Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Automatisering og elektronikkdesign	Vårsemesteret 2023 Åpen eller Konfidensiell
Forfatter: Thomas Davidsen Matre, Nils Helge H. Sandsbråten, Joar Landa	
Fagansvarlig: Morten Tengesdal Veileder: Morten Tengesdal	
Tittel på bacheloroppgaven: UiS Subsea: Kommunikasjon og video Engelsk tittel: UiS Subsea: Communication and video	
Studiepoeng: 3 x 20	
Emneord: Kommunikasjon, Kretskort, Videosending, CAN-Buss, Ethernet, TCP, Python	Sidetall: 319 Vedlegg/annet: A: Testrapporter B: Skjemategninger C: Mekaniske vedlegg D: I/O-lister E: Gantt og månedsrapporter Stavanger 15. mai 2023

Forord

Bacheloroppgaven markerer slutten på det treårige studiet i Automatisering og Elektronikkdesign ved Universitetet i Stavanger. Det å jobbe med prosjekt for UiS Subsea har vært veldig givende og spennende, tross alle arbeidstimene som er blitt lagt inn i prosjektet. Oppgaven har bydd på flere utfordringer og har vært krevende, noe som har resultert i et stort læringsutbytte. Blant erfaringene vi tar med oss, er erfaringen om hvordan det er å samarbeide med flere grupper for å nå et felles mål. Noen av oss tar med disse erfaringene videre inn i Masterstudiet, mens andre bygger videre på dette i arbeidslivet.

Først og fremst en stor takk til alle studentene som har vært involvert i prosjektet. Hver og en av gruppene har lagt inn en innsats og har bidratt for å komme i mål med prosjektet.

En ekstra stor takk til styreleder Otto Ljosdal, daglig leder Tomas Choat, og styrets nestleder Vejørn Riiser som har viet tiden sin til å veilede oss, på tross av Masterstudium og jobb. Alle har skrevet bachelor for UiS Subsea tidligere, og besitter god kunnskap om organisasjonen. Innspillene og erfaringene deres fra tidligere har vært svært verdifulle og hjelpsomme.

Takk til Kristian Thorsen og Jon Fidjeland for all hjelpen de har gitt relatert til PCB-design. Dette har vært svært hjelpsomt da ingen på gruppen hadde noen forkunnskaper for PCB-design når startskuddet for prosjektet gikk.

En takk til alle sponsorene som har bidratt med midler, komponenter, material og rabatter, eller som har vært involvert på andre måter. Uten dere ville det vært vanskelig å realisere prosjektet med gitt budsjett.

Det rettes en takk til Universitetet i Stavanger som har gjort det mulig å skrive bachelor for en så spennende studentorganisasjon som UiS Subsea. Det er blitt lagt til rette for en bachelor som både er teoretisk og praktisk.

Sist men ikke minst - en ekstra stor takk til veileder Morten Tengesdal. Morten har, fra start til slutt i prosjektet, kommet med gode innspill og uvurderlige tilbakemeldinger. Hans engasjement til gruppen og prosjektet har vært helhjertet, og skal ha en del av æren for det samlede resultatet av prosjektet.

Sammendrag

UiS Subsea er en studentorganisasjon som har formål med å utvikle undervannssystem. Vi har i år bestått av ni grupper som har basert sine bacheloroppgaver på å lage ROV-en Yme og flyteren Balder. Fartøyene skal konkurrere i MATE ROV Competition, og er derav utformet for å løse oppgavene knyttet til konkurransen. Prosjektet er et tverrfaglig prosjekt, der det har vært et samarbeid på tvers av ulike studieretninger.

Denne rapporten omhandler vår oppgave rundt utvikling av et system for kommunikasjon og videosending til et undervannsfartøy. I rapporten er prosessen for design, utvikling, produksjon og testing dokumentert. Det blir gjenomgått relevant teori for alle delene av oppgaven. Teorien av temaene og dokumentasjonen, legger grunnlaget for valgene som er tatt for de enkelte kapittelene. Kapittelene er seksjonert i ulike deler av oppgaven som tar for seg problemstillinger og løsninger rundt disse.

Kommunikasjonen internt ombord i ROV-en går over CAN-buss. Det ble valgt å kombinere en CAN-modul sammen med kommunikasjonskontrolleren *Nvidia Jetson Nano*, for å realisere dette grensesnittet. For at de andre kretskortene i ROV-en skal kommunisere på CAN-bussen, ble det lagd en CAN-busskrets som alle inkluderte på sine kort. Kontroll og tilbakemeldinger mellom ROV-en og kontrollstasjonen blir sendt med TCP-protokollen over Ethernet. Video blir hentet inn fra tre ulike kamera gjennom kommunikasjonskontrollen, hvor de deretter blir komprimert. Det brukes et stereokamera og to USB-kamera, der stereokamera brukes av bildebehandlingsgruppen. Videodataen sendes med RTP-protokollen til kontrollstasjon over Ethernet med UDP-protokollen.

Design av elektronikkhus og kamerahus er tegnet i *Fusion 360*. Innvendig design for montering av kretskort er så produsert ved bruk av 3D-printing. Vanntette kamerahus er maskinert for å utvendig montering av USB-kamera. Kretskort for sammenkobling av systemene og kretskort for kommunikasjonskontroller er designet i *Altium*, og bestilt fra *JLCPCB*.

Kommunikasjonskontrolleren opererer som et bindeledd mellom ROV-en og kontrollstasjonen. For å realisere bindeleddet er det utviklet kode i Python for mottak, sending og behandling av CAN-bussmeldinger samt TCP-meldinger. Det er tatt i bruk tråder for at programvaren skal operere som et sanntidssystem. Behandling av videodataen som sendes blir gjort med *gStreamer* og tar i bruk maskinvareakselerert enkoding og dekoding for energieffektiv og rask komprimering av videostrømmene. Et rammeverk for implementering av CAN-buss på mikrokontrollerene i systemene er også utviklet.

Prosjektmedlemmene har gjort en stor innsats for å få ferdigstilt ROV-en. I slike omfattende prosjekt, vil det alltid være rom for forbedringspotensiale i sin egen oppgave - samt prosjektet i sin helhet. Diskusjonskapittelet tar for seg erfaringene vi har gjort gjennom prosjektet. Her vil også hva som har fungert bra og hva som er eventuelle forbedringer bli diskutert. Gruppen har et fungerende sluttprodukt og er fornøyd med resultatet.

Innhold

Forkortelser og uttrykk

- **ABS** - ***A****crylonitrile* ***B****utadiene* ***S****tyrene*, en type termoplast som brukes i 3D printere og som tåler temperaturer opp mot 100 grader.
- **ACK** - ***A****cknowledge*, Bit som settes for å bekrefte mottak av data
- **BE** - ***B****ig* ***E****ndian*
- **BRP** - ***B****aud* ***R****ate* ***P****rescaler*, Parameter for setting av CAN-busshastigheten.
- **CAD** - ***C****omputer* ***A****ided* ***D****esign*, Konstruksjon og tenknisk tengning ved hjelp av datamaskiner.
- **CAN** - ***C****ontroller* ***A****rea* ***N****etwork*, Bussprotokoll brukt i prosjektet for kommunikasjon mellom mikrokontrollerne.
- **CPU** - ***C****entral* ***P****rocessing* ***U****nit*
- **CRC** - ***C****yclic* ***R****edundancy* ***C****heck*, Syklisk redundanssjekk er en type feildeteksjon brukt i digitale nettverk.
- **CSI** - ***C****amera* ***S****erial* ***I****nterface*, Grensesnitt mellom kamera og vertsprosessor.
- **ENIG** - ***E****lectroless* ***n****ickel* ***i****mmersion* ***g****old*, Prosess brukt ved produksjon av kretskort.
- **FDCAN** - ***F****lexible* ***D****ata* ***C****ontroller* ***A****rea* ***N****etwork*, Bussprotokoll brukt i prosjektet for kommunikasjon mellom mikrokontrollerne. Videreutvikling av CAN.
- **Full-Duplex** - Når kommunikasjon mellom to enheter foregår parallellt over to linjer.
- **FDM** - ***F****used* ***D****eposition* ***M****odeling* (Stratasys sitt eget navn på 3D printing).
- **Gcode** - Programmeringsspråk for CNC maskiner og 3D-printere
- **GPIO** - ***G****eneral-****P****urpose* ***I****nput/****O****utput*
- **HAL** - Innebygde funksjoner som skal forenkle programmering av maskinvare, og lage koden mer universell for andre mikrokontrollere i STM32-serien.
- **Half-Duplex** - Når kommunikasjon mellom to enheter kun går en vei om gangen over en linje.
- **HASL** - ***H****ot* ***a****ir (****s****older)* ***l****eveling*, Prosess brukt ved produksjon av kretskort.
- **HDPE** - ***H****igh* ***D****ensity* ***P****oly* ***E****thelen*, Plastmateriale som er slitesterkt. Brukest i alt fra plastposer til vannrør.
- **I2C** - ***I****nter-****I****ntegrated* ***C****ircuit*. Half duplex kommunikasjonsbuss
- **IFS** - ***I****nter-****F****rame* ***S****pacing*
- **IP** - ***I****nternet* ***P****rotocol*
- **IC** - ***I****ntegrated* ***C****ircuit*, Integrert elektrisk krets
- **Kontrollstasjon** - Samlebegrep for datamaskin med brukergrensesnitt stasjonert ved overflaten som brukes til kontroll og styring av ROV-en.
- **LAN** - ***L****ocal* ***A****rea* ***N****etwork*
- **LE** - ***L****ittle* ***A****rea* ***E****ndian*
- **LSB** - ***L****east* ***S****ignificant* ***B****yte* ***E****ndian*
- **MAN** - ***M****etropolitan* ***A****rea* ***N****etwork*
- **MSB** - ***M****ost* ***S****ignificant* ***B****yte* NAL - Network Abstraction Layer
- **NAL** - ***N****etwork* ***A****bstraction* ***L****ayer*
- **PC** - ***P****ersonal* ***C****omputer*, Datamaskin

- **PCB** - ***P**rinted **C**ircuit **B**oard*
- **PCIe** - ***P**eripheral **C**omponent **I**nterconnect **E**xpress*, PCIe eller PCI Express er en buss brukt i datamaskiner for tilkobling av modulkort som for eksempel skjermkort, lydkort og lagringskontrollere.
- **PLA** - ***P**olyactic **A**cid*, Vanlig plastmateriale å bruke i 3D-printing.
- **PWM** - ***P**ulse **W**idth **M**odulation*, Firkantpulser som er breddemodulert for å sende signal.
- **PETG** - ***P**oly**E**thylene **T**erephthalate **G**lycol*, en type termoplast som brukes i 3D printere og som har mange av de samme egenskapene som ABS, men tåler ikke fult så høye temperaturer.
- **RTP** - ***R**eal-time **T**ransport **P**rotocol*
- **ROV** - ***R**emote **O**perated **V**ehicle*, I denne oppgaven er det snakk om en undervannsdrone.
- **SBC** - ***S**ingle **B**oard **C**omputer*
- **Simplex** - Når kommunikasjon mellom to enheter bare går en vei.
- **SOF** - ***S**tart **O**f **F**rame*, Markerer første bit av data på CAN-bussen
- **SPI** - ***S**erial **P**eripheral **I**nterface*, Kommunikasjonsstandard for synkron seriellkommunikasjon over korte avstander.
- **TCP** - ***T**ransmission **C**ontrol **P**rotocol*
- **TIG** - ***T**ungsten **I**nert **G**as*, Buesveising som bruker en ikke-forbrukbar wolframelektrode og ekstern tilsats.
- **OSI** - ***O**pen **S**ystems **Interconnection***
- **Thruster** - En type propell som ofte brukes på ROV-er eller til å manøvrere båter der de har lite plass. Det er thruster som brukes i selve rapporten.
- **Tomgang/hvilemodu** - Utrykker en normaltilstand for en IC. Ikke er aktiv eller endrer seg ikke. For eksempel når klokken på SPI-bussen ikke kjører.
- **UDP** - ***U**ser **D**atagram **P**rotocol*
- **USB** - ***U**niversal **S**erial **B**us*, industristandard ofte brukt mellom eksterne enheter og datamaskiner.
- **UiS** - ***U**niversitetet **i** **S**tavanger*
- **WAN** - ***W**ide **A**rea **N**etwork*

Kapittel 1

Introduksjon - Subsea

Kapitteloversikt

1.1 Om UiS Subsea

Dette kapittelet er blitt skrevet i felleskap av prosjektledelsen for UiS Subsea. Hensikten med kapittelet er blant annet å gi en innsikt i MATE-konkurransen som gruppen skal delta i. Det vil også gis en introduksjon av studentorganisasjonen UiS Subsea i sin helhet, samt en introduksjon av oppgavene til de underordnede gruppene som skriver bacheloroppgave.

UiS Subsea er en studentorganisasjon ved Universitetet i Stavanger som har engasjert studenter innenfor undervannsteknologi helt siden 2013. Hovedmålet til organisasjonen er å tilby studenter erfaring med å jobbe, og samarbeide i et prosjekt bestående av forskjellige ingeniørdisipliner.

I år består UiS Subsea av totalt ni ulike bachelorgrupper fra ingeniørdisiplinene data, elektro og maskin, som jobber mot et felles mål om å designe og produsere en fjernstyrt undervannsfarkost, ROV[1].

Årets ambisjoner var basert på å forbedre fjorårets ROV-design, med hovedmål om å gjøre den nye ROV-en enklere å vedlikeholde, mer effektiv, samt oppgradere programvaren for å rette produktet inn mot en autonom undervannsfarkost, AUV [2]. Årets farkost vil ikke bli helt autonom, men skal kunne utføre noen av MATE sine autonome oppgaver. På årets prosjekt er det to maskiningeniørgrupper, som er ansvarlig for å designe og produsere manipulator, ramme og elektronikkhus til ROV-en og flyter. Det er fem elektroingeniørgrupper, hvor fire av gruppene jobber med ROV-en, og en gruppe jobber med flyteren. Elektrogruppene på ROV-en har ansvar for krafthåndtering, sensorer, regulering & navigasjon, og kommunikasjon mellom de ulike elektriske systemene, samt operatørstasjonen. Elektrogruppen som jobber med Flyteren har ansvar for all elektronikk til flyteren. Det er to dataingeniørgrupper som er ansvarlig for å sende og motta kommandoer, samt data mellom ROV-en og kontrollstasjonen(*Topside*). Kontrollstasjonen viser informasjon fra ROV-en i et *GUI*[3], og behandler også bildedata for de autonome oppgavene.

UiS Subsea har over flere år bygget ROV-er og deltatt i internasjonale konkurranser. Årets mål er å delta på *MATE ROV Competition* i Denver, USA. Dette skaper et grunnlag for problemløsning og samarbeid, med formålet om å skape et positivt og sunt miljø for læring og utvikling av tekniske løsninger. UiS Subsea gir studenter muligheter til å kommunisere og samarbeide med næringslivet i området. Flere selskaper er svært engasjerte i denne type prosjekter, og tilbyr ofte komponenter, ekspertise og andre ressurser via sponsoravtaler. For å videre forbedre forholdet mellom organisasjonen og næringslivet, holder UiS Subsea årlig en begivenhet kalt *Subsea dagen*, hvor selskaper fra *subsea*-næringen kan ha presentasjoner for å reklamere for seg selv. Både UiS Subsea, Universitetet i Stavanger og selskapene genererer positiv eksponering fra denne begivenheten.

Figur 1.1: UiS Subsea logo.

Tidligere år har organisasjonen lidd fra en mangel på kontinuitet, da studentene kommer inn, skriver bachelor og forlater organisasjonen før neste års studenter kan lære fra dem. I tillegg har det vært lite kommunikasjon mellom tidligere og nåværende bachelorstudenter. I år har fjorårets prosjektleder og sponsoransvarlig besluttet å sitte i styret for 2023 i UiS Subsea, slik at de kan bidra med å gi råd og veilede årets prosjekt. Dermed blir kunnskap og erfaringer videreført, som gir en brattere og mer effektiv læringskurve for årets bachelorstudenter.

Årets prosjektledelse består av følgende roller og personer:

- **Prosjektleder:** Joar Rodrigues de Miranda
- **Prosjektleder konkurranse-koordinator:** Thomas Matre
- **Teknisk leder elektro:** Jesper A. Flatheim
- **Teknisk leder data:** Filip Sølvberg Herrera
- **Teknisk leder maskin:** Aleksander Schei

[1] Remotely Operated Vehicle
[2] Autonomous Underwater Vehicle
[3] Grafisk brukergrenssnitt

1.2 Årets bacheloroppgaver

Årets prosjekt engasjerer 24 studenter, fordelt på totalt ni oppgaver. De ulike gruppene og oppgavene blir kort presentert i de kommende delkapitlene.

1.2.1 Design av ROV og flyter

Hovedoppgaven for denne maskiningeniørgruppen går ut på å designe og bygge rammen og elektronikkhuset til ROV-en, samt flyteren. Komponentene er utviklet ved bruk av *produktutviklingsprosessen*. Primært er fokuset til designet å kunne koble alle de individuelle komponentene sammen til en fungerende ROV, samtidig tilfredstille kravene som stilles i konkurransene.

Kjerneoppgaver som løses av ROV-designgruppen, er materialvalg, dimensjonering, strukturell analyse, flytanalyse, beregning av oppdrift og stabilitet. På grunn av miljøutfordringene som verden står ovenfor, er det sekundære fokuset i utviklingsprosessen gjenvinning og bærekraft. Hvor målet er å minimere miljøavtrykket i denne prosessen, og ved effektiv bruk av DFE, vil resultatet innebære redusert kostnad og produksjonstid, samt forbedret produktkvalitet.

1.2.2 Manipulator

Oppgaven til den andre maskiningeniørgruppen består av å utvikle og designe en funksjonell manipulator. Fokuset vil være at manipulatoren skal være relativt enkel og funksjonell, samt at den tilfredsstiller de kravene som er satt i MATE- og TAC-konkurransene. Dette innebærer å være kreativ og løsningsorientert, for å optimalisere manipulatordesignet til å best mulig kunne utføre disse oppgavene. Dette innebærer å bestemme antall frihetsgrader, hvilke mekaniske systemer som skal benyttes, samt velge de mest hensiktsmessige materialene for å kunne tåle belastningene og forholdene manipulatoren utsettes for. Manipulatoren må designes til å være kompatibel med de elektriske systemene.

1.2.3 Kraftforsyning

Kraftmodulen sin hovedoppgave er å regulere og fordele inngangsspenningen som forsynes fra land, samt sikre mot overbelastning og kortslutninger på fartøyet. Med en inngangsspenning på $48V$ på ROV-en, er det nødvendig med spenningsregulering for at de ulike komponentene i ROV-en skal fungere. Kraftmodulen skal også gjøre nødvendige strømmålinger for å verne om overforbruk av kraft.

