vi ulike innstillinger vi kan ha på kamera som vist på figur 2.10.

Det første vi kan endre i innstillinger er Compressions. Her kan vi velge H265 eller H265X(H264). H265 gir bedre bildekvalitet men vi vil få større forsinkelse. H265X gir dårligere bildekvalitet, men mindre forsinkelse. Med tanke på at vi skal fjernstyre en undervannsrobot vil h265X høres ut som et naturlig valg. Men vi er usikre på hvor stor forsinkelsen er så dette må vi teste.

Det andre vi kan endre på er Resolution. Her har vi fått et krav fra Bildegjenkjenning og autonom kjøring gruppen at vi må bruke 1080P på grunn av bildegjenkjenning-oppgaven.



Figur 2.10: Kameratest

FPS er hvor mange ganger bildet oppdateres i løpet av ett sekund. Her ønsker vi mest mulig og bruker derfor det høyeste vi har muligheten til, nemlig 25.

Quality her går det fra 'worst' til best. Dette sier noe om kvaliteten på bildet. Vi vil i utgangspunktet satse på best så lenge ikke forsinkelsen blir for stor.

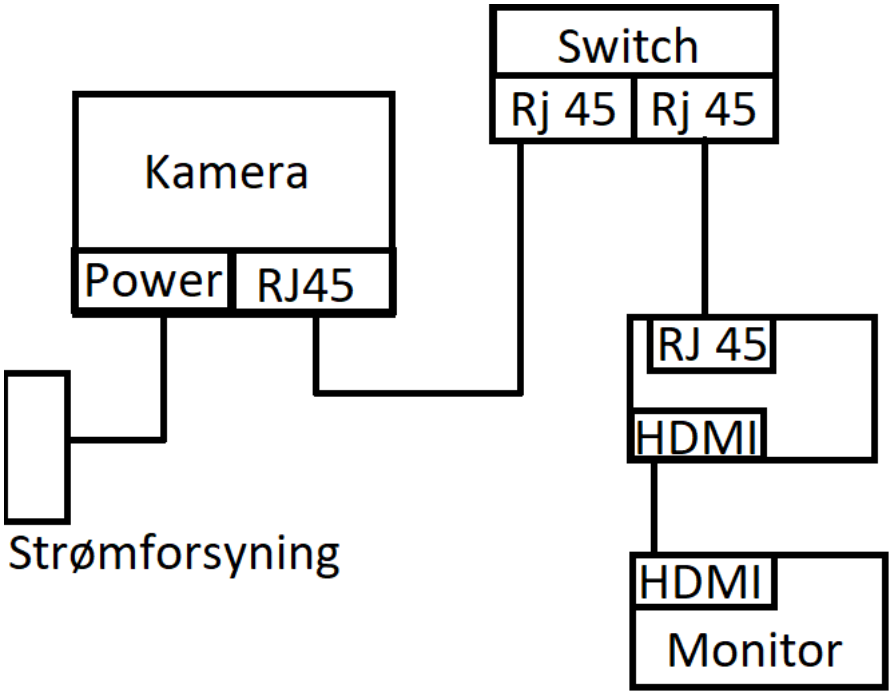
Baseline, mainline og highline. I teorien er det slik av hvis du velger highline så skal du i utgangspunktet få den samme kvaliteten, men med mindre forsinkelse. Det er ikke mange år siden det kun var baseline på markedet. Men det har kommet flere produsenter på markedet og dermed har main og highline kommet også.

Baseline er en enkel profil med lavt kompresjonsforhold. Det er kompatibel med flere opptakere.

Mainline er en mellomprofil med middels kompresjonsforhold, og har standard profilstilling. Denne profilen er kompatibel med de fleste opptakere.

Highline er en kompleks profil med høy kompresjonsforhold. Denne blir brukt for eksempel på HD-tv applikasjoner. [68]

Når vi skal teste kobler vi opp kamera og filmer direkte på monitoren hvor vi har en stoppeklokke slik at vi får målt tidsforsinkelsen. Vi har koblet det opp som vist på figur 2.11.



Figur 2.11: Blokkskjema test

Figur 2.12: Kamera test blokkskjema

Etter å ha prøvd alle ulike innstillingene innser vi at det er veldig liten forskjell på dem. Vi velger derfor å gå for innstillingen med minst forsinkelse.

Test av forsinkelser :

Test 1: H265 , 25 FPS , 1080P, Kvalitet : Best, Baseline	
Test 1	430ms
Test 2	320ms
Test 3	330ms
Test 4	330ms
Test 5	350ms
Gjennomsnitt	352ms

Test 2:H265 , 25 FPS , 1080P , High Profile, Kvalitet : best	
Test 1	270ms
Test 2	350ms
Test 3	270ms
Test 4	330ms
Test 5	310ms
Gjennomsnitt	306ms

Test 3:H265X , 25 FPS , 1080P , High Profile, Kvalitet : best	
Test 1	200ms
Test 2	230ms
Test 3	240ms
Test 4	200ms
Test 5	200ms
Gjennomsnitt	214ms

Test 4: H265X , 25 FPS , 1080P , Baseline Profile, Kvalitet : best	
Test 1	160ms
Test 2	130ms
Test 3	270ms
Test 4	200ms
Test 5	180ms
Gjennomsnitt	188ms

Test 5: H265X , 25 FPS , 1080P , Baseline Profile, Kvalitet : Verst	
Test 1	130ms
Test 2	160ms
Test 3	170ms
Test 4	170ms
Test 5	130ms
Gjennomsnitt	152ms

2.2.1 Oppsummering av test

Vi ser i testen at det er forskjell mellom de 5 forskjellige testene. Fra første til siste test er det i gjennomsnitt 200 ms forskjell på etterslepet. Med tanke på at vi skal fjernstyre en undervannsrobot er det viktig at vi finner en kombinasjon med best mulig kvalitet og minst mulig etterslep. Det er viktig med lavest mulig etterslep slik vi klarer å kjøre ROV-en uten problemer. Vi går for test 3 med tanke på etterslep og kvalitet. Vi hadde også en test der vi ikke hadde festet kamera og hadde de løst. De testene fikk vi veldig varierte resultater av og gikk heller for å montere kamera fast i et stativ. Vi har ut ifra testene konkludert med at kamera er godt nok for vårt behov og har bestemt oss for å bruke dette i ROV-en og Mikro ROV-en. Resten av jobben med kamera som bildebehandling og videostrøm fra brukergrensesnittet blir gjort av bildebehandlingsgruppa.

Kapittel 3

Kommunikasjon

Innhold

3.1 Kommunikasjonsbehov

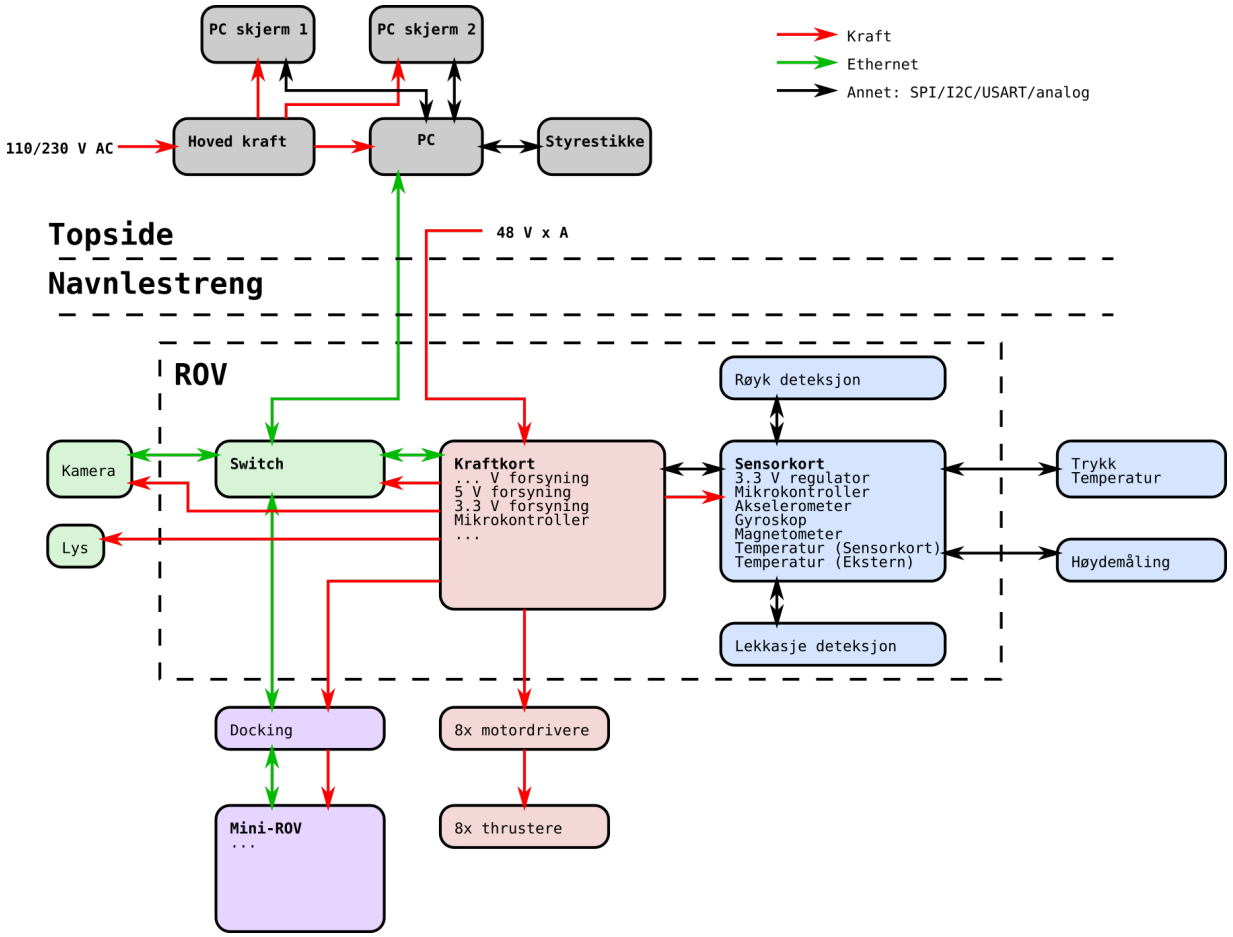
I ROV-en har vi behov for et kommunikasjonssystem som gjør det mulig at flere noder (Mikro ROV, kraftforsyningskort) skal kunne kommunisere med hverandre. Denne informasjonen skal så bli sendt opp til datamaskinen på topside slik vi kan lese av den dataen vi ønsker. Systemet må være i stand til å sende og motta data fra de forskjellige nodene på raskest mulig måte slik vi unngår mye forsinkelser under kjøring og styring av ROV-en. Vi må også sette opp kommunikasjon mellom kamera og datamaskinen og i tillegg styring til lys.

3.1.1 Alternativ

For kommunikasjonen mellom datamaskinen på topside og ROV-en har alle gruppene tatt en felles beslutning for å bruke en nettverksbasert løsning. Vi har da valget mellom å bruke standard Ethernet-kabel eller fiberoptisk kabel som vårt transmisjonsmedium. Vanlig Ethernet-kabel er en billigere og mindre plasskrevende løsning, men har ikke like rask overføring som fiber gir oss. Vi har derfor bestemt oss for å bruke en fiberoptisk kabel som transmisjonsmedium. Dette gir oss veldig god overføringshastighet og rask kommunikasjon.

I fiberkabler kan man få en enorm overføringshastighet. En fiberkabel har blitt testet til å klare å overføre data med en hastighet på hele 100 terabit per sekund. I en Ethernet-kabel kan vi få til en overføringshastighet på opptil 10 gigabit per sekund, som er en del mindre enn mulig i fiberkabler. Ethernet har blitt brukt i tidligere års ROV-er og har fungert bra, men vi ønsker å prøve og gjøre dette enda bedre ved bruk av fiberoptikk. [27]

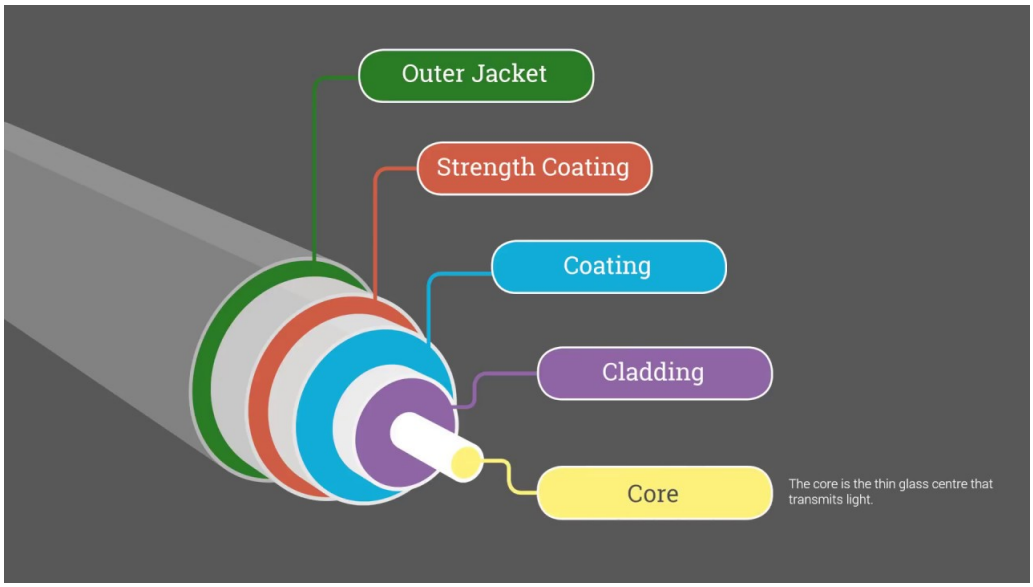
Vi har i år bedre plass inne i kuppelen på grunn av motorkontrollerne er plasserte på utsiden slik at det ikke blir problem å få plass til fiber. Inne i ROV-en skal vi sette opp en svitsj slik vi kan koble inn alle nodene i systemet sammen. Vi har da en fiberkabel fra datamaskinen på topside ned til svitsjen. Alle nodene i ROV-en blir så koblet til svitsjen via en Ethernet-kabel.
Blokkskjema over hvordan oppsettet er tenkt er vist i figur 3.1.



Figur 3.1: Blokkskjema over ROV

3.1.2 Fiberoptisk kommunikasjon

En fiberkabel er bygget opp av flere forskjellige materialer som er delt inn i flere lag. [7] Selve kjernen i fiberkabelen er laget av veldig rent glass, det blir så lagt en kappe rundt kjernen for å beskytte denne. Kappen er og ofte laget av rent glass, men finnes også i plast. Etter kappen blir det så lagt til enda et lag rundt hver fiberleder som kalles primærbelegg. Primærbelegget skal beskytte kabelen mot støt slik at glasset ikke blir ødelagt. Det siste laget i en fiberkabel kalles sekundærbelegg og avhenger av hvor vi skal bruke kabelen. Til fiberkabler som skal brukes innendørs blir dette som oftest realisert med et plastlag. Til utendørs bruk blir fiberkablene delt inn i et eller flere små rør som er fylte med fett. Kabelen blir så isolert med et lag med kevlargarn. Figur 3.2 viser et eksempel på hvordan en fiberkabel er bygget opp.



Figur 3.2: Oppbygning til en fiberkabel [10]

I fiberoptisk kommunikasjon blir et lys sendt gjennom kjernen i fiberkabelen via en laser eller en lysdiode. Siden fiberkommunikasjon bruker lys, trenger vi ikke å tenke på støy fra nærliggende kabler slik som i koaksial, CatX eller andre kobberkabler. Dette gjør at fiber har en mye raskere overføringshastighet og fungerer bedre på lengre distanser enn Ethernet-kabler. Ulempene med fiber er at det er en veldig dyr løsning i forhold til vanlig Ethernet. Pris per meter koster vesentlig mer, og vi må i tillegg tenke annerledes når vi skal kjøpe svitjsen. Hadde vi hatt Ethernet, kunne vi hatt en vanlig svitsj med kun RJ45 porter. Når vi skal bruke fiberkabel må vi ha en SFP (Small Form-factor Pluggable transceiver) inngang på svitsjen. Mer om SFP vises i kapittel 3.1.3.1. Vi må også passe på å ikke få for stor bøy på selve kabelen. For store bøyninger på kabelen fører til at fiberen knekker eller at lyset ikke blir reflektert inn mot kjernen og kan forsvinne ut i kappen rundt fiberen. [7]

3.1.3 Forskjellige fibertyper

Det er to hovedtyper fiber. Single-modus og multi-modus fiber. Single-modus fiber har en kjerne-diameter på mindre enn $9\mu m$ og brukes til å sende en lys kun en vei gjennom hele kabelen. En multi-modus fiber har en kjerne med diameter på $50\mu m$ eller $62.5\mu m$ dette gjør at vi kan sende signalet på flere måter. I single-modus fiber må man ha flere fiberledere i kabelen for å kommunisere begge veier samtidig, dette slipper vi ved bruk av multi-modus fiber. Her er det plass til å både sende og motta samtidig. Single-modus har mindre demping enn multi-modus, dette gjør at vi kan bruke single-modus over veldig lange lengder uten problem. Single-modus kan brukes til lengder på flere kilometer, men multi-modus brukes når lengdene er mindre enn 4-500 meter. [8]

3.1.3.1 Multi-mode kabel

Vi har bestemt oss for å bruke multi-modus fiber i ROV. Dette var den billigste løsningen og vi har ikke behov for veldig lange kabellengder. Vi har vært i kontakt med Nexans som har sendt oss en kabel som er laget for bruk i ROV. Denne kabelen inneholder 6 Single-mode fiber og 6 Multi-mode fiber i tillegg inneholder den og 2x6mm^2 strømledninger. Denne kabelen er utstyrt med armering og god isolasjon som gjør at den er veldig robust og tåler å ligge under vann i lengre tider. Noe som kan bli problem er vekten av kabelen som også går med i totalvekten i ROV-en som vi må prøve å holde minst mulig med tanke på MATE-konkurransen. I databladet for kabelen som vi fikk er det oppgitt at kabelen veier 805kg per kilometer med kabel, som tilsvarer rundt 20kg for en kabel på 25m. Maks vekt for ROV-en med kabel må være under 35kg for å få lov til å delta, og under 28kg for å få poeng. Dette kan bli et problem med kabelen vi fikk fra Nexans. En mulighet for å løse dette er å fjerne det ytterste laget med isolasjon og stålarmeringen rundt kabelen slik den blir lettere. Kabelen vil fortsatt fungere bra under vann, men ikke tåle like mye bøyinger og er ikke like robust. Et annet problem hvis vi skal bruke denne kabelen er å få den tilkoblet til ROV-en. Dette kan enten gjøres ved en hybrid plugg som har inngang til fiber og strøm eller en koblingsboks på utsiden av ROV-en der kablene blir tilkoblet og skjøtet inn i ROV-en. Pluggene vi har funnet som gjør det mulig med hybrid kabel er for dyre i forhold til budsjettet og med en koblingsboks får vi ikke muligheten til å ta av kabelen like lett som med en plugg.

På grunn av dette har vi blitt nødt til å se på alternative løsninger. Den andre løsningen vi har vurdert er å bestille en vanlig duplex multimode fiberkabel som vi fester i lag med strømkablene. Dette må så tettes slik at det kan bli brukt under vann. Vi må da ha en kontakt for strøm og en egen for fiber. Med denne løsningen kan vi få en mye lettere navlestreng og plugger med kun fiber er en del billigere enn hybride kontakter. For å koble fiberen til baksiden av ROV-en har vi funnet en kontakt på nettsidene til Farnell. Disse kontaktene blir produsert av Bulgin og er laget for tilkobling av duplex multimode fiberkabel. Ifølge datablad har kontaktene en IP grad på 68 når de ikke er tilkoblet og en IP grad på 69K når de er tilkoblet. IP68 betyr at kontaktene er helt støvtette og tåler å ligge i vann over lengre tid, IP69K er en tysk standard som sier oss at kontaktene tåler høyt trykk, høye temperaturer og sterke vannstrømninger. I figur 3.3 ser vi hvordan pluggene ser ut.

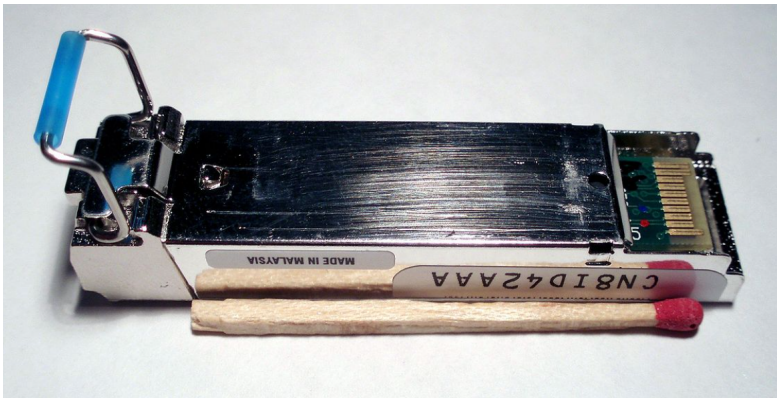
Vi har valgt å gå for den alternative løsningen etter som det ikke var mulig å finne hybridkontakter som var innenfor budsjettet og navlestrengen blir en del tyngre enn planlagt ved bruk av kabelen fra Nexans.

(a) Bulgin PXF6052A

(b) Bulgin PXF6050A

Figur 3.3: Kontakter for fibertilkobling [20]

Fra denne kontakten legger vi en patchekabel i fiber inn til svitsjen. I svitsjen vi skal bruke er det lagt til rette for en SFP(Small form-factor pluggable transceiver) port som vi kobler fiberkabelen til. En SFP er en liten plugg som blir brukt i fiber og telekommunikasjon. I svitsjen vi skal bruke er det lagt til rette for en SFP port. Inne i SFP-en er det et grensesnitt som gjør at vi kan gjøre om Ethernet-signalet til et signal som fiberen kan transportere opp til datamaskinen. Figur 3.4 viser hvordan en SFP ser ut, her i forhold til en fyrstikk.



Figur 3.4: Eksempel på en SFP-plugg [29]

3.1.4 Fiber kabel G2-50/OM2 BB AXAI-I/0

Vi skal bruke G2-50/OM2 BB AXAI-I/0 som vist på figur 3.5.

Figur 3.5: Fiberkabel

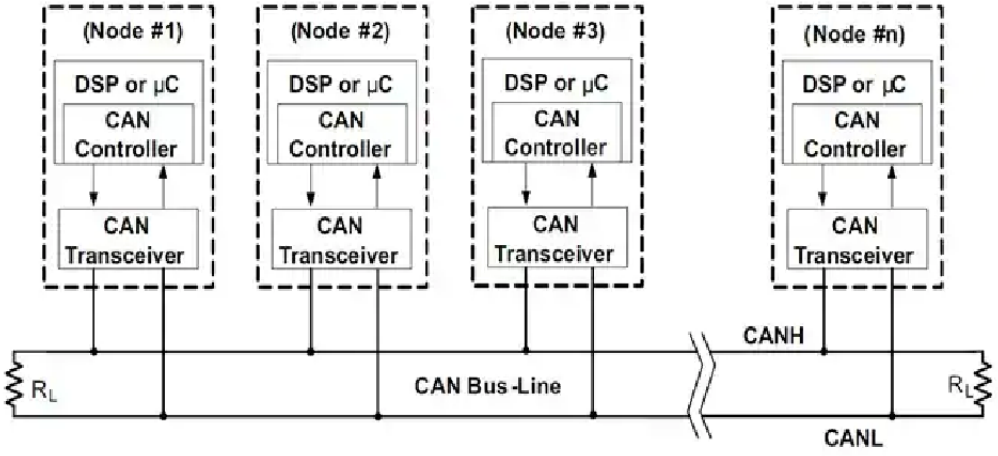
Vi valgte denne fiberkabelen for dette var en billig kabel og en multimodekabel. Kabelen er også svært lett, den veier 0.4 kg på 20 meter. Kabelen kostet oss 5.51 kr per meter. Vi bestilte 30 meter slik at vi hadde en god sikkerhetsmargin. Det som også var viktig med denne kabelen var at den passet til pluggen vi har kjøpt. Kabelen har et tverrsnitt på 50/125 som er det tverrsnittet pluggene er kompatible med. Kabelen er og vanntett, men skal for sikkerhetsskyld utstyres med en ekstra strømpe over for beskyttelse.

3.2 Kommunikasjon internt i ROV

For å kunne kommunisere med mikrokontrollerne over Ethernet må vi sette opp en kommunikasjon slik at mikrokontrollerne og datamaskinen topside kan forstå hverandre. Vi har sett på flere metoder for å løse dette. Vi har blant annet sett på CAN-bus, I^2C/Ethernet og en SPI/Ethernet-modul. Noen mikrokontrollere har også mulighet til å koble seg direkte til Ethernet. Dette hadde vi mulighet til på mikrokontrolleren til hoved ROV-en, men det var ikke mulig å få en liten nok mikrokontroller med Ethernet i mikro ROV-en.