1.2.4 Kommunikasjon

Kommunikasjonsoppgaven er tredelt. Videodelen innebærer å hente video fra ROV-en og sende denne til kontrollstasjonen. Kommunikasjonsdelen innebærer å videresende kontrolldata fra kontrollstasjonen til ROV-systemene, samt videresende prosessdata fra ROV-systemene til kontrollstasjonen. Den siste delen av oppgaven innebærer å lage et rammeverk for sammenkobling av kretskortene som realiserer grensesnittet mellom disse.

1.2.5 Regulering og styring

For at ROV-en skal være manøvrerbar, må det utvikles et reguleringssystem. De mest sentrale delene i et slikt system er valg av motorer og thrusterene. Denne gruppen har også ansvaret for styring av manipulatoren. Systemet skal realiseres gjennom utvikling av kretskort.

1.2.6 Sensorsystem

Sensorsystemet innebærer å hente inn rådata fra ulike sensorer, behandle den, for så å distribuere denne videre over kommunikasjonsgrensesnittet. Sensordata som kreves til regulering er orienteringen, samt dybden ROV-en befinner seg på. Orienteringsdata registreres fra en IMU(*Inertial Measurement Unit*), mens dybden måles ved hjelp av en trykksensor.

Sensorsystemet er også ansvarlig for å rapportere om kritiske feil, som for eksempel vannlekkasje, eller for høy temperatur i elektronikkhuset. Systemet realiseres gjennom utvikling av kretskort med en mikrokontroller som skal sørge for tilstrekkelig databehandling.

1.2.7 Flyter

Flyteren som skal designes og produseres, er en autonom farkost til bruk under vann. Autonome flytere brukes til å hente inn data om miljøet under vann.

I konkurransen skal flyteren utføre to vertikale profiler. En vertikal profil innebærer at flyteren starter med å synke kontrollert ned til bunnen, og deretter stiger opp til overflaten. Under den vertikale profilen skal flyteren måle temperatur og trykk. Trykkmålingene brukes til kartlegging av de ulike dybdene til flyteren under den vertikale profilen. Når flyteren når overflaten, skal dataen sendes trådløst til kontrollsystemet på land, hvor en operatør analyserer dataen.

1.2.8 Bildebehandling

Hovedoppgavene til bildebehandligsgruppen går ut på å utføre deloppgavene til MATE ROV Competition, som innebærer å behandle bildedata for å løse ulike oppgaver. Disse oppgavene krever kamerasyn og autonomi. Gruppen har ansvar for å løse følgende oppgaver:

Autonom Docking Denne oppgaven går ut på å plassere ROV foran en dockingstasjon, hvor ROV kan se et referansepunkt inne i stasjonen og autonomt kjøre inn å parkere.

3D-modellering av syke korallhoder Her skal det bli brukt kamerasyn til å måle størrelsen på korallhodet, samt kunne lages en 3D-modell av objektet. Til slutt skal det vurderes hvorvidt korallen er syk, basert på mengden hvitfarge som er synlig på objektet.

Telle frosker i transekt : I denne oppgaven skal ROV kjøre langs en transekt, og telle antall frosker på havbunnen.

Analyse av sjøgressvekst ROV-en skal her utføre analyse og sammenligne to felt med sjøgress på havbunnen, for å så detektere om det er positiv eller negativ plantevekst.

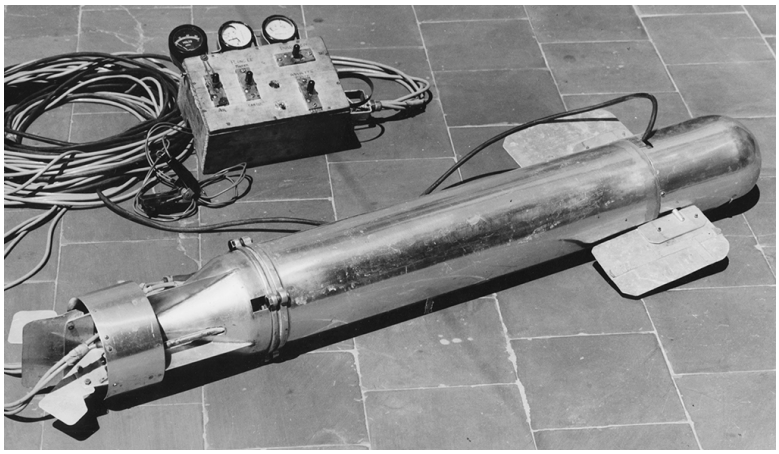
1.2.9 GUI

Oppgaven går ut på å utvikle et system for å overvåke og styre ROV-en. For å oppnå dette, må det implementeres et system for å sende kommandoer og styringsdata fra operatrørstasjonen til ROV-en. Dette gjøres i samarbeid med kommunikasjonsgruppen. Det må også designes et brukergrensesnitt som viser videostrømmer og sensordata fra ROV i sanntid. Det skal prioriteres at grensesnittet er brukervennlig, og viser nødvendig informasjon på en oversiktlig måte. I tillegg skal ROV-en være enkel for en ROV-operatør å kjøre.

1.3 Om ROV prosjektet

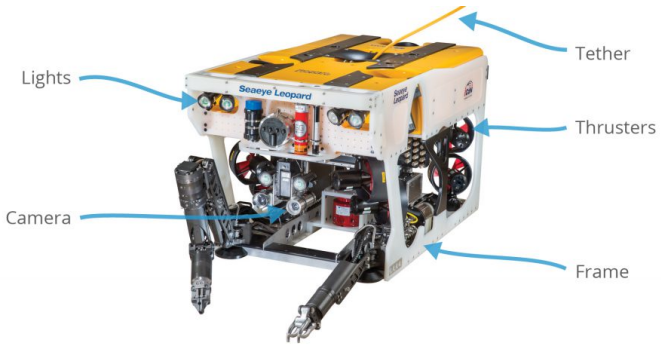
1.3.1 ROV-historie

Utviklingen av den første ROV-en kan bli spores tlibake til Dimitry Rebikoff, hans ROV er vist i figur 1.2 [1]. Den ble laget i 1953 og fikk navnet *Poodle*. Den er komplett med en tjoret forbindelse, og mulighet for styring fra land.



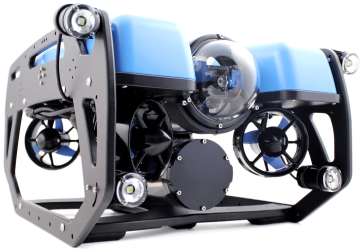
Figur 1.2: Verdens første ROV [1]

Subsea-næringen har kommet langt siden den gang, hvor de moderne ROV-ene er annerledes i både form og funksjon. På 1960-tallet brukte den amerikanske marinen ROV-er til å lokaliser og gjenvinne tapt undervannsutstyr. 20 år senere var det over 500 ROV-er i verden, bygget til å håndtere flere ulike oppgaver. De fleste ROV-ene utførte arbeid i det kommersielle markedet. Vanlige komponenter på standard ROV-er er thrustere, tjor, kamera, lys, rammeverk, kontrollstasjon, oppdriftselementer. Noen av disse komponentene er vist i figur 1.3 [1].

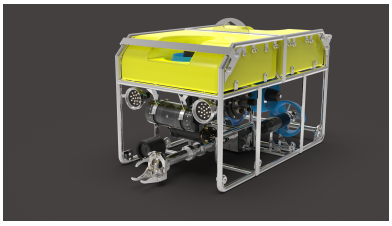


Figur 1.3: Viser vanlige komponenter på en ROV [1]

Moderne ROV-er er designet for å utføre bestemte oppgaver, eksempelvis: observasjon, inspeksjon, grøfting, nedgraving, intervensjon og konstruksjon. Noen ROV-er kan brukes til flere oppgaver, mens andre er spesiallaget for en spesifikk oppgave. Det er 7 hovedklasser av ROV-er, *Klasse I* til *Klasse VII* er vist i figur 1.4 [2].



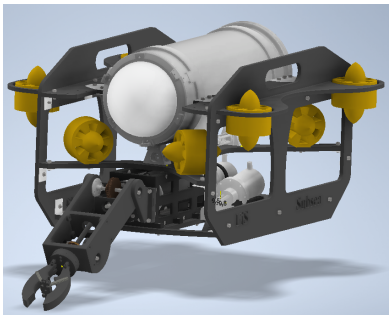
(a) Klasse I: Ren observasjon. Bildet er hentet fra [3].



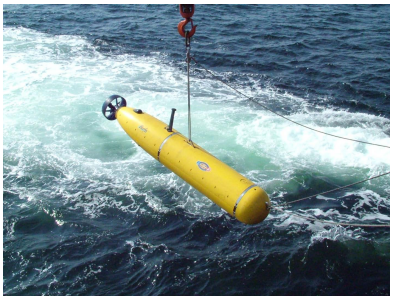
(b) Klasse II: Observasjon med alternativ nyttelast. Bildet er hentet fra [4].

(c) Klasse III: Kjøretøy for arbeid. Bildet er hentet fra [5].

(d) Klasse IV: Kjøretøy for arbeid på havbunnen. Bildet er hentet fra [6].



(e) Klasse V: Prototyper eller utviklingskjøretøy.



(f) Klasse VI: Autonome undervannskjøretøy (AUV). Bildet er hentet fra [7].

(g) Klasse VII: Høyhastighets vedlikeholdskjøretøy. Bildet er hentet fra [8].

Figur 1.4: Viser de ulike ROV-klassene.

Klasse I:

Det er flere fordeler og begrensninger med hver klasse. *Klasse I* består av rene observasjonskjøretøyer som er begrenset til videoobservasjon, men som er svært manøvrerbare. Vanligvis er dette små kjøretøyer utstyrt med videokamera, lys og thrustere. De kan ikke påta seg noen andre oppgaver uten betydelige endringer.

Klasse II:

ROV-er til observasjon med alternativ nyttelast har i utgangspunktet like muligheter som en ren observasjons-ROV. Vanligvis med tilleggsfunksjonalitet, som manipulator, kameraovervåkning, ekkolodd og katodisk beskyttelsesmålesystem.

Klasse III:

Her er det ROV-er laget for arbeid, og er store nok til å bære ekstra sensorer og/eller manipulatorer. De har semi-autonome evner, kjent som multipleksingssevne, som gjør det mulig å bruke tyngre utstyr uten å være *direkte koblet* gjennom en tjor. Videre har de nok stabilitet og oppdrift til å bære ekstra avtakbart utstyr uten tap av funksjonalitet. Denne klassen er større enn de tidligere nevnte, med 3 underklasser basert på effekt:

1. **Klasse III A:** Arbeidsklassekjøretøy $<$ 100 Hp
2. **Klasse III B:** Arbeidsklassekjøretøy 100 Hp to 150 Hp
3. **Klasse III C:** Arbeidsklassekjøretøy $>$150 Hp

Klasse IV:

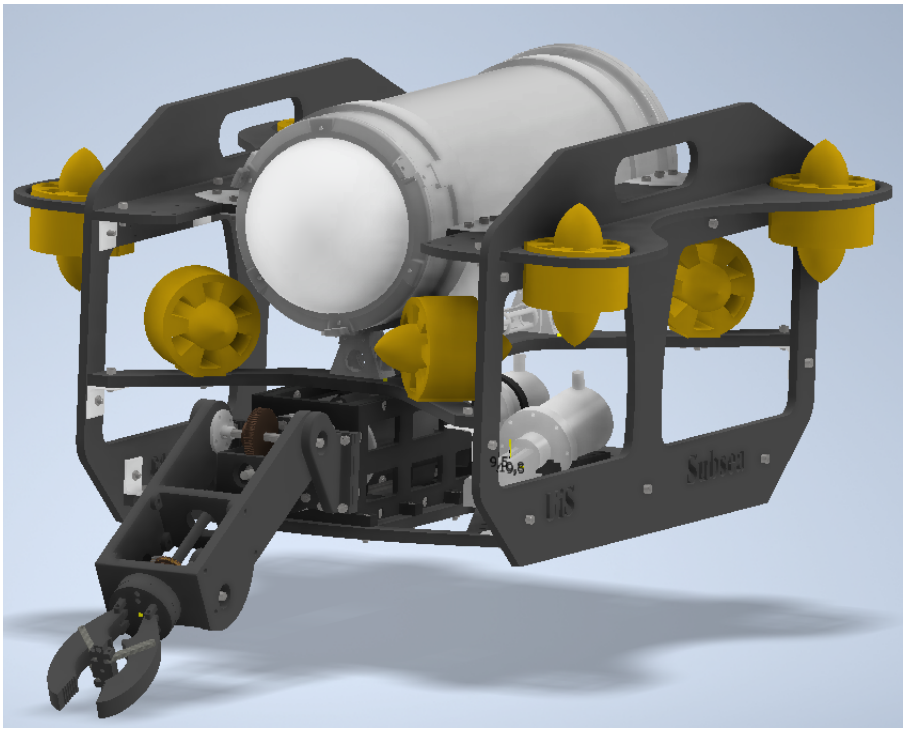
Disse er arbeidskjøretøyer som brukes på havbunnen. De manøvrerer ved hjelp av hjul, beltetrekksystem, thrusterpropeller og vannjet, eventuelt en kombinasjon av de nevnte. Disse ROV-ene er vanligvis enda større enn *Klasse III*, hvor hovedformålet ved arbeid under vann er mudring, gruvedrift, grøfting, utgraving og annet konstruksjonsarbeid under vann.

Klasse V:

Denne klassen består av prototype- eller utviklingskjøretøy. ROV-er i denne klassen er under utvikling, og dermed ikke tilstrekkelig testet. De fleste spesialkjøretøyer eller engangsprototyper plasseres i denne kategorien. Både *Klasse VI* og *Klasse VII* tilhører denne klassen i henhold til norsk standard for ROV-er. Dette kommer av at de fortsatt er under utvikling, hvor det kun er noen få utvalgte selskaper som produserer denne typen ROV-er.

1.3.2 Yme

ROV-en, navngitt Yme, som utvikles og produseres i dette prosjektet er en *Klasse II ROV*, med 6 frihetsgrader og kompakt design. Målet er å skape et fartøy som baserer seg på gode eksisterende løsninger, samt påfører et lite miljøavtrykk i samsvar med FNs bærekraftsmål.

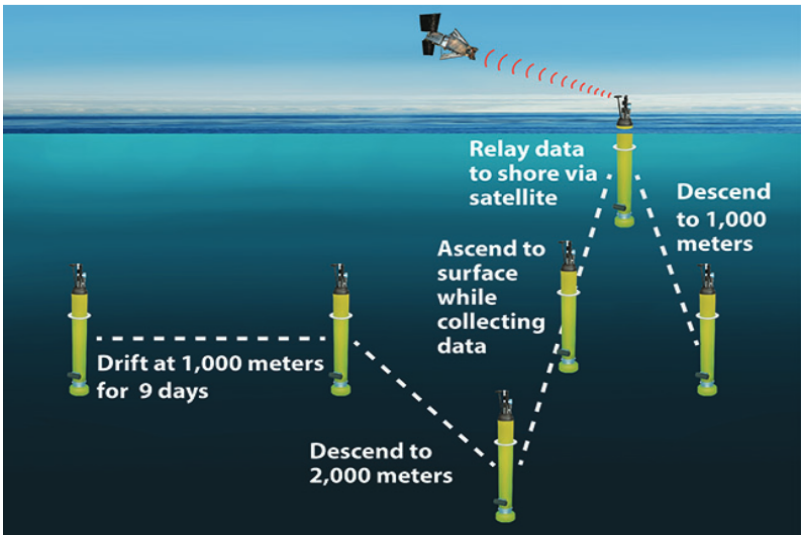


Figur 1.5: 3D-modell av Yme

Figur 1.5 viser en 3D-modell av ROV-en Yme. ROV-en styres av en egen kontrollstasjon som kommuniserer via en tjor. Målet i prosjektet er at Yme skal kunne operere og fungere på 100 meters dyp, selv om designet utvikles for å kunne konkurrere i MATE-ROV konkurransen som utspilles i svømmebasseng på 5-10 meters dybde. Designet er modulært, noe som medfører at man enkelt kan erstatte defekte deler. Det modulære designet legger til rette for å forbedre og viderutvikle ROV-en, samt legge til autonome funksjoner.

1.3.3 Generelt om flytere

Vitenskapelige flytere har vært i bruk i lang tid, helt siden Henry Melson Stommel kom opp med ideen tilbake i 1955 [9]. Formålet med en flyter var å spore og observere dype drivstrømmer. De første flyterene ble produsert av aluminium med en dybdebegrensning på 4500 meter. En kan regulere dybden til flyteren ved å bruke en oppdriftsmotor med forhåndsprogrammerte høyder. En type flytere er biogeokjemiske, og bruker en rekke optiske og kjemiske sensorer for å samle data [10].



Figur 1.6: Skjematikk av en flyter sin syklus. [11]

Figur 1.6 viser en normal syklus til en flyter. Først går den ned til en dybde på 1000 meter, og driver i 5 til 10 dager mens den samler inn data. Denne rutinen gjentas ved 2000 meter, for så å stige opp til overflaten. Ved overflaten overføres dataen til satellitter. En gjennomsnittlig flyter vil ha en livssyklus på cirka 5 år.

1.4 MATE - Marine Advanced Technology Education

Informasjonen under, samt alle figurene er hentet fra arrangørenes hjemmesider [12], [13].

ROV-en som UiS Subsea utvikler i år, er i samsvar med spesifikasjonene fastsatt av den internasjonale konkurransen *MATE ROV COMPETITION*. Denne konkurransen arrangeres av organisasjonen *MATE*[4], som er et partnerskap mellom amerikanske organisasjoner etablert i 1997. Partnerene inkluderer hovedsakelig skoler, forskningsinstitutter, myndigheter og institusjoner for marinteknikk. Formålet med partnerskapet er å arbeide for å forbedre marintekniske utdanninger, og dermed forberede den fremtidige amerikanske arbeidsstyrken for kommende maritime operasjoner.

Figur 1.7: MATE logo.

I 2021 overførte *MATE* ansvaret for studentaktiviteter til *Marine Advanced Technology Education for Inspiration and Innovation*, også kjent som *MATE II*. Disse har ansvar for å å arrangere den årlige *MATE*-konkurransen. Formålet er å utfordre studentene til å anvende og utvide sin ingeniørfaglige kunnskap for å løse undervannsmessige utfordringer. UiS Subsea deltar i Explorer-klassen, som er rettet mot studenter innen høyere teknisk utdanning.

Figur 1.8: MATE II logo.

1.4.1 MATE ROV Competition

Informasjonen om konkurransen er hentet fra konkurransemanualen [14] og arrangørens hjemmeside [12].

Konkurransen fortsetter i samme stil som de to siste årene, med temaer knyttet til FNs tiår for havforskning (2021-2030). FNs tiår for havforskning er et initiativ som tar sikte på å øke kunnskapen om havene globalt, med mål om å sikre at samfunnet anvender denne kunnskapen. Dette bidrar til å nå FNs bærekraftsmål [15]. Årets oppgave innebærer å bygge en ROV samt en flyter, som kan utføre vitenskapelig arbeid knyttet til disse temaene. Aktuelle tema for året inkluderer utvikling og drift av ren energi, samt overvåking og prøvetaking for å bidra til å gjenopprette biologisk mangfold.

Figur 1.9: MATE Competition logo

Poenggiving

Under konkurransen deles poengene ut i tre segmenter. For produktdemonstrasjon, skal deltakerne utføre tre praktiske oppgaver innen en tidsramme på 15 minutter. Dersom oppgavene fullføres før tiden, tildeles det ekstrapoeng. Det gis ett ekstrapoeng for hvert minutt. Ekstrapoeng gis også dersom ROV-en veier mindre enn 25 kg. Under konkuransen får man også poeng for godt samarbeid.

De praktiske oppgavene utfordrer de operative egenskapene til ROV-en. Det andre segmentet gir poeng for teknisk dokumentasjon og presentasjon av organisasjonen. Det siste segmentet gir poeng for sikkerheten til ROV-en, og analysen av farene knyttet til driften av denne.

[4] *The Marine Advanced Technology Education*

Tabellen nedenfor viser en oppsummering av poengstruktur for hvert segment i konkurransen. Detaljer om hvordan poeng tildeles for dokumentasjon, markedsføring og sikkerhet er ikke inkludert.

Produktdemonstrasjoner	
Oppgaver	300 poeng
Tidsbonus	10 poeng
Vektrestriksjoner	10 poeng
Organisatorisk effektivitet	10 poeng
Prosjektering og kommunikasjon	
Teknisk dokumentasjon	100 poeng
Produktpresentasjon	100 poeng
Marketsføring	50 poeng
Bedriftens spesifikasjonsark	20 poeng
Bedriftsansvar	20 poeng
Sikkerhet	
Gjennomgang av sikkerhetsdokumentasjon	20 poeng
Sikkerhetsinspeksjon	30 poeng
Sikker jobbanalyse	10 poeng
Totalt	**680 poeng**

Oppgave 1: (Maritim fornybar energi)

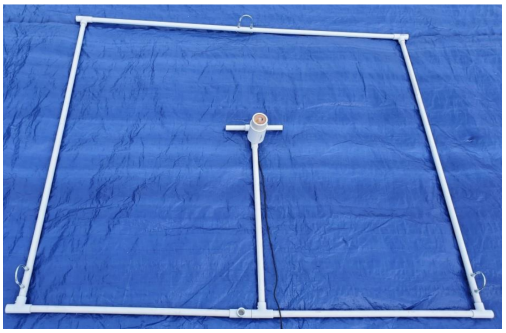
FNs bærekraftsmål [15] [14]:

7 Ren energi til alle

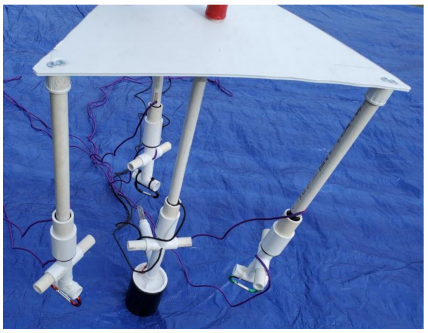
12 Ansvarlig forbruk og produksjon

Den første oppgaven er designet for å simulere installasjonen av flytende solcellepaneler i en etablert flytende vindkraftpark. Oppgaven inkluderer også fjerning av biologisk begroing, og manøvrering av ROV-en autonomt eller manuelt inn i en parkeringsstasjon.

Oppgave 1.1: Installere flytende solcellepaneler



(a) Havbunnsanker for installasjon



(b) Flytende solcellepanel

(c) Fortøyningskrok

- Manøvrere solcellepanelene mellom de tre eksisterende vindturbinene: 10 poeng.
- Feste de tre fortøyningene fra solcellepanelene til ankerpunktene: 15 poeng.

- Fjerne dekselet fra inngangen til kraftporten: 5 poeng.
- Koble til pluggen fra solcellepanelene: 10 poeng.

Oppgave 1.2: Fjerne biologisk begroing fra flytende vindturbiner

Den biologiske begroingen simuleres enten med rødfarget PVC-rør festet med borrelås, eller med rørrensere laget av chenille som er tvunnet sammen.

(a) Biologisk begroing på struktur

(b) Biologisk begroing på tau

- Fjerne 1-2 enheter av biologisk begroing: 5 poeng.
- Fjerne 3-5 enheter av biologisk begroing: 10 poeng.
- Fjerne 6 enheter av biologisk begroing: 15 poeng.

Oppgave 1.3: Manøvrere ROV-en i en parkeringsstasjon

Figur 1.12: Parkeringsstasjon for en ROV

- Manøvrere autonomt inn i parkeringsstasjonen: 15 poeng.

- Manøvrere manuelt inn i parkeringsstasjonen: 10 poeng.

Oppgave 2A: Korallrev og blått karbon

FNs bærekraftsmål [15], [14]:

13 Stoppe klimaendringene

14 Livet i havet

Denne oppgaven er delt inn i to deler, 2A og 2B. Del A omfatter vitenskapelige arbeidsoppgaver som å skanne et korallrev og identifisere organismer ved bruk av eDNA. Videre innebærer oppgaven å påføre UV-lys på syke korallrev, samt inspisere og installere et miljøvennlig forankringssystem for å beskytte sjøgress på havbunnen.

Oppgave 2.1: Måle, modellere og finne sykdom på en korallrev

Figur 1.13: Korallhode med hvite flekker.

- Måle korallrevets diameter: 5 poeng.
- Måle korallrevets høyde: 5 poeng.
- Måle korallrevets areal av syke områder: 5 poeng.
- Lage en 3D-modell autonomt: 15 poeng.
- Lage en 3D-modell manuelt i CAD: 5 poeng.

Alle mål har et toleranseområde på 2 cm og kan utføres enten autonomt eller manuelt, ved å bruke objekter med kjente størrelser som referanse.

Oppgave 2.2: Identifisere korallrevorganismer ved eDNA

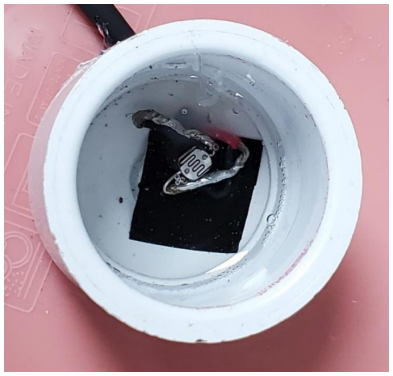
(a) Vannposekobling

(b) Kobling for uttak av prøve

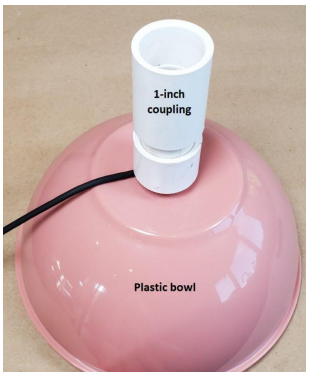
a) Hente vannprøve i flaske: 10 poeng.

b) Identifisere fisketyper basert på tre prøver av eDNA utgitt fra arrangør: 5 poeng.

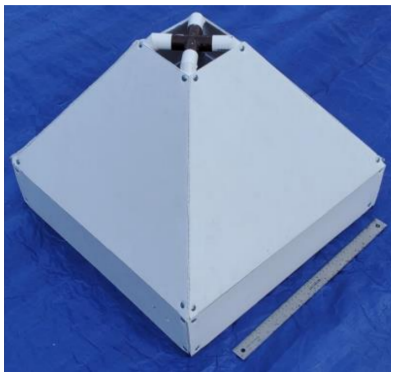
Oppgave 2.3: Administrere Rx-middel til sykt koral



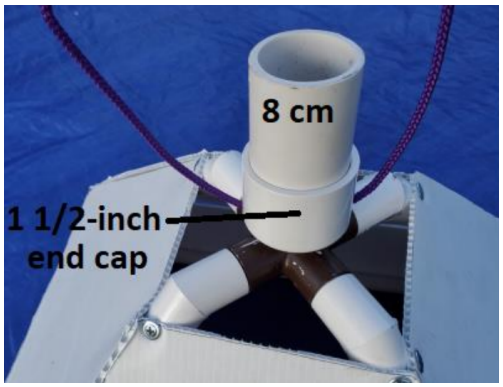
(a) Photoresistor.



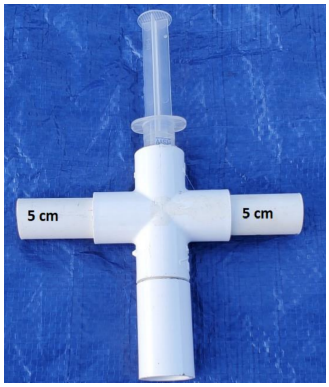
(b) Kobling for lys.



(c) Telt.



(d) Kobling for injeksjon.



(e) Sprøyte.

a) Posisjonere UV-lys over sykt område: 5 poeng.

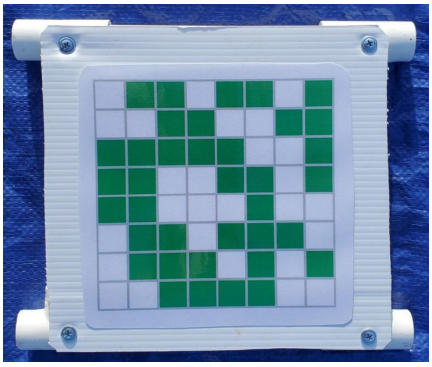
b) Skru på lys og behandle: 5 poeng.

c) Plassere telt over korallrev: 10 poeng.

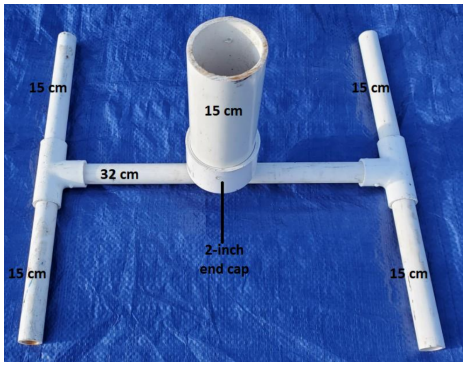
d) Plassere sprøyte i åpningen til teltet: 5 poeng.

e) Tømme innholdet i sprøyten i teltet: 5 poeng.

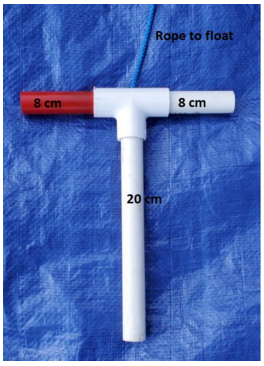
Oppgave 2.4: Overvåke og beskytte sjøgresshabitat



(a) Sjøgress.



(b) Eco-Mooring base.



(c) Fortøyningspunkt.

a) Identifisere om et sjøgresshabitat har rehabilitert seg, forblitt uendret eller forverret seg i løpet av tre måneder basert på bilder: 5 poeng.

b,c) Installere *Eco-Mooring* system på havbunnen i en base og deretter rotere forankringspunktet 720° i basen: 10 poeng.

Oppgave 2B: Innsjøer og elver

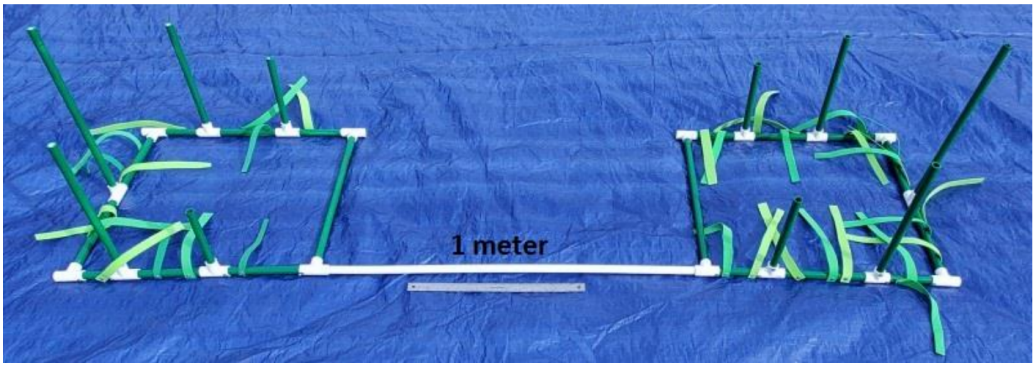
FNs bærekraftsmål [15] [14]:

13 Stoppe klimaendringene

14 Livet i havet

Oppgave 2B er rettet mot vitenskapelig arbeid på innsjøer. Oppgavene inkluderer å lete etter fisk og sjekke om de tilhører en invaderende art, samt slippe ut yngel der det er trygt. Videre skal ROV-en inspisere tau, fjerne større objekter, kjøre over en transektlinje, telle antall frosker og installere et kamera på bunnen.

Oppgave 2.5: Re-introdusere utrydningstruede arter av Northern Redbelly Dace yngel



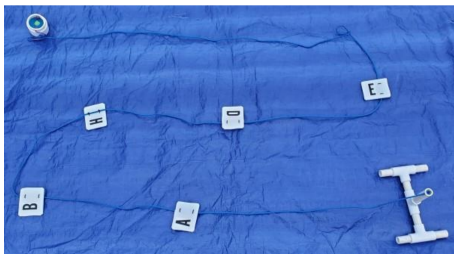
(a) Habitatområdene

(b) Yngel

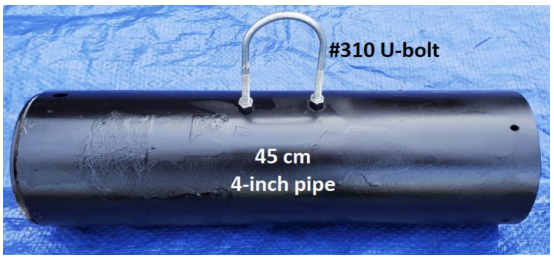
(c) Eksisterende fiskerase

- Undersøke to områder for å finne trygge steder å plassere yngel: 10 poeng.
- Klimatisere yngel i et trygt område: 5 poeng.
- Slippe ut yngel i et trygt område: 10 poeng.

Oppgave 2.6: Sørge for helse og sikkerheten av Dillion-reservatet



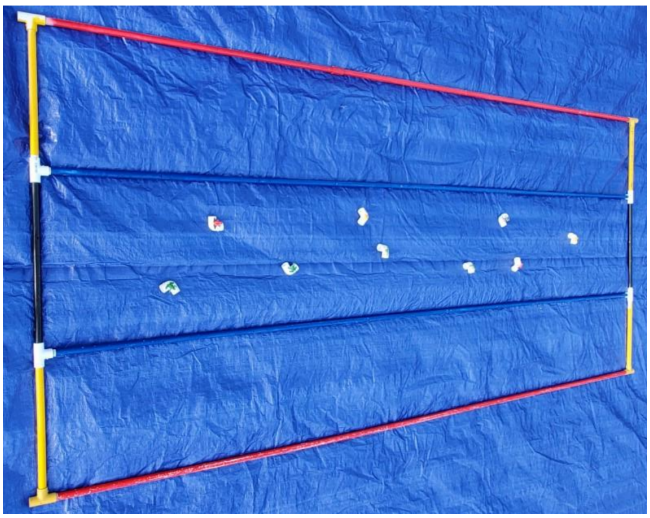
(a) Tau med bokstaver



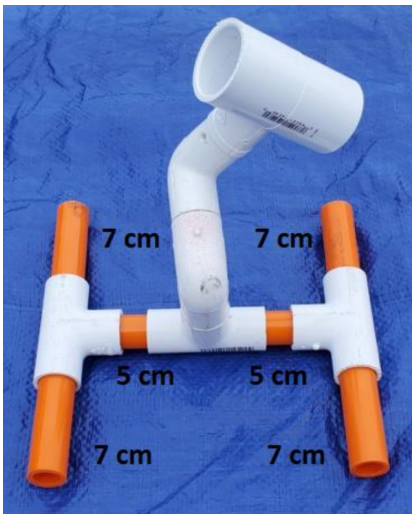
(b) Tungt objekt som skal ryddes

- Inspisere tauet til en bøye og vise frem 10 bokstaver som er festet langs tauet: 10 poeng.
- Vise frem dokumentasjon på ROV-en sin løftekapasitet: 5 poeng.
- Løfte et objektet på maksimalt 120 Newton opp av vannet: 10 poeng.
- Bringe objektet til land: 5 poeng.

Oppgave 2.7: Overvåke truet Lake Titicaca frosker



(a) Område som transektlinjen ROV-en skal kjøre over



(b) Kameraet som skal plasseres

a) Kjøre transektlinje og holde videobildet innefor strekene: 10 poeng.

b) Telle antall frosker som er innenfor område: 5 poeng.

c) Installere et kamera på et angitt område: 5 poeng.

1.4.2 Oppgave 3: MATE Floats

FNs bærekraftsmål [15] [14]:

13 Stoppe klimaendringene

Denne oppgaven innebærer å bygge en flyter som kan sende data ved havoverflaten.

Oppgave 3.1: MATE Floats 2023

- Designe og bygge en flyter med vertikal profil: 5 poeng.
- Flyteren skal kommunisere med land før den begynner å synke: 10 poeng.
- Flyteren synker til bunn, og stiger opp etter å ha truffet bunnen: 10 poeng.
- Flyteren sender tiden den brukte til land: 10 poeng.

1.4.3 Restriksjoner og Krav

Konkurransen stiller fysiske restriksjoner til ROV-en med tanke på størrelse, vekt, operasjonsmiljø, samt noen elektriske krav.

- **Miljø:** ROV-en skal kunne operere i fersk-, klor- eller saltvann i temperaturer mellom 15°C og 30°C.

- **Materialer:** ROV-en skal kunne operere på minimum 4 meter dybde. Det er også en vektbegrensning på 35 kg.
- **Tjorlengde:** Tjoren må være lang nok til å operere i et område som er 10 meter fra vannkanten og 4 meter dypt. Kontrollstasjonen kan være opptil 3 meter fra kanten av bassenget.
- **Thrustere:** Thrusterene skal ha beskyttelse og minimum innkapslingsgrad på *IP20*. Motorene til thrusterne skal være designet for å operere under vann.
- **Elektrisk:** Arrangøren stiller med kraftforsyning til ROV-en på 30 A og 48 VDC. Konvertering til lavere spenninger må skje på ROV-en. Det skal også være et overbelastningsvern på 150% av det nominelle strømforbruket til ROV-en.
- **Flyter:** Batteriene som brukes ombord må være av typene: *AAA*, *AA*, *A*, *A23*, *C*, *D*, eller *6LR61 alkaliske batterier*. Flyterne skal også være beskyttet av en sikring på 7.5 A. Det skal være en trykkavlastningsventil med diameter på minimum 2.5 cm.