3.2.1 CAN-buss

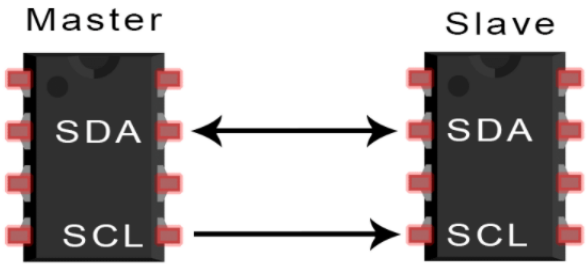
Controller Area Network eller CAN-bus er en protokoll som gjør at mikrokontrollere og andre enheter i et system kan kommunisere med hverandre uten bruk av en sentral datamaskin. CAN-protokollen er utviklet for og blir brukt i biler og andre kjøretøy, men er også veldig anvendelig i andre kommunikasjonssystem. Det er to forskjellige typer CAN, High-speed CAN og low-speed CAN. High-speed CAN er det som oftest blir brukt i kjøretøy og i industrien. High-speed har en hastighet på 1Mbit/s. I High-speed CAN settes det inn en 120Ω motstand i hver ende. Dette blir gjort fordi CAN-bussen må avsluttes. Motstandene brukes for å undertrykke refleksjoner i CAN-bussen og for å tilbakestille bussen til sin inaktive tilstand. Low-speed CAN har en hastighet på rundt 125Kbit/s og blir brukt til å koble sammen flere grupper med enheter. De forskjellige enhetene i et slikt system blir kalt for noder. Nodene er koblet sammen med to ledninger fra en tvunnet parkabel, som sender signalene CAN-High og CAN-Low. Alle nodene må ha en CPU(Central Processing Unit) som bestemmer hvilke signal som skal sendes og hva signalet den mottar betyr. En CAN-Kontroller som lagrer bitene fra CAN-bussen til det er klart til og leses av prosessoren og en CAN-transceiver som konverterer datastrømmen fra CAN-bussen til noe som er forståelig for CAN-kontrolleren. Bildet under viser et eksempel over hvordan en CAN-buss er koblet. Her er alle nodene tilkoblet med CANH(CAN-High) og CANL(CAN-LOW) og to 120Ω motstander i hver ende. Figur 3.5 viser et eksempel over hvordan CAN-bussen blir koblet opp.[9]



Figur 3.6: CAN-Buss eksempel [16]

3.2.2 Inter Integrated Circuit

I^{2}C eller Inter Integrated Circuit er en protokoll for overføring av seriell data. [53] I^{2}C kan fungere med flere mastere som kontrollerer en eller flere slaver eller flere slaver som er koblet til en master slik på samme måte som SPI grensesnittet. I I^{2}C blir det kun tatt i bruk to ledninger til kommunikasjon. SDA og SCL. SDA(Serial Data) brukes til kommunikasjon mellom master og slave. Her både sender vi og mottar over data over en ledning. SCL(Serial Clock) er klokkesignalet vi bruker, dette klokkesignalet blir alltid styrt av masteren i systemet. Figur 3.7 viser en kobling mellom en master og en slave i en I^{2}C buss. Normalt vil farten i en slik buss ligge på 100Kbps,



Figur 3.7: I^{2}C kobling [53]

men finnes også med høyere overføringshastighet opptil 5Mbps. I I^{2}C blir meldingene delt opp i frameseller rammer som inneholder adressen til slaven vi skal kommunisere med og dataen vi skal overføre. [53] Vi har mulighet for å koble dette signalet til en modul som omformer signalet til en Ethernet-pakke. Dette gjør at vi kan koble mikrokontrolleren til Ethernet og sende pakker via denne modulen. Det finnes også slike moduler som bruker en SPI-buss. Vi skal gå mer inn på SPI i kapittelet under.

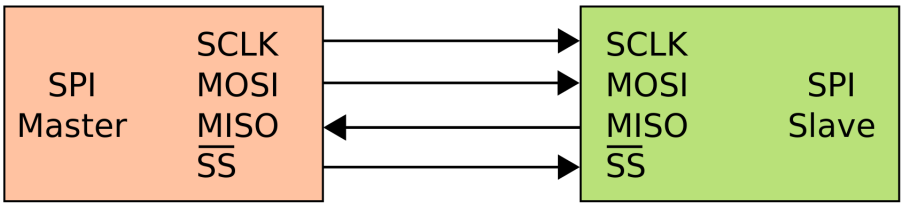
3.2.3 SPI

SPI(Serial Peripheral Interface) er en databuss som ble utviklet av Motorola på 1980-tallet. SPI blir ofte brukt til kommunikasjon i innebygde datamaskiner og systemer. SPI bruker en master/slave oppbygning. Dette betyr at vi har en hoveddatamaskin eller enhet som kontrollerer de andre enhetene i systemet(slavene). Ved bruk av SPI-bussen kan man oppnå en overføringshastighet på opptil 60Mbps. [23] Slavene er koblet til Masteren med fire signal:

- SCLK er klokkesignalet som blir styrt av masteren i systemet
- MOSI(Master Out Slave In) er signalet som blir sendt fra masteren til første slave i systemet
- MISO(Master In Slave Out) er signalet som blir sendt fra slaven og mottas av masteren

- SS(Slave Select) eller CS(Chip Select) er ofte et aktivt lavt signal som masteren bruker for å velge hvilken slave som skal motta eller sende signal.

I figur 3.8 ser vi hvordan en master og en slave blir koblet sammen.



Figur 3.8: Master og slave tilkobling [12]

Når kommunikasjonen starter velger masteren et klokkesignal med en frekvens som slavene kan håndtere. Masteren velger så slaven med SS signalet, etter det sender den et bit via MOSI signalet og slaven sender et bit på MISO signalet. Dette gjentas hver klokkesyklus. [12]

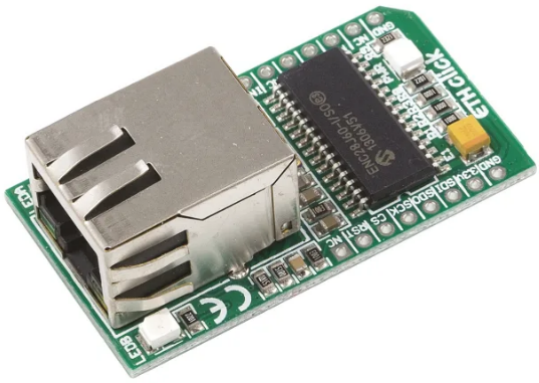
Vi har nå sett på noen alternativ til kommunikasjon. Vi har videre valgt å prøve å realisere dette ved bruk av en SPI til Ethernet omformer. Dette var en veldig billig løsning og passet bra til vårt behov. CAN-bus har blitt brukt i prosjektet tidligere år og vi vil da prøve en ny løsning. I^2C har ikke blitt brukt tidligere, men vi har valgt SPI ettersom dette har en raskere overføringshastighet og det var veldig mange forskjellige moduler og eksempler på bruk av en slik løsning.

3.2.4 SPI til Ethernet

For at mikrokontrolleren skal kunne kommunisere over Ethernet, må vi ha en måte å omforme SPI signalene fra mikrokontrolleren til et Ethernet-signal. Dette kan gjøres ved å bruke en SPI/Ethernet- modul som vi kan koble direkte til mikrokontrolleren. Vi har vurdert to alternativ av slike moduler. Figur 3.9 viser et bilde av kretskortet Mikroe-971 hvor ENC28J60 blir brukt.

3.2.4.1 MIKROE-971

Det første alternativet er MIKROE-971 fra RS components. Denne modulen er bygget opp av en ENC28J60 chip og en RJ45 Ethernet kontakt. ENC28J60 Brikka og RJ45 kontakten er plassert på et kretskort fra mikroBus som gjør at vi enkelt kan koble denne til pinnene på mikrokontrollerne. ENC28J60 er bygget opp av blokkersom har forskjellige funksjoner.



Figur 3.9: ENC28J60 [25]

- Vi har et SPI-grensesnitt som gjør at vi kan kommunisere med mikrokontrolleren og modulen vår. SPI-en kan bruke et klokkesignal på opptil 20MHz
- En SRAM buffer for datapakkene som blir sendt og mottatt. SRAM-en er bare på 8Kbyte, noe som kan være litt lite i vårt tilfelle
- En MAC(Medium Access Control) modul som er en adresse som hver enhet som kan kobles til internett har. Dette kan være enheter som skrivere, svitsjer, datamaskiner og lignende. Alle enhetene har hver sin unike adresse. MAC-adressen er et ord som inneholder 48bit og skrives ofte i heksadesimalt format. "08:21:58:61:ff:eaer et eksempel på hvordan en MAC-adresse skrives.
- Vi har også noe som kalles det fysiske laget. Her blir det definert hvordan man skal overføre informasjon og hvilken frekvens man skal ha på overføringen. Det fysiske laget har også ansvar for oppkoblingen til de forskjellige enhetene vi skal ha forbindelse med.

Ut ifra spesifikasjonene og databladet til MIKROE-971 ser vi at denne modulen fungerer bra til å koble mikrokontrolleren til Ethernet. Noe som kan bli et problem med denne er SRAM- bufferet på 8Kbyte og tidligere har det vært noen problemer med å få modulen til å fungere slik den skal. [24]

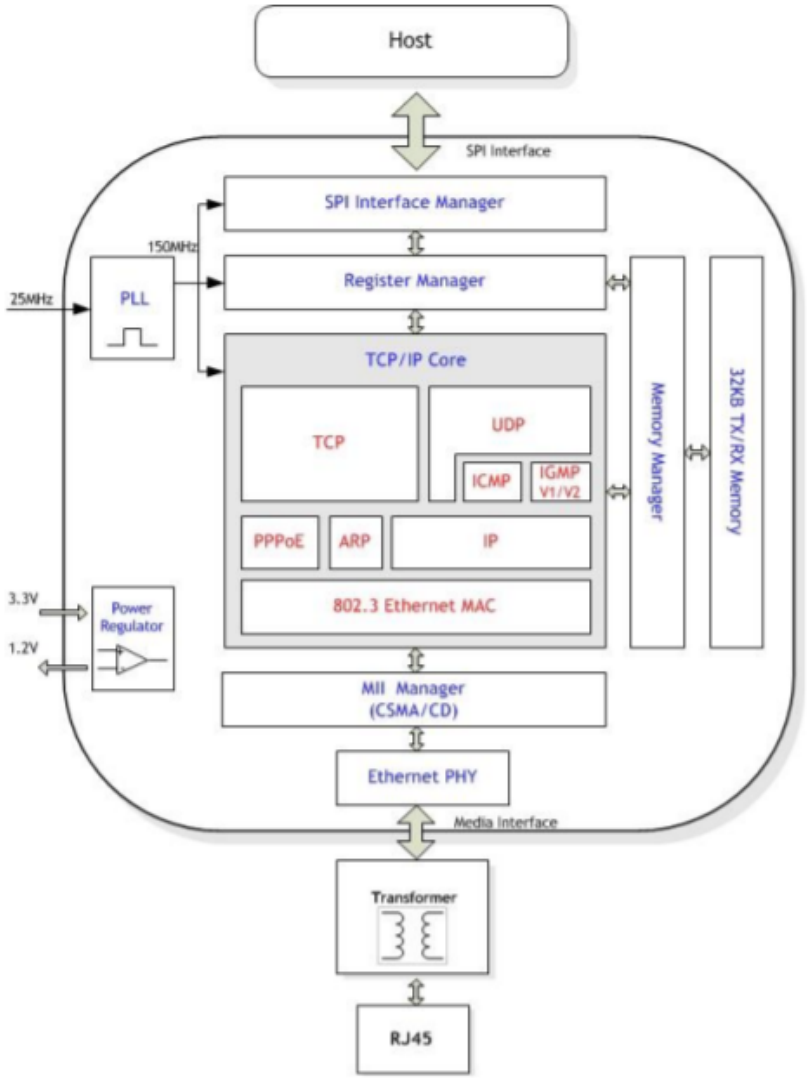
3.2.4.2 MIKROE-1718

Den andre modulen vi har sett på er også koblet til et kretskort fra mikroBus, men her brukes det en W5500 chip fra WIZnet. W5500 har en litt annen oppbygning enn ENC28J60, men begge kan brukes til Ethernet-tilkobling fra mikrokontrolleren. W5500 har en innebygd buffer på 32Kbyte som er 4 ganger større enn bufferet i ENC28J60. W5500 støtter i teorien SPI hastigheter på opptil 80MHZ, men har kun blitt testet med hastighet opptil 33.3MHz. Modulen fra Wiznet har også

en del mer dokumentasjon og datablad på nettet som gjør programmeringen enklere for oss. På grunn av dette har vi bestemt oss for å bruke MIKROE-1718 med W5500 i ROV-en. Brikka har en innebygget TCP/IP kjerne som har samme virkemåte som forklart i 2.1.1.1.

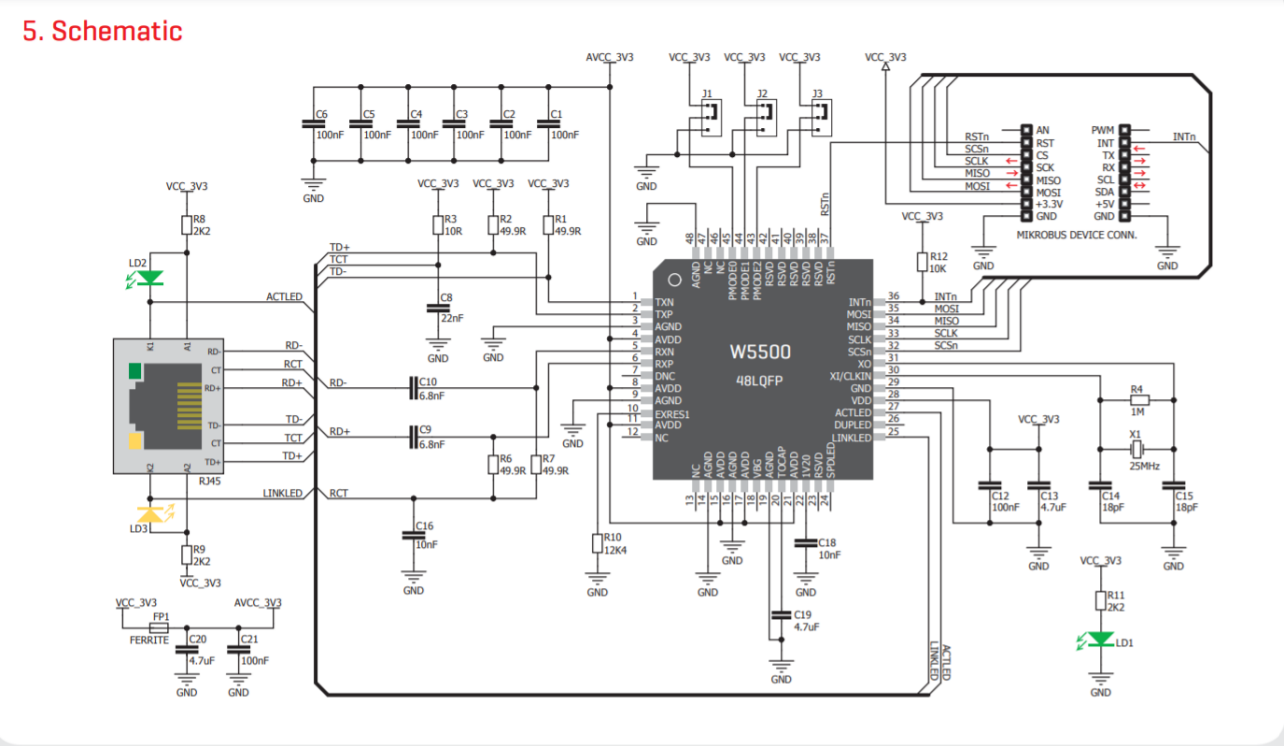
3.2.5 W5500 - virkemåte

W5500 er bygget opp av flere forskjellige blokker som alle har sine egne funksjoner og oppgaver. I figur 3.10 ser vi oppbygningen til W5500. Brikka har en innebygd TCP/IP kjerne som gjør at det eneste vi trenger å programmere er SPI grensesnittet og det fysiske laget til Ethernet. Her kan man velge mellom flere forskjellige typer internettprotokoller for overføring og hvordan man skal kommunisere med datamaskinen.



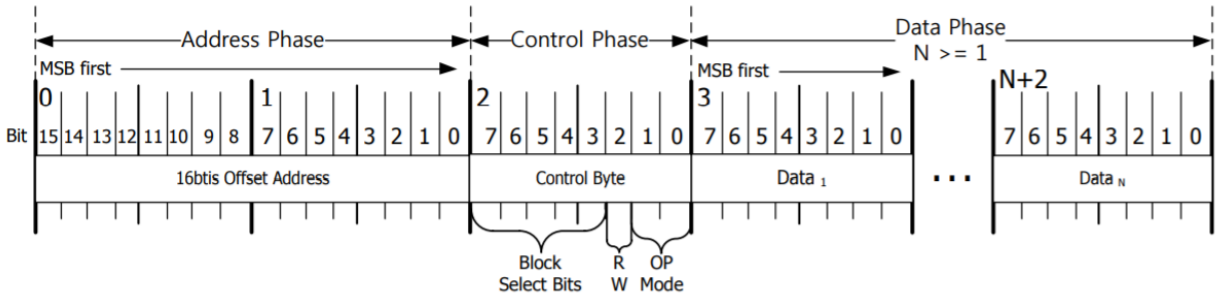
Figur 3.10: Blokkskjema av W5500 [26]

Figur 3.11 viser hvordan Brikka blir koblet til RJ45-pluggen og kretskortet W5500 har 4 pinner som blir brukt til SPI tilkobling, pinnene blir så koblet til mikroBus-kortet. Dette blir så koblet til masteren i systemet, som her er mikrokontrolleren. Her kobler vi til SPI-signalene MOSI, MISO, CS(SCSn) og SCLK fra modulen vår til SPI inngangen på mikrokontrolleren. I tillegg til dette trenger vi en ledning til spenning og en ledning til jord. Når vi velger SPI grensesnittet kan vi velge mellom VDM(Variable Data mode) eller FDM(Fixed Data Mode). I FDM blir CS koblet til jording, dette gjør at vi kun kan bruke SPI-grensesnittet til W5500. Dette gjør at vi kun kan ha en slave på SPI-bussen. Vi har valgt å koble CS til mikrokontrolleren slik vi har mulighet til å koble flere slaver til SPI-bussen om dette trengs.



Figur 3.11: Koblingsskjema av mikrobus og W5500 [26]

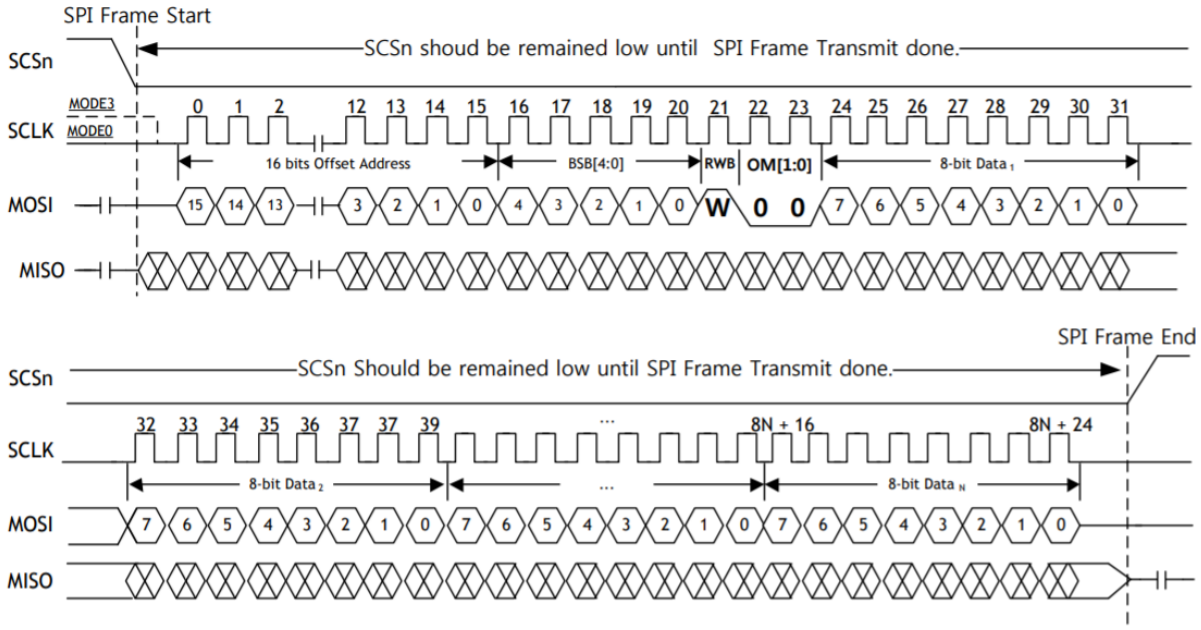
Hver SPI-melding blir sendt i 3 faser. Adressefasen, kontrollfasen og datafasen. I adressefasen blir det spesifisert en 16 bits adresse for register til W5500 eller for Tx/Rx-minne. Tx(Transmit) minne brukes for data som skal sendes, Rx(Recieve) brukes for data som blir mottatt. Her blir det altså gitt adresse til registeret til W5500 eller til et minne for sending og mottak av data. Kontrollfasen spesifiserer blokken som den adressen vi angir i første fase tilhører. Her spesifiserer vi også om vi skal lese eller skrive til SPI-bussen. Det siste vi gjør i kontrollfasen er å definere om vi bruker VDM eller FDM. I siste fase av SPI-meldingen blir det spesifisert hvor mange biter med data som skal sendes.



Figur 3.12: Eksempel på et oppsett av en SPI-melding [26]

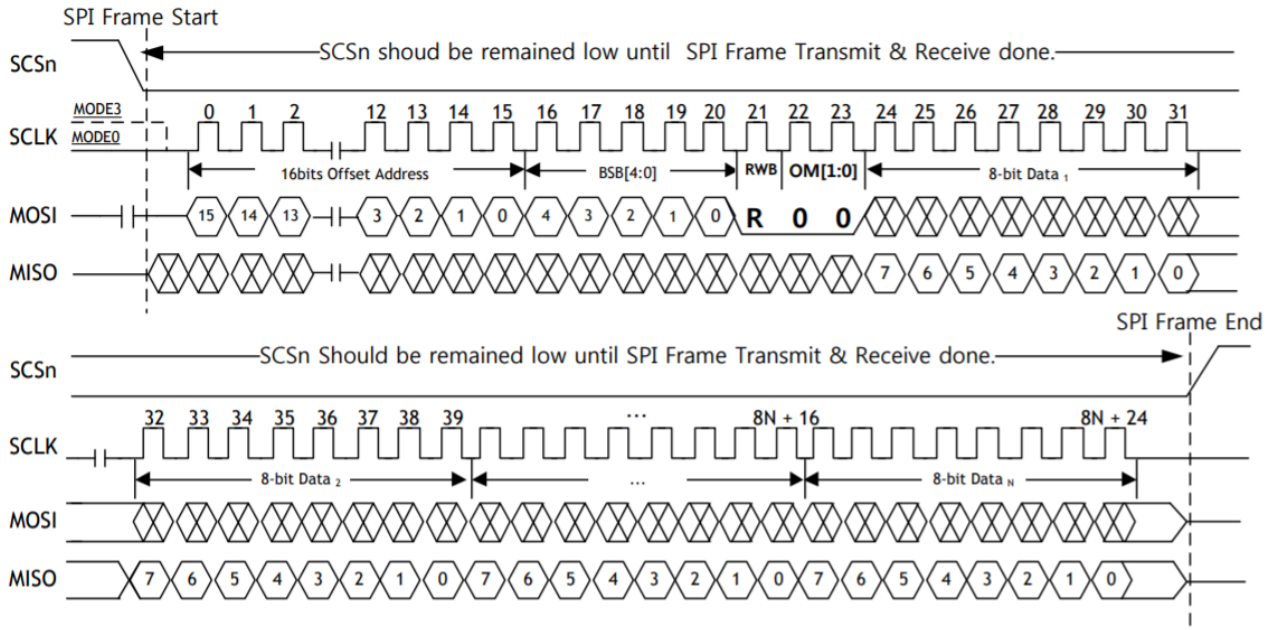
I figur 3.12 ser vi et eksempel på hvordan dataen i SPI-bussen blir satt opp. Her blir det først angitt en adresse, etter adressen blir satt går vi inn i kontrollfasen hvor vi velger hvilken blokk adressen tilhører og om vi skal lese eller skrive til SPI-bussen. Siste del av meldingen er datafasen hvor selve dataen vi skal sende blir plassert.

I figur 3.13 under ser vi hvordan alle linjene mellom W5500 og mikrokontrolleren kommuniserer. Her vises et eksempel på når SPI-en er i skrivemodus. Her starter det med at mikrokontrolleren gjør CS(Chip Select) lav eller SCSn som de blir kalt på figuren. Denne forblir lav til prosessen er fullført. Vi ser at når vi er i skrivemodus er de første 16 bitene en adresse. De neste 4 bitene er blokken den tilhører etterfulgt av bitene som blir satt etter hvilken modus vi har. Her er RWB(Read/Write bit) satt til W som er definert som 1 og betyr at vi er i skrivemodus. Resten av prosessen går til sending av dataen vi skal sende. Denne dataen blir så sendt fra MOSI(Master Out Slave In)-linjen.



Figur 3.13: Skriving til SPI-bussen eksempel [26]

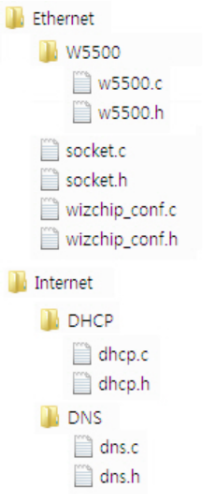
Lesing av SPI-en skjer på nesten samme måte som ved skriving. I figur 3.14 ser vi et eksempel av lesing fra SPI-en. Her Ser vi at starten er ganske lik, men her er RWB(Read/Write bit) lik R som er definert som null. Dataen som skal leses blir og sendt over MISO(Master In Slave Out)-linjen.



Figur 3.14: Lesing fra SPI-bussen eksempel [26]

3.2.5.1 W5500 driver

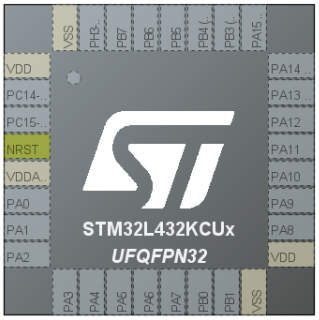
For W5500 har det blitt laget en driver av Wiznet som gjør at vi enkelt kan bruke modulen på flere forskjellige mikrokontrollere uten og kode så ekstremt mye. All konfigurasjon og initialisering av funksjoner blir gjort i denne driveren. Det vi må fikse videre er det fysiske laget. Altså hvilke oppgaver programmet skal utføre. Denne driveren laster vi ned via Wiznet sin github side. Etter å ha lastet ned denne driveren importerer vi denne inn i prosjektet med mikrokontroller-koden. Driveren kalles for ioLibrary som betyr Internet offload Library, dette biblioteket er bygget opp av mange filer og programmer som gjør at vi kan bruke W5500 til en rekke forskjellige internettprotokoller. IoLibrabry inneholder filer som gjør at vi kan bruke internettprotokoller som SNTP, DNS, TFTP, FTP, SNMP, DHCP, HTTP og MQtt. Mer om disse internettprotokollene blir forklart i kapittel 5.1.1. Her finner vi også filer for konfigurering av W5500 Brikka og en fil som konfigurerer forskjellige "sockets" på W5500. Figur 3.15 viser strukturen til filene som driveren ioLibrary inneholder. Disse filene må inkluderes i programmet for mikrokontrolleren. I denne driveren er det også en fil som forteller oss hvilke funksjoner som finnes i biblioteket og hvordan de skal bli brukt. I neste kapittel skal vi lage et enkelt program for å teste kommunikasjonen mellom PC-en og mikrokontrolleren over Ethernet.