1.5 Kommunikasjon og video

Vår oppgave er å utvikle et robust kommunikasjonssystem for ROV-en Yme. Denne oppgaven består av flere sentrale deloppgaver som må løses. Blant disse oppgavene, skal det sørges for at kretskort kan kommunisere seg imellom, kontrolldata skal sendes til ROV-en og prosessdata skal sendes til kontrollstasjonen. ROV-en skal kunne styres fra land for å løse oppgaver knyttet til konkurransen MATE. Dette stiller krav til at kommunikasjonen mellom ROV og kontrollstasjon skal skje med så lav forsinkelse som mulig, slik at det er mulig for operatøren å styre ROV-en presist. Det skal også monteres flere kamera, et stereokamera i front, et kamera som vender nedover, og et for oversikt over manipulatoren. Kameraene i front, under og for manipulatoren blir plassert for at operatøren av ROV-en skal få et overblikk over operasjonen ROV-en utfører. Bildestrømmen fra kameraene skal tas inn og komprimeres, før den sendes videre opp til kontrollstasjonen for å redusere brukt båndbredde på kommunikasjonslinjen. Kommunikasjonslinjene må ha tilstrekkelig båndbredde både mellom ROV og kontrollstasjon, samt internt mellom kretskort for at ønsket datamengde kan overføres med tilstrekkelig lav forsinkelse.

Det skal også lages et design internt i elektronikkhuset, som skal holde kretskortene og koble de sammen på et modulært vis. Det er essensielt at konstruksjonen av denne løsningen hindrer skade på elektronikken den skal huse, og må designes deretter. Koblingsmetoden skal også legge til rette for viderekobling mot plugger for å redusere ledninger internt. Det skal sørges for kjøling av kretskortene, og at kortene har tilstrekkelig med plass til sine komponenter. To vanntette kamerahus skal også lages slik at det kan posisjoneres for bedre overblikk over ROV-en.

Det skal lages et kretskort for å realisere funksjonene til kommunikasjonen. Dette skal passe inn i det interne designet. En mal for kretskort med noen felles kretsutlegg skal også deles ut til de andre gruppene.

Det skal lages rammeverk for implementering av valgt kommunikasjonsprotokoll på mikrokontrollere for å realisere kommunikasjonen mellom kretskortene i ROV-en. Gruppen har også i oppgave å lage en oversikt over all kommunikasjonen som skal foregå slik at det er mulig for resten av gruppene å implementere ønsket funksjonalitet.

Kapittel 2

Elektronikkhus

Kapitteloversikt

Dette kapittelet handler om elektronikkhuset på ROV-en. Det vil gå gjennom grunngivelser for valg som ble gjort. Ta for seg design og konstruksjon, samt ta en oppsummering av resultatet.

2.1 Problemstilling

Elektronikkhuset har tre ulike oppgaver:

- Huse all elektronikken med tilstrekkelig plass innad
- Beskytte elektronikken mot ytre farer som vann, trykk og slag
- Avlede varmen som blir avgitt av elektronikken i elektronikkhuset

Huse all elektronikken - Inne i elektronikkhuset skal det huses flere kretskort. Disse kortene skal festes på en sikker måte som forsikrer at ingen komponenter har eller kan komme i kontakt med hverandre eller kapslingen. Elektronikkhuset må derfor ha en viss størrelse for å ha tilstrekkelig plass til elektronikken.

Vann, trykk og slag - For at ROV-en skal kunne operere i vann, må den kunne beskytte delene sine som ikke tåler vannet eller trykket. Det er derfor viktig at elektronikkhuset holder seg tett, selv under høyt trykk. Komponenter og elektronikkort er svake for direkte slag. Det er derfor viktig at elektronikken blir minst mulig påvirket av slag på fartøyet.

Varme - Elektronikken vil utvikle varme. Denne må avledes slik at driftstemperaturen ikke blir for høy. Dersom den blir for høy, vil elektronikken være i fare for å slutte og fungere. Elektronikken har en driftstemperatur som ønskes å opprettholdes for optimal drift. Det er vanskelig å drive med varmeavledning når man ikke har mulighet for utskifting av luften inne i elektronikkhuset. Til gjengjeld er elektronikkhuset omringet av vann, noe som gir muligheter for varmeavledning mot vannet.

For de tre oppgavene trenger innkapslingen ulike egenskaper. Basert på hva material den er laget av, vil den ha ulik evne for avledning av varme og styrke mot trykk og slag.

Vekt er også en viktig faktor, da manøvreringsevnen til fartøyet ønskes å være best mulig for å løse oppgaver. Det er også av betydning for konkurransen at vi har et fartøy som er under 35 kg. Dersom fartøyet er under 25 kg, får vi ekstrapoeng - noe vi setter oss som et mål.

For styrke så skal kapslingen i konkurransen ned til en dybde på omtrent 4 meter, likevel ønsker vi å dimensjonere ROV-en for å nå en dybde på 100 meter. Dette krever mye mer av et material, siden vi har krav til dimensjoner for kapslingen.

2.2 Behovsspesifikasjon

- Skal holde elektronikken trygt i beholderen
- Skal holde stereokamera i front
- Skal ha utkikkspunkt for kamera
- Skal være vanntett ned til spesifisert dybde
- Skal ha plass til all elektronikken
- Skal tåle slag på kapsling
- Skal ha innføringer til kabler fra kontrollstasjon
- Skal være mulig å drive vedlikehold på
- Skal avlede varme
- Skal være korrosjonsbestandig

2.3 Funksjonsspesifikasjon

- Skal ikke deformeres på 100m dybde
- Skal kunne avlede 60W med varme med 60℃ i driftstemperatur
- Skal ha en kuppel som har plass til et stereokamera
- Skal ha feste for tilt til stereokamera ved bruk av servo
- Skal ha eget kamerahus for kamera ved bunn
- Skal ha under 15L med oppdrift

2.4 Material- og konstruksjonteori

2.4.1 Teori eksterne deler

2.4.1.1 Styrke

Elektronikkhuset vil oppleve utvendig trykk som presser mot kapslingen. Det er derfor viktig at materialet og formen ikke deformeres eller knekker under trykket. Som nevnt i funksjonsspesifikasjon ønskes det at ROV-en skal tåle en dybde på 100 m. Dersom vi bruker formelen for hydrostatisk trykk (2.1) tatt fra kilde [16], kan vi regne ut hvor mye trykk det er på 100 meters dybde.

$$p = p_0 + \rho g h \tag{2.1}$$

- p : Trykk
- p_0 : Atmosfæretrykket
- ρ : Massetetthet
- g : Akselerasjon
- h : Dybde under vann

Inne i elektronikkhuset er det mulig å anta atmosfæretrykk. Ved å sette inn verdier for ρ, g og h, kan vi finne hvilket trykk det vil være under vannet. Verdier brukt for massetetthet til vann er sjøvann. Siden det er atmosfæretrykk inne i kapslingen, vil denne utgjevne seg med atmosfæretrykket utenfor. Verdier er satt inn og regnet på ligning (2.2)

- ρ : $1020\,\mathrm{kg/m^3}$
- g : $9.81\,\mathrm{m/s^2}$
- h : $100\,\mathrm{m}$

$$\begin{aligned} p &= \rho g h \\ p &= 1020 \cdot 9,81 \cdot 100 \\ p &= 1000.62\,\mathrm{kPa} \end{aligned}$$

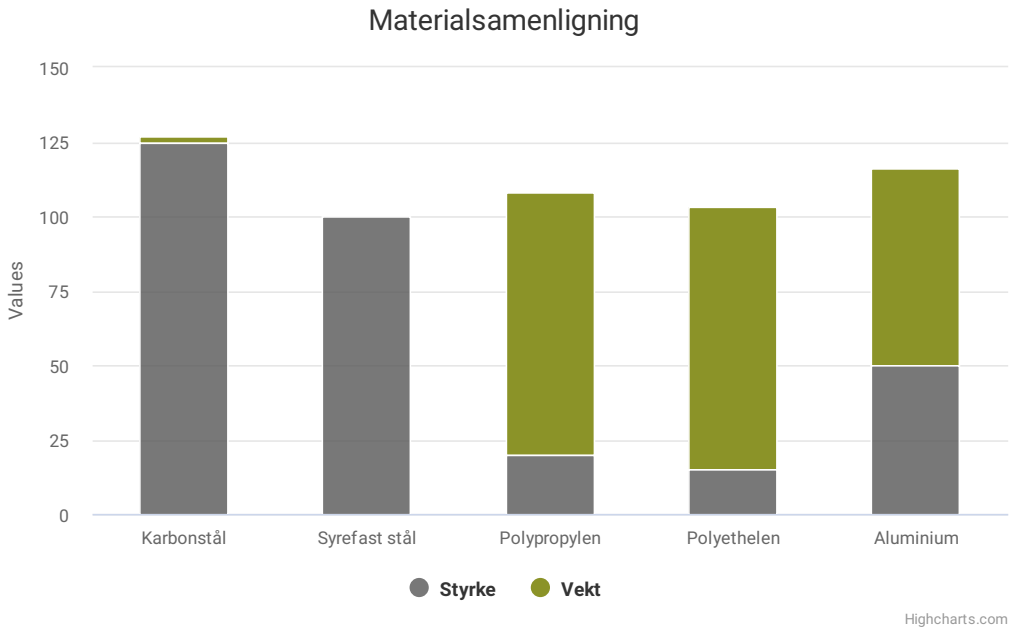
$$p \approx \textbf{10 bar} \tag{2.2}$$

Kapslingen sin flytegrense må derfor være over påkjenningen som kommer ved 10 bar hydrostatisk trykk. Flytegrensen forteller hva som er det maksimale trykket den tåler før den går over til plastisk deformasjon. Dersom kapselen går over til plastisk deformasjon, kan den kollapse eller svekkes over tid. Det er også mulig at komponenter inne i elektronikkhuset kan bli ødelagt av en deformasjon, da komponenter inne i kapslingen ikke tåler kreftene som blir påført.

Flytegrense -[17]

Flytegrense er en teknisk, praktisk grenseverdi som forteller hvor mye metaller og legeringer tåler av belastninger før det oppstår en observerbar plastisk deformasjon ved strekk eller trykk.

Materialene i figur 2.1 er sammenlignet med både flytegrense og egenvekt. Siden det er en klar relasjon mellom veggtykkelse og flytegrense, er det mest relevant å se hvilken som presterer best på begge områder. Dersom materialet har lav flytegrense og vekt, vil det være mulig å bruke mer av materialet for å få den styrken som ønskes for kapslingen.



Figur 2.1: Samenligning av materialer, figuren nytter visuell framstilling, der materialet skårer på styrke og hvor lav egenvekt materialet har. Syrefast stål er satt som standard med 100 i styrke og 0 i vekt. Verdier er tatt fra kilde [18]

Fra figuren kommer karbonstål best ut, og syrefast kommer dårligst ut. Matrialene ligger likevel veldig likt, når de blir justert for vekt. Selv om polypropolen og polyethylen har lavere flytgrense, er egenvekten deres lav nok til at den kompenserer for dette ved at man kan ha en større veggtykkelse for økt styrke. Karbonstål og aluminium kommer likevel best ut i samenligningen.

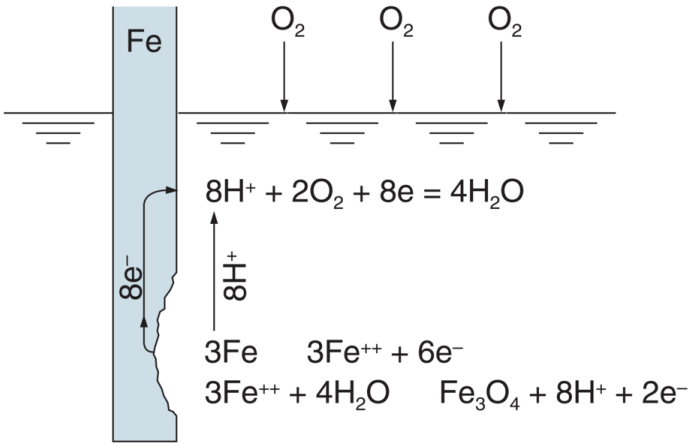
2.4.1.2 Korrosjonsbestandighet

Materialet må også være korrosjonsbestandig. Dette er fordi korrosjon vil ha betydelig effekt på styrken som elektronikkhuset har over tid.

Korrosjon - [19]

Korrosjon er oppløysing av metalliske materiale (metall og legeringar), hovudsakleg på grunn av spontane elektrokjemiske reaksjonar. Rust er eit folkeleg namn på produktet av korrosjon av jern.

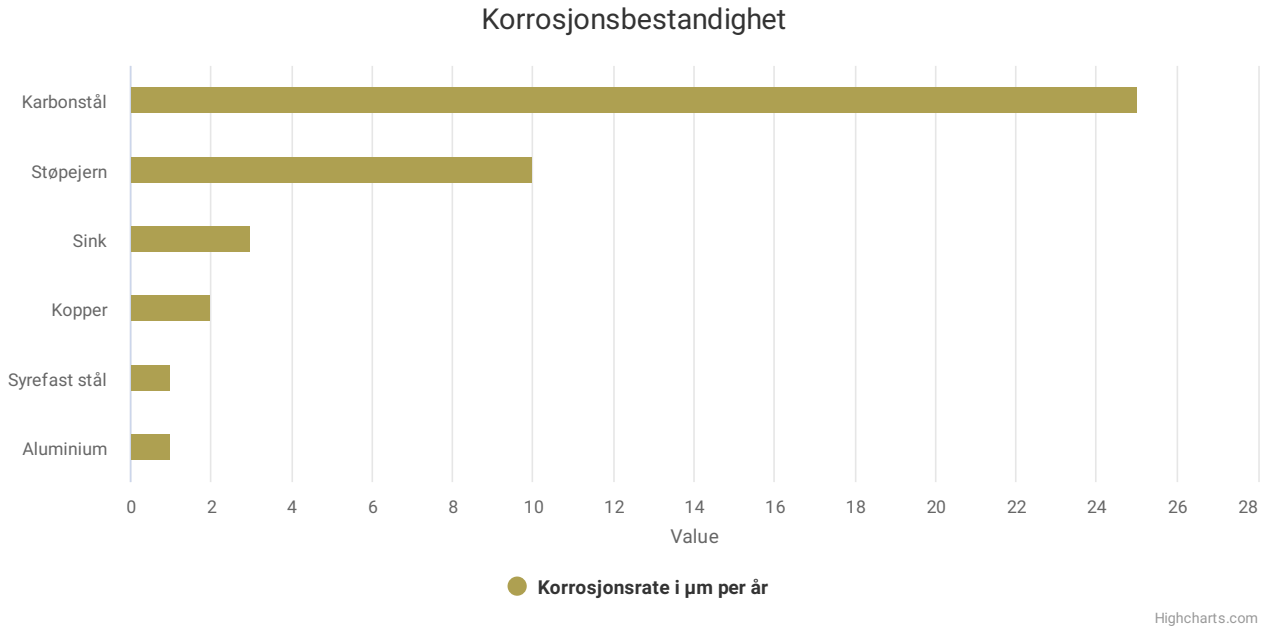
Ved korrosjon i maritimt er jern spesielt påvirket. Jernet vil da gjennomgå en oksidasjonsreaksjon. Denne reaksjonen fjerner materiale fra jernet som er i kontakt med vannet. Se figurer 2.2a og 2.2b for visuell framstilling av korrosjon og rust.



(a) Kjemisk reksjon for korrosjon [19]

(b) Eksempelbilde for rust [20]

Selv om karbonstål har det beste forholdet mellom flytegrense og egenvekt, er det et lite korrosjonsbestandig material. Maritimt forhold tærer fort på karbonstål, og materialet vil fort bli skadet. På figur 2.3, er det vist hvordan ulike materialer sin korrosjonsbestandighet er i et maritimt miljø. ROV-en skal tåle maritimt miljø med minst mulig skade. Alle materialeler vil oppleve en viss korrosjon ved i dette miljøet, men for å redusere denne faktoren er materialer med god korrosjonsbestandighet å foretrekke.



Figur 2.3: Korrosjonsrate for ulike matrialer i et maritimt miljø [21]. Tall er ikke nøyaktige, men brukes til visuell fremstilling.

Ut fra figuren er karbonstål et lite korrosjonsbestandig materiale. Materialer som aluminium og syrefast stål er materialer som tåler slike miljø bedre, og har en lav korrosjonsrate.

2.4.1.3 Form og dimensjoner

Basert på veggtykkelse og form, vil kapslingen tåle ulike belastninger. Et krav for kapslingen er også at den skal romme elektronikken. Det må derfor være dimensjonert slik at alle komponenter får tilstrekkelig plass inne i elektronikkhuset.

2.4.1.4 Varmeoverføring

Elektronikk som er i drift genererer varme, og hvis varmen ikke fjernes - kan den ødelegge eller begrense ytelsen til elektronikken. I tilfelle av en undervanns-ROV er det ikke mulig å bytte ut luften inne i kapslingen, og det er derfor nødvendig å ha en annen måte å fjerne varmen på. Heldigvis er vannet som omgir ROV-en vanligvis kaldt nok til å fungere som et kjølemiddel og holde temperaturen inne i kapslingen på driftstemperaturen til elektronikken. Så selv om det kan være en utfordring å overføre varmen fra elektronikken til utsiden av kapslingen, kan vannet rundt ROV-en være en effektiv måte å kjøle ned elektronikken på.

Det er ønskelig å få varmeoverføringen så høy som mulig. Både mellom luften i elektronikkhuset til kapslingen, men også fra kapslingen til vannet på utsiden. Se figur 2.4 for en illustrasjon for dette.

Figur 2.4: Overføring mellom luft inne i elektronikkhuset, kapslingen og vannet utenfor.

For å regne varmeoverføring, kan vi bruke Newtons kjølelov [25]. Denne er vist i ligning 2.3.

$$\begin{aligned} \dot{Q} &= h \cdot A \cdot (T_1(t) - T_2(t)) \\ \dot{Q} &= h \cdot A \cdot \Delta T(t) \end{aligned} \tag{2.3}$$

- Q : Varmeoverføring
- h : Varmeoverføringskoeffisient
- A : Overfalteareal i kontakt
- $\Delta T(t)$:Temperaturforskjell mellom objekt en og to

Varmeoverføringskoeffisienten mellom materialene blir påvirket av mange faktorer som hvordan overflaten er, bevegelser i veske og gass og mer. Varmeoverføringskoeffisienten er derfor vanskelig å finne for et spesielt tilfelle, og vil som regel variere basert på overflate og forhold rundt omgivelser. Varmeoverføringskoeffisienten blir likevel påvirket av den termiske konduktiviteten til et materiale. Materiale med bedre termisk konduktivitet, vil gi bedre varmeavledning [23].

Termisk konduktivitet - [26]

Termisk konduktivitet er mål for evnen et stoff har til å lede varme. SI-enheten for termisk konduktivitet er W/(K·m), altså watt/(kelvin-meter). Termisk konduktivitet kalles også varmeledningsevne.

Ulike materialer har ulik termisk konduktivitet. Siden det skal avledes varme ut fra elektronikkhuset, må den termiske ledeevnen til materialet være tilstrekkelig god nok til å avlede varmen som blir avgitt fra elektronikken. For å holde driftstemperaturen lavest mulig, er det viktig å velge et materiale med høy termisk konduktivitet. Høyere termisk konduktivitet vil gi høyere samlet varmeoverføringskoeffisienten for varmeavledningen mellom to material.

I tabell 2.1, er det sammenlignet termisk konduktivitet hos forskjellige material.

Material	**Termisk konduktivitet W/m, K**
Stål	50.2
Aluminium	205.5
Polyethelen	0.5
Polypropolen	0.11

Tabell 2.1: Samenligning av termisk konduktivitet. Tall tatt fra kilde [27]

Polypropolen og polyethelen har dårlige egenskaper for varmeavledning. Aluminium derimot har bra egenskaper for varmeavledning. Siden dette er en viktig egenskap til elektronikkhuset, vil dette være en viktig faktor for valg av materiale.

Varmeoverføringskoeffisienten er varierende for ulike material, og blir påvirket av hvilke tilstander som er rundt overføringen. For eksempel dersom væske og gass strømmer over en overflate vil varmeoverføringskoeffisienten være større mellom de to materialene, enn om gassen eller væsken er stasjonær. En vifte inne i elektronikkhuset vil lage en strøm av luft som øker varmeoverføringskoeffisienten mellom luften og kapslingen.

2.4.2 Teori interne deler

2.4.2.1 Konstruksjonsmetode

Delene som skal produseres må være presise, og det må derfor velges en produksjonsmetode som muliggjør presis produksjon. Dersom det er for lave toleranser[1], kan elektronikken løsne, noe som kan forårsake skade på elektronikken.

Produksjonsmetoder kan være følgende:

- Dreiing
- Fresing
- Termoforming
- 3D-printing

Termoforming - [29]

Termoforming er det å forme et oppvarmet halvfabrikat av et termoplastisk materiale.

Dreiing og fresing er gode metoder for konstruksjon, men krever et spesielt design for å fungere. Det koster også mer enn de andre, da det fjerner materiale fra legemet.

Termoforming er en effektiv måte å konstruere, men krever enten spesielt utstyr, eller enkle design som skal endres fra halvfabrikata.

Dersom delen allerede er modellert, er det klart enklest å 3D-printe. Dette er en vanlig måte å prototype deler på grunn av den lave kostnaden ved produksjon. Selv om det ofte er tidskrevende, skjer prosessen uten at det krever manuelt oppsyn.

3D-printing er en teknologi som har økt i funksjonalitet og effektivitet de siste årene. Det er en konstruksjonsmetode som bruker en overflate og bruker lagvis konstruksjon.

Alle design blir delt inn i flere lag. Basert på hvor høy oppløsning en ønsker, kan man redusere i lagstørrelse for å øke oppløsning. Selv om designet kommer som en 3D-modell som har X-, Y- og Z-akser - vil en *slicer*[2], gjøre dette om til mange lag som kan konstrueres på to akser.

En samenligning for å forstå 3D-printing bedre er å se på et eksempel fra byggebransjen. Når en bygger et hus med murstein, legges det **lagvis** på med murstein helt til den ønskede høyden er oppnådd.
Svakheten med muring er at man må ha **støtte** under for å mure over. Det kan mures litt utenfor det samme laget, så lenge balansepunktet til den nye mursteinen er innenfor arealet til den forrige. På samme måte er svakheten med 3D-printing at den ikke kan printe der det ikke er noe materiale under som den kan bygge på.

Denne typen 3D-printing er ***FDM*** - ***F**used **D**eposition **M**odeling* . Det finnes også andre typer for 3D-printing som ***SLA og SLS***, men de er dyrere metoder og mindre tilgjengelig på UiS.

[1] Toleranse er innenfor teknikk en betegnelse på tillatte avvik fra nøyaktige mål, for eksempel i dimensjonene til en utskiftbar maskindel. [28]

[2] Et program som deler 3D-modellen om til mange 2D-lag

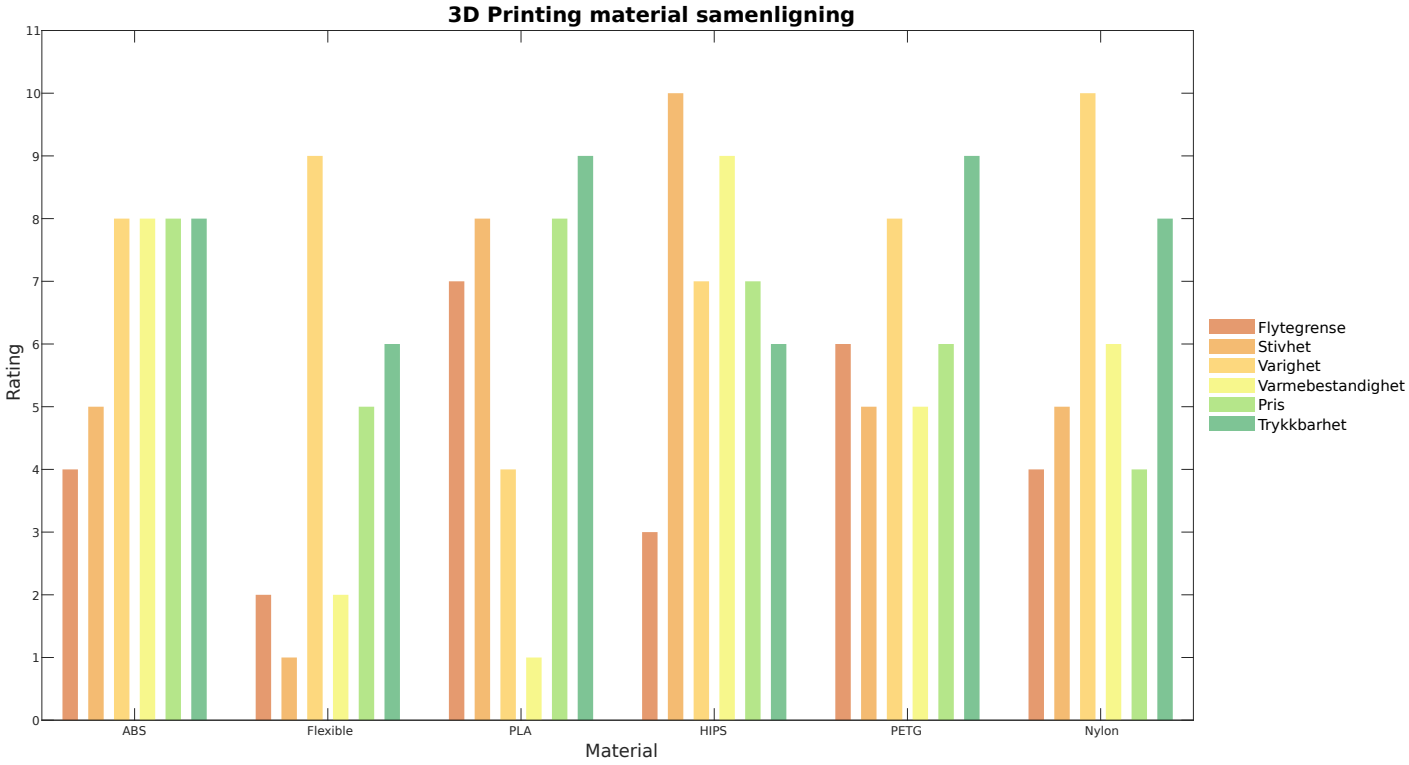
2.4.2.2 Styrke og materiale

Dersom kapslingen tåler det utvendige trykket vil de interne delene av elektronikkhuset ikke bli påvirket av det eksterne trykket, og trenger derfor ikke den samme styrken som de eksterne delene. De innvendige delene skal brukes til å feste elektronikkortene til rammeverket til kapslingen. Disse kortene må likevel beskyttes mot bråe bevegelser som kan forekomme.

For å unngå at det er elektrisk forbindelse mellom kapslingen og elektronikkortene er det mulig å bruke et materiale som er isolerende. Forbindelser mellom elektronikkortene og kapslingen kan føre til at uønskede potensial kan komme utenfra og påvirke elektronikken inne i elektronikkhuset.

For å festes i elektronikkhuset må materialet være fast, men fleksibelt. Elektronikk tåler dårlig bråe bevegelser, og en naturlig fjøring i materialet gir en god måte å redusere faren for dette.

Ikke alle materialer kan brukes til 3D-printing. Selv om det i senere tid er blitt me populært med printing av metaller, er det fortsatt ulike plastikker som er vanlig. På figur 2.5, er ulike materialer og deres egenskaper knyttet til 3D-printing fremstilt.



Figur 2.5: Samenligning av materialer for 3D-printing, verdier tatt fra kilde [30]

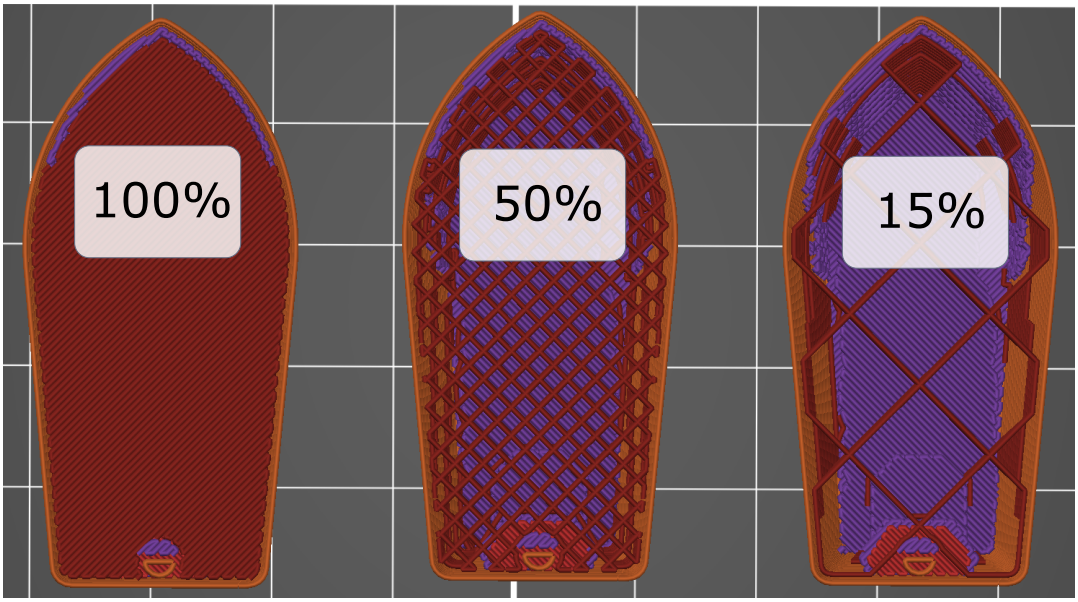
Fra grafen kan man se at ulike typer plastmaterial har ulike kvaliteter, og må velges basert på hva prosjektet ønsker og krever. En viktig faktor er *Trykkbarhet*, noe som sier noe om hvor enkelt det er å få et solid og bra resultat fra en print. Rimeligere printere er ofte bare laget for å printe materialer som er lette å printe.

2.4.2.2.1 Bruk av 3D-printing for undervannsbruk

Denne underseksjonen tar deler av materialet fra tidligere bachelor for kommunikasjon [31].

3D-printede produkt er ikke kjent for å være brukt mye til undervannsbruk. Likevel er det flere komponenter som kan printes, da det er mindre krav til noen av kompoentene. Det blir sett i større grad, og det er til og med noen som har eksprimentert med å lage en 3D-printet undervannsdrone. ***CPS -Custom Printed Submarine***[32], er en drone som består av kun printede eksterne deler. Denne baserer seg på et akrylrør for interne deler, men bruker 3D-printing i kombinasjon med epoksy for å vanntette deler. Dronen menes å ha en dybderating på 85 meter, uten lekkasje inn til elektronikken. Selv om mye ikke er vanntett, er det mange av delene som heller ikke trenger å være det, så lenge det ikke holdes luft inne i kapslingen.

3D-printing kan bruke ulik grad av innfyll, dette bestemmer hvor mye av materiale som er solid i den ferdige delen. Et innfyll av 15% vil for eksempel resultere i et solid skall, men med 85% luft inne i delen. En figur av dette vises på 2.6



Figur 2.6: Samenligning av innfyll i 3D-printing.

Styrken på en del med 100% innfyll vil være mye bedre enn en med 15%, dersom de testes på statisk trykk. Likevel er det ikke like vanlig at 3D-printede deler må tåle statisk trykk, og det er derfor vanlig å bruke en lavere mengde innfyll for å få en den som er sterk nok med mindre material.

Det er også mulig at luft fanges inne i modellen selv om det er 100% innfyll, da linjer med materiale blir lagt på siden av hverandre med lite til ingen overlapp. Dette resulterer i små luftbobler som ligger inne i delen. Denne innestengte luften vil lage en trykkforskjell mellom utsiden av delen og internt i delen. Den vil da skape svakheter i konstruksjonen. Selv om dette er en svakhet, vil påkjenningen ved mindre innfyll være mye større. Som nevnt i fjorårets bachelor, vil påkjenningen ved maks bassengdybde, som er 4 meter - være lite alvorlig for den 3D-printede delen. Trykkforskjell på 4 meter vil være omtrent 0.4 bar. Til samenligning er bruddstyrken til materialet ABS på 400 bar [30]. Selv om bruddstyrken er høyere, vil likevel **lagene** på delen være det som bryter først.

Hvert lag ved 3D-printing skal legges over hverandre og smeltes ilag. Ved printing vil det likevel være litt variable tilstander for hvert lag, for eksempel kan omgivelsestemperatur variere noe. Dette gjør at det er fare for at laget ikke smeltes sammen like bra. Dette fører til svakheter i konstruksjonen, noe som gjør at hvert lag er et eventuelt bruddanvisning[3].

[3] "Bruddanvisning er sprekker, riss eller andre strukturelle feil i et materiale eller en konstruksjonsdel som kan resultere i brudd.[33]"

Deler vil likevel få en vanntetthet ved at det må penetreres flere lag igjennom delen før den lekker vann. Materialene ABS og PETG er foretrukene plasttyper for bruk i vann, da de er vannresistente. PLA er en vanlig plasttype til 3D-printing, men materialet absorberer vann og egner seg derfor dårlig som vanntett.

2.5 Erfaringer fra tidligere

Her vil det bli sett på erfaringer som tidligere grupper har rapportert fra tidligere år. Dette tas med for å få lærdom fra tidligere feil og for å ta med seg det som er bra videre. Hele denne seksjonen er basert på rapporten med lignende oppgave i fjor [31].

2.5.1 Plass

Designet i fjor ble laget for å ha plass til fem stk elektronikkort. Det var totalt tre elektronikkort som ble montert. De ble montert i en bakplate og lå horisontalt langs ROV-en. Det var også ledig rom og plugger til flere kort, noe som er bra for utvidelse.

Likevel ville det vært vanskelig å legge til flere kort, da avstanden mellom PCI-kontaktene var for liten for å få plass til mer. Noen av kortene hadde påmonterte komponenter som var så store at det hadde vært et fysisk hinder for andre kort dersom de skulle monteres i ledige PCI-kontakter.

Tas med videre:

- Omtrent samme mål på elektronikkhus, da dette var tilstrekkelig for den elektronikken som skal monteres i elektronikkhuset
- Mulighet for utvidelse av flere kort

Endres på:

- Horisontal montering av elektronikkort omgjøres til vertikal. En vertikalt montering vil gi mer modulært design og gi mulighet for kort med mindre størrelse.

2.5.2 Konstruksjon

Det ble i fjor konstruert lokalt på UiS, da det var verksted som egner seg til maskinering tilgjengelig for bruk. Materialer var av typen halvfabrikat[4], for å redusere svinn og gjøre det enklere å maskinere til ferdig form.

Rørkroppen var i form av rør i aluminium. Dette ble så dreid[5] innvendig og utvendig for å få rette dimensjoner. På bakstykket og framstykket ble det brukt både dreiing og fres[6] for å maskinere ut innmat fra et solid stykke med aluminium. Mye av boringen ble også tatt med fres.

[4] "Halvfabrikat, industrivare som er bearbeidet, men som ikke er et ferdig utviklet produkt.[34]"
[5] "Dreiing er en enkel måte for å presist fjerne materiale fra et rotasjonslegeme [35]."
[6] "En fres vil fjerne materiale ved å bruke et roterende skjæreverktøy [36]."

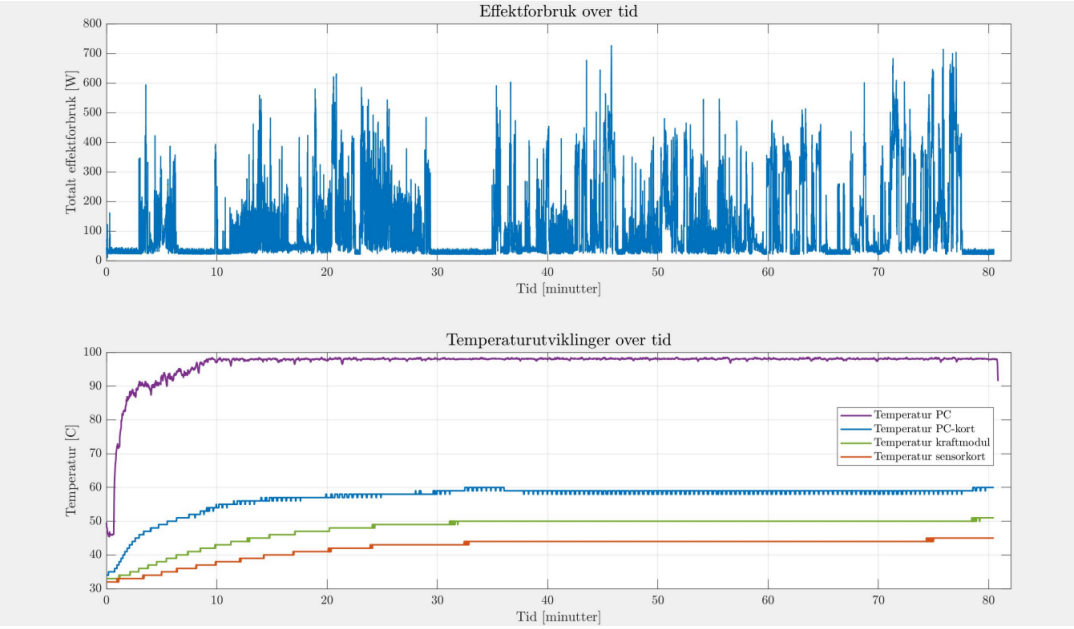
Designet var enkelt å maskinere, noe som resulterte i mindre tid brukt på verkstedet. Designet tok hensyn til hvilke halvfabrikat som er tilgjengelig på markedet, og fant en fin balanse mellom å modifisere og å kjøpe ferdig.