Figur 3.15: IoLibrary oppbygning [30]

3.2.6 W5500 - Nucleo-L432KC test

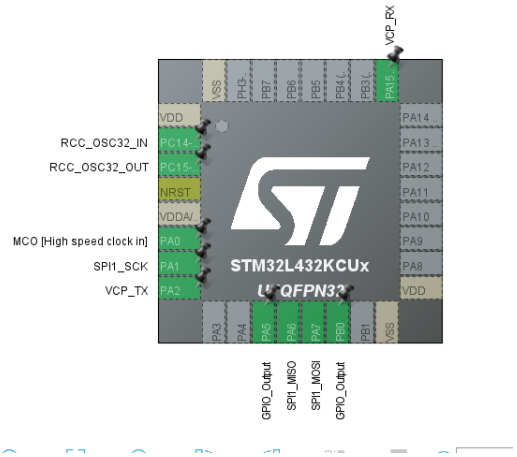
For å kunne lage en kommunikasjon mellom W5500 og mikrokontrolleren til mikro ROV-en bruker vi STM32CUBE for å lage programmet. Ved å bruke STM32CUBE får vi opp selve mikrokontroller-brikka slik vi enkelt kan velge hvilke pinner vi skal bruke og spesifisere pinnene til hvilke signal vi skal bruke. Når vi først starter STM32Cube velger vi hvilken mikrokontroller vi skal programmere på, som her er Nucleo-L432KC. På figur 3.16 ser vi hvordan pinnene på mikrokontrolleren er. Her kan vi nå enkelt velge hvilke pinner vi skal bruke. Vi aktiverer nå SPI-modulen til mikrokontrolleren



Figur 3.16: STM32L432KC

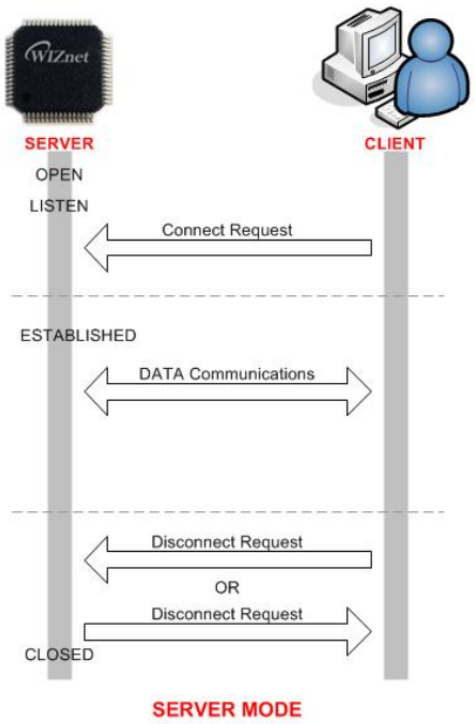
og får da opp hvilke pinner som SPI-en skal kobles til. Når vi aktiverer denne får vi opp MISO og MOSI-signalet, men vi trenger også CS(chip select) og SCLK til SPI bussen. Til CS har vi valgt pinne PB0 og SCLK blir koblet til PA1. PB0 blir bare brukt som en vanlig GPIO(General purpose input/output). Vi har også brukt pinne PA5 som en GPIO output. Denne pinnen blir brukt som

en reset som gjør at vi kan tilbakestille W5500. Pinnene PA2 og PA15 blir brukt til kommunikasjon med USARTen. USART er den kommunikasjonsmetoden som gjør at vi kan lese og sende meldinger til en virtuell port på PC-en. Denne blir i vårt tilfelle brukt til å sjekke om alt går etter planen. Pinnene PC14, PC15 og PA0 er klokkene til mikrokontrolleren. PA0 er høyhastighetsklokken til mikrokontrolleren og PC14 og PC15 blir brukt til lavhastighetsklokken. Figur 3.17 viser hvordan pinnene blir koblet opp.



Figur 3.17: Pinner som blir brukt på Nucleo-L432KC

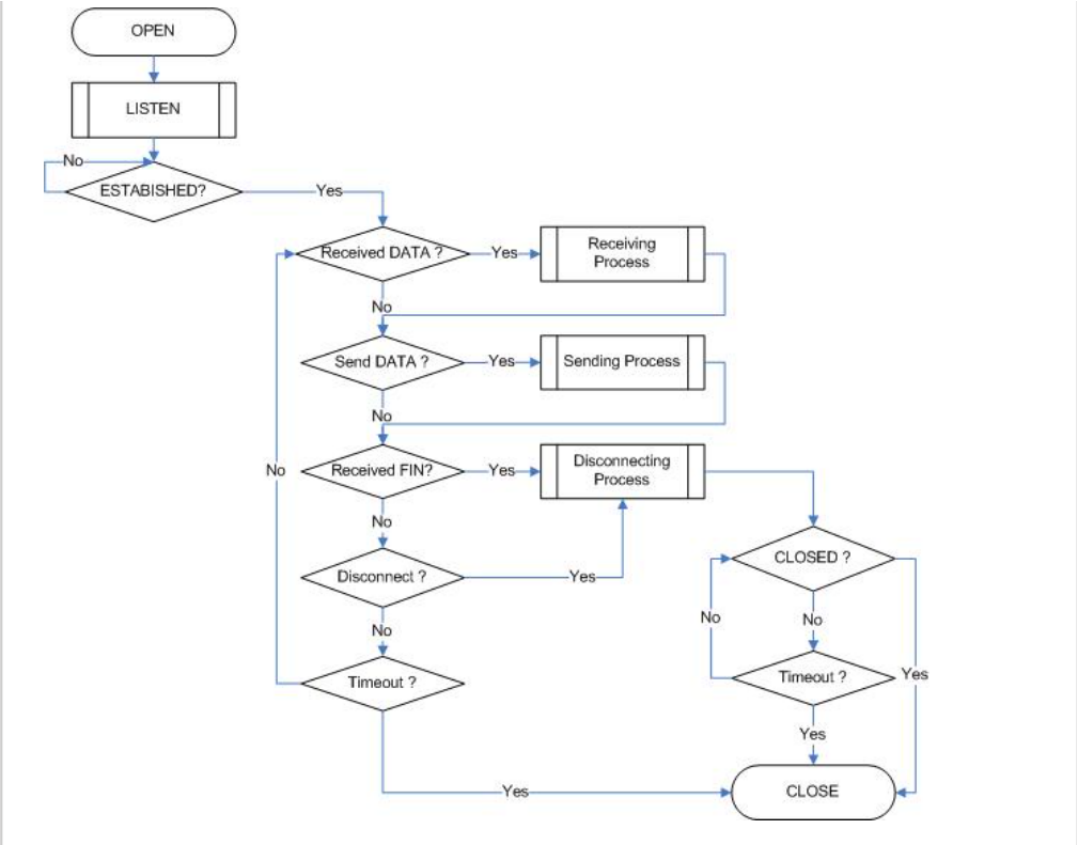
Når vi nå har valgt alle pinnene vi trenger kan vi begynne å bygge koden. Dette er kun et test-program som vi kan bruke til å teste kommunikasjonen. Programmet i ROV-en vil være litt mer komplisert og bruke flere pinner enn det vi viser nå. Når vi nå generer koden i STM32CUBE får vi opp en kode som har konfigurert alle pinnene for oss. Vi må i denne filen legge til funksjonene som beskriver virkemåten til programmet og inkludere alle filene vi trenger.



Figur 3.18: TCP-server virkemåte [11]

I testprogrammet vårt skal vi lage et program for sending og mottak av TCP-pakker. Her skal vi bruke W5500-brikka som en TCP-server. I figur 3.18 ser man hvordan kommunikasjonen mellom en tjener og en klient fungerer over TCP. Her åpnes tjeneren og lytter etter om det kommer noen tilkoblinger. Når klienten svarer, blir det etablert en tilkobling mellom tjener og klient. Når tilkoblingen er etablert kan informasjon sendes og mottas fra begge parter. Etter sendingen er fullført kobles klienten og tjeneren fra hverandre og tjeneren lukkes. Figur 3.19 viser et flytskjema av hvordan TCP- kommunikasjonen foregår.

Her ser man alle stegene som tas i en TCP-prosess. Etter tilkobling er etablert går programmet videre og sjekker om data skal mottas. Hvis svaret er "Ja" starter programmet og ta imot dataen før prosessen går videre. Hvis svaret er "Nei" går prosessen videre og sjekker om det er data som skal sendes. Hvis data skal sendes venter programmet til dette er fullført før det går videre. Når sending eller mottak av data er fullført kan vi koble oss fra klienten.



Figur 3.19: Flytskjema over TCP kommunikasjon [11]

Vi har i testprogrammet fulgt en bruksanvisning for tilkobling av W5500. [36] Etter vi har fått koden for initialisering av pinnene må vi tenke på hvilke funksjoner vi skal bruke i programmet. For å få driveren til å virke med resten av programmet må vi velge hvilke filer vi skal inkludere. Dette gjøres i main.c som er hovedfilen for programmeringen. Her importerer vi filen socket.h"og "wizchip_conf.h". Dette gjøres ved å bruke koden.

```
#include "wizchip_conf.h";
#include "socket.h";
#include <string.h>;
#include <stdio.h>;
#include <stm32l4xx_hal.h>
```

Vi har også brukt noen andre bibliotek som er innebygd i C. Bibliotekene vi har brukt er <string.h> som er et bibliotek som gjør at vi kan bruke mange funksjoner for sending og kopiering av biter og andre karakterer. <stdio.h> som betyr Standard Input Output. Dette biblioteket gjør at vi kan skrive og lese til flere forskjellige enheter. [31]

Vi har også tatt i bruk STM32 sitt HAL(Hardware Abstraction Layer) bibliotek. Dette er et bibliotek som gjør det enklere for oss å velge hvilke perifermoduler vi skal bruke og hvilke parameter og konfigurasjoner som vi skal bruke. Det er også noen ulemper ved bruk av HAL-biblioteket. Debugging eller feilsøking blir vanskeligere ved bruk av HAL, men vi har valgt å bruke dette siden vi får en enklere jobb med å lage programmet.

Videre i programmet har vi definert noen meldinger som gjør at vi får utskrift på den virtuelle porten til mikrokontrolleren. Her brukte vi et program som heter Terminal v1.9b for å koble oss til denne porten. Her må vi definere hvilken baudrate og hvor mange bit vi skal bruke. Baudrate er informasjonshastigheten ved signaloverføring. Baud betyr det samme som et symbol per sekund. Vi bruker 115200 som baud rate som betyr at vi sender 115200 symboler per sekund. I vårt tilfelle er det ikke så viktig med hvilken baud-rate vi velger, men begge enhetene må ha samme verdi. Når vi har satt opp terminalprogrammet til å kunne lese av verdier fra mikrokontrolleren må vi i koden vår legge til de meldingene vi vil få opp på terminalen.

Under private define har vi definert noen meldinger som vi ønsker å få opp i terminalen. Meldingene vi har definert er vist under.

```
char msg[60];
#define SEPARATOR
    "================================================\r\n"
#define WELCOME_MSG             "STM32Nucleo Ethernet configuration\r\n"
#define NETWORK_MSG             "Network configuration:\r\n"
#define IP_MSG                  "IP ADDRESS:  %d.%d.%d.%d\r\n"
#define NETMASK_MSG             "NETMASK:     %d.%d.%d.%d\r\n"
#define GW_MSG                  "GATEWAY:     %d.%d.%d.%d\r\n"
#define MAC_MSG                 "MAC ADDRESS: %x:%x:%x:%x:%x:%x\r\n"
#define GREETING_MSG            "Well done guys! Bye!\r\n"
#define CONN_ESTABLISHED_MSG    "Connection established:
    %d.%d.%d.%d:%d\r\n"
#define SENT_MESSAGE_MSG        "Sent a message. Close the socket!\r\n"
#define WRONG_RETVAL_MSG        "Something went wrong; return value:
    %d\r\n"
#define WRONG_STATUS_MSG        "Something went wrong; STATUS: %d\r\n"
#define LISTEN_ERR_MSG          "LISTEN Error!\r\n"
```

Her har vi definert noen meldinger som gjør det enklere å bruke disse videre i koden. Her har vi lagt inn noen meldinger som viser nettverkskonfigurasjonen til W5500 og noen velkomstmeldinger. Vi får også opp på terminalen om det er feil i programmet, eller vi ikke klarer å koble oss til W5500. Vi har også definert en streng med 60 karakterer som heter msg. Under er det definert en kode for å printe ut meldinger til terminalvinduet.

```
#define PRINT_STR(msg) do   {                                   \
```

```
  HAL_UART_Transmit(&huart2,(uint8_t*)msg,strlen(msg),100);       \
} while(0)

#define PRINT_HEADER() do  {
                  \
HAL_UART_Transmit(&huart2,(uint8_t*)SEPARATOR, strlen(SEPARATOR), 100
    );
HAL_UART_Transmit(&huart2,(uint8_t*)WELCOME_MSG, strlen(WELCOME_MSG),
    100);
HAL_UART_Transmit(&huart2,(uint8_t*)SEPARATOR, strlen(SEPARATOR), 100
    );
} while(0)
```

Dette gjør at vi kun kan bruke PRINT_HEADER(); og PRINT_STR(); for å få frem meldingene på terminalen. Vi har også definert en for utskrift av nettverkskonfigurasjonen.

Neste steg er å lage noen funksjoner for lesing og skriving til SPI bufferet. Vi har her også lagt til funksjoner for CS(Chip Select).

```
void cs_sel() {
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_0, GPIO_PIN_RESET); //CS LOW
}
void cs_desel() {
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_0, GPIO_PIN_SET); //CS HIGH
}
uint8_t spi_rb(void) {
    uint8_t rbuf;
    HAL_SPI_Receive(&hspi1, &rbuf, 1, 0xFFFFFFFF);
    return rbuf;
}
void spi_wb(uint8_t b) {
    HAL_SPI_Transmit(&hspi1, &b, 1, 0xFFFFFFFF);
}
```

Void cs_sel() skriver til pinne 0 på mikrokontrolleren og gjør denne lav, dette betyr at chip select er 0. void cs_desel() gjør at pinne PB0 blir høy, altså at chip select er 1. Spi_rb bruker funksjonen HAL_SPI_Receive til å lese en byte fra SPI grensesnittet. Spi_wb bruker funksjonen HAL_SPI_-Transmit til å skrive en byte til SPI grensesnittet. Når vi har definert funksjonene går vi inn i main() funksjonen. Her blir modulene som GPIO, SPI og USART initialisert. Her legger vi også til noen funksjoner og variabler.

```
int main(void)
{
```

```
  /* USER CODE BEGIN 1 */
    uint8_t retVal, sockStatus;
    uint8_t bufSize[] = {2, 2, 2, 2};
    reg_wizchip_cs_cbfunc(cs_sel, cs_desel);
    reg_wizchip_spi_cbfunc(spi_rb, spi_wb);
    wizchip_init(bufSize, bufSize);
wiz_NetInfo netInfo = { .mac= {0xc4,0x41,0x1e,0x83,0x0b,0xc3},  //
   Mac address
.ip      = {192, 168, 2 ,101}, // IP address
.sn      = {255, 255, 255, 0},  // Subnet mask
.gw      = {192, 168, 2, 1}};       // Gateway address
wizchip_setnetinfo(&netInfo);
wizchip_getnetinfo(&netInfo);
PRINT_NETINFO(netInfo);
```

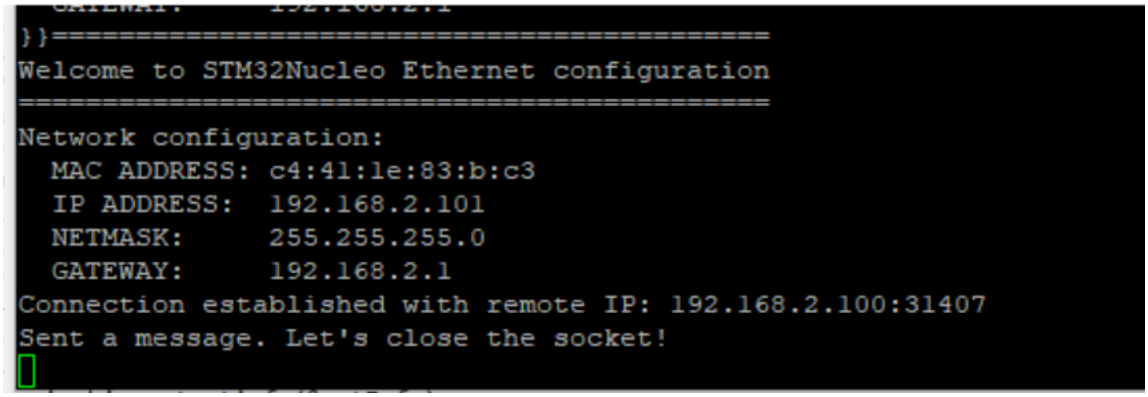
Funksjonene reg_wizchip_cs_cbfunc og reg_wizchip_spi_cbfunc er ferdige funksjoner som vi finner i driveren for W5500. Her legger vi inn parameterne som vi definerte tidligere i programmet. Vi legger og til tre variabler. retVal, bufsize og sockStatus. I wizchip_init funksjonen legger vi inn bufsize, dette er størrelsen på bufferet til de forskjellige sockets"i w5500. Her bruker vi standard buffer størrelse som er 2Kb per socket. Vi må også legge til nettverkskonfigurasjonen som vi skal bruke til W5500. Her har vi lagt inn en IP-adresse, Subnet maske, en mac -adresse og gateway-adressen. Vi bruker så en funksjon for å sette disse variablene i W5500 og en funksjon for å hente ut variablene slik vi kan printe dem ut ved bruk av PRINT_NETINFO. Når vi har satt opp nettverkskonfigurasjon til W5500 kan vi begynne å lage kode slik vi vil at programmet skal fungere. Her har vi lagt inn følgende kode [36]. Det er i denne kodesnutten vi nå kan velge hva som blir sendt og hva som skal mottas fra datamaskinen. Dette gjøres ved å endre eller legge til funksjoner som for eksempel send() eller recv() inne i while-løkka. I send() funksjonen må man spesifisere hvilke socket som er i bruk, en peker til hva som skal sendes og lengden på det som skal sendes. I recv() tar inn samme parameter som send() bare her er pekeren på hvor dataen som blir mottatt skal plasseres.

```
reconnect:
    if((retVal = socket(0, Sn_MR_TCP, 80, SF_TCP_NODELAY)) == 0) {
        if((retVal = listen(0)) == SOCK_OK) {
            while((sockStatus = getSn_SR(0)) == SOCK_LISTEN)
                HAL_Delay(100);
            while(1) {
                if((sockStatus = getSn_SR(0) == SOCK_ESTABLISHED) ){
                    uint8_t remoteIP[4];
                    uint16_t remotePort;
                    getsockopt(0, SO_DESTIP, remoteIP);
                    getsockopt(0, SO_DESTPORT, (uint8_t*)&remotePort);
```

```
                    sprintf(msg, CONN_ESTABLISHED_MSG, remoteIP[0],
                        remoteIP[1], remoteIP[2], remoteIP[3],
                        remotePort);
                    PRINT_STR(msg);

                    if((retVal = send(0, GREETING_MSG,
                        strlen(GREETING_MSG)) ==
                        (int16_t)strlen(GREETING_MSG)))
                         PRINT_STR(SENT_MESSAGE_MSG);
                    else {
                         sprintf(msg, WRONG_RETVAL_MSG, retVal);
                         PRINT_STR(msg);
                    }
                    break;
                }
                else {
                    sprintf(msg, WRONG_STATUS_MSG, sockStatus);
                    PRINT_STR(msg);
                    break;
                }
            }
        } else
            PRINT_STR(LISTEN_ERR_MSG);
    } else {
        sprintf(msg, WRONG_RETVAL_MSG, retVal);
    PRINT_STR(msg);
    }
disconnect(0);
close(0);
goto reconnect;
```

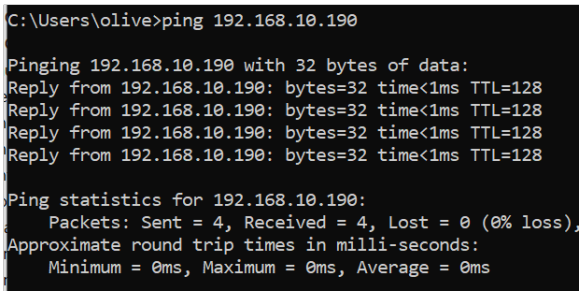
Denne koden gjør at vi lager en TCP-server som sender en melding til en socket og avslutter tilkoblingen etter meldingen er sendt. Kommunikasjonen fungerer som vist i første del av kapittelet. Her må vi definere hvilke socket vi skal bruke. I dette eksempelet bruker vi socket 0. Vi må også legge inn flere parameter i socket-funksjonen. Her må vi velge hvilken internettprotokoll vi skal bruke, hvilken port vi skal koble oss til og noe som kalles flag". Vi har her valgt TCP protokoll med tilkobling til port 80. I den siste parameteren i socket-funksjonen velger du forskjellige typer for overføring, her har vi valgt TCP uten forsinkelse. Andre alternativer her er for eksempel multicast for UDP protokollen. Videre i programmet tar bruker vi listen(0) funksjonen til å lese av verdiene som blir gitt av socket-funksjonen. Hvis vi klarer å koble oss til socket 0 går programmet videre. Vi skal da få en melding om dette i terminalen. Etter vi har fått en tilkobling blir meldingen sendt og

tilkoblingen avsluttes. Etter tilkoblingen avsluttes går vi tilbake igjen opp til starten av reconnect". Når vi nå skal kjøre programmet ser vi i terminalen at vi får opp noen meldinger. I Figur 3.20 ser



Figur 3.20: Nettverkskonfigurasjon i terminal vinduet

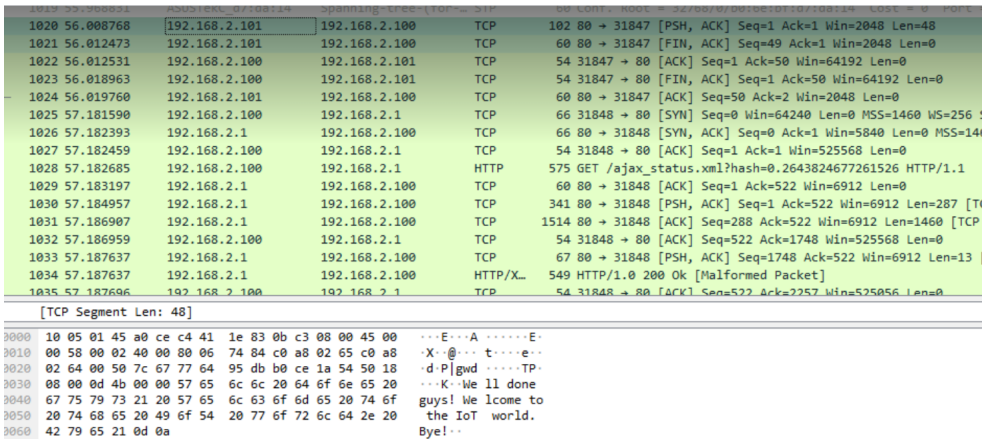
vi at nettverkskonfigurasjonen vi satte opp blir printet ut i terminalvinduet. Her kan vi se nederst på figuren at vi får en melding om at tilkoblingen er i orden og at meldingen blir sendt. Videre har vi også testet tilkoblingen ved og "pinge" ip adressen vi har satt. Dette gjøres ved å bruke ledetekstprogrammet som følger med på PC-en. Her skriver vi inn "ping 192.168.10.190", PC-en sender da et signal til IP-adressen til W5500 og sjekker om vi får svar. Denne testen ble gjort på et annet nettverk enn det vi brukte på Uis, vi har av den grunn en annen IP-adresse i eksempelet. Figur 3.21 viser resultatet vi får når vi pinger IP-adressen. Her ser vi at vi sender 4 pakker til



Figur 3.21: Resultat av ping testen

IP-adressen og vi får svar tilbake.[36]

Vi har også brukt et program som heter Wireshark. Wireshark er et gratis program som blir brukt til feilsøking, analysering og utvikling at nettverksprotokoller. Ved brukt av dette programmet har



Figur 3.22: Utdrag fra Wireshark

vi muligheten til å følge med på alle pakkene som blir sendt mellom PC-en og mikrokontrolleren vår. I figur 3.22 ser vi et utdrag fra wireshark. Her ser vi at det blir sendt pakker mellom port 80 og port 31847. En port i datakommunikasjon er et adressepunkt i en forbindelse mellom enheter som kommuniserer med hverandre. En port er et tall på 16 biter og kan ha verdi fra 0 til 65535. Det blir fordelt porter til egne kommunikasjonsprotokoller. For eksempel Port 20 blir brukt til FTP(File Transfer Protocol). Vi har her bare valgt port 80. Vi ser også nederst på figuren at pakken inneholder velkomstmeldingen vår.

3.2.6.1 Oppsummering av test

I testen ser vi at vi klarer å få til en kommunikasjon med SPI/Ethernet-modulen. Vi får til å sende TCP pakker over fra mikrokontrolleren til en nettside på PC-en. Send() funksjonen i programmet som vi finner i ioLibrary gjør at vi kan velge hva vi skal sende til nettsiden. Videre nå skal vi prøve å utvikle dette programmet slik vi leser inn fra sensorene og sender dette til en nettside. Vi brukte veldig mye tid på å få til denne kommunikasjonen og prøvde veldig mye forskjellige løsninger, men fant til slutt ut problemet. Det første problemet vi fant ut av var hastigheten på SPI overføringen. Denne var tidligere for høy. I databladet er det oppgitt at W5500 støtter hastigheter opp til 33.3MHz. Ved hastigheter opp mot 33.3MHz fikk vi problemer med koden og måtte endre denne til en lavere hastighet. Dette var veldig enkelt å endre, men det tok tid å finne problemet. Et annet problem som oppsto, var med Ethernet- tilkoblingen og gi modulen en IP-adresse. Vi prøvde her også mange forskjellige løsninger med å velge IP-adresse selv og bruke denne i programmet uten noe nytte. Etter mye testing og feiling prøvde vi å koble opp en gammel router som vi fant på Subsea-rommet på UiS og koblet denne til en svitsj og til en PC på skolen. Når vi koblet opp ruteren fikk vi opp en nettside hvor vi kunne velge IP-adresse til ruteren og klientene. Etter at vi satte opp ruteren og laget et lokalt nettverk funket programmet etter hensikten.

3.3 Svitsj

En Svitsj er en nettverkskomponent som samler informasjon fra ulike noder i et nettverk. Du kan få en svitsj i mange forskjellige størrelser, alt fra 4 porter til flere hundre. Du kan ha forskjellige hastigheter på portene i svitsj : 10,100 og 1000 MBps er de mest vanlige. En svitsj er bygd opp slik at den tar imot data i form av rammer. Det blir deretter lagret hvilken port og hvilken MAC-adresse det blir sendt fra. I første omgang blir ett signal sendt til alle portene, men hvis de bruker de samme MAC-adressene og portene vil svitsjen huske dette og kun sende det til de aktuelle portene. Dette er en stor fordel med å velge svitsjen i forhold til HUB som ikke bruker MAC-adresse og sender alltid signal til alle porter. [28]

3.3.1 Valg av Svitsj

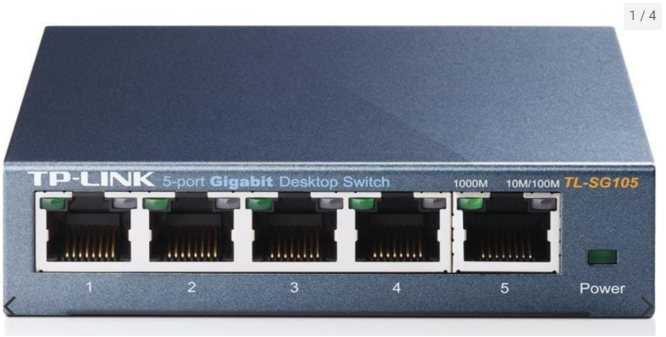
Når vi skal velge svitsj er det flere ting å forholde oss til : Mest mulig kompakt, rimelig pris, under 10W, 5 RJ45 plugger, 1 SFP port til fiber. Det er også viktig at den har riktig hastighet i inngangene minst 100 Mbs i RJ45 og 1000 Mbs på fiberinngangen. Dette for å få rask nok respons.

Navn	PLANET Svitsj	MEDICAL MICRO SVITSJ	Volktek IEN-8415L
Størrelse	162x102x28	90x45x60	136x109.5x31
Pris	500NOK	6000NOK	1927NOK
Effekt	5W	4.5W	10W

Vi ser på PLANET-svitsjen [14] at den er litt for stor. Vi skal ha denne i ett 6 toms rør på en halv meter med mange andre komponenter. Det er derfor viktig at den er så kompakt som mulig. Videre ser vi på Medical micro svitsj[15]. Denne hadde vært perfekt hadde det ikke vært for prisen. Vi har et budsjett på totalt 8000,- på hele oppgaven og kan derfor ikke bruke 6000 på en svitsj. Ut ifra det vi har funnet går vi for Volktek IEN-8415L[17]. Vi har sett på en del forskjellige svitsjer og muligheten for å lage en svitsj selv, men bestemte oss for å bruke noen svitsjer som var tilgjengelige på universitetet. Vi valgte dette på grunn av en del problemer som oppsto under programmeringen slik vi ikke fikk brukt så veldig mye mer tid på forskjellige svitsjløsninger. Den første svitsjen vi har valgt er GEU-0521 fra LevelOne. Denne svitsjen har muligheten til å koble til 4 Ethernet-porter på 1000Mbps(Megabit per sekund) og en SFP-port til fiberkabelen. Svitsjen blir tilkoblet 5V og var veldig liten, noe som gjør at den passer veldig bra i ROV-en. Vi har også fjernet dekselet på svitsjen slik at den blir enda litt mindre. Siden denne svitsjen kun har 4 Ethernet-porter var vi nødt til å bruke en annen svitsj i tillegg. Figur 3.23 viser et bilde av LevelOne svitsjen

Figur 3.23: LevelOne GEU-0521 [62]

Den andre svitsjen vi har valgt å bruke er TP-LINK TL-SG105. Denne svitsjen har 5 Ethernet-porter som er på 1000Mbps(Megabit per sekund) som er veldig rask overføring. Denne svitsjen blir tilkoblet en spenning på 9V som vi får tildelt av kraftforsyningsgruppen.



Figur 3.24: TP-LINK TL-SG105 Svitsj [61]

Denne svitsjen var litt for stor med dekselet på slik at vi måtte skru av dette og kun bruke selve kretskortet. I figur 3.24 ser vi hvordan denne svitsjen ser ut. Svitsjene sine kretskort blir så montert på en skinne som plasseres i kuppelen på ROV-en.

Kapittel 4

Belysning

Belysning

4.1 Lys behov

Det er ingen oppgaver i MATE konkurransen der de blir nødvendig med noe særlig lys ettersom vi skal kjøre i opplyst basseng, men kan allikevel være greit å utstyre ROV-en med belysning slik at vi kan utføre andre arbeidsoppgaver der de ikke er like mye lys. Vi har blitt enige med kraftforsyningsgruppa om å sette av 40W til belysning i ROV-en. Til belysning har vi vurdert flere forskjellige løsninger. vi har vurdert ett lys fra JMroboticts og en LED-diode fra DigiKey.

4.1.1 Lys alternativ

Det første alternativet vi har sett på er et lys fra JMrobotics. Dette lyset er laget for å brukes til ROV-er så vi slipper å tenke på å gjøre lyset vanntett.

Lampen kan kobles til spenninger fra 7 til 48V. Lampen gir ut spenninger fra 10-48V avhengig av spenningen inn. Dette blir styrt av en liten mikrokontroller og driver som er inne i selve lampen. Strømtrekket til lampen avhenger av inngangsspenningen. Strømmen blir regnet ut ved bruk av formelen $15/V$ altså 15 delt på inngangsspenningen. Lyset bruker da 0.3125A ved en inngangsspenning på 48V.

I lampen blir det brukt en LED diode med en lysfarge på 6200K. Vi har ikke satt noe krav til hvor mange Kelvin lyset skal ha, men 6200K gir oss et veldig hvitt lys. LED dioden gir et lys på 1500lumen som er et ganske sterkt lys. Vi ønsker derfor å ha muligheten til og dimme lyset ned. Dette kan gjøres ved å bruke et PWM signal. Dette lyset gir oss muligheten til og dimme lysstyrken med et PWM signal på 3-48V. Ulempene med dette lyset er i hovedsak prisen og størrelsen. Et lys koster cirka 1000kr og vi har tenkt å ha et lys fremme på ROV-en og et lys bak. Dette gir en totalpris på rundt 2000kr som tar mye av budsjettet vårt. Vi ønsker å ha et lys som bygger minst mulig ut fra ROV-en og dette kan bli et problem ved bruk av lyset fra JMrobotics som er nesten 7cm langt. I figur 4.1 ser vi hvordan lyset fra JMrobotics ser ut. [18]

Lumen Subsea	Light for ROV
Pris	980kr
Spenning ut	10-48V
Lysstyrke	1500 lumen
Vekt	102g
Effekt	15W

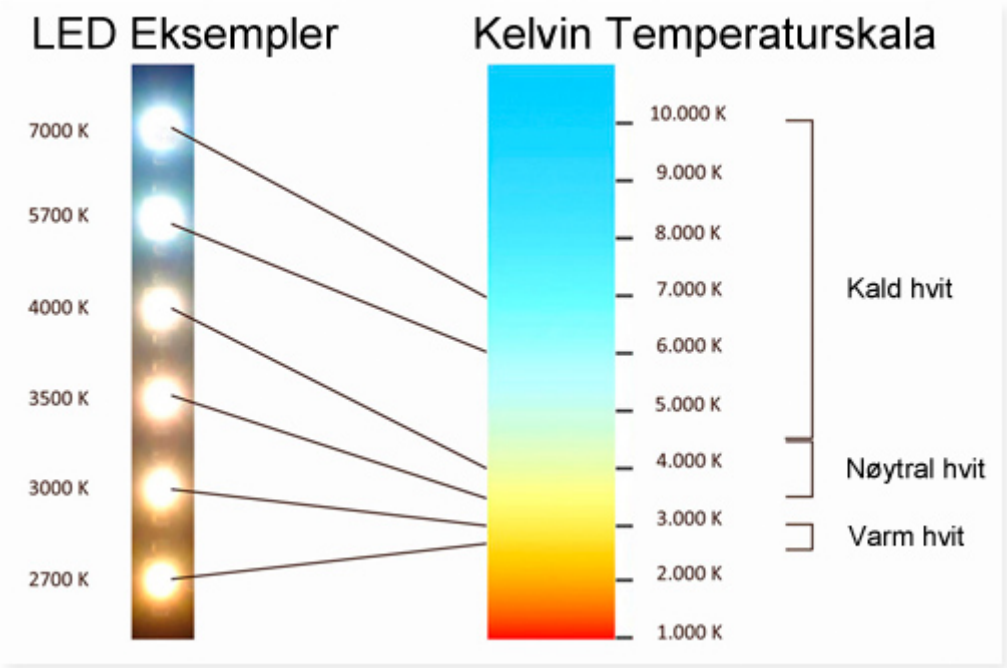
Figur 4.1: Lys fra JMrobotics [18]

Et annet alternativ vi har sett på er å bruke LED dioder som blir plassert rundt på ROV-en. Vi har sett på LED dioden BXRE-27G0800-D-73 fra DigiKey. I figur 4.2 ser vi lysdioden vi har valgt og se nærmere på.

Figur 4.2: LED diode fra DigiKey [19]

Denne LED dioden trekker rundt 6W, dette gjør at vi kan ha 6 slike LED-dioder rundt omkring ROV-en der de blir behov for lys. Denne dioden har en lysfarge på 2700K, noe som gjør at vi får et varmere lys enn lampen fra JMrobotics. Ut ifra figur 4.3 under ser vi en skala over forskjellige kelvin verdier. Det var ønskelig med en Kelvin verdi på rundt 4-5000K, men etter testing av lyset

fant vi ut at dette ga en fin farge og et bra lys. Denne dioden trenger en inngangsspenning på 16.7V for å fungere optimalt. Dioden trekker heller ikke veldig mye strøm, dioden bruker 350mA ved normalt bruk og kan maksimalt bruke 700mA.



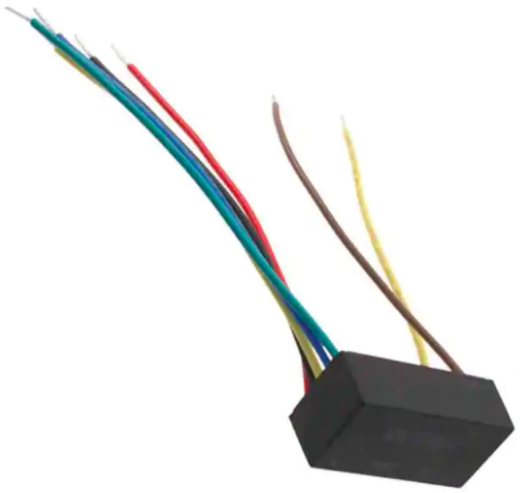
Figur 4.3: Kelvin skala [21]

Spenningen til LED dioden er 16.7V noe som vi ikke får ut ifra ROV-en sitt kraftkort. Det blir derfor nødvendig å sette inn en driver som vi kan koble lysene via.

Denne driveren må være i stand til å operere på 12-48V og kunne styre lys med effekt på 36W.

En av driverne vi har sett på er BSF-RECOM-RCD-48-0.70/W fra ballastshop.com.

Figuren 4.4 viser driveren BSF-RECOM-RCD-48-0.70/W.

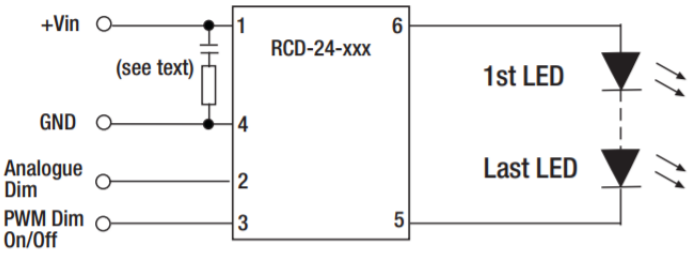


Figur 4.4: BSF-RECOM-RCD-48-0.70/W Driver til led lys [22]

Denne driveren har en inngangsspenning på 9-60V og tåler opptil 39W. Driveren har et strømforbruk på maksimalt 700mA. Vi har også muligheten til å koble til et PWM signal slik at vi kan justere lys styrken. Den er også veldig liten og koster rundt 250kr. Driveren er av typen buck som betyr at spenningen ut av driveren er mindre enn inngangsspenningen. [22]
Vi har og sett på noen andre drivere. Denne første vi har sett på har en inngangsspenning på 6-36V og kan drive lys opptil 37W. Driveren har mulighet for tilkobling av PWM signal. Denne driveren har et strømforbruk på 300-700mA. I figur 4.5 ser vi et bilde av driveren.

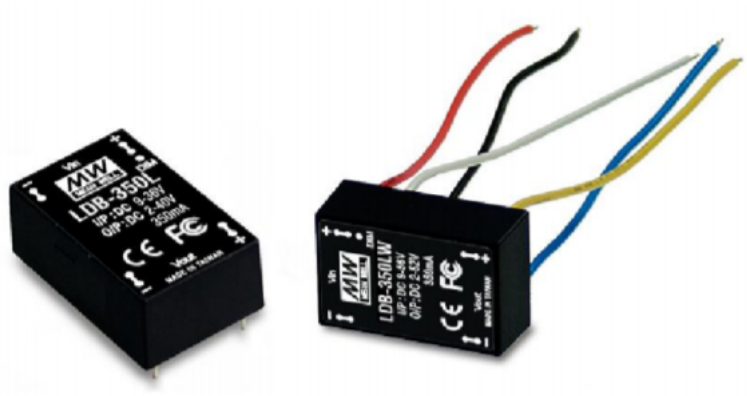
Figur 4.5: RCD-24-1.20/W/X3 LED driver [66]

Driveren fra digikey var også en del billigere og passer til kravene og behovet vi har satt. Led-driveren får sin strømforsyning fra kretskortet til kraftforsyningen. Her kobler vi også driveren til jording og til et PWM signal slik vi har mulighet for dimming. Figur 4.6 viser hvordan driveren skal kobles. Her blir LED-diodene koblet i serie på utgangen til driveren. Mellom +Vin og GND ser vi det er plassert en kondensator og en motstand i serie. Dette er for kretser hvor vi har veldig lav impedans eller skal legge kabler over lengre lengder(>10m). I ROV-en er lednings strekkene såpass korte at det ikke blir behov for det i vårt tilfelle. Denne driveren er og av typen buck.



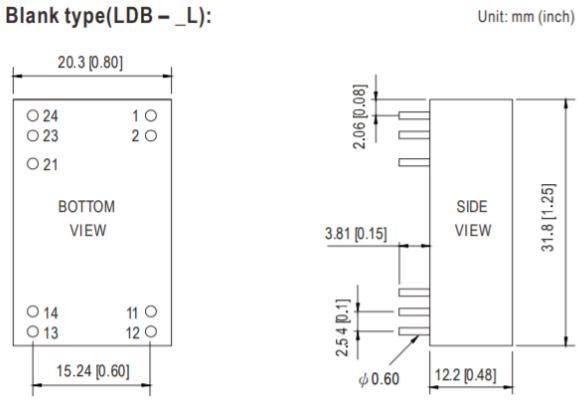
Figur 4.6: Koblingsskjema LED-driver [66]

Den siste driveren vi har sett på var fra elfadistrelec.no som heter LDB-600L [32]. Denne driveren var av type Buck/Boost. Dette betyr at driveren har mulighet for og brukes for å gi ut en lavere spenning enn inngangsspenningen, eller brukes til å gi ut en høyere spenning en tilført. Dette passet veldig bra som oss ettersom vi får 12V fra kraft kortet og kan da justere spenningen slik vi får 16.7V ut. Driveren gir ut en strøm mellom 300-600mA. Dette avhenger av spenningen vi gir inn på driveren. Vi har her og mulighet for å bruke PWM til dimming. [33] I figur 4.7 ser vi



Figur 4.7: LDB-600L driver fra Elf [32]

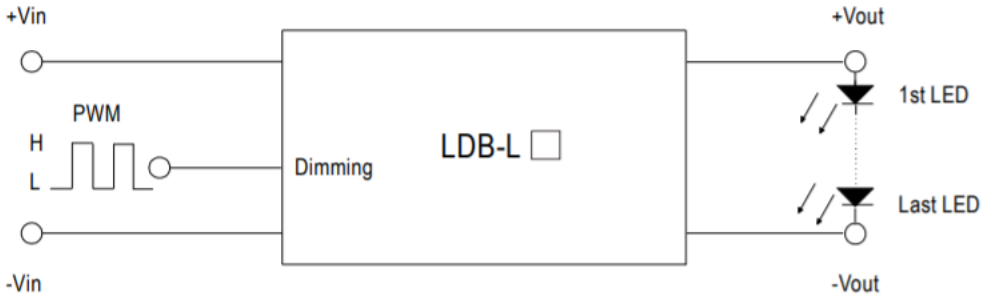
et bilde av driveren LDB-600L. LDB-600L koster bare 85kr som er en del billigere enn de andre driverene. På grunn av prisen og muligheten for å håndtere både buck og boost har vi valgt å gå for denne driveren. Driveren kan håndtere opptil 18W med lys slik at vi må ha to slike drivere hvis vi skal ha 6 LED-dioder. Figur 4.8 viser pinnene til driveren. På figuren ser vi hvilke pinner som er brukt for spenning og dimming. Her bruker vi pinne 24 og 1 for spenning inn og pinne 13 og 12 ut til lysene. Pinne 21 skal brukes til PWM dimming.



Pin No.	Output	Comment
1,2	-Vin	Don't connect to -Vout
11,12	-Vout	LED - Connection
13,14	+Vout	LED + Connection
21	PWM DIM	ON/OFF and PWM Dimming (Leave open if not used)
23,24	+Vin	DC Supply

Figur 4.8: Pinnene som skal kobles til på driveren [33]

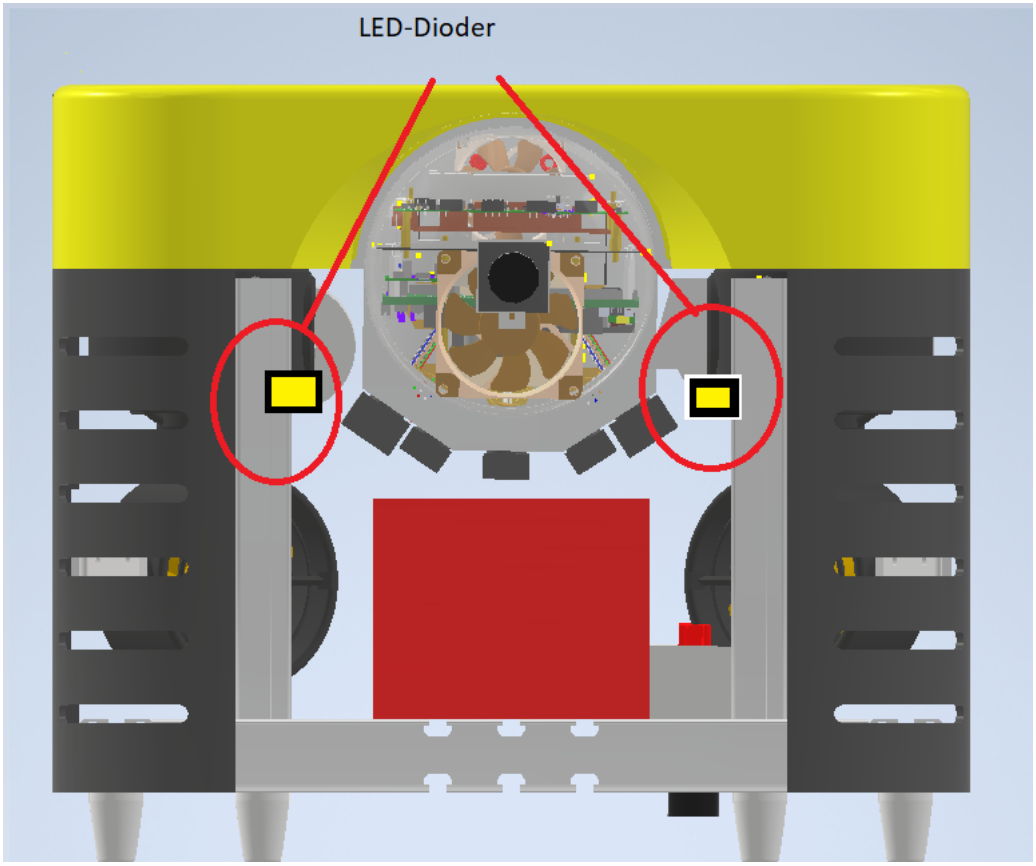
I figur 4.9 ser man et koblingsskjema av driveren. Her blir og lysene koblet i serie. Vi må her ta i bruk 2 drivere. LED-lysene blir da koblet i serie med 3stk til hver driver.



Figur 4.9: Kobling av LDB-600L [33]

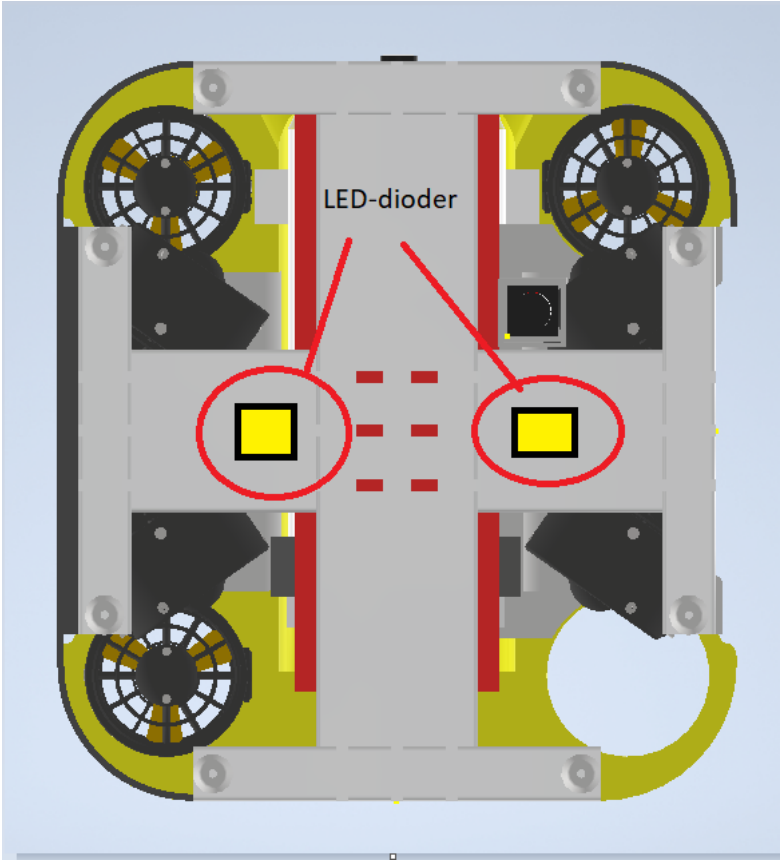
4.1.2 Plassering av lys

Lysene blir plassert rundt ROV-en. Her skal vi feste lysdiodene på rammen til ROV-en og gjøre dette vanntett med bruk av en blank epoxy. Epoxy er en veldig slitesterk maling som gir en blank overflate etter behandling. Epoxy kan brukes til maling, liming og reparering av diverse overflater. De kan brukes på plastikk, metall, tre og lignende. Epoxy lim blir ofte brukt til å reparere båter, biler og møbler. Figur 4.10 og figur 4.11 viser sånn ca hvordan lysene vil bli plassert på ROV-en.



Figur 4.10: Plassering av lys fremme på ROV-en

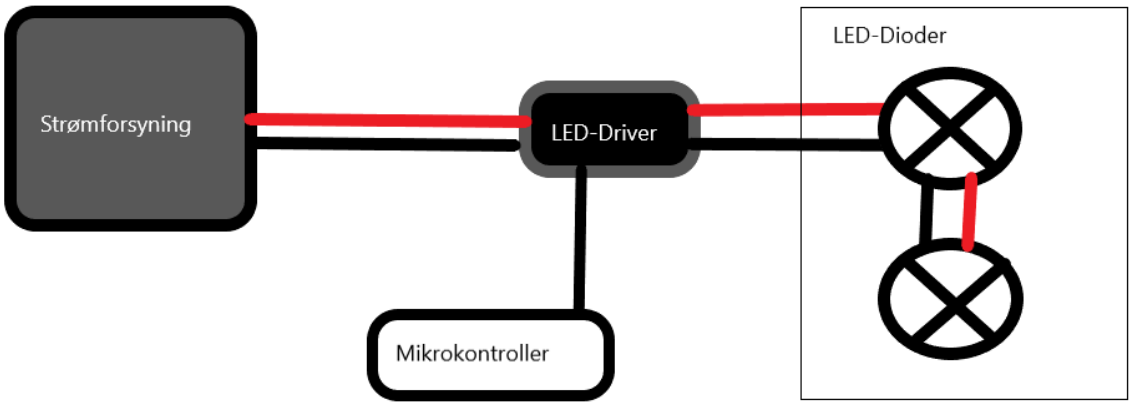
I starten var planen å ha 6 lys på ROV-en som skulle bli plassert rundt ROV-en, men vi har nå valgt en ny løsning med 4 LED-dioder. Her har vi valgt to lys fremme og to lys under ROV-en. Dette gjør at vi kan lyse opp steder hvor vi trenger og ha god sikt hele tiden. Vi får med denne lysplasseringen god oversikt over begge kameraene. For lysene fremme på ROV-en er det laget fester til diodene som enkelt kan justeres opp og ned etter behov. Diodene blir festet i en mutter som kan styres opp og ned. Lysene under ROV-en blir festet med epoxy i rammen til ROV-en.