Tas med videre:

- Mulighet for å maskinere på UiS sitt verksted
- Design som kan lages fra halvfabrikata

2.5.3 Varme

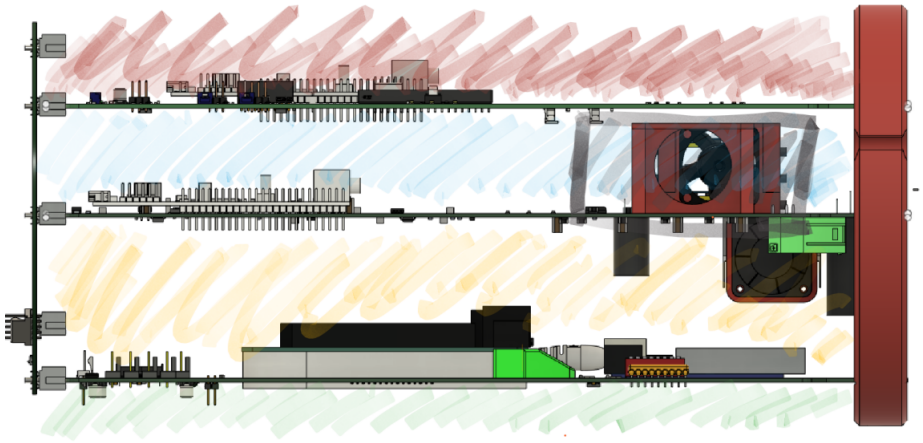
Det ble gjort en test av varme og en beregning. Denne testen vises i figur 2.7. Den varte i omtrent i 80 minutter, og ble utført over vann. Testen bestod av at det ble gjort bildebehandling og sporadisk pådrag på thrustere. I testen vises det at temperaturen øker drastisk ila. de første 10 minuttene i området rundt PC-kortet. Stabiliseringstemperaturen ligger på omtrent 60 °C, noe som er over driftstemperatur. Temperaturen målt inne i PC-en er mye høyere enn dette. Når den når omtrent 98 °C, stabiliserer temperaturen seg. I rapporten skriver de en antagelse om at det grunnes at hastigheten justeres ned for å kompensere for økningen i temperatur. Den vil på den måten holde temperaturen under det maksimale nivået(T_{jmax}[7]) på 105 °C



Figur 2.7: Viser temperaturutvikling fra ROV-en i fjorårets design. [31]

De andre områdene ligger på 50 °C eller under, og holder en stabil drift. Slik som elektronikkortene ble plassert i fjor, var det lite rom for gjennomstrømming mellom kortene. Dette var hensikten, og skulle lage ulike *varmesoner*. Varmesonene kan ses på figur 2.8 i rød, blå, gul og grønn. Det var montert vifter for å sirkulere luften som var innad i tre av de fire varmesonene. Formålet med dette var å øke varmeoverføringen fra luften til aluminiumskapslingen. Designet med liggende elektronikkort og varmesoner står at det er tatt fra tidligere år som har lignende effektbudsjett.

[7] "Høyeste driftstemperatur [37]"



Figur 2.8: Viser varmesoner fra ROV-en i fjorårets design. [31]

Varmeoverføring er noe som må tas hensyn til når en ROV skal utvikles. Det er tydelig at dette var et gjennomtenkt design med god utføring. Aluminiumet gir god ledeevne for luften til å avlede varme fra elektronikken. Bruk av vifter er gjort i alle soner som når en høy temperatur. Kjøleribber er montert på komponenter som avgir høy effekt. Testen der driftstemperaturen på PC-en er nærme tJMax, ble utført over vann. Det kan antas at dersom dette var under vann, ville resultatet gitt en lavere makstemperatur.

Behovet for varmesone kan anses å være mindre viktig. De ulike kretskortene i elektronikkhuset anses å være like kritisk for funksjonen av ROV-en, og at det blir en samlet driftstemperatur anses som positivt. Dette er såfremt den holder seg under driftstemperaturen til elektronikken som er montert. Varmesoner er noe som kan være bra dersom varme utvikler seg til brann i elektronikkhuset, siden dersom brannen er isolert vil det være vanskeligere for den å spre seg. Likevel er en brann i elektronikkhuset usannsynlig og konsekvenser av dette vil være store uansett om brannen begrenses eller ikke.

Tas med videre:

- Aluminium som materiale for varmeavledning
- Stort overflateareal for varmeavledning
- Vifter plassert rundt varme områder i elektronikkhuset
- Kjøleribber på komponenter med høy effekt

Endres på:

- Inndeling i varmesoner
- Bruk av PC med høy effekt

2.5.4 Styrke

Styrkeberegningene fra rapporten er gjort i *Fusion 360*. Der er det mulig å simulere trykk mot konstruksjoner som er laget i programmet. Fra denne testen blir det lest verdier med sikkerhetsfaktor på hver del. Denne omfatter alle delene som er konstruert, og tester fra 6 til 10 bar. Fra testen er alle

deler utenom *bakplaten* over en sikkerhetsfaktor på 3. Bakplaten vil derfor være det svakeste leddet i beregningene. Denne er likevel over en sikkerhetsfaktor på 3 ved 6 bar. Noe som er innenfor målet fra i fjor da den skulle tåle en dybde på 50 meter.

Dette viser at tykkelsen på delene av elektronikkhuset var korrekt beregnet for kravene de skulle møte. Materiale ble også fjernet av rørkroppen for å gjøre elektronikkhuset lettere, slik at kapslingen ikke var overdimensjonert.

Tas med videre:

- Simulering av design i Fusion 360
- Teste på ulike typer trykk
- Fjerne materiale slik at elektronikkhuset er lavest mulig vekt

2.6 Valg av tilvirkningsmetode

I denne seksjonen utdypes det hva som er lagt vekt på ved bestemmelsen av tilvirkningsmetode og materiell. Det belyses hvilke faktorer som ligger bak begrunnelsen og hvorfor de er viktige.

Det følger først en del om eksterne deler[8], så kommer det en del om de interne delene[9].

2.6.1 Kapsling

2.6.1.1 Design

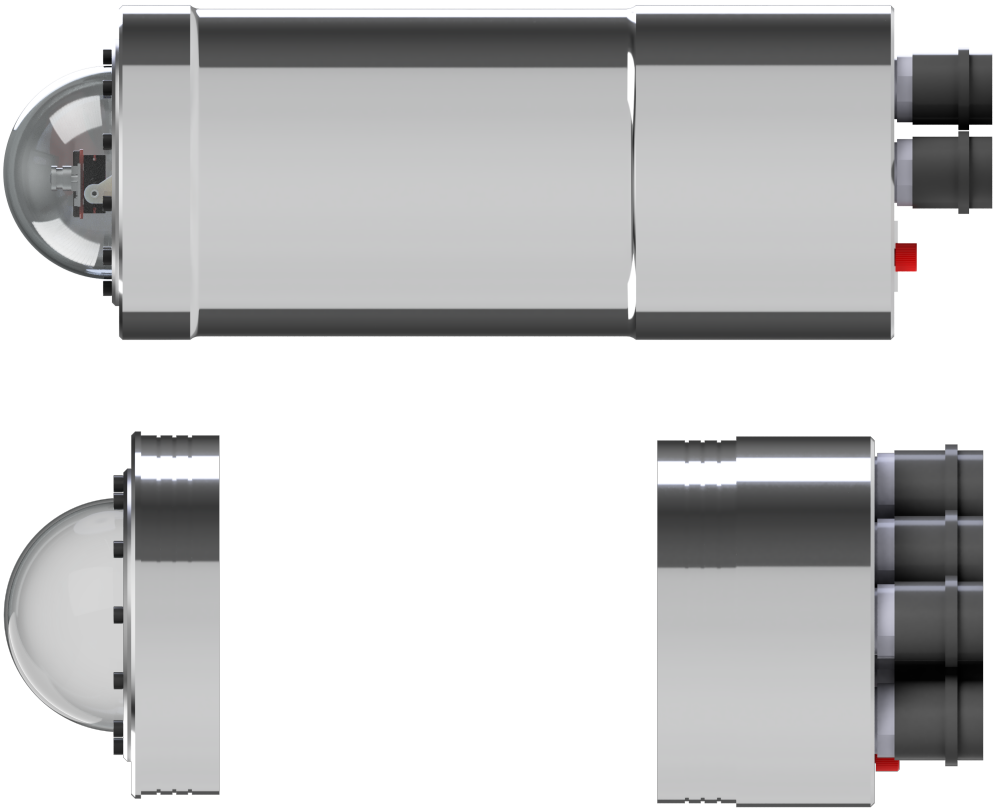
Designet som ble valgt var en **rund sylinder**. Dette designet er solid for utvendig trykk, og enkelt å maskinere ut fra halvfabrikata. Det er også rimelige halvfabrikata å få i form av rør og bolt. Ilag med maskingruppen ble også dimensjonene for kapslingen satt til:

- **Lengde:** 435,00 mm
- **Innvendig diameter:** 180,00 mm
- **Utvendig diameter:** 182,00 mm
- **Volum:** 11,07 Liter
- **Innvendig overflateareal:** 296 880 mm^2

Kapslingen er så designet for å ha to ender på hver side som tettes med O-ringer rundt en hylse som presses inn i rørkroppen til elektronikkhuset. Den ene vil være framenden som holder kuppelen, og den andre vil være bakenden som holder alle gjennomføringene. Dette designet er i stor grad inspirert fra forrige års design med lignende utforming [31].

Et bilde av ferdig design er vist på figur 2.9.

[8] Deler som er i berøring med vannet.
[9] Deler som er på innsiden av kapslingen.



Figur 2.9: utforming av elektronikkhus kapslingen

O-ringene er skjært spor til slik at de kun komprimeres med 20% av original størrelse. På denne måten presser de mot kapslingen slik at vannet ikke trenger inn. Kuppelen er satt i front og festet med en ring som presser kuppelen inntil framdelen. Bakdelen har gjennomføringen for kontaktpunkt til plugger som er satt i enden. Her er det satt av litt ekstra plass slik at det er mulig å føre gjennom ledninger til elektronikken på innsiden.

Valget av gjennomføringer var et fellesvalg for gruppen, og falt på `MacArtney` sine `SubCon - Connectors`. Det ble valgt flere ulike typer plugger etter formål til de forskjellige gruppene.

2.6.1.2 Materiale

Etter evalueringen av materialer sin styrke, korrosjonsbestandighet, varmeavledningsevne og tilgjengelighet - ender valget på **aluminium**. Dette materialet er også nokså rimelig i forhold til noen av alternativene, og er enkelt å få tak i som halvfabrikata.

Ved å bruke dimensjonene til elektronikkhuset kan vi regne ut hvor stor varmeavledning aluminiumskapslingen vil ha til luften inne i huset. Det er tatt utgangspunkt i at varmeavledningen fra vannet til aluminiumen er god nok til å anta at kapslingen er lik vanntemperaturen. Utregningen tar utgangspunkt i formel for varmeoverføring (2.3), gitt i tidligere seksjon om teori 2.4.1.4. Først må vi fylle inn verdier for:

- h : $5\ \mathrm{W/(m^2K)}$
- A : $0.296\,\mathrm{m}^2$

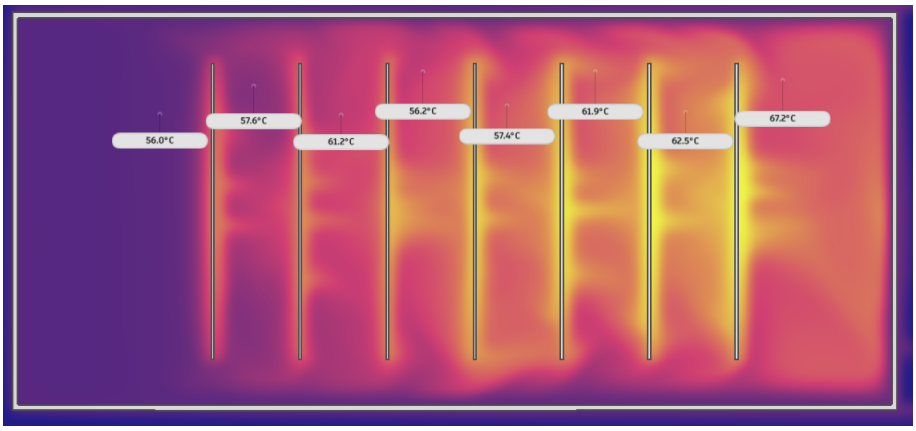
- $\Delta T(t)$:Antar vanntemperatur til å være 20℃ og makstemperatur er satt til 60℃, som gir 40℃ i temperaturforskjell.

Verdien for varmeoverføringskoeffisienten er tatt fra kilde [38]. Verdien for gas ved atmosfærisk trykk er oppgitt i området 5 til 35 W/(m^{2}K). Det er brukt det laveste, men det antas at denne egentlig er høyere da det er vifter som sirkulerer luften inne i kapslingen.

Se ligning (2.4) for utregning.

$$\begin{aligned} \dot{Q} &= h \cdot A \cdot \Delta T(t) \\ \dot{Q} &= 5 \cdot 0,296 \cdot 40 \\ \dot{Q} &= 59.2\,\text{W} \end{aligned} \tag{2.4}$$

Her ser vi at funksjonsspesifikasjonen med 60W er tilnærmet oppnådd ved å bruke den laveste varmeoverføringskoeffisient som var oppgitt. Dette stemmer bra med en enkel simulering som er gjort i Fusion 360. I figur 2.10 er det gjort en simulering av en forenklet versjon av elektronikkhuset, der overflatearealet er det samme, og der det er satt inn syv plater som genererer 10W hver. Det er da forventet at temperaturen er over 60℃ i elektronikkhuset, siden den totale effekten avgitt er 70W.



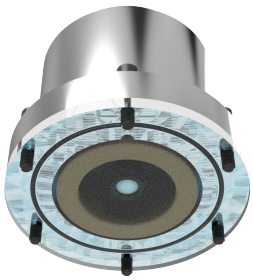
Figur 2.10: Varmetest der omgivelsetemperatur er satt til 20℃. Simuleringen er gjort i luft, ikke i vann

Fra figuren ser vi at temperaturen mellom kortene er mellom 55℃ til 70℃. Kortene er satt med 40mm avstand mellom hverandre, men har ingen luftstrøm fra vifter som veksler ut luften. Dette vil antageligvis ha stor effekt på varmeavledningen.

2.6.2 Kamerahus

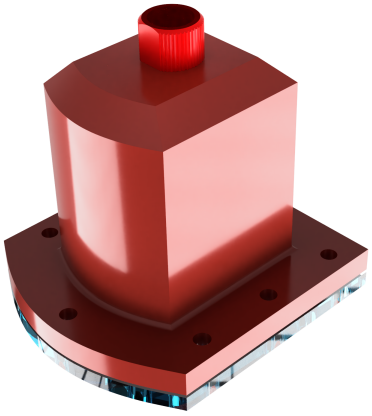
Det skal lages to kamerahus for å huse et kamera som vender mot bunnen, og et kamera som er rettet mot manipulatoren. Kamerahusene blir like og lages i aluminium likt som elektronikkhuset. Huset dreies ned fra et stykke aluminium, slik at det er hullrom inne. Det lages så en lippe for å ha gjennomgående skruer for å feste platen av polykarbonat. Se figur 2.11a og 2.11b.

(a) Topp med Blue Robotics cable penetrator.



(b) Bunn med polykarbonat for gjennomsiktighet.

Det lages også et kamerahus for manipulatoren som kan monteres ved behov. Dette ble printet, da det ikke var noe behov for at dette skal tåle mer enn fire meter dybde. Bilde av dette vises på figur 2.12.



Figur 2.12: Printbart kamerahus.

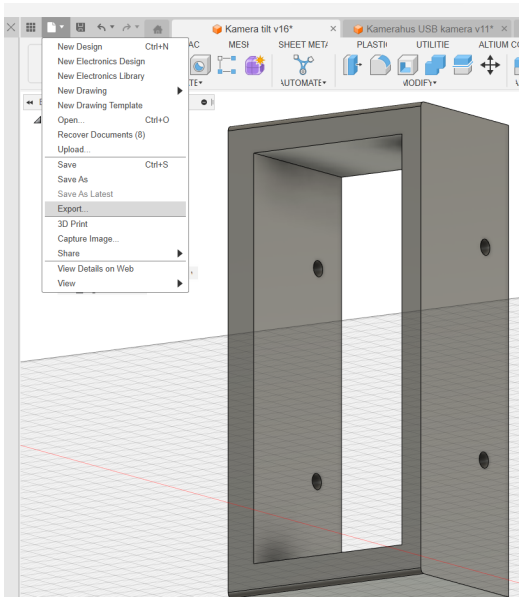
Dette kameraet printes i **ABS** med 60% innfyll. Designet er tatt fra samme design for kamerahus i aluminium, og bruker en polykarbonatplate for gjennomkikkshull.

2.6.3 Interne deler

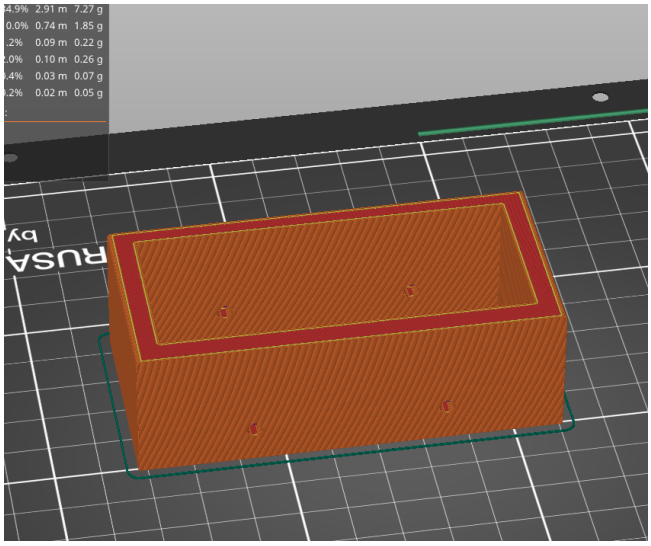
I denne seksjonen vil det gås igjennom hvilke valg som ble gjort av de interne delene til elektronikkhuset samt kamerahuset.

2.6.3.1 Materiale og konstruksjonsmetode

De interne delene har vi valgt å lage med 3D-printing så fremt det er mulig å lage på denne måten. Siden alt ble 3D-modellert, var dette en enkel måte å realisere design som allerede er laget. For å gjøre dette kan en 3D-fil eksportes om til en **STL** eller **3MF** fil. Programmet for CAD var *Fusion 360*. Eksportering av denne filtypen er enkelt å gjøre i Fusion, og vises på figur 2.13a.