Figur 4.11: Plassering av lys under ROV-en

4.1.3 Test av belysning

For testing av belysningen har vi loddet på noen ledninger på lysdiodene slik vi har mulighet til å koble dem til driveren. LED driveren blir så koblet opp mot en strømforsyning som vi har på labben. Her kan vi stille inn forsyningen til den spenningen vi ønsker. Strømforsyningen vi har brukt er av typen TTi PL303QMD-P QUAD-MODE DUAL POWER SUPPLY. Når vi kobler opp kretsen slik og stiller inn forsyningen på 12V stilles denne automatisk ned til 8V. Måling av utgangen ved inngangsspenning gir oss 16.7V. Vi må ha en utspenning fra forsyning på 12V og utgangsspenning fra driveren på 16.7. Dette problemet kan løses ved bruk av motstander i serie med driveren. Figur 4.12 viser hvordan koblingen av lyset ble gjort på labben.



Figur 4.12: Test oppkobling av lys

I figuren ser vi hvordan oppkoblingen er. Her har vi to ledninger + og - fra strømforsyningen som blir koblet til Vin+ og Vin- på driveren. Vout+ og Vout- fra driveren blir koblet til + og - på LED-dioden. PWM signalet blir koblet til en PWM utgang fra mikrokontrolleren. Når vi legger inn en motstand på 67Ω får vi ut en spenning på 16V fra driveren med en inngangsspenning på 12V. Til PWM signal har vi brukt pinne PE13 på mikrokontrolleren til hoved ROV-en. Denne pinnen blir satt til å være en PWM utgang. PWM signalet bruker den interne klokken til ROV-en som klokke kilde.

4.1.4 Oppsummering av test av belysning

Vi har testet med driveren fra digikey og elfadistrelec og fant ut at sistnevnte fungerte best til vårt bruk. Her kan vi få 16.7V ut og 12V inn ved bruk av motstander i serie med driveren. Lysene var veldig sterke og gir en fin opplysning av ROV-en. Vi har ikke fått til dimmingen på noen god måte ennå. Dette vil heller ikke bli veldig prioritert før bachelor innleveringen, men skal videreutvikles frem mot konkurransen.

Kapittel 5

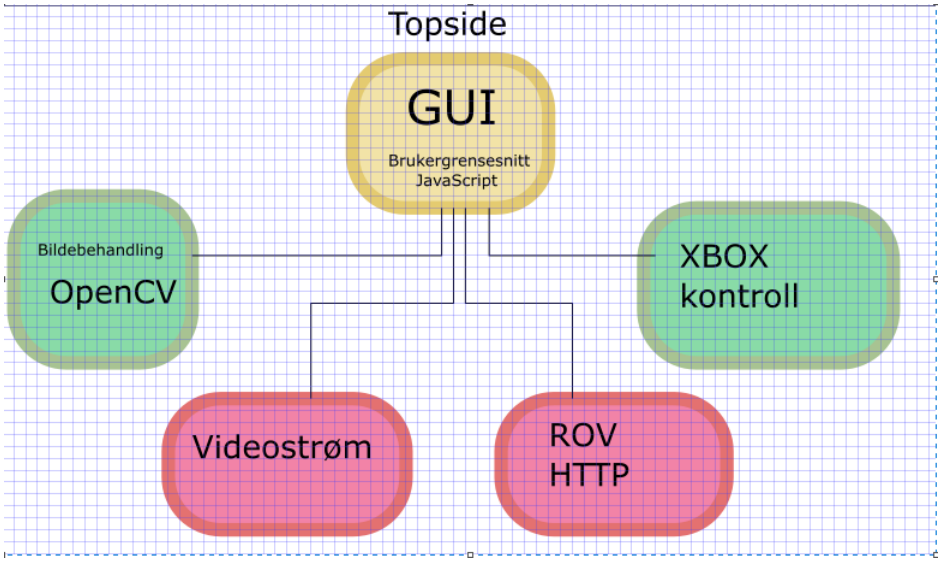
Programvare i ROV

Innhold

5.1 Program til kommunikasjon i ROV

Vi har nå brukt en del tid på å få til en tilkobling mellom PC-en og mikrokontrolleren over Ethernet og vi forstår nå bedre hvordan dette fungerer.

Nå skal vi gå igjennom valg av språk og internett-protokoll i programmet som faktisk skal brukes. Her er planen å ha en webserver som skal lese av dataen fra sensorene på sensorkortet og lese av reguleringsparameterne i motorene. Dette skal så vises på brukergrensesnittet som vi skal bruke. Her skal vi kunne lese av dataen til diverse sensorer og sende nye reguleringsparameter til ROV-en. Figur 5.1 viser hvordan kommunikasjonen skal fungere.



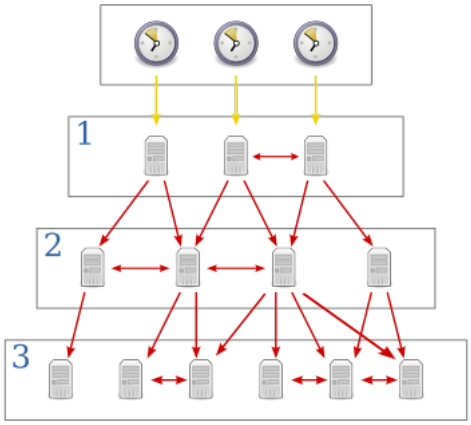
Figur 5.1: Blokkskjema over kommunikasjon oppsettet

5.1.1 Valg av internettprotokoll og programmeringsspråk

Dette blir et mer avansert program enn det vi har brukt i testen som vi forklarte i kapittel 3.2.6. For å realisere programmet kan man bruke mange forskjellige protokoller for å gjøre dette. W5500 støtter som sagt flere typer. Først en liten gjennomgang av de forskjellige protokollene som brukes og støttes av w5500 -brikka før. Her må vi nok bruke en kombinasjon av flere av protokollene for å få til et bra oppsett.

5.1.1.1 SNTP

SNTP(Simple Network Time Protocol) er en forenklet versjon av NTP(Network Time Protocol) som er en internett protokoll som blir brukt for å synkronisere et klokkesignal mellom datamaskinen og andre enheter i nettverket. NTP er en av de eldste protokollene og har vært i drift siden 1985. Målet med NTP er å synkronisere klokkene til alle enhetene som er tilkoblet i løpet av noen millisekund. Figur 5.2 viser et eksempel over hvordan et NTP system er koblet opp. De gule pilene i figuren viser en direkte tilkobling og de røde pilene viser en nettverkstilkobling. Her lytter klientene etter oppdateringer fra tjeneren. Øverst i figuren ser man tidsstyringsenhetene. Dette kan være enheter som for eksempel GPS(Global Positioning System) eller andre klokker med veldig høy presisjon. Dette blir så koblet til datamaskiner hvor datamaskinens klokker blir synkronisert i løpet av noen få millisekund. Tilkobling går så videre over nettverket til andre datamaskiner som synkroniserer klokkene.

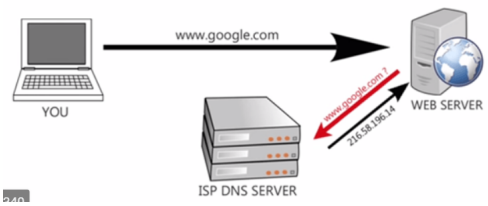


Figur 5.2: NTP topologi [45]

I NTP brukes det UDP på port 123 som transportmetode. Dette er litt annerledes enn metoden vi har brukt med TCP. SNTP og NTP bruker samme metode for overføring og samme prinsipp, men SNTP blir ofte brukt i innebygde systemer hvor nøyaktighet ikke er like viktig. [45]

5.1.1.2 DNS

DNS(Domain Name System). I denne protokollen blir det gitt navn til IP-adresser. Dette gjør at vi slipper å gå rundt og huske på alle IP-adressene til forskjellige nettsider. DNS har også muligheten til å gjøre andre ting enn å gi navn til adresser. Vi kan bruke DNS til å be om informasjon om diverse opplysninger om PC-en. Vi kan også bruke DNS til å sørge for å sende e-poster til rett mottaker. En DNS-server blir brukt av nesten alle nettsider vi kan besøke. Når vi søker etter en nettside på nettleseren vil datamaskinen sende en melding til DNS-serveren som igjen sender tilbake hvilket domene navn nettsiden vi vil besøke har. [46]



Figur 5.3: Eksempel på hvordan DNS fungerer [57]

Figur 5.3 viser et eksempel på hvordan en DNS-server fungerer. Her med et søk fra brukeren som web-serveren sender til DNS-serveren og spør om IP-adressen til www.google.com. DNS-serveren sender så tilbake IP-adressen og google dukker opp i nettleseren.

5.1.1.3 TFTP

TFTP(Trivial File Transfer Protocol). Denne protokollen brukes i hovedsak til sending og mottakelse av filer, men kan også brukes til andre ting som sikkerhetskopiering og til å starte datamaskiner uten disk.[47] TFTP kan også brukes til å laste inn nettsider til en HTTP-server. Med TFTP kan vi kun sende og motta, endringer som sletting og flytting av filer er ikke mulig.

TFTP bruker også UDP som sin transport protokoll. Her sender verten i systemet ut forespørsler til den noden vi ønsker å sende eller motta filer fra. Her sendes filene i blokker på 512 biter, har vi mindre enn dette vil filoverføringen avsluttes. TFTP brukes som et tillegg til FTP og ikke en erstatning. TFTP har litt mindre funksjoner enn FTP og kan brukes hvor det kun trengs enkle funksjoner for overføring. [35]

5.1.1.4 FTP

FTP(File Transfer Protocol). Denne protokollen blir også brukt til overføring av filer mellom en server og en klient. Forskjellen på FTP og TFTP er i hovedsak måten meldingene transporteres. I TFTP brukes det UDP og i FTP brukes det TCP.

Her lytter FTP-serveren til nettverket hvor vi kan koble oss til med en FTP-klient. Når tilkoblingen blir godkjent kan vi lese og skrive til tjenerens filsystem. Her er det også mulig å endre navn på filer, flytting og sletting av filer. Klienten og tjeneren i FTP kommunikasjonen er dataprogrammer som vi kan kjøpe eller lage selv på datamaskinen. Her trengs det et dataprogram som er en FTP-klient på en datamaskin og et dataprogram som virker som en FTP-tjener på en annen.

Det er og noen fordeler og ulemper med denne protokollen. Det er mange programmer som bruker FTP som er enkle og ta i bruk. På grunn av dette er det billig å få tak i FTP- programvare. Ulempene med FTP er at kommunikasjonen mellom datamaskinene er ukryptert. Det gjør at

informasjonen som sendes kan leses av alle som har tilgang til kommunikasjonen til datamaskinen. Det kan og oppstå problemer med brannmuren ved bruk av FTP. Noen ganger vil brannmuren stoppe en filoverføring hvis FTP prøver å koble seg til en annen port en den som er gitt.

FTP har vært i bruk lenge, men i januar i år har denne protokollen blitt deaktivert på mange nettlesere. Nå er det mest vanlig å bruke HTTP i stedet for FTP. [48]

5.1.1.5 SNMP

SNMP(Simple Network Management Protocol) er en internettprotokoll som blir brukt til å samle inn og organisere informasjon om enheter som er koblet til nettverket. Denne informasjonen kan endres slik at vi kan endre virkemåten til diverse enheter. Enheter som bruker SNMP er svitsjer, rutere, printere og lignende. [49] SNMP er bygget opp av fire forskjellige deler.

- SNMP-agent
- SNMP-enheter
- SNMP-leder
- Database

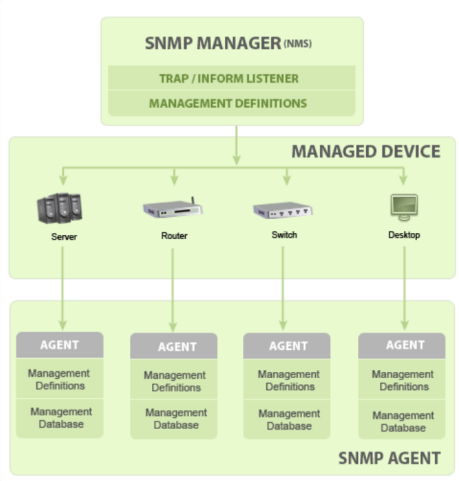
SNMP-agenten er en programvare som kjører på en enhet. Her blir det samlet inn informasjon om diskplass og diverse nettverksinformasjon. Når SNMP-lederen blir spurt om informasjon sender agenten dette tilbake til styringssystemet. Enheter har som oftest installert en SNMP-agent fra før.

SNMP enhetene er de forskjellige enhetene som agentene kjører på.

SNMP-lederen er en programvare som agentene sender informasjon til. Lederen sender ut forespørsel til agentene med jevne mellomrom. Her ber lederen agentene om å sende tilbake oppdateringer om nettverksbruken.

Den siste delen i et SNMP-system er en database som inneholder alle objektene som kan styres og endres på en enhet. I et SNMP-system finnes det flere forskjellige funksjoner. Her er det blant annet funksjoner som lar oss hente ut verdier fra flere variabler og funksjoner som lar oss endre verdiene på flere variabler. Dette gjør at SNMP er en mulig løsning som bruk i ROV-en der vi i hovedsak skal endre variabler over nettverket. [59]

Figur 5.4 viser et eksempel over hvordan et SNMP-system er bygget opp. Her har vi SNMP-lederen øverst som er koblet til diverse enheter på nettverket. Alle enhetene har så en innebygd agent som kommuniserer med lederen.



Figur 5.4: SNMP eksempel [58]

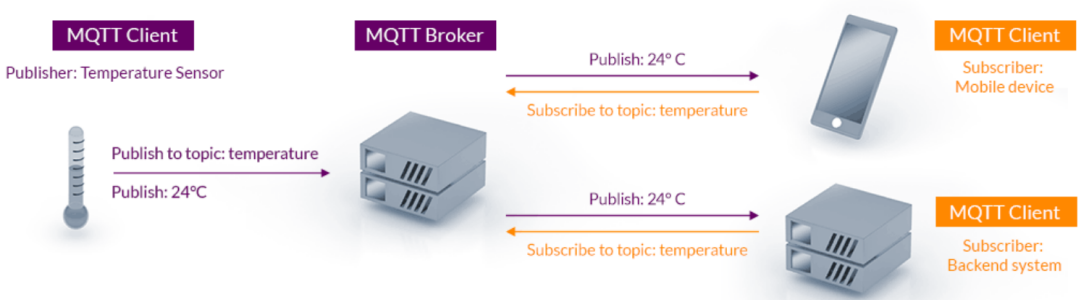
5.1.1.6 DHCP

DHCP(Dynamic Host Configuration Protocol) er en protokoll som lar oss automatisk generere IP-adresser til klienter som er tilkoblet en server. Dette gjør at vi kan enkelt kommunisere mellom alle enhetene i et nettverk. Her blir det også brukt UDP som transportprotokoll. En DHCP-operasjon blir delt inn i fire forskjellige faser: Oppdagelse, tilbud, forespørsel og bekreftelse. Klienten sender først ut en oppdagelsesmelding på nettverket som blir plukket opp av DHCP- tjeneren. Når tjeneren mottar denne meldingen, sender tjeneren ut et tilbud til klienten. Klienten sender så ut en melding om at den bekrefter tilbudet som tjener har gitt. Når tjeneren mottar bekreftelsesmeldingen, er konfigurasjonen fullført. [50]

5.1.1.7 MQTT

MQTT(Message Queuing Telemetry Transport) blir brukt til å transportere meldinger mellom enheter i et nettverk. Protokollen ble opprettet i 1999, da ble den brukt for å overvåke oljerørledninger. Nå brukes MQTT i mange forskjellige bransjer som logistikk, produksjon, transport og i smarthus.

Her brukes det vanligvis TCP som transportprotokoll, men kan MQTT støtter også noen andre protokoller for transport. I MQTT-protokollen blir det brukt en tjener som mottar alle meldingene fra klientene og videresender meldingene til rett klient. En MQTT-klient kan være mye forskjellig som for eksempel en mikrokontroller. [51]



Figur 5.5: MQTT eksempel med bruk av temperatur sensor [60]

I figur 5.5 ser vi et eksempel av et MQTT-oppsett. Her ved bruk av en temperatursensor. Temperatursensorer sender informasjon til MQTT-tjeneren som kan kobles til forskjellige enheter. På enhetene kan vi da lese av temperatursensoren.

5.1.1.8 HTTP

HTTP(Hypertext Transfer Protocol) er den protokollen som blir mest brukt til å sende informasjon på internett. HTTP er en forespørsel/respons type protokoll. I HTTP blir det som oftest opprettet en TCP-forbindelse mellom en nettside og en klient. Når forbindelsen mellom tjeneren og nettsiden er etablert venter tjeneren på en forespørsel fra klienten. Når forespørselen er mottatt av tjeneren blir det sendt tilbake en respons. Her er det noen forskjellige responsstrenger vi kan motta. For eksempel "200 OK" betyr at forespørselen er i orden. "404 Not Found" betyr at den forespurte adressen ikke finnes. Det er flere slike statuskoder for forskjellige forespørsler. I HTTP finner vi mange metoder for å hente ting eller endre ting på en tjener. De mest brukte metodene her er GET og POST. GET brukes når vi skal hente ut informasjon fra nettsiden. For eksempel kan vi hente ut sensordata fra en temperatursensor. POST brukes når vi skal endre data på en tjener. For eksempel kan vi endre verdien på en variabel. [52]

Figur 5.6: Eksempel på en HTTP-forespørsel og respons [67]

I figur 5.6 ser vi et eksempel på bruk av W5500 som HTTP-tjener. Her sendes det GET og POST-forespørsel fra datamaskinen. Dette blir så mottatt av tjeneren som sender tilbake en respons om alt er i orden etterfølgt av dataen vi vil ha tak i.

Nå har vi sett på en del forskjellige måter vi kan bruke SPI/Ethernet- modulen på. Vi må nok ta i bruk flere av protokollene vi har gått gjennom. Vi skal nå se litt videre på hvordan vi kan realisere programmet vi skal ha på topside. Det finnes flere muligheter for å lage en webserver. Dette kan for eksempel gjøres i JavaScript, Python og C++. Det finnes flere muligheter enn de som er oppgitt, men dette er de mest brukte.

5.1.1.9 JavaScript

Figur 5.7: Javascript logo [56]

JavaScript ble utviklet i 1995 og er et høynivå programmeringsspråk. I figur 5.7 ser vi logoen til JavaScript. Høynivå betyr at programmeringsspråket er mer abstrakt enn lavt nivå programmeringsspråk. Høynivå har ofte flere ferdige funksjoner enn lavt nivå og er enklere å bruke for mer avanserte program. Vi trenger her ikke å programmere så mye i datamaskinens maskinkode. JavaScript er en av de mest brukte programmeringsspråk innen web utvikling. Vi kan bruke språket til både enkle skriptprogram og mer avanserte webapplikasjoner. Alle nettlesere som er i bruk idag kan enkelt ta i bruk Javascript uten behov for ekstra programvare. JavaScript kan brukes til både objektorientert og funksjonell programmering. [56]

5.1.1.10 Python

Figur 5.8: Python logo [55]

Python er et objektorientert programmeringsspråk som ble oppfunnet i 1989.I figur 5.8 ser vi logoen til Python. Det har lenge vært et av verdens mest brukte programmeringsspråk. Python har en veldig enkel og lettleselig syntaks som gjør at Python-programmer er enkle å forstå seg på og enkle å skrive. For å bruke Python på en enkel måte må vi laste ned et utviklingsverktøy hvor vi kan skrive og kjøre programmet. Tidligere i undervisningen på skolen har vi brukt PyCharm

som blir laget av JetBrains som utviklingsverktøy. Dette er et veldig greit og oversiktlig program til å skrive Python-kode. Det finnes også veldig mange ekstra utvidelser til Python som er enkle og installere i PyCharm. Her kan vi bruke tillegg som FastAPI eller flask som er rammeverk for å bygge webservere og webapplikasjoner. [55]

5.1.1.11 C++

Figur 5.9: C++ logo [54]

I figur 5.9 ser vi logoen til C++. C++ er også et høynivå programmeringsspråk som ble laget rundt 1983-1985. Dette er en utvidelse av programmeringsspråket C som vi bruker til å programmere i mikrokontrolleren. C++ har også muligheten til å støtte objektorientert programmering, data abstraksjon, funksjonell programmering og generisk programmering. C++ består i hovedsak av to deler. Dette er kjernespråket og standardbiblioteket. Standardbiblioteket inneholder de fleste bibliotekene vi finner i C, men har også her muligheten til å bruke andre datatyper og algoritmer. Kjernespråket bygger også på språket C, men her har vi i tillegg mange funksjoner og klasser som vi ikke har tilgang til i C. [54]

5.1.2 HTTP-Server og brukergrensesnitt

Vi har nå sett på flere muligheter for protokoller for overføring over internett og programmeringsspråk. Vi har bestemt oss for å bruke W5500 som en HTTP-server. Dette fordi vi har muligheten til å enkelt koble oss opp mot en nettside hvor vi kan ha kontroll på alle variablene. HTTP overføring er også raskere enn overføring av TCP-pakker. [13] Det er også enkle HTTP-metoder for lesing og endring av variabler som er enkle å ta i bruk i for eksempel JavaScript eller Python. Test av HTTP-serveren sin hastighet og overføringstid vil bli mer forklart i kapittel 7.2

Når vi bruker W5500 som en HTTP-server har vi muligheten for å lage en nettside og funksjonene til nettsiden i samme program som vi programmerer mikrokontrolleren. Vi har også muligheten til å lage en nettside eller webserver i en av programmeringsspråkene som er listet over.

Videre skal denne serveren kobles opp mot et GUI(Graphical User Interface) som vi skal lage i samarbeid med bildebehandlingsgruppa. Vi skal her lage en fil som inneholder alle variablene som skal leses av eller skal endres i mikrokontrolleren. Her har vi variabler fra regulatoren, XBOX-kontrollen, sensorverdier, strømforbruk og motorpådrag. Variablene skal være i JSON(Javascript

Object Notation) som er en tekstbasert standard for å formatere filer. Her kan vi ta inn tall, bokstaver og andre datatyper å gjøre om til en felles datatype som er enkel å lese.

Videre skal denne filen sendes frem og tilbake mellom mikrokontrolleren og nettsiden. Her skal vi kunne gå inn og endre verdier på nettsiden vår slik at de også endres i mikrokontrolleren.

Brukergrensesnittet skal skrives i JavaScript hvor vi bruker HTTP- funksjoner som POST og GET for å hente ut informasjon fra nettsiden og endre verdier. Her skal vi bruke POST for å endre variablene som for eksempel reguleringsparameter og lignende. GET bruker vi for å hente ut variabler som for eksempel sensordata og lignende.

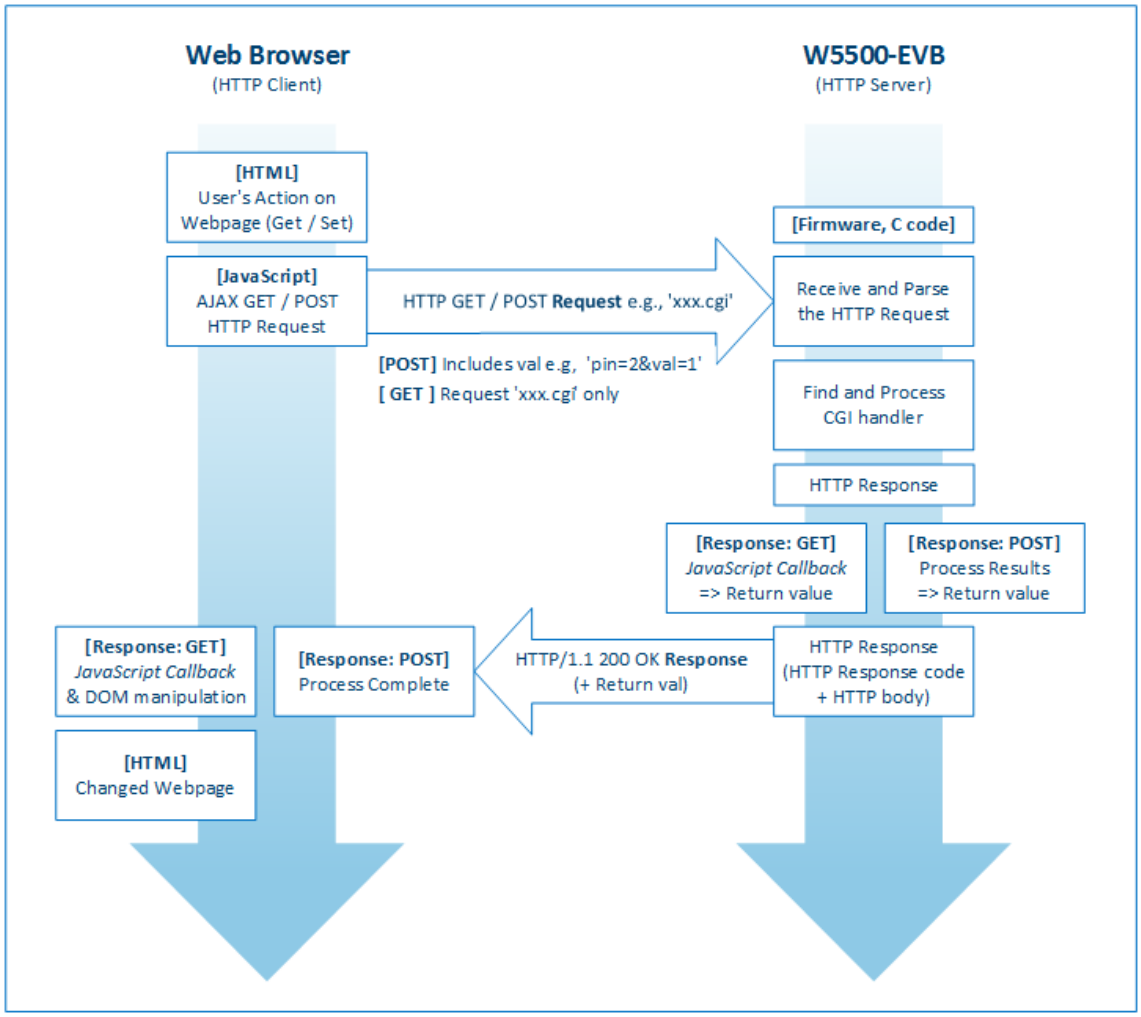
Det finnes mange måter å lage en HTTP-server på W5500-brikka. Vi har utforsket en del med forskjellige løsninger vi har funnet på nettet. Nesten alle manglet veldig mye dokumentasjon og de fleste var programmert i andre utviklingsverktøy eller i andre programmeringsspråk. De var heller ikke laget for å brukes med kretskortet vi har hvor W5500 er plassert.