(a) Eksportering fra Fusion 360



(b) Importert i PrusaSlicer

Du velger filen og hvilke deler som skal eksporteres ved å velge hva som er synlig, og derretter eksportere tegningen til valgt filtype. På figur 2.13b er filen vist i programmet *PrusaSlicer*, etter den er importert inn i dette programmet. Dette programmet gjør filen om igjen til filformatet `gcode`, som igjen er det 3D-printeren trenger for å legge lagene til delen. Printerene som blir brukt til printing, er av typen **Prusa i3 MK3S+**. Denne vises på figur 2.14a. Noen deler ble også printet på Joar sin egen Voron V0.1, som vises på figur 2.14b.

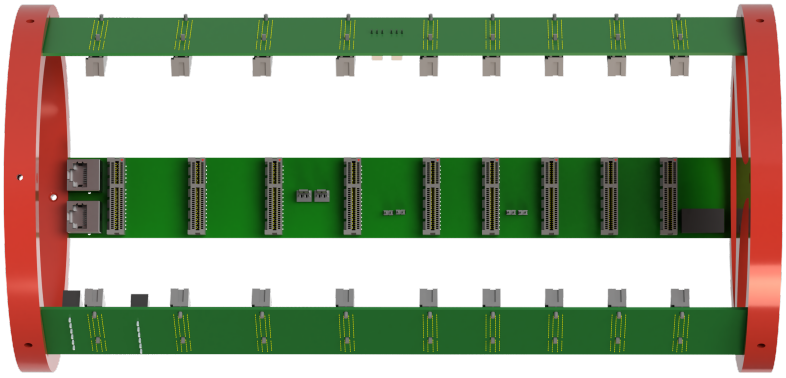
(a) Prusa i3 MK3S+ [39]

(b) Voron V0.1 [40]

Printeren fra Prusa-en har et printområde på 25x21x21cm, og har derfor stort nok område til å printe alle deler vi trenger. Den er tilgjengelig for oss å bruke på 3D-printerlabben og kan printe **PETG**, **PLA** og **ABS**. Dersom den skal printe ABS, må det brukes en som er i lukket kammer for avgasser. Voron V0.1 har et mindre printareal, men siden den har innkapsling er den bedre for printing av ABS. Selv om gassen fra ABS ikke er regnet som giftig, har den negative effekter som lukt, sløvhet og hodepine for personer som blir utsatt for den[41]. Et kammer reduserer dette.

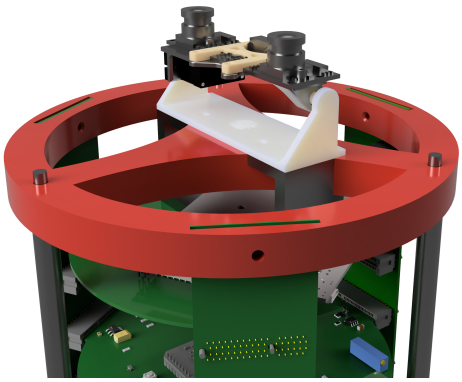
2.6.3.2 Design og festeringer

Designet som ble valgt er et design som består av tre bakplater, der kretskortene skal plugges i de tre bakplatene som holder kretskortene. Kretskortene blir bedre beskrevet i kapittel om kretskort 6. Designet av dette kan også sees på figur 2.15.

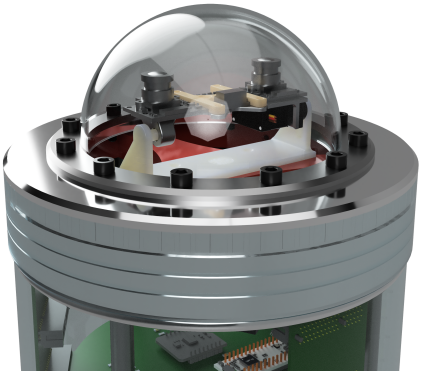


Figur 2.15: Internt design av elektronikkhus.

Det ble brukt to 3D-printede ringer på hver ende av bakplatene. Disse brukes til å feste kortene i, og er laget med små toleranser. På denne måten vil bakplatene ligge fast i ringene ved hjelp av friksjon. For å feste bakplatene ytterligere, er det laget hull til en skrue for å sikre at denne ikke sklir ut. Se figurer 2.16a og 2.16b.



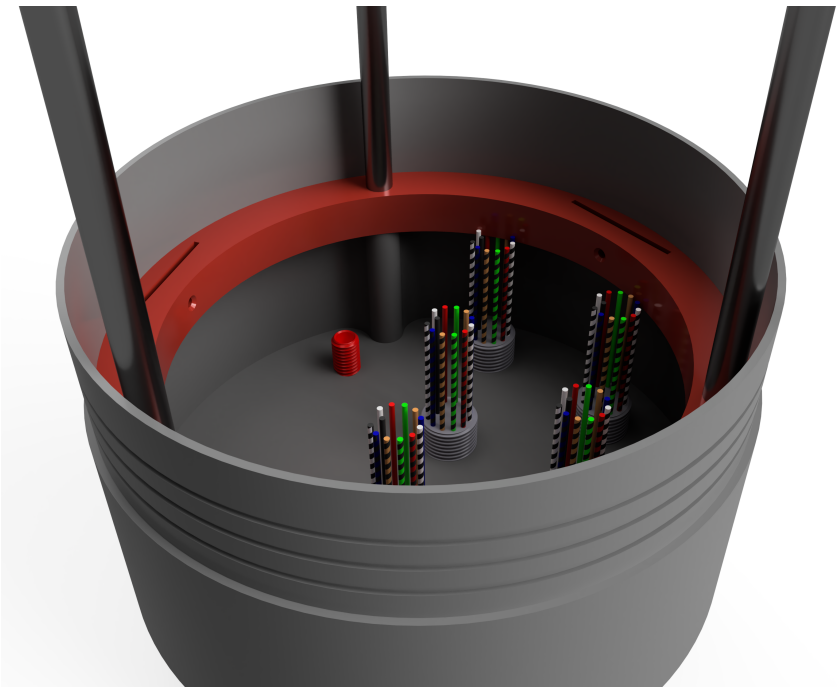
(a) Kameraholder uten kuppel



(b) Kameraholder med kuppel

Stereokameraet er festet i ringen som er forut i ROV-en. For å styre tilt er denne montert i en eget-designet konstruksjon som bruker en servo for rotasjon. For å sikre feste av ringene i hver ende og for å lage et solid rammeverk, er det laget tre stag som går parallelt med bakplatene. De er festet i hver ring, og har også som hensikt å avlaste bakplatene fra å mekanisk holde ringene på plass.

Formålet med dette designet er å skille **innsiden** av elektronikkhuset fra **kapslingen**. På denne måten skal en kunne sette hele konstruksjonen inn i kapslingen til elektronikkhuset. Den interne delen skal deretter festes i bakdelen av kapslingen. Se figur 2.17

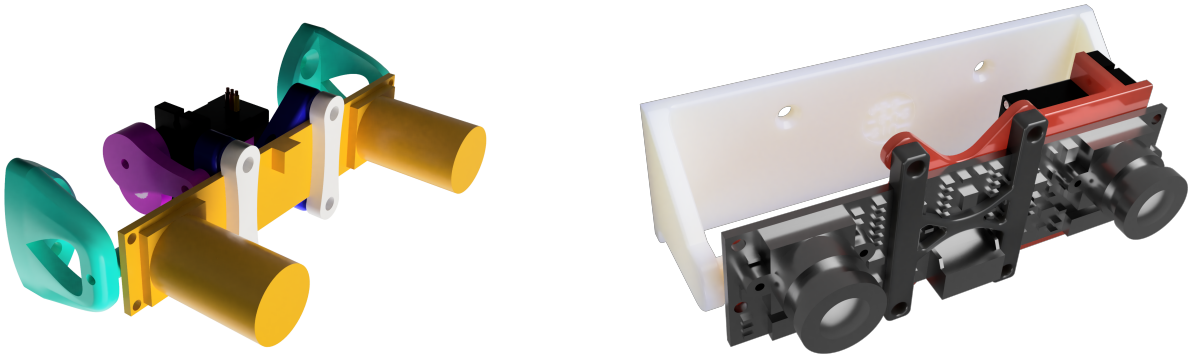


Figur 2.17: Festepunkt til stag.

Stagene har gjenger i enden, og festes i bunndelen av kapslingen sine hull. Hele konstruksjonen vil da bli festet i bunndelen, og det er ikke behov for å feste noe i framdelen av kapslingen. Konstruksjonen vil bli solid med gjennomgående stag som holder strukturen selv om den er inne i kapslingen.

2.6.3.3 Kameraholder

Kameraholder er laget for å bruke en servo direkte for å styre tilten på kamera, vist på figur 2.18b. Det er tatt utgangspunkt i design fra i fjor som er vist på figur 2.18a. Svakheten som ble sett, var at designet bruker flere løse komponenter for å justere tilt. Flere deler utgjør flere svakheter, og det kan bli økt friksjon. Det ble også brukt små akslinger av metall for å gjennomføre dette designet.



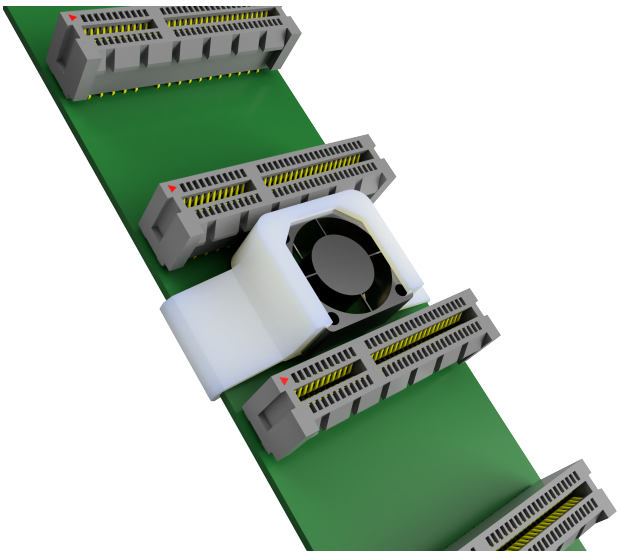
(a) Fjorårets design av kamera tilt

(b) Nytt design av kamera tilt

Det nye designet er laget for å unngå å gjøre det i tre deler. Designet er også laget for å være mer solid, da festene fra i fjor knakk under transport. Ideen om å bruke et modulært design der størrelsen på kamera kan variere, er direkte kopiert da denne virket veldig bra.

2.6.3.4 Vifteholder

Vifteholderen ble laget slik at viftene enkelt kunne skyves inn i et kammer, og klipse på delen til bakplaten. Se figur 2.19.

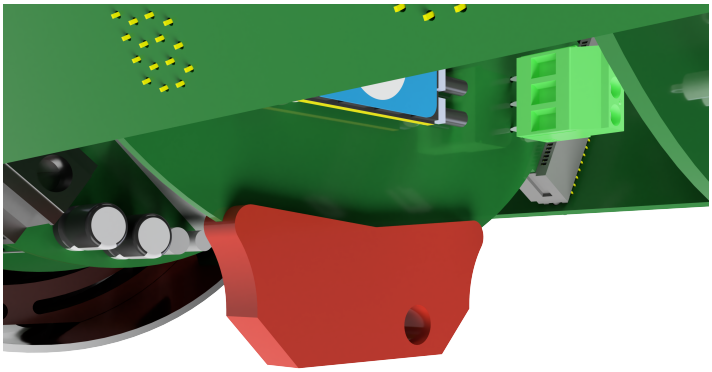


Figur 2.19: Vifteholder festet på bakplate

Vifteholderen er også plassert slik at den ikke ligger helt inntil bakplaten. Den ligger litt over for å kunne trekke inn luft under seg.

2.6.3.5 Holder for lekkasjesensor

For å ha feste av lekkasjesensorer langs bunnen av kapslingen, er det laget holdere for lekkasjesensorene. De er laget slik at de kan skyves på kreskortene for å nå helt til bunnen av kapslingen. Se figur 2.20.



Figur 2.20: Holder for lekkasjesensor

Denne enten limes til printet, eller stripses til delen gjennom å bruke hullet. Delen kan monteres på de

kortene en ønsker.

2.6.3.6 Kamerahus internt

For å holde kameraet inne i kamerahuset, ble det printet en del som holder det i rett høyde fra glasset. Denne festes så i kapslingen til kamerahuset for å holde kameraet i rett retning med en skrue igjennom et hull. Se figur 2.21.

Figur 2.21: Holder for kamera i kamerahus

2.7 Resultat

2.7.1 Konstruksjon

2.7.1.1 Elektronikkhus

Det ferdige elektronikkhuset ble laget av gruppen for ROV design, og våres design ble derfor ikke behov for å lage. Det er likevel laget for å passe våres innvendige design, og utformingen er lik nok til å bruke i utregninger.

Det er derfor ikke gjort beregninger for om denne tåler det utvendige trykket, da den andre gruppen har gjort sine egne konklusjoner på området.

2.7.1.2 Kamerahus

For å gjenbruke materialer, ble det tatt en emne av aluminiumkapp fra verkstedet for å lage huset. Emnet hadde et gjennomgående hull, noe som gjorde at toppen måtte sveises på. Toppen ble TIG-sveiset på før videre dreiing. Innstillinger for TIG-sveising var:

- AC spenning

- 210 Ampere strømtrekk
- Wolfram tupp 3,2mm
- Tilsats 4043 for 6061 aluminium

Emnet ble så dreid ned til 77mm for å redusere vekt av huset. Det lages en lippe for å skru fast de gjennomgående skruene som fester polykarbonatplaten til kamerahuset. Bilder av ferdig montering vises på figur 2.22.

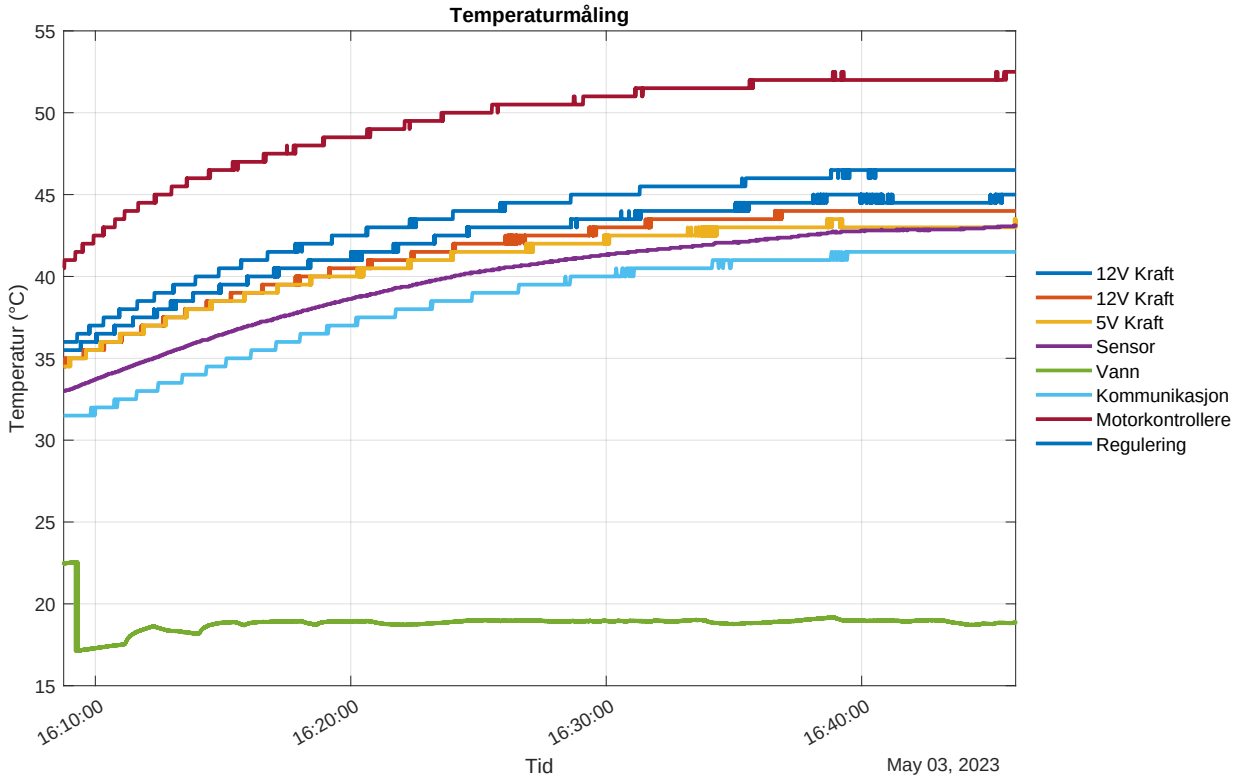
Figur 2.22: Bilder av kamerahus

To stk hull til penetratorer ble montert, da det originalt var planen å ha kabelen for det andre USB-kamera gjennom den samme pluggen. Kabelen skulle da skjøtes gjennom kamerahuset og videre for lettere å gjøre kablingen vanntett. Dette ble droppet, da det var ledige plugger til å ha en annen kabel til det andre kamerahuset. Hullet ble derfor tettet med en blindplugg.

Platen under for å feste huset til ROV-en ble også laget av gruppen ut av tilgjengelig kapp fra verkstaden.

2.7.2 Varme

På figur 2.23[10] er det vist temperaturmåling fra en vanntest. Grafen viser en kjøretid på litt under 40 minutter. Under testen var det gjort sporadisk kjøring av thrustere.



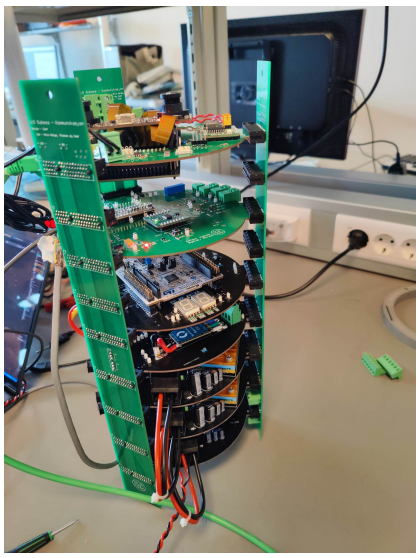
Figur 2.23: Temperaturmåling på kort under vanntest

Fra grafen ser vi at temperaturen stiger jevnt på alle kortene under kjøringen. Temperaturen for de ulike kortene stabiliserer seg mot slutten av kjøringen. Vanntemperaturen viser under 20°C under hele tidsperioden. Det kortet som når høyest temperatur er kortet som har motorkontrollerene på seg. Dette kortet har fire motorkontrollere som skal drive manipulatoren, som utvikler mye varme ved drift. Det når en makstemperatur på 52,5 °C. Kortet har ingen vifter, og lite varmeavledning. Dette ble ikke tenkt på ved design av kortet, selv om kontrollerene er kjent for å bli varme. Likevel er temperaturen som er oppnådd ikke et problem for elektronikken.

2.7.3 Design

Det ferdige designet for det interne elektronikkhuset ble montert med ringer for å holde kretskortene. Et bilde vises på figur 2.24.

[10]For å hente ut denne dataen ble det laget en kodesnutt som filtrerer ut den dataen en ønsker fra logfilen. Denne ble så delt med andre grupper som de videre brukte for å lage sine grafer.