Til slutt fant vi en server som var laget i C-kode i STMCubeIDE [65]. Vi lastet ned filene til PC-en og la dem inn i STMCubeIDE. Her var det en del funksjoner vi måtte endre på og andre funksjoner vi måtte lage. Vi fikk til å lage serveren og sende noe data opp til nettleseren.

Vi har derfor bestemt oss for å bruke dette som et rammeverk for HTTP-serveren. Vi må videre legge til alle variablene som skal kunne endres og alle variablene vi skal kunne hente ut. Slik som programmet er nå blir alle filene laget i STMCubeIDE. Her blir det også implementert noen funksjoner fra JavaScript og HTML(HyperText Markup Language)- kode for å legge ut data på nettsiden. HTML er et markeringsspråk som lar oss lage et oppsett av en nettside. Her kan vi legge til knapper og velge plassering av diverse ting rundt på nettsiden.

Vi skal altså skrive C kode, Javascript og HTML i STMCubeIDE som skal kunne kobles opp mot brukergrensesnittet som blir laget i JavaScript.

Figur 5.10 viser et blokkskjema over hvordan HTTP-tjeneren fungerer. Her ser vi at vi bruker nettleseren som en klient. Nettsiden vi har laget blir altså laget i HTML-kode. Her har vi noen funksjoner i JavaScript som vi bruker til å sende POST og GET-forespørsel til HTTP-serveren som er i mikrokontrolleren. Her lager vi noen funksjoner i C for å behandle forespørslene. Den behandlede dataen blir så sendt ut fra tjeneren til nettleseren via en responsmelding. Hvis responsen er i orden vil dette fullføre prosessen og nettsiden vil endre seg alt etter hva vi ønsker å gjøre. I vårt tilfelle blir dataen fra XBOX-kontrollen lest inn via et program som motorkontrollgruppa har laget. Dataen blir så behandlet og gjort om til data som kan brukes. Disse verdiene blir så lagt inn i en POST-forespørsel som sendes ned til mikrokontrolleren. Her blir variabelen gitt den verdien som XBOX-kontrollen har. Vi har også laget variabler for diverse sensorer. Verdiene til sensorene blir så lagret i forskjellige variabler. Disse verdiene kan vi så få tak i ved å bruke en GET-forespørsel fra datamaskinen på overflaten.



Figur 5.10: HTTP serveren sin virkemåte [64]

Mer om selve koden og hvordan funksjonene fungerer kommer i kapittel 6 som handler om konfigurasjon av kommunikasjonsoppsettet. Her skal vi gå gjennom virkemåten til programmet og hvordan vi enkelt kan sette opp nettsiden slik koden kan bli brukt og videreutviklet av fremtidige studenter på UIS-Subsea.

Kapittel 6

Konfigurasjon av Mikro-ROV

Innhold

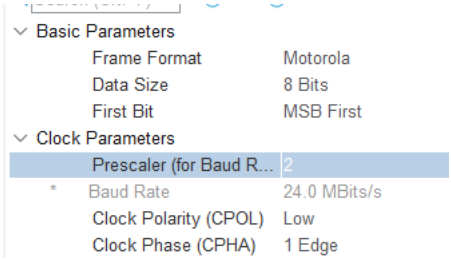
6.1 Oppkobling av SPI/Ethernet modul

Her skal vi nå først gå gjennom konfigurasjonen av SPI/Ethernet- modulen til mikro ROV-en. I oppkoblingen til mikro ROV-en bruker vi like mange pinner på mikrokontrolleren som vi gjorde i testen i kapittel 3.2.6. På mikrokontrolleren til mikro ROV-en bruker vi de samme pinnene som vist i kapittel 3.2.6. Gruppa som har ansvar for mikro ROV-en skal også bruke en del pinner på mikrokontrolleren, men her skal vi kun gå igjennom det som trengs for SPI/Ethernet-modulen. Programmet er enda under utvikling slik at det ferdige produktet inneholder litt flere ting enn det vi går igjennom her, men slik det er nå har vi en HTTP-server hvor vi kan lese og skrive til variablene vi har i C fra Python via POST og GET-forespørsel. Vi må også endre på noen parameter for SPI-bussen.

I figur 6.1 ser vi hvilke parameter som kan endres. Her må vi endre "Data Size" fra 4 bit til 8 bit slik at vi kan overføre flere biter i en melding.

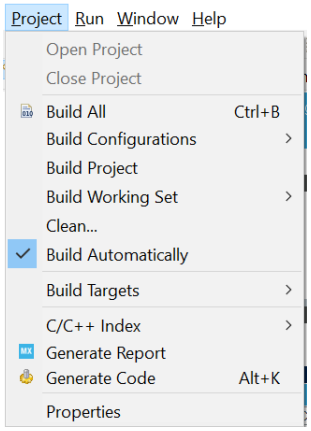
Vi må også sjekke klokkefrekvensen til SPI-grensesnittet. I teorien skal modulen kunne håndtere klokkesignal opptil 80 MHz, men har bare blitt testet opp til 33.3MHz. Vi må altså passe på at klokkesignalet er under 33.3MHz. Vi ser her at SPI-bussen sin hastighet er 24Mbit/s som er ganske høyt til å være SPI. Vi kan her enkelt halvere frekvensen ved å endre på "Prescaler". Ved å justere "Prescaler" opp vil frekvensen halveres. Setter vi prescaler til 4 vil frekvensen bli dividert på 4

osv. Vi har prøvd ut forskjellige hastigheter, men ser ikke noe tegn til forskjell på. Vi har da valgt å bruke 24Mbit/s som vår overføringshastighet.



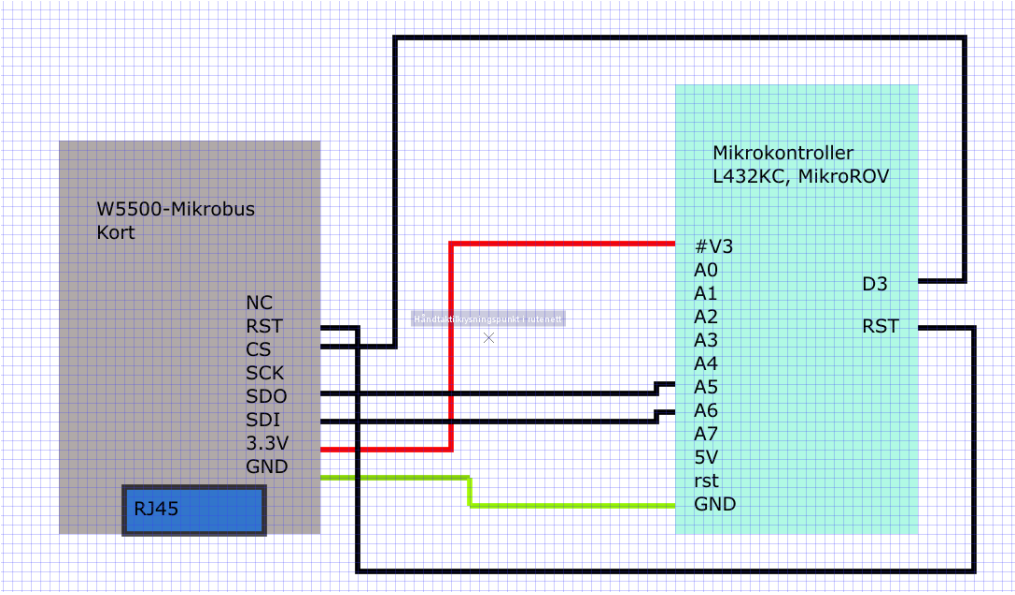
Figur 6.1: SPI parameter

Etter at vi har fått satt opp pinnene vi skal bruke og resten av gruppene har fått tildelt sine pinner trykker vi på "Generate Code" som er vist i figur 6.2, STM32CubeIDE lager da et oppsett og legger til alle funksjonene og parametrene vi velger.



Figur 6.2: Generate Code

For selve oppkoblingen trenger vi 7 ledninger mellom SPI- modulen og mikrokontrolleren. Dette er 4 ledninger til SPI- grensesnittet, en til 3.3V, en til jording og en ledning til "reset" -pinnen på mikrokontrolleren. Reset gjør at vi kan tilbakestille W5500-brikka, noe som må gjøres når vi laster inn et nytt program. Denne pinnen trenger ikke å velges i STMCubeIDE. Figur 6.4 og figur 6.5 viser hvordan vi skal koble opp modulen med mikrokontrolleren til Mikro ROV-en.



Figur 6.3: Koblingsskjema av SPI modul i Mikro ROV

6.2 Programmering av Mikro ROV

Når alt er koblet opp riktig og pinnene er konfigurert riktig kan vi begynne å programmere i STM32CubeIDE. Det første vi gjør da er å importere driveren fra Wiznet som vi brukte i test-kapittelet [30]. Her skal vi i tillegg ta i bruk følgende filer som vi finner på GitHub-sidene til Wiznet.[65]

- webpage.h som inneholder noen eksempeler på nettsider som vi skal modifisere og legge til nye
- userHandler.c som inneholder noen ferdige funksjoner for sending av HTTP-forespørsler
- userHandler.h som er en header-fil for userHandler.c. Det vil si en fil som inneholder funksjonserklæringer og definisjoner som skal brukes i flere forskjellige filer.

Vi må også ta i bruk alle HTTP-filene som vi finner i driveren fra Wiznet. Filene her inneholder flere funksjoner og metoder for initialisering av en HTTP-server som vi skal forklare etter hvert. Når vi har fått inn alle filene begynner vi å inkludere filene i "main.c" -filen som er hovedprogrammet.

main.c :

```
#include <stdio.h>
#include "wizchip_conf.h"
#include "httpServer.h"
```

```
#include "webpage.h"
#include "wizchip_init.h"
#include <stdlib.h>
#include <string.h>
#include "main.h"
```

Etter at vi har inkludert alle filene blir variablene vi skal bruke definert.

```
#define MAX_HTTPSOCK 4
#define DATA_BUF_SIZE 2048
```

Her definerer vi hvor mange "sockets" vi skal bruke til HTTP-serveren og størrelsen på databuffere. Vi har her valgt 2048 bytes og 4 "sockets". Videre definerer vi nettkonfigurasjonen til modulen vår. Her har vi valgt en tilfeldig MAC-adresse. Skal du ha en egen MAC-adresse må denne kjøpes fra IEEE(Institute of Electrical and Electronics Engineers) som er en organisasjon som har mange standarder som blir brukt innen Ethernet. Vi satte opp et lokalt nettverk på ruteren våres. Her fikk vi tildelt IP-adresse, nettverksmaske(sn), nettverkgateway(gw) og DNS -tjeneren.

Vi har også laget noen RX_BUF som er størrelsen på dataen som skal mottas og TX_BUF er størrelsen på dataen som skal sendes. Socknumlist blir brukt til å nummerere hvilke sockets som er i bruk.

```
iz_NetInfo defaultNetInfo = { .mac = {0x00,0x08,0xdc,0xff,0xee,0xdd},
                              .ip = {192,168,3,101},
                              .sn = {255,255,255,0},
                              .gw = {192,168,3,1},
                              .dns = {8, 8, 8, 8},
                              .dhcp = NETINFO_STATIC};
uint8_t RX_BUF[DATA_BUF_SIZE];
uint8_t TX_BUF[DATA_BUF_SIZE];
uint8_t socknumlist[] = {0, 1, 2, 3};
```

Vi har også en del variabler som skal tas i bruk til styring av ROV-en. Her har vi status av motorene i ROV-en. Styring av lys og motorer. Lesing av strøm, spenning og andre sensorer i ROV-en.

```
// Motor Control mROV //
int8_t motor_vertikal_1 = 0;
int8_t motor_vertikal_2 = 0;
int8_t motor_horisontal_1 = 0;
int8_t motor_horisontal_2 = 0;
//---------------------------//
//Kontroll data mROV //
int8_t mROV = 0;
```

```
int8_t lys = 0;
int8_t bryter_ext = 0;
//-----------------------------------------//

//Status data mROV //
int8_t strom_1 = 0;
int8_t strom_2 = 0;
int8_t spenn_1 = 0;
int8_t spenn_2 = 0;
//-------------------------------//
```

Dette er variablene vi skal ha oversikt over i mikro ROV-en. Etter vi har definert variablene i main.c definerer vi dem også i main.h som globale variabler slik vi kan bruke variablene i alle filene i programmet.

Vi må nå inn i filen wizchip_init.h som er en fil for hvor vi legger til funksjoner for initialisering av w5500. Her definerer vi noen variabler og funksjoner for chip select og lesing og skriving til SPI-grensesnittet.

```
extern SPI_HandleTypeDef hspi1;
#define WIZCHIP_SPI                 hspi1
#define WIZCHIP_CS_PIN              GPIO_PIN_0
#define WIZCHIP_CS_PORT             GPIOB

void WIZCHIPInitialize();

void csEnable(void);
void csDisable(void);
void spiWriteByte(uint8_t tx);
uint8_t spiReadByte(void);
```

Når vi har definert funksjonene i wizchip_init.h kan vi lage funksjonene i wizchip_init.c. Her lager vi funksjoner for lesing og skriving til SPI-bussen.

```
uint8_t i;
void WIZCHIPInitialize(){
    csDisable();
    reg_wizchip_spi_cbfunc(spiReadByte, spiWriteByte);
    reg_wizchip_cs_cbfunc(csEnable, csDisable);
    uint8_t tmp;
    //w5500, w5200
#if _WIZCHIP_ >= W5200
    uint8_t memsize[2][8] = { {2,2,2,2,2,2,2,2},{2,2,2,2,2,2,2,2}};
```

```
#else
    uint8_t memsize[2][4] = { {2,2,2,2},{2,2,2,2}};
#endif
    if(ctlwizchip(CW_INIT_WIZCHIP,(void*)memsize) == -1)
    {
        //myprintf("WIZCHIP Initialized fail.\r\n");
        printf("WIZCHIP Initialized fail.\r\n", 1, 10);
      return;
    }
    /* PHY link status check */
    do {
        if(ctlwizchip(CW_GET_PHYLINK, (void*)&tmp) == -1)
        {
            printf("Unknown PHY Link status.\r\n", 1, 10);
          return;
        }
    } while (tmp == PHY_LINK_OFF);
}

void csEnable(void)
{
    HAL_GPIO_WritePin(WIZCHIP_CS_PORT, WIZCHIP_CS_PIN,
       GPIO_PIN_RESET);
}

void csDisable(void)
{
    HAL_GPIO_WritePin(WIZCHIP_CS_PORT, WIZCHIP_CS_PIN, GPIO_PIN_SET);
}

void spiWriteByte(uint8_t tx)
{
    uint8_t rx;
    HAL_SPI_TransmitReceive(&WIZCHIP_SPI, &tx, &rx, 1, 10);
}

uint8_t spiReadByte(void)
{
    uint8_t rx = 0, tx = 0xFF;
    HAL_SPI_TransmitReceive(&WIZCHIP_SPI, &tx, &rx, 1, 10);
    return rx;
```

```
}
```

I funksjonen WIZCHIPInitialize() blir w5500 initialisert. Denne funksjonen var ferdig og kom med driveren vi lastet ned slik vi ikke trenger å gjøre noen endringer her. Denne funksjonen sjekker hvilken wiznet-brikke vi bruker. Vi bruker W5500. Det blir så valgt en størrelse på minnet som vi skal ha tilgjengelig i brikka. Vi legger også til en variabel kalt "i" som skal bli brukt i en funksjon i while-løkka i main.c filen. csEnable og csDisable er funksjoner for chip select. De er nødvendige i SPI slik vi vet hvilken slave vi skal kommunisere med. spiWriteByte og spiReadByte er funksjoner for lesing og skriving til SPI-grensesnittet.

Videre i main.c filen går vi inn i main() funksjonen. Her legger vi til funksjonen vi har for initialisering av W5500.

```
wizchip_setnetinfo(&defaultNetInfo);
 //Initialisering av W5500 //
WIZCHIPInitialize();
// Setter nettverkskonfigurasjonen vi har definert. //
wizchip_setnetinfo(&defaultNetInfo);
httpServer_init(TX_BUF, RX_BUF, MAX_HTTPSOCK, socknumlist);
 while (1)
  {
      for(i = 0; i < MAX_HTTPSOCK; i++) httpServer_run(i);
      }
```

Her bruker vi også funksjonen httpServer_init som initialiserer en HTTP-serveren. Denne blir lagt til før while-løkka

Inne i while-løkka legger vi til denne kodelinja som gjør at serveren kjører hele tiden. Etter main() funksjonen lager vi to funksjoner.

```
 void print_network_information(void)
{
    wizchip_getnetinfo(&defaultNetInfo);
    printf("Mac address: %02x:%02x:%02x:%02x:%02x:%02x\n\r",
    defaultNetInfo.mac[0],
    defaultNetInfo.mac[1]
    ,defaultNetInfo.mac[2],
    defaultNetInfo.mac[3],
    defaultNetInfo.mac[4],
    defaultNetInfo.mac[5]);
    printf("IP address : %d.%d.%d.%d\n\r",defaultNetInfo.ip[0],
    defaultNetInfo.ip[1],
    defaultNetInfo.ip[2],
```

```
    defaultNetInfo.ip[3]);
    printf("SM Mask     : %d.%d.%d.%d\n\r",defaultNetInfo.sn[0],
    defaultNetInfo.sn[1],
    defaultNetInfo.sn[2],
    defaultNetInfo.sn[3]);
    printf("Gate way    : %d.%d.%d.%d\n\r",defaultNetInfo.gw[0],
    defaultNetInfo.gw[1],
    defaultNetInfo.gw[2],
    defaultNetInfo.gw[3]);
    printf("DNS Server  : %d.%d.%d.%d\n\r",defaultNetInfo.dns[0],
    defaultNetInfo.dns[1],
    defaultNetInfo.dns[2],
    defaultNetInfo.dns[3]);
}

void wep_define_func(void)
{
    // Index page and netinfo
    reg_httpServer_webContent((uint8_t *)"index.html", (uint8_t
       *)index_page);              // index.html        : Main page
       example
    reg_httpServer_webContent((uint8_t *)"netinfo.html", (uint8_t
       *)netinfo_page);          // netinfo.html
    reg_httpServer_webContent((uint8_t *)"netinfo.js", (uint8_t
       *)w5x00web_netinfo_js); // netinfo.js
    reg_httpServer_webContent((uint8_t *)"dio.html", (uint8_t
       *)dio_page);                  // dio.html          :
    reg_httpServer_webContent((uint8_t *)"dio.js", (uint8_t
       *)w5x00web_dio_js);          // dio.js            :
    reg_httpServer_webContent((uint8_t *)"sensor.json", (uint8_t
       *)ROV_data);
    reg_httpServer_webContent((uint8_t *)"sensor.html", (uint8_t
       *)sensor_page);
    // AJAX JavaScript functions
    reg_httpServer_webContent((uint8_t *)"ajax.js", (uint8_t
       *)w5x00web_ajax_js);
```

Den første funksjonen brukes til å printe ut nettverksinformasjonen slik vi kan sjekke om dette stemmer overens med verdiene vi har satt. I den andre funksjonen bruker vi reg_httpserver_-webContent som er en funksjon som er laget av Wiznet. Det er her vi definerer de forskjellige nettsidene vi skal ha. Her har vi en hovedside(index_page) hvor vi har knapper som lar oss gå

inn på de andre sidene. Vi har en nettside for nettverksinformasjon og en side som har oversikt over sensorene. Dette har vi brukt til å teste om variablene endrer seg når vi bruker POST og GET-forespørsler. I ROV-en skal vi bare bruke et brukergrensesnitt som har oversikt over alle variablene.

Vi må lage oppsettet til hver side i HTML og funksjonene til nettsiden i JavaScript. Her kan vi veldig enkelt legge til og fjerne nettsider etter behov. Programmet er enda under utvikling slik at det ferdige produktet inneholder litt flere ting enn det vi går igjennom her, men slik det er nå har vi en HTTP-server hvor vi kan få opp data av variablene vi har og styre lys og diverse fra nettsiden.

For å bruke funksjonene vi har definert her må vi inn i main() funksjonen og kalle på funksjonene. Dette gjøres slik

```
print_network_information();
wep_define_func();
```

Vi har ennå ikke fått tak i noen sensorer som vi faktisk skal bruke i ROV-en. Slik at vi nå ikke kan lese av praktiske verdier, men vi har koblet opp potensiometeret og har fått dette til å fungere. På samme måte som testen i kapittel 3.2.6

Videre må vi ha en funksjon som sender alle variablene ut til nettsiden i JSON-format. Her har vi laget funksjonen make_json_variabler. Denne funksjonen tar inn alle variablene og printer ut til nettsiden.

```
void make_json_variabler(uint8_t * buf, uint16_t * len)
{
    // DHCP: 1 - Static, 2 - DHCP
    *len = sprintf((char *)buf, "sensorCallback({\"mROV\":\"%d\",\
                    \"bryter_ext\":\"%d\",\
                        \"lys\":\"%d\",\
                    \"motor_vertikal_1\":\"%d\",\
                    \"motor_vertikal_2\":\"%d\",\
                    \"motor_horisontal_1\":\"%d\",\
                    \"motor_horisontal_2\":\"%d\",\
                    \"strom_1\":\"%d\",\
                    \"strom_2\":\"%d\",\
                    \"spenn_1\":\"%d\",\
                    \"spenn_2\":\"%d\",\
                    });",
                    mROV, bryter_ext, lys, motor_vertikal_1,
                        motor_vertikal_2, motor_horisontal_1,
                        motor_horisontal_2, strom_1, strom_2, spenn_1
                        ,spenn_2
```

```
                    );
}
```

For å få dataen printet ut legger vi den inn i predefined_get_cgi_processor hvor alle funksjonene for GET blir brukt. Dette gjøres på følgende måte:

```
uint8_t predefined_get_cgi_processor(uint8_t * uri_name, uint8_t *
    buf, uint16_t * len)
{
    uint8_t ret = 1;     // ret = 1 means 'uri_name' matched
    //uint8_t cgibuf[14] = {0, };
    int8_t cgi_dio = -1;
    int8_t cgi_ain = -1;

    if(strcmp((const char *)uri_name, "get_variabler.cgi") == 0)
    {
        make_json_variabler(buf, len);
    }
```

Her er det enkelt å utvide programmet ved bruk av flere "else if" - påstander i funksjonen. Når vi legger til flere if-påstander går programmet gjennom og sjekker om navnet stemmer overens med navnet på JavaScript-funksjonen som vi lager. Her har vi brukt "get_variabler.cgi" som vår Javascript-funksjon. Når programmet sjekker at navnet stemmer overens blir funksjonen inne i if-påstanden kjørt. Nå er GET-metodene fullført og vi kan nå begynne på POST-funksjonene. Det er her vi gjør det mulig å faktisk oppdatere variablene i C via annen programvare. For å bruke POST-metoden har vi først laget en funksjon som tar inn alle variablene vi skal bruke. Denne funksjonen har vi kalt for set_data.

```
int8_t set_data(uint8_t * uri)
{
    int8_t * param;
    //uint8_t val = Variabel; // var 0
        if ((param = get_http_param_value((char *)uri, "mROV" ))) /
        {
            mROV=(int8_t)ATOI(param,10);

        }
        if ((param = get_http_param_value((char *)uri, "lys" )));
        {
            lys=(int8_t)ATOI(param,10);

        }
```

```
if ((param = get_http_param_value((char *)uri,
   "motor_vertikal_1" )))
{
    motor_vertikal_1=(int8_t)ATOI(param,10);

}
if ((param = get_http_param_value((char *)uri,
   "motor_vertikal_2" )))
{
    motor_vertikal_2=(int8_t)ATOI(param,10);
}
if ((param = get_http_param_value((char *)uri, "bryter_ext"
   )))
{
    bryter_ext=(int8_t)ATOI(param,10);
}
if ((param = get_http_param_value((char *)uri,
   "motor_horisontal_1" )))
{
    motor_horisontal_1=(int8_t)ATOI(param,10);
}
if ((param = get_http_param_value((char *)uri,
   "motor_horisontal_2" )))
{
    motor_horisontal_2=(int8_t)ATOI(param,10);
}
if ((param = get_http_param_value((char *)uri, "strom_1" )))
{
    strom_1=(int8_t)ATOI(param,10);
}
if ((param = get_http_param_value((char *)uri, "strom_2" )))
{
    strom_2=(int8_t)ATOI(param,10);
}
if ((param = get_http_param_value((char *)uri, "spenn_1" )))
{
    spenn_1=(int8_t)ATOI(param,10);
}
if ((param = get_http_param_value((char *)uri, "spenn_2" )))
{
    spenn_2=(int8_t)ATOI(param,10);
```

```
        }
        return 0 ;
    };
```

Her har vi nå en funksjon som inneholder alle variablene som skal kunne endres og leses av. Denne funksjonen blir så lagt inn i predefined_se_cgi_processor som er funksjonen som gjør at vi kan POSTE. Dette gjøres slik:

```
uint8_t predefined_set_cgi_processor(uint8_t * uri_name, uint8_t *
    uri, uint8_t * buf, uint16_t * len)
{
    uint8_t ret = 1;     // ret = '1' means 'uri_name' matched
    if(strcmp((const char *)uri_name, "set_data.cgi") == 0)
    {
        val = set_data(uri);
        *len = sprintf((char *)buf, "%d", val);

    }
    else
    {
        ret = 0;
    }
    return ret;
}
```

Her også er det veldig enkelt å legge til flere funksjoner ved bruk av "else if" -påstander under if-løkka. Dette kan være diverse funksjoner til skriving og lesing til mikrokontrolleren. Her kan vi også legge til funksjoner for å styre lysdioder og andre ting på mikrokontrolleren. Funksjonene blir også definert helt øverst i filen.

```
int8_t set_data(uint8_t * uri);
```

ATOI-funksjonen som blir brukt i set_data er en funksjon som gjør om bokstaver og tegn til tallverdier. Dette er en funksjon som allerede finnes i C, men den fungerte dårlig ved POSTing av negative tall. ATOI gjør om hvert tegn i en streng til den ASCII(American Standard Code for Information Interchange) verdien den har. ASCII er en standard for å sende tekst mellom datamaskiner. Der alle bokstaver, tall og tegn har sin egen tallverdi fra 0 til 127. Et minustegn har verdien 45 i denne tabellen slik at når vi POSTet et negativt tall fikk vi ut 45 pluss den tallverdien vi egentlig ville ha. Vi har derfor endret funksjonen slik at hvis det første tegnet i meldingen er et minustegn skal ATOI-funksjonen hoppe over dette tegnet. Dette tallet blir så ganget med -1 slik at det blir negativt.

```
int16_t ATOI(
    int8_t * str,    /**< is a pointer to convert */
    int8_t base /**< is a base value (must be in the range 2 - 16) */
    )
{
        unsigned int num = 0;
        signed int minus = 1;
        if  (*str == '-') {
            minus= -1;
            *str++;  }
        while ((*str !=0) && (*str != 0x20)) // not include the
           space(0x020)
                num = num * base + C2D(*str++);
    return num*minus;
}
```

Etter endring av funksjonen kan vi bruke negative tall uten problem. Når funksjonene er ferdig laget kan vi begynne å lage JavaScript og HTML-koden. Den første vi gjør er JavaScript-koden. Her har vi laget en funksjon som tar inn verdiene fra alle variablene.

```
#define ROV_data        "function sensorCallback(o){"\
            "$('txtmROV').value=o.mROV;"\
            "$('txtstrom_1').value=o.strom_  "\
            "$('txtstrom_2').value=o.strom_2;"\
            "$('txtspenn_1').value=o.spenn_1;"\
            "$('txtspenn_2').value=o.spenn_2;"\
            "$('txtmotor_horisontal_1').value=o.motor_horisontal_1;"\
            "$('txtmotor_horisontal_2').value=o.motor_horisontal_2;"\
            "$('txtbryter_ext').value=o.bryter_ext;"\
            "$('txtlys').value=o.lys;"\
            "$('txtmotor_vertikal_1').value=o.motor_vertikal_1;"\
            "$('txtmotor_vertikal_2').value=o.motor_vertikal_2;"\
            "}"\
            "function getsensordata(){"\
            "var oUpdate;"\
            "setTimeout(function(){"\
                "oUpdate=new
                   AJAX('get_variabler.cgi',function(t){"\
                   "try{eval(t);}catch(e){alert(e);}"\
                              "});"\
                  "oUpdate.doGet();},300);"\
                  "setTimeout('getsensordata()',500);"\
```

```
                    "}"
```

Her har vi en JavaScript-funksjon hvor vi definerer alle variablene i Mikro ROV-en og en funksjon som oppdaterer variablene i et gitt tidsintervall, her 500ms. Den andre JavaScript-kodesnutten vi trenger er en funksjon som setter variablene til den verdien vi sender fra brukergrensesnittet. Her har vi tatt koden som allerede er laget i driveren vi lastet ned og modifisert den slik at passer til våre variabler.

```
#define set_data1   "function setXbox(o){"\
                    "var p=o.attributes['pin'].value;"\
                    "/*var v=$('txtdio_s'+p).value;*/"\
                    "var v=o.attributes['s'].value;"\
                    "dout=new AJAX('set_data.cgi',
                       function(t){try{eval(t);}catch(e){alert(e);}});"\
```

Når dette er klart kan vi begynne å lage selve nettsiden hvor alt informasjonen vises. Dette må gjøres i HTML-kode i webpage.h filen. Her lager vi bare en helt standard side hvor vi får alle variablene på linje og et felt hvor verdien til variablene står. Her starter vi med å velge dokumenttype som her er HTML. Videre har vi laget en overskrift som viser hvilken side vi er inne på. Vi har så lagt til et felt for hver variabel og et felt hvor verdien til variablene står.

```
#define sensor_page
"<!DOCTYPE html>"\
    "<html>"\
        "<head>"\
        "<title>ROV motor info</title>"\
        "<meta http-equiv='Content-Type'content='text/html;
           charset=utf-8'>"\
        "<style>"\
        "label{float:left;text-align:left;width:70px;}"\
        "li {list-style:none;}"\
        "</style>"\
        "<script type='text/javascript' src='ajax.js'></script>"\
   "<script type='text/javascript' src='sensor.json'></script>"\
        "</head>"\
        "<body onload='getsensordata();'>"\
        "<div>"\
        "ROV motor info"\
        "</div>"\
        "<br>"\
        "<ul>"\
        "<li><label for='txtmROV'>mROV:</label><input id='txtmROV'
```

```
            name='mROV' type='text' size='20'
            disabled='disabled'/></li> "\
        "<li><label for='txtstrom_1'>Strom_1:</label><input
            id='txtstrom_1' name='Strom_1' type='text' size='20'
            disabled='disabled'/></li> "\
        "<li><label for='txtstrom_2'>Strom_2:</label><input
            id='txtstrom_2' name='Strom_2' type='text' size='20'
            disabled='disabled'/></li> "\
        "<li><label for='txtspenn_1'>Spenn_1:</label><input
            id='txtspenn_1' name='Spenn_2' type='text' size='20'
            disabled='disabled'/></li> "\
        "<li><label for='txtspenn_2'>Spenn_2:</label><input
            id='txtspenn_2' name='Spenn_2' type='text' size='20'
            disabled='disabled'/></li> "\
             "</ul>"\
             "</body>"\
    "</html>"
```

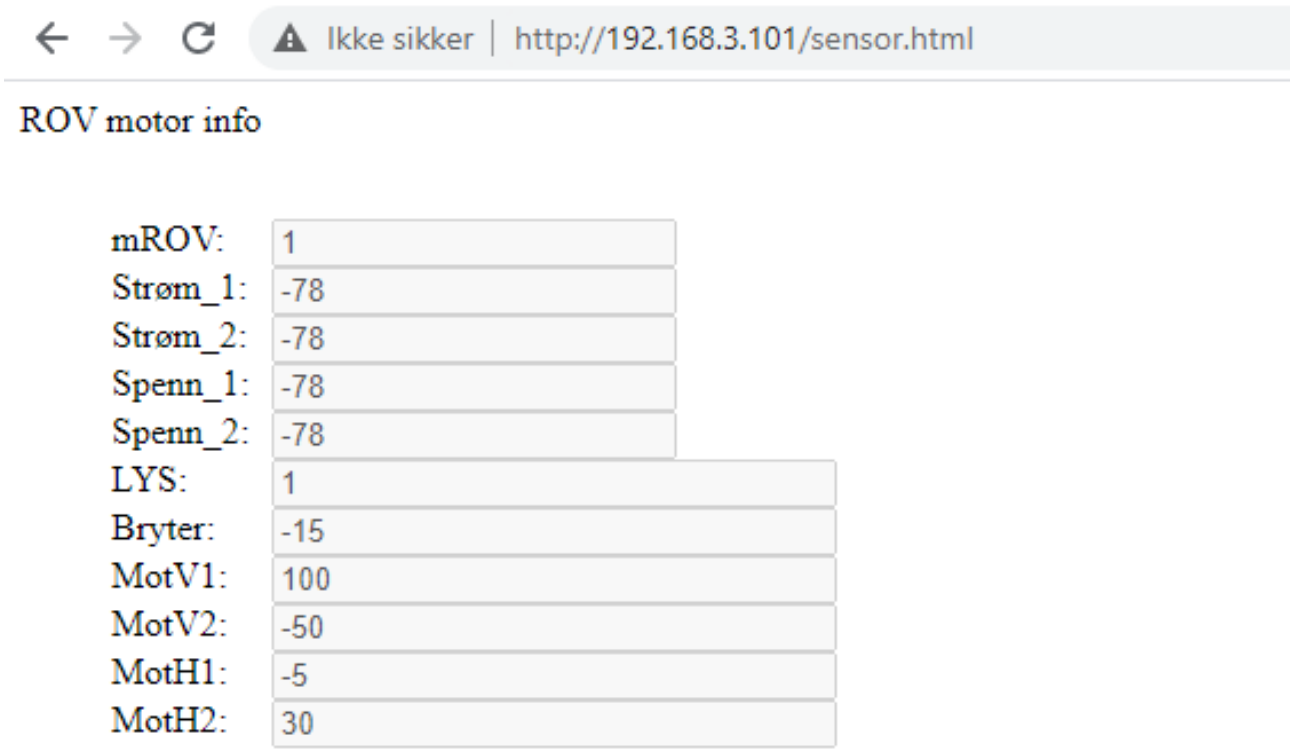
Her er et utsnitt av koden for nettsiden. Her blir ikke alle variablene vist siden koden blir veldig lang, men for flere variabler er det bare å legge inn samme kodelinjer for variablene som ønskes å ha på siden. Når vi nå kjører dette programmet kan vi søke på IP-adressen i nettleseren. Vi får da opp en nettside. Alle variablene er satt til null, men ved bruk av Python klarer vi og endre verdiene.

I figur 6.4 ser vi hvordan Python-koden ser ut. Når vi kjører denne etter HTTP-serveren er startet ser vi at variablene endres. Her har vi nå bare satt inn tilfeldige verdier selv, men dette skal bli oppdatert automatisk ved bruk av XBOX-kontrollen.

```python
import requests
import time

url2 = 'http://192.168.3.101/get_netinfo.cgi'
url3 = 'http://192.168.3.101/set_data.cgi'

mini_rov_kontroll = {'motor_vertikal_1' : '100',
                     'motor_vertikal_2' : '-50',
                     'motor_horisontal_1' : '-50',
                     'motor_horisontal_2' : '30',
                     'mROV' : '1',
                     'lys' : '1',
                     'bryter_ext' : '0'}

r1 = requests.get(url2)
r = requests.post(url3, data=mini_rov_kontroll)
print(r.text)
print(r1.text)
```

Figur 6.4: Python kode for å endre variabler

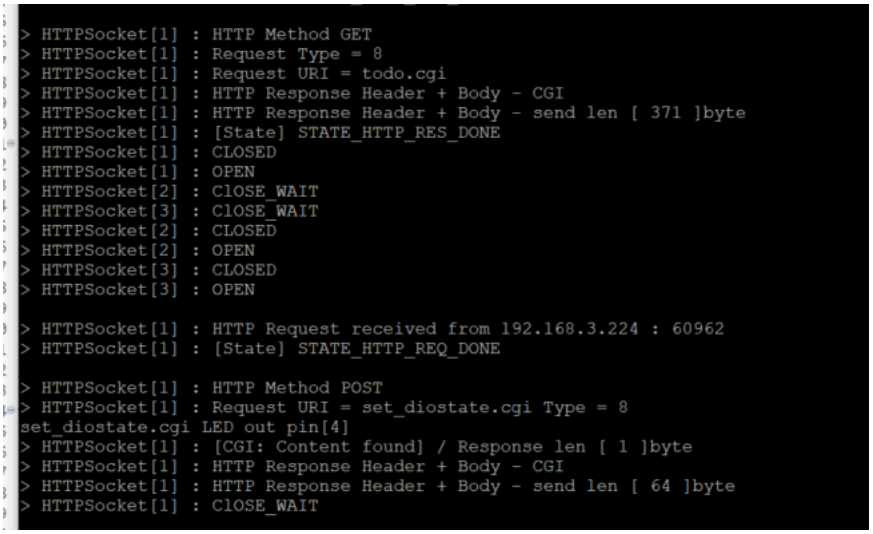
I figur 6.5 ser vi hvordan nettsiden ser ut etter vi har endret noen av variablene. Her kan vi enkelt lese av alle variablene på datamaskinen på overflaten. Vi har nå bare satt inn tilfeldige verdier, men i det ferdige produktet skal motorene ha en verdi mellom -100 og 100 som er pådraget. Lys er enten 0 eller 1 som betyr av eller på, samme med mROV og Bryteren. Strøm og spenningsmåling skal bli styrt av sensorene når vi skal kjøre programmet sammen med mikro ROV-gruppa.



Figur 6.5: Nettsiden for variablene i Mikro ROV-en

I figur 6.6 ser vi et skjermbilde fra terminalvinduet på datamaskinen. Her ser vi meldingene vi får når vi sender en POST eller en GET-forespørsel.

I figur 6.7 ser vi et utdrag fra wireshark. Her ser vi pakkene som blir sendt mellom wiznet-brikka og datamaskinen. Vi får her en HTTP- kode på 200 som betyr at alt er i orden.



Figur 6.6: POST og GET forespørsel

192.168.3.224	192.168.3.101	TCP	54 49893 → 80 [ACK]
192.168.3.224	192.168.3.101	HTTP	416 GET /netinfo.js
192.168.3.101	192.168.3.224	TCP	132 80 → 49888 [PSH,
192.168.3.101	192.168.3.224	HTTP	822 HTTP/1.1 200 OK
192.168.3.224	192.168.3.101	TCP	54 49888 → 80 [ACK]

Figur 6.7: Utdrag fra wireshark

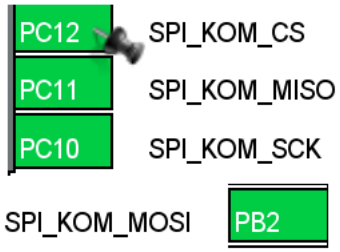
Nå har vi altså et program hvor vi kan lese av alle verdiene vi trenger og kan gi dem den verdien vi ønsker. Det som gjenstår er å samkjøre med mikro ROV- gruppa og få det til å fungere i lag med deres kode.

I neste kapittel skal vi gå gjennom hvordan vi konfigurerer programmet på mikrokontrolleren til hoved ROV-en.

Kapittel 7

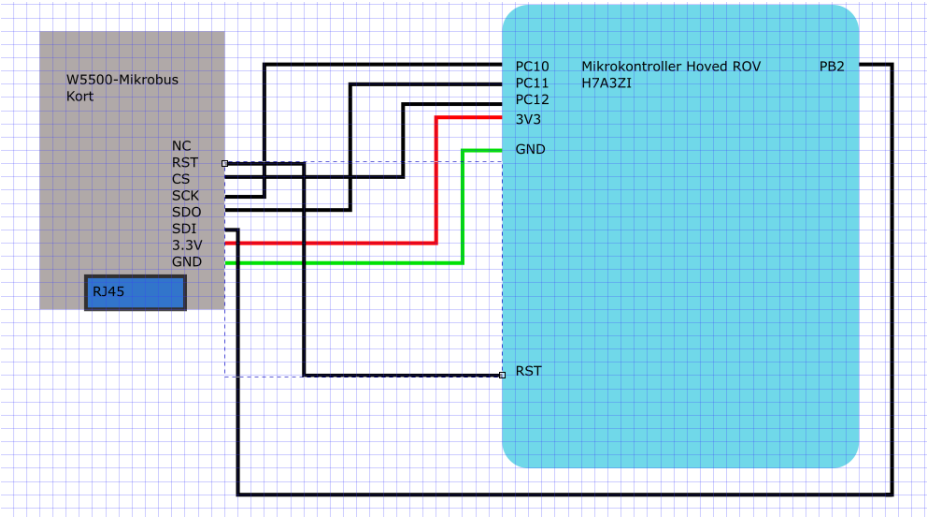
Konfigurasjon av Hoved-ROV

Nå skal vi gå gjennom konfigurasjonen av hoved ROV-en. Det meste her er likt som i mikro ROV-en. Hovedforskjellen er pinnene som er i bruk og variablene som skal leses av. Vi starter her på samme måte og velger hvilke pinner som skal brukes. Her bruker vi SPI3 til vår SPI-buss. Her blir pinne PC11, PC10 og PB2 satt av til SPI grensesnittet. Vi velger også pinne PC12 som en GPIO output som vi bruker som chip select. Figur 7.1 viser hvordan dette skal se ut.



Figur 7.1: Pinner vi bruker til SPI på mikrokontrolleren H7A3ZI

Etter det er gjort kan vi koble opp ledningene mellom W5500 og mikrokontrolleren. I figur 7.2 vises et koblingsskjema over oppkoblingen. Dette og er gjort på samme måte som vist tidligere.



Figur 7.2: Koblingsskjema av SPI modul i hoved ROV

Etter at alt er koblet opp riktig kan vi legge inn koden slik vi gjorde i kapittel 6 om mikro-ROV. Her brukes alle de samme filene, bare med litt endringer. Her må vi endre IP-adresse slik at ikke begge serverene kjører på samme IP. Vi må også bytte til en annen MAC-adresse slik at de blir konfigurert som en egen enhet på nettverket.

```
wiz_NetInfo defaultNetInfo = { .mac = {0x00,0x08,0xdc,0xea,0xee,0xdd},
                               .ip = {192,168,3,102},
                               .sn = {255,255,255,0},
                               .gw = {192,168,3,1},
                               .dns = {8, 8, 8, 8},
                               .dhcp = NETINFO_STATIC};
```

Variablene vi skal ha oversikt over i hoved ROV-en er vist i kodesnutten under. I hoved ROV-en er det en del ting vi må ha kontroll over. Her har vi en del flere sensorer og motorer som må kunne leses av og styres.

```
int8_t manuel = 1;
int8_t start = 1;
int8_t teller = 102;
int8_t tid = 102;
int8_t dybde = 102;
int8_t temp = 21;
int8_t lekasje = 0;
int8_t hoyde=0;
int8_t temp_vann = 0;
int8_t lys = 0;
int8_t mROV = 0;
int8_t manip_paa =0;
int8_t aks_x = 121;
int8_t aks_y = 122;
int8_t aks_z = 100;
int8_t vinkel_x = 121;
int8_t vinkel_y = 111;
int8_t vinkel_z = 100;
int8_t strom_1 =0;
int8_t strom_2 =0;
int8_t strom_3 =0;
int8_t spenn_1 = 0;
int8_t spenn_2 = 0;
int8_t spenn_3 = 0;
int8_t hhf =0;
int8_t hvf =0;
int8_t hvb= 0;
int8_t hhb = 0;
int8_t vvf = 0;
int8_t vhf =0;
int8_t vvb = 0;
int8_t vhb = 0;
int8_t  manip1 =0;
int8_t  manip2 =0;
int8_t  manip3 =0;
int8_t  manip4 =0;
int8_t  depth_regulator = 0;
int8_t  stamp_regulator = 1;
```

```
int8_t  rull_regulator = 1;
int8_t  KP_depth= 0.125;
int8_t  KP_stamp = 0.99;
int8_t  KP_rull = 1.44;
int8_t  KI_depth =0.25;
int8_t  KI_stamp =0.56;
int8_t  KI_rull =4.22;
int8_t  KD_depth =0.5;
int8_t  KD_stamp =0.24;
int8_t KD_rull =5.11;
int8_t bryter_ext = 0;
int8_t venstre_x = 14;
int8_t venstre_y = 14;
int8_t hoyre_x = 14;
int8_t hoyre_y = 14;
int8_t skalering = 14;
```

Når alle variablene har blitt definert og funksjonen til variabelen er ferdig kan vi kjøre programmet. Vi skal i neste kapittel gjennomgå en test opp mot programmet vårt.

7.1 Test med HTTP-server mot Python

For å teste programmet har vi laget et testskript i Python. Python programmet sender ut forespørsler til HTTP-tjeneren som er på mikrokontrolleren. Vi skal her prøve å endre verdien til noen variabler. Python programmet lagrer tiden det tar mellom hver test i en liste og printer dette ut. Her kan vi se hvor lang tid vi bruker mellom hver forespørsel til mikrokontrolleren. Vi skriver og ut et gjennomsnitt av tiden på alle forespørslene som blir sendt.

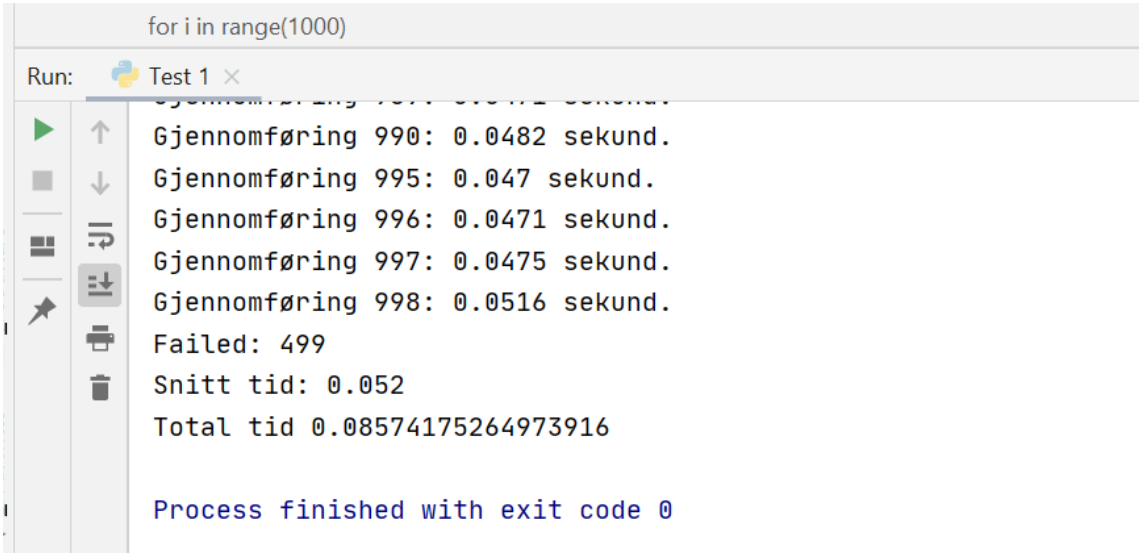
Koden under viser hvordan Python programmet ser ut. Her har vi en liste over noen variabler som vi skal endre i mikrokontrolleren. Vi har så en funksjon som lagrer tiden til alle forespørslene som blir sendt. Vi har og en for-løkke som sender 1000 forespørsler til IP-adressen. Verdiene her blir printet ut i konsoll vinduet når vi kjører programmet. Her kan vi lese av tiden mellom hver forespørsel.

```
import requests
import time
start_time = time.time()
url3 = 'http://192.168.3.102/set_xbox.cgi'
```

```
Liste = { 'venstre_y':'90' ,'venstre_x':'98' ,
'hoyre_x':'97','hoyre_y':'96','manip1':'95'
,'manip2':'94','manip3':'93',
 'manip4':'92','skalering':'91'
 ,'manuel':'90','mROV':'89',
 'manip_paa':'88','depth_regulator':'87'
 ,'stamp_regulator':'86','rull_regulator':'85',
 'lys':'84','KP_depth':'60','KP_stamp':'59'
 ,'KP_rull':'58','KI_depth':'57',
 'KI_stamp':'56','KI_rull':'55'
 ,'KD_depth':'54','KD_stamp':'53','KD_rull':'52'}

def tidstest(data,url ):    # Funskjon som tar tiden til hver GET
   request, foruten de som faar feil
try:
    t1 = time.perf_counter() # Tidsur
    r = requests.get(url3)   # Get request
    print(r.text)
    tidsbruk =  time.perf_counter() - t1
    # tiden paa hver get request
    return tidsbruk
    except:
    return 1111

snitt = []   # Liste med snitttid
Feiltid = []   # Liste med Feiltid
failed = 0
for i in range(1000):  #  Funskjon som sender 1000 GET requester til
#Mikrokontrolleren, samt legger sammen tiden
    n = round(tidstest(Liste, url3), 4)

    if n != 1111:
        snitt.append(n)

    else:
        failed += 1

# Rekner ut snitt tiden paa feilkode og snitt tid paa GET requester.
Totaltid = time.time() - start_time
tid = round(sum(snitt)/len(snitt),3)
Fungerendetid = sum(snitt)
```

```
Failtid = Totaltid - Fungerendetid
FailtidSnitt = Failtid / failed
Total = FailtidSnitt + tid
print(f"Failed: {failed}")
print(f"Snitt tid: {tid}")
print('Total tid' , Total)
```

Når vi kjører programmet og sender 1000 forespørsler til mikrokontrolleren fikk vi en gjennomsnittstid på 0.085 sekund per forespørsel. Dette inkluderer også forespørslene som gir feilkode. Vi ser ut ifra resultatet i figur 7.3 at vi fikk 499 feilkoder av 1000 forespørsler. Vi har ikke fått til å fikse dette problemet, men vi har konkludert med at en tid på 0.085 sekund per forespørsler er nok til å gi bra nok kjøring av ROV-en. Vi skal fortsette å prøve og optimalisere koden slik vi ikke får like mange forespørsler med feilkode.



Figur 7.3: Resultat fra test

Vi har gjort en del feilsøking for å finne ut hva som gjør at vi får så mange feilkoder, men ikke fått til å fikset dette. Grunnen til at vi får disse feilkodene er mest sannsynlig at vi bruker en HTTP-server som oppretter en tilkobling hver gang vi sender en forespørsel. Dette kan gjøre at HTTP-tjeneren ikke får fullført forespørselen før neste forespørsel blir sendt. En HTTP-tjener må svare på alle forespørslene som blir sendt og dette ender opp med feilkoder når tjeneren ikke er klar til å motta data.

Kapittel 8

Markedsføring

Vi har også hatt ansvar for markedsføringdelen av Uis Subsea. Her har det meste gått i å oppdatere hjemmesiden og facebook-siden med nye sponsoravtaler og andre nyheter som har kommett. Vi har også tatt i bruk Instagram og LinkedIn for å prøve å få flere folk interesserte i UiS Subsea. Vi har også laget en presentasjon som vi fremførte på UiS sin Åpen dag. Vi skulle egentlig stå på stand på Uis, men på grunn av koronasituasjonen som pågår nå måtte vi presentere Uis Subsea over Zoom. Dette var en grei måte å fremføre på, men vi fikk ikke snakket med folk som hørte på presentasjonen. I tillegg til dette har vi brukt litt tid på å fikse klær til Uis Subsea. Her har vi fått laget gensere, t-skjorter og caps hvor alle sponsorene har fått sin egen logo.

Vi har designet klær for Uis Subsea gjennom nettsiden Spreadshirt. Vi laget gensere, t-skjorter, munnbind og capser. Vi brukte en nettside som heter https://www.remove.bg/ til å fjerne bakgrunnen på bildene fra sponsorene slik at det ville se bedre ut. Klærne er vist i figur 8.1-8.4

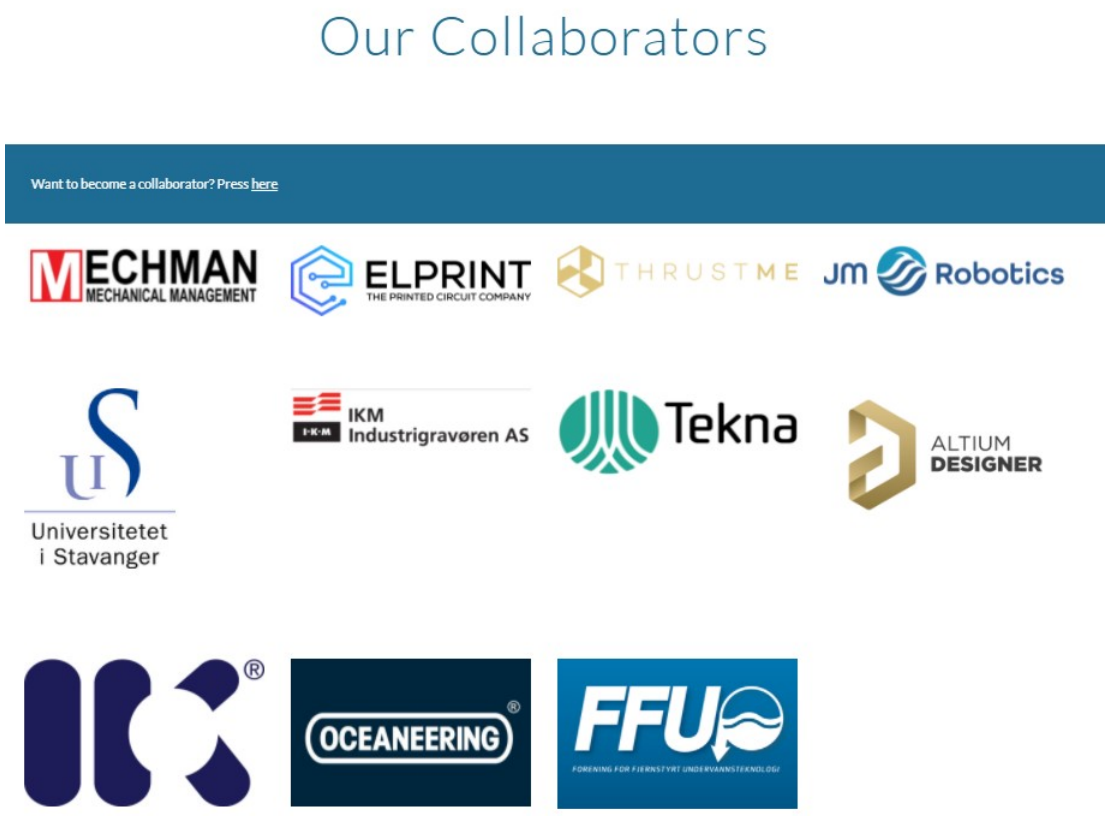
Figur 8.1: Caps

Figur 8.2: T-skjorte

Figur 8.3: Genser

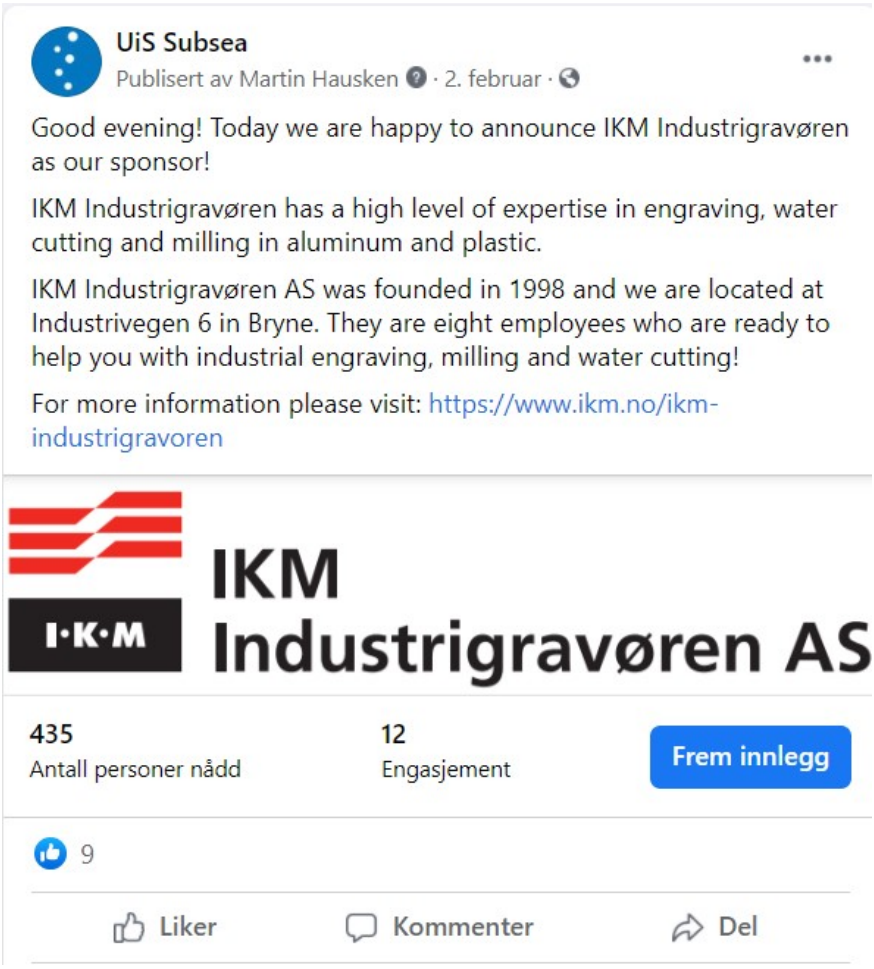
Figur 8.4: Munnbind

Vi har hatt en avtale med sponsorene våre at vi skal markedsføre dem ved å ha logoen på klærne våre, nettsiden og klistermerke på ROV-en. Vi har på nettsiden lagt en egen side for sponsorer, med bilde og link til hjemmesiden ved å trykke på bilde. som vist i figur 8.5.



Figur 8.5: Nettside

På Facebook har vi vært aktive med å presentere de nye bedriftene som har valgt og sponse oss. Et eksmepel fra IKM er vist i figur 8.6.



Figur 8.6: Facebook innlegg

Kapittel 9

Konklusjon og Diskusjon

9.1 Kamera

Kameraene vi valgte i ROV-en fungerer veldig bra. Vi klarer å sende videostrømmen opp til datamaskinen over nettverket. Dette fungerer uten noe særlig etterslep og har en god bildekvalitet. Delene til kamerahuset vi har laget passer bra sammen og maskiningeniørstudentene har 3D-printet fester til kamerahuset for oss. Det som gjenstår her, er å feste kamerahuset til rammen på ROV-en og legge kabler opp til svitsjen.

Arbeidet med å finne kamera, lage kamerahus og testingen har gått veldig bra. Her møtte vi på få utfordringer.

9.2 Kommunikasjonssystem

Det har vært litt usikkerheter rundt løsningen på fiber. Vi har fått tak i fiberkabel og fiberkontakter som vi egentlig skulle bruke, men MacArtney har valgt å sponse oss med dette. Vi har ikke fått mulighet til å teste fiberkommunikasjonen før bachelor-innlevering på grunn av problemer med MacArtney og sponsor av fiber-kontakter. MacArtney skulle levere fiber-kontakter og fiberkabel, men dette har vi ennå ikke fått tak i. Vi må da gå for vår opprinnelige løsning med kabelen fra FOSS fiber og fiberkontaktene fra Farnell hvis ikke utstyret kommer i nærmeste fremtid.

Kommunikasjonen slik den er nå fungerer som den skal. Vi har mulighet til å både sende og motta data fra datamaskinen på overflaten. Det har vært ganske mye prøving og feiling med SPI/Ethernet- modulen. Både SPI og Ethernet var nye tema for oss slik at vi har brukt veldig mye tid på å lære oss mer om dette. Vi fikk litt kunnskaper om dette etter hvert som vi begynte å programmere på mikrokontrolleren. Ut ifra testene vi har gjort i kapittel 7.1 ser vi at vi kan

sende GET og POST-forespørsler til ROV-en med en hastighet på rundt 10Hz. Dette skal være raskt nok til å kjøre uten noe særlig problemer. Noe som kan bli et problem er at halvparten av forespørslene gir en feilkode. Vi har ikke klart å finne ut hva som gjør at det blir feil, men dette vil bli jobbet mer med frem mot konkurransen.

Arbeidet med kommunikasjonen har vært svært krevende. Dette er fordi vi hadde lite kunnskap om dette fra før, i tillegg til at dette ikke har blir gjort i Uis Subsea før. Hvis vi skulle gjort dette på nytt hadde vi nok gått for et CAN-bus system, som har blitt brukt før. Da kunne vi også fått en del hjelp fra tidligere bachelor-oppgaver.

9.3 Belysning

Belysningen funker slik den skal og gir et veldig bra lys. Det som gjenstår her er å feste lysene til ROV-en og legge kabler inn til strømforsyningen. Vi har ikke fått til noen god måte å dimme lyset på per dags dato, men dette skal vi prøve å løse før konkurransen.

Videre skal vi jobbe med å optimalisere programmet for kommunikasjon slik vi får en bedre kode som igjen gir oss bedre kjøreegenskaper. Vi må også koble ferdig LED-diodene på ROV-en og feste kamerahuset.

Kilder

[1] TCP/IP Model: Layers and Protocol — What is TCP IP Stack?, https://www.guru99.com/tcp-ip-model.html [Accessed 20 Jan. 2021]

[2] Customer 1st. Communications. (2017). Customer 1st Communications. [online] Available at: https://www.c1c.net/blog/analog-vs-digital-security-cameras-cctv/ [Accessed 10 Jan. 2021]

[3] Grandado kamera https://cutt.ly/KbZbdqL [Accessed 30 Jan. 2021]

[4] Pngwing.com. (2021). Inter frame H.264/MPEG-4 AVC Intra-frame Video codec, others, angle, white, text png — PNGWing. [online] Available at: https://www.pngwing.com/en/free-png-tcpdc [Accessed 9 May 2021].

[5] Bidragsytere til Wikimedia-prosjektene. "H.264." Wikipedia.org, Wikimedia Foundation, Inc., 8 June 2007, https://no.wikipedia.org/wiki/H.264 Accessed 10 Jan. 2021.

[6] The Broadcast Bridge. "H.264 Versus HEVC: Understanding the Differences." Thebroadcastbridge.com, The Broadcast Bridge, 28 Nov. 2017, www.thebroadcastbridge.com/content/entry/10029/h.264-versus-hevc-understanding-the-differences. [Accessed 10 Jan. 2021.]

[7] —. "Lysleder Laget Av Glass Eller Plast." Wikipedia.org, Wikimedia Foundation, Inc., 16 Aug. 2006,https://no.wikipedia.org/wiki/Fiberoptikk [Accessed 15 Jan. 2021.]

[8] "Single Mode vs Multimode Fiber: What's the Difference?" Blog, 18 Jan. 2021, https://community.fs.com/blog/single-mode-cabling-cost-vs-multimode-cabling-cost.html [Accessed 18 jan. 2021]

[9] Wikipedia Contributors. "CAN Bus." Wikipedia, Wikimedia Foundation, 15 Jan. 2021, https://en.wikipedia.org/wiki/CAN_bus. Accessed 18 Jan. 2021.

[10] Carritech (2017). Fiber Optics Explained. YouTube. Available at: https://www.youtube.com/watch?v=F7H0KJP6_is [Accessed 14 Apr. 2021].

[11] Wizwiki.net. (2017). products:w5500:application:tcp_function [Document Wiki]. [online] Available at: https://wizwiki.net/wiki/doku.php/products:w5500:application:tcp_function [Accessed 15 Apr. 2021].

[12] Wikipedia Contributors (2021). Serial Peripheral Interface. [online] Wikipedia. Available https://en.wikipedia.org/wiki/Serial_Peripheral_Interface [Accessed 3 May 2021].

[13] https://www.facebook.com/profile.php?id=100008204471322 (2020). HTTP vs TCP : Detailed Comparison» Network Interview. [online] Network Interview. Available at: https://networkinterview.com/http-vs-tcp-know-the-difference/ [Accessed 29 Apr. 2021].

[14] PLANET - Switch 5-p + 1-p1000X Gigabit SFP 5x10/100/1000Mbps Gigabit, https://deal.no/planet-switch-5-p-1-p1000x-gigabit/cat-p/c0/p9898072?fbclid=IwAR0H5-imspxGpBimyYipspQyM5hfZLVgMcecmzbPxigMfKpunVct-d--X8. [Accessed 29 Apr. 2021]

[15] MICROSENS - 6-Port GbE Medical Micro Switch G6 https://www.microsens.com/product/6-port-gbe-medical-micro-switch-g6 [Accessed 18 Jan. 2021]

[16] "CAN Example - ATmega32M1 - STK600 - Microcontroller - Eewiki." Digikey.com, 2016, www.digikey.com/eewiki/display/microcontroller/CAN+Example+-+ATmega32M1+-+STK600[Accessed 18 Jan. 2021.]

[17] 5x10/100/1000 + 1x100/1000 SFP, DIN, -10°C, 20-60VDC, Lite Foss AS https://www.fossfiberoptikk.no/5x10-100-1000-1x100-1000-sfp-din-10-c-lite-managed [Accessed 20 Jan. 2021.]

[18] "Underwater Light for ROVs, AUVs, and Other Subsea Applications." Blue Robotics, 21 Jan. 2021, https://bluerobotics.com/store/thrusters/lights/lumen-r2-rp/ Accessed 21 Jan. 2021.

[19] "BXRE-27G0800-D-73 Bridgelux — Optoelectronics — DigiKey." Digikey.no, 2021, www.digikey.no/product-detail/en/bridgelux/BXRE-27G0800-D-73/976-1712-ND/7491533?Accessed21Jan.2021.

[20] Farnell.com. (2020). Fiber Optic Connector, Duplex, LC, Singlemode, 9µm / 125µm, Nylon (Polyamide) Body. [online] Available at: https://no.farnell.com/bulgin/pxf6050c/fibre-conn-lc-duplex-sm-9-125um/dp/3265665 [Accessed 9 May 2021].

[21] "Kelvin (K) - LysFramtid." Lysframtid.com, 2021,www.lysframtid.com/kelvin-k Accessed 21 Jan. 2021.

[22] "RCD-48-0.70/W RECOM Power CC Buck LED Driver - 39W 700mA." BallastShop, 2021, https://ballastshop.com/rcd-48-0-70-w-recom-power-cc-buck-led-driver-39w-700ma/?fbclid=

IwAR0ouTizhMc6o34WRDXY7ClTptDneJ8VPkmAwq66-7PetanYrMi-8IESpG4#topOfPage
Accessed 21 Jan. 2021.

[23] Analog.com. (2021). Isolating SPI for High Bandwidth Sensors. [online] Available at: https://www.analog.com/en/technical-articles/isolating-spi-for-high-bandwidth-sensors.html [Accessed 3 May 2021].

[24] (ENC28J60 Ethernet Controller Features • IEEE 802.3TM Compatible Ethernet Controller • Fully Compatible with 10/100/1000Base-T Networks • Integrated MAC and 10Base-T PHY • Supports One 10Base-T Port with Automatic Polarity Detection and Correction • Supports Full and Half-Duplex Modes • Programmable Automatic Retransmit on Collision • Programmable Padding and CRC Generation • Programmable Automatic Rejection of Erroneous Packets • SPI Interface with Clock Speeds up to 20 MHz Buffer • 8-Kbyte Transmit/Receive Packet Dual Port SRAM • Configurable Transmit/Receive Buffer Size • Hardware Managed Circular Receive FIFO • Byte-Wide Random and Sequential Access with Auto-Increment • Internal DMA for Fast Data Movement • Hardware Assisted Checksum Calculation for Various Network Protocols) https://no.mouser.com/datasheet/2/272/eth-click-manual-v100-1487212.pdf

[Accessed 13 Apr. 2021]

[25] "MikroElektronika ETH Click ENC28J60 Ethernet Evaluation Kit MIKROE-971." Rs-Online.com, 2020,https://no.rs-online.com/web/p/communication-wireless-development-tools/7916432/?cm_mmc=NO-PLA-DS3A-_-google-_-CSS_NO_NO_Raspberry_Pi_ [Accessed 12 Apr. 2021]

[26] (W5500 Datasheet) http://wizwiki.net/wiki/lib/exe/fetch.php/products:w5500:w5500_ds_v109e.pdf [Accessed 1 Apr. 2021]

[27] Router-switch.com. (2017). (Update 2020) Ethernet Cable or Fiber Optic Cable? Ethernet vs. Fiber – Router Switch Blog. [online] Available at: https://blog.router-switch.com/2017/12/ethernet-cable-or-fiber-optic-cable-ethernet-vs-fiber/ [Accessed 14 Apr. 2021].

[28] Switch https://no.wikipedia.org/wiki/Switch Accessed 6 Feb. 2021.

[29] Wikipedia Contributors. "Small Form-Factor Pluggable Transceiver." Wikipedia, Wikimedia Foundation, 3 Feb. 2021, https://en.wikipedia.org/wiki/Small_form-factor_pluggable_transceiver Accessed 3 Feb. 2021.

[30] Wizwiki.net. (2013). products:w5500:driver [Document Wiki]. [online] Available at: http://wizwiki.net/wiki/doku.php/products:w5500:driver [Accessed 11 Feb. 2021].

[31] Cplusplus.com. (2020). (stdio.h) - C++ Reference. [online] Available at: http://www.cplusplus.com/reference/cstdio/[Accessed 11 Feb. 2021].

https://newsandviews.dataton.com/codecs_revisited_what_are_h264_and_h265 [Accessed 15 Feb. 2021]

[32] Elfa Distrelec Norge. (2015). DC/DC-omformer 18W 600mA. [online] Available at: https://cutt.ly/kbZbeBz [Accessed 19 Apr. 2021].

[33] Elfadistrelec.no. (2021). [online] Available at: https://www.elfadistrelec.no/Web/Downloads/_t/ds/ldbl_eng_tds.pdf [Accessed 19 Apr. 2021].

[34] Blue Robotics. (2021). M10 Cable Penetrator for 6mm Cable. [online] Available at: https://bluerobotics.com/store/cables-connectors/penetrators/penetrator-10-25-a/ [Accessed 19 Apr. 2021].

[35] Keil.com. (2020). TFTP Server. [online] Available at: https://www.keil.com/pack/doc/mw/Network/html/group__net_t_f_t_ps___func.html [Accessed 20 Apr. 2021].

[36] Noviello, Carmine. "Adding Ethernet Connectivity to a STM32-Nucleo." Carmine Noviello - a Blog about Programming and Electronics, Carmine Noviello - A blog about programming and electronics, 28 Aug. 2015, www.carminenoviello.com/2015/08/28/adding-ethernet-connectivity-stm32-nucleo/ [Accessed 17 Feb. 2021.]

[37] MATE logo,https://www.facebook.com/materovcompetition/photos/1250385541654400 - [Accessed: 06.01.21"]

[38] MATE Competition Logo, https://files.materovcompetition.org/images/logos/MATEROVCompetition5683x2662.png - [Accessed: 06.01.21]

[39] BlueROV2, https://bluerobotics.com/store/rov/bluerov2/ - Accessed: 08.01.21

[40] Seabed Dredger,https://www.novasub.com/project-details/seabed-dredger/#1532340251241-e0d5a862-6c83 - Accessed: 09.01.21

[41] Serpent Seaview,https://www.seaviewsystems.com/toolbox/serpent-2/ - Accessed: 09.01.21

[42] MATE ROV Competition manual - Explorer, MATE Center 2020, http://files.materovcompetition.org/2021/2021_EXPLORER_Manual_14Sept2020.pdf - Accessed: 09.01.21

[43] Trykksensor_TE_Connectivity, https://no.rs-online.com/web/p/absolute-pressure-sensor-ics/8937200/ - Accessed: 13.01.21

[44] Skjermdump av konkurranseoppgaver, https://www.youtube.com/watch?v=KWNBOUqVIPQ&t=6s&ab_channel=MATECenter - Accessed: 19.01.21

[45] Bidragsytere til Wikimedia-prosjektene (2008). Network Time Protocol. [online] Wikipedia.org. Available at: https://no.wikipedia.org/wiki/Network_Time_Protocol [Accessed 19 Feb. 2021].

[46] Bidragsytere til Wikimedia-prosjektene (2005). Domain Name System. [online] Wikipedia.org. Available at: https://no.wikipedia.org/wiki/Domain_Name_System [Accessed 19 Feb. 2021].

[47] Wikipedia Contributors (2021). Trivial File Transfer Protocol. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Trivial_File_Transfer_Protocol [Accessed 19 Feb. 2021].

[48] Wikipedia Contributors (2021). File Transfer Protocol. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/File_Transfer_Protocol[Accessed 19 Feb. 2021].

[49] Wikipedia Contributors (2021). Simple Network Management Protocol. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol [Accessed 19 Feb. 2021].

[50] Bidragsytere til Wikimedia-prosjektene (2005). DHCP. [online] Wikipedia.org. Available at: https://no.wikipedia.org/wiki/DHCP [Accessed 19 Feb. 2021].

[51] Wikipedia Contributors (2021). MQTT. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/MQTT [Accessed 22 Feb. 2021].

[52] Bidragsytere til Wikimedia-prosjektene (2004). HTTP. [online] Wikipedia.org. Available at: https://no.wikipedia.org/wiki/HTTP [Accessed 22 Feb. 2021].

[53] https://www.facebook.com/circuitbasic (2016). Basics of the I2C Communication Protocol. [online] Circuit Basics. Available at: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/ [Accessed 2 Mar. 2021].

[54] Bidragsytere til Wikimedia-prosjektene (2004). programmeringsspråk. [online] Wikipedia.org. Available at: https://no.wikipedia.org/wiki/C%2B%2B. [Accessed 11 Mar. 2021].

[55] Bidragsytere til Wikimedia-prosjektene (2004). programmeringsspråk. [online] Wikipedia.org. Available at: https://no.wikipedia.org/wiki/Python [Accessed 11 Mar. 2021].

[56] Bidragsytere til Wikimedia-prosjektene (2005). høynivå-programmeringsspråk. [online] Wikipedia.org. Available at: https://no.wikipedia.org/wiki/JavaScript [Accessed 11 Mar. 2021].

[57] Admin (2019). Beste gratis og offentlige DNS-servere fra 2019 - IPTV Norge - NORSK IPTV Tjeneste. [online] IPTV Norge - NORSK IPTV Tjeneste. Available at: https://iptvnorge.xyz/beste-gratis-og-offentlige-dns-servere-fra-2019/[Accessed 11 Mar. 2021].

[58] ManageEngine (2021). Network Monitoring Software by ManageEngine OpManager. [online] ManageEngine OpManager. Available at: https://www.manageengine.com/network-monitoring/what-is-snmp.html [Accessed 14 Mar. 2021].

[59] Scarpati, J. (2017). Simple Network Management Protocol (SNMP). [online] SearchNetworking. Available at: https://searchnetworking.techtarget.com/definition/SNMP [Accessed 20 Apr. 2021].

[60] Mqtt.org. (2020). MQTT - The Standard for IoT Messaging. [online] Available at: https://mqtt.org/ [Accessed 20 Apr. 2021].

[61] Komplett.no. (2021). TP-LINK TL-SG105 Switch. [online] Available at: https://www.komplett.no/product/821778/datautstyr/nettverk/switch/tp-link-tl-sg105-switch?gclid=CjwKCAjwgOGCBhAlEiwA7FUXknB5TSS9AJVHjTFGHMz7USdKQimoclm7tTpmSK66F9rPN_0Z1qDBwBoCMRMQAvD_BwE&gclsrc=aw.ds# [Accessed 23 Mar. 2021].

[62] Multicom.no. (2021). LEVELONE - GEU-0521 4 PORT 10/100/1000MBPS SWITCH + 1 SFP SLOT IN PERP. [online] Available at: https://www.multicom.no/levelone-geu-0521-4-port-10/cat-p/c/p4468958 [Accessed 23 Mar. 2021].

[63] GitHub. (2021). WIZnet. [online] Available at: https://github.com/Wiznet [Accessed 23 Mar. 2021].

[64] Wiznet (2015). Wiznet/HTTPServer-LPC11E36-LPCXpresso. [online] GitHub. Available at: https://github.com/Wiznet/HTTPServer_LPC11E36_LPCXpresso [Accessed 23 Mar. 2021]

[65] WIZnet-ioLibrary (2021). WIZnet-ioLibrary/W5x00-HTTPServer. [online] GitHub. Available at: https://github.com/WIZnet-ioLibrary/W5x00-HTTPServer [Accessed 23 Mar. 2021].

[66] Selection Guide. (n.d.). [online] . Available at: https://recom-power.com/pdf/Lightline_DC-DC/RCD-24.pdf [Accessed 5 Apr. 2021].

[67] Instructables (2015). Create a Web Server Using W5500-EVB. [online] Instructables. Available at: https://www.instructables.com/Create-a-web-server-using-W5500-EVB/ [Accessed 9 May 2021].

[68] Baseline,Highline, mainline https://cutt.ly/GbZv0GF [Accessed 20 Apr. 2021].