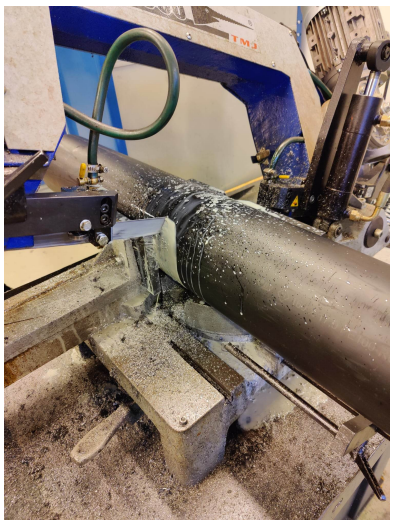
Figur 2.24: Kretskortstabel montert med alle kretskort.

Designet viste seg sterkt og montering var enkelt. Selv uten ringene var konstruksjonen relativt rigid. Dette gjorde at testing kunne foregå i tidlig fase før maskinering av bunndel til elektronikkhus var ferdig.

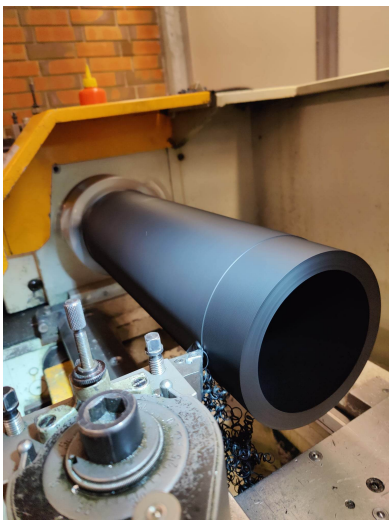
2.8 Flyter konstruksjon

Den mekaniske gruppen ba oss hjelpe med konstruksjon av flyteren sin kapsling. Bakgrunnen var at denne skulle lages av HDPE, og Joar er sertifisert plastsveiser og har mye erfaring med materialet.

Flyterkroppen består av et rør på **140mm** i ytre diameter og **120mm** i indre diameter. Flyteren skal være **400mm** lang. Dette skal sveises tett i den ene enden med en plate. Røret som ble bestilt var et rør på **160mm** i ytre diameter, og **120mm** i indre diameter. Røret var **1000mm** langt. Dette betydde at røret måtte kuttes og dreies ned. Dette vises på figurer 2.25a og 2.25b.



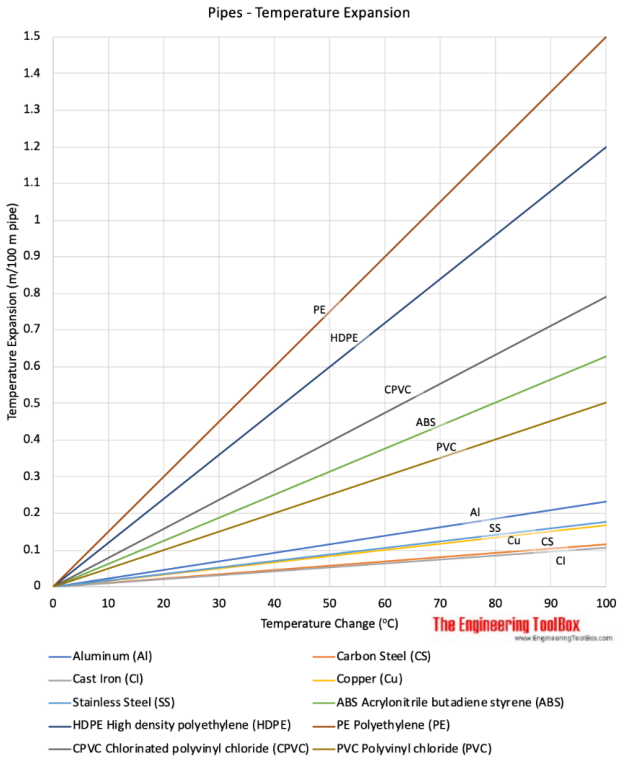
(a) Kutting av flyterkroppen



(b) Dreiing av flyterkroppen

Figur 2.25

HDPE er et materiale som egner seg dårlig til presis konstruksjon. Materialet er meget bøyelig og utvider seg mye ved oppvarming. Siden materialet blir varmt ved dreiing, er nøyaktige mål nesten umulig å få til. På figur 2.26 er det vist hvordan dette materiale utvider seg i forhold til andre material. Det var derfor viktig å la materialet kjøle seg litt før en tar endelige mål for diameter og lengde.



Figur 2.26: Utvidelse av material basert på varmeutvikling.

Etter røret var rett lengde, ble det kuttet ut en platebit av samme PE-kvalitet med større diameter enn røret. Siden materialet er så mykt, kan dette behandles med enkle redskaper. Sager som er laget for trevirke fungerer like bra som sager laget for metall. I dette tilfellet ble det brukt en stikksag for å kutte en **10mm** plate.

Denne kunne da sveises på gjennom ekstrudering. Til forskjell for buesveising[11], bruker ekstrudering varmt materiale for å sammenføye deler. For at delene skal feste seg til det ekstruderte materialet, blir det lokalt oppvarmet med varmluft.

Sveiseapparatet som ble brukt er et `Leister Fusion 2`, vist på figur 2.27.

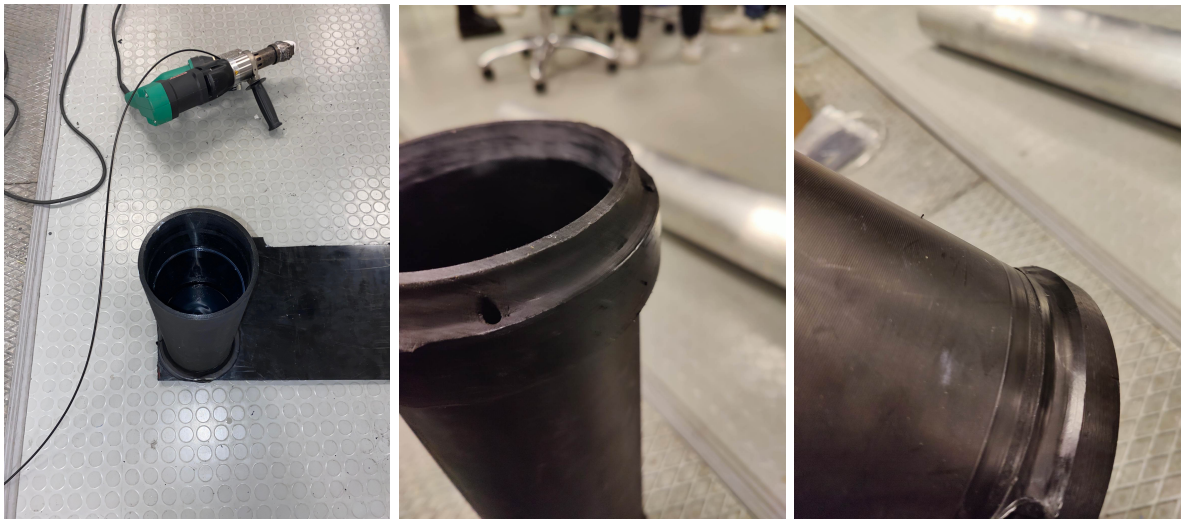
[11]"Buesveising omfatter flere smeltesveisemetoder som benytter en elektrisk bueutladning, en elektrisk lysbue, som varmekilde."[42]

Figur 2.27: Leister Fusion 2[43]

Dette sveiseapparatet bruker tråd med HDPE som blir presset igjennom en oppvarmet dyse som smelter materialet og mater det ut. Dysen blir varmet opp av varmluften som kommer fra undersiden av apparatet. Denne luften går deretter ut på oppsiden av teflonstykket, og brukes så for å varme delene som skal sveises ilag.

HDPE oksiderer likt som aluminium, og det må derfor slipes av et lag for at sveisen skal holde. Ulikt fra aluminium kommer denne oksideringen mye senere, og oppstår raskest under varme. En bacheloroppgave fra NTNU testet kvaliteten av sveiser med ulike tider for oksidering og konkluderer slikt: "På bakgrunn av de testene vi har gjort, konkluderer vi med en teori om at oksidlaget ikke har relevant betydning for sveisekvaliteten dersom man forholder seg til en periode på ti dager"[44].

Siden materialets smeltepunkt er mye lavere enn aluminium eller stål, er det viktig å huske å ikke ha for mye bevegelse i delene før sveisen kjøler seg godt ned. Dersom dette hastes, kan sveisen deformeres. Den ferdige sveisen vises på figur 2.28.



Figur 2.28: Ferdig sveist med enkel vanntest

Resultatet var en tett topp på flyteren. Toppen ble så kappet av platen og dreid rund. De ble kappet og sveist et håndtak på flyteren som manipulatoren på ROV-en skal bruke for å ta tak i flyteren. Dette er en utfordring på MATE som gir poeng. I tillegg ble det dreid til en ring for å feste skruer til bunnstykket av flyteren. Dette ble dreid av det samme emnet som ble brukt til rørkroppen.

Kapittel 3

Kommunikasjon

Kapitteloversikt

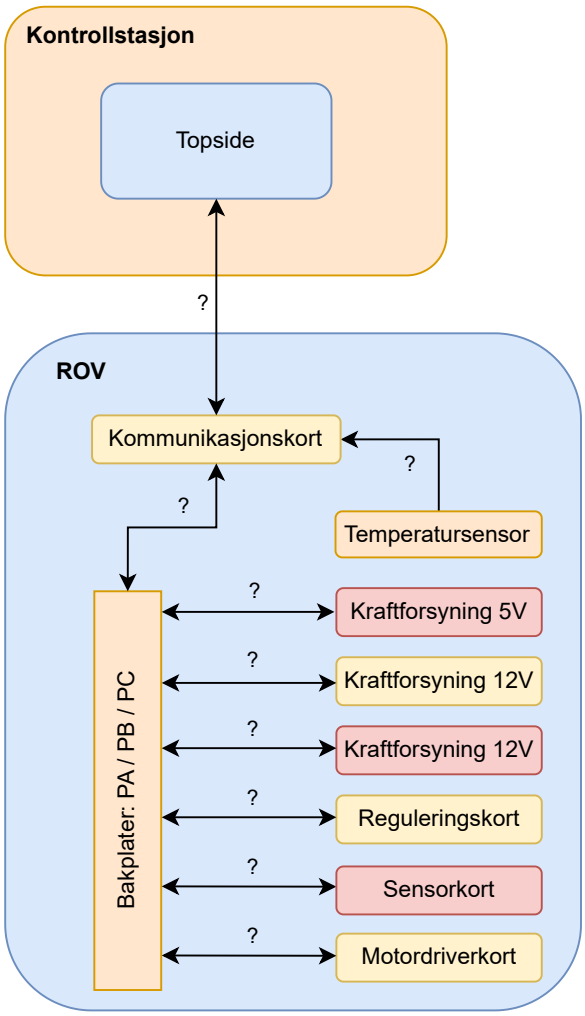
Dette kapittelet tar for seg kommunikasjonen ombord i ROV-en, samt mellom ROV-en og kontrollstasjon. Det vil gås gjennom virkemåten til de ulike kommunikasjonsstandardene som er blitt vurdert, og grunngivelser for valgene som ble gjort.

3.1 Problemstilling

Problemstillingen som omhandler kommunikasjon i dette prosjektet kan deles opp i to ulike kategorier: *Kommunikasjon internt ombord i ROV* og *kommunikasjon med kontrollstasjon*.

Kommunikasjon internt ombord i ROV - Alle de forskjellige kretskortene ombord i ROV-en skal ha en måte å kommunisere sammen seg i mellom over korte avstander. Dette bør være en kommunikasjonsstandard med relativt høy overføringshastighet, samtidig som den er robust og driftssikker. Eventuelle brudd eller forstyrrelser på kommunikasjonen kan være kritisk for driften.

Kommunikasjon med kontrollstasjon - Fra ROV-en er det nødt å gå en kommunikasjonslinje til kontrollstasjonen. Dette må være en kommunikasjonsstandard med en høy overføringshastighet som fungerer over lengre avstander. Det stilles også høye krav til stor båndbredde, da det også skal overføres videostrømmer over denne linjen.



Figur 3.1: Blokkskjema som illustrerer problemstillingen. "?" skal erstattes med kommunikasjonsmetoder

3.2 Behovsspesifikasjon

- Kommunikasjonsstandard brukt internt bør ha en relativt høy overføringshastighet
- Kommunikasjonsstandard brukt internt må være robust mot forstyrrelser og fungere over en kort avstand
- Kommunikasjonsstandard mellom ROV og kontrollstasjon må ha en relativt høy overføringshastighet og båndbredde til kommunikasjon og videostrøm

3.3 Funksjonsspesifikasjon

- Kommunikasjonsstandardene internt ombord i ROV-en skal kunne sende 16 pakker med en frekvens på 20Hz, det blir 320 pakker hvert sekund.
- Kommunikasjonstandardene for sending av data fra ROV til kontrollstasjon skal kunne sende 16 pakker med en frekvens på 10Hz, det blir 160 pakker hvert sekund.
- Kommunikasjonstandarden internt skal kunne fungere feilfritt på avstander inntil 1 meter
- Kommunikasjonstandarden mellom ROV og kontrollstasjon skal opprettholde ønsket kommunikasjonshastighet på 100m
- Kommunikasjonstandarden mellom ROV og kontrollstasjon skal ha nok båndbredde til fire videostrømmer, samt styre- og informasjonssignaler.
- Forsinkelsen på videostrøm bør være lavere enn 200ms, men det kan aksepteres inntil 400ms.
- Kommunikasjonsstandard som realiserer lesing av temperatur fra temperatursensor (gitt av sensorgruppe)

3.4 Kommunikasjonsstandarder

I dette delkapittelet presenteres ulike kommunikasjonsprotokoller og standarder som kan brukes for å realisere kommunikasjonen mellom:

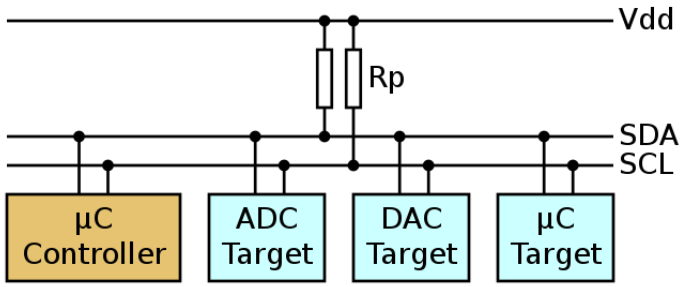
- Kontrollstasjonen og ROV-en
- Kretskortet og kameraer
- Komponenter på kretskortet
- Kretskort og andre kretskort

I slutten av delkapittelet kommer begrunnelsen for de ulike valgene som er tatt.

3.4.1 I^2C

Dette kapittelet er delvis basert på Wikipediasidene for I^2C ([45] og [46]).

I^2C (*Inter Integrated Circuit*) er en synkron seriellbuss som brukes i en master/slave-konfigurasjon. I dag brukes hovedsaklig I^2C for å koble lavhastighetskretser opp mot en prosessor eller mikrokontroller[45] over korte distanser, gjerne internt på et kretskort.



Figur 3.2: Eksempeloppsett for kommunikasjon over I^2C med en kontroller (master), tre målnoder (slaver) og opptrekksmotstander[45]

.

3.4.1.1 Design

Det elektriske designet for I^2C-bussen består av to bidireksjonale "*open-drain*"-linjer - seriell data (SDA) og seriell klokke (SCL) - og "*pull-up*"-motstander. Typisk brukt driftsspenning for I^2C er enten +5V eller +3,3V. Dette er vist i figur 3.2 ovenfor. I^2C-bussen kjøres på "*half-duplex*", som vil si at to enheter kan kommunisere med hverandre, men kommunikasjonen går ikke i begge retninger samtidig som ved "*full-duplex*". En forklaring på begrepene, "pull-up"-motstand og "open-drain" er gitt i de to boksene nedenfor.

Pull-Up Resistor - [47]

"In electronic logic circuits, a pull-up resistor (PU) or pull-down resistor (PD) is a resistor used to ensure a known state for a signal. It is typically used in combination with components such as switches and transistors, which physically interrupt the connection of subsequent components to ground or to VCC. Closing the switch creates a direct connection to ground or VCC, but when the switch is open, the rest of the circuit would be left floating (i.e., it would have an indeterminate voltage)."

Open-Drain - [48]

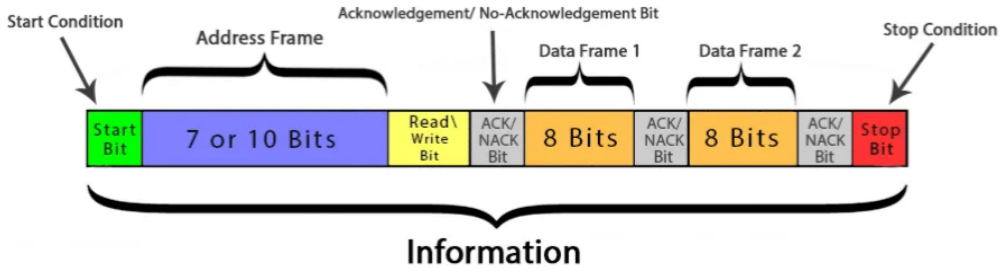
"It is also called Open Collector which is a transistor having a connection with the ground pin. In a simple way, it behaves like a switch that gets connected or disconnected depending on the input signal. This open-drain configuration permits different devices to establish communication bi-directionally on one wire. This is generally considered as a mode where it provides only pull-down functionality. This is the basic open-drain definition."

Overføringshastigheten til I^2C-bussen og, i noen tilfeller hvor mange noder som kan kobles til bussen, er bestemt av hvilket modus som blir kjørt. Standardmodus på hastigheten er 100 Kbps, men "Ultra-fast"-modus[45] kan kjøre 5 Mbps. Antall noder som kan kobles på bussen begrenses av antall ledige adresser, men også av kapasitansen på busslinjene.

I databladet for I^2C-bussstandarden utgitt av NXP Semiconductors er det oppgitt en maksimal kapasitans på 400pF på hver linje[49]. Dette er for å opprettholde kravet til maksimal stigningstid på 1000ns i standardmodus. I praksis betyr dette at busskabelen må holdes relativt kort, da kapasitansen må begrenses til et minimum. Siden den maksimale kapasitansen på linjene er 400pF, vil den praktiske lengden på bussen ikke kunne være mer enn et par meter.

3.4.1.2 Kommunikasjon over I^2C

I^2C-bussen bruker 7- og 10-bits adressering, men 10-bits adressering er sjeldent brukt. Ved 7-bits adressering vil en ha $2^7 = 128$ adressser, men noen av disse adressene er reservert, så i realiteten er det kun 107 adresser som kan brukes[45]. I figur 3.3 nedenfor er det gitt en oversikt over en generell melding på I^2C-bussen kan se ut.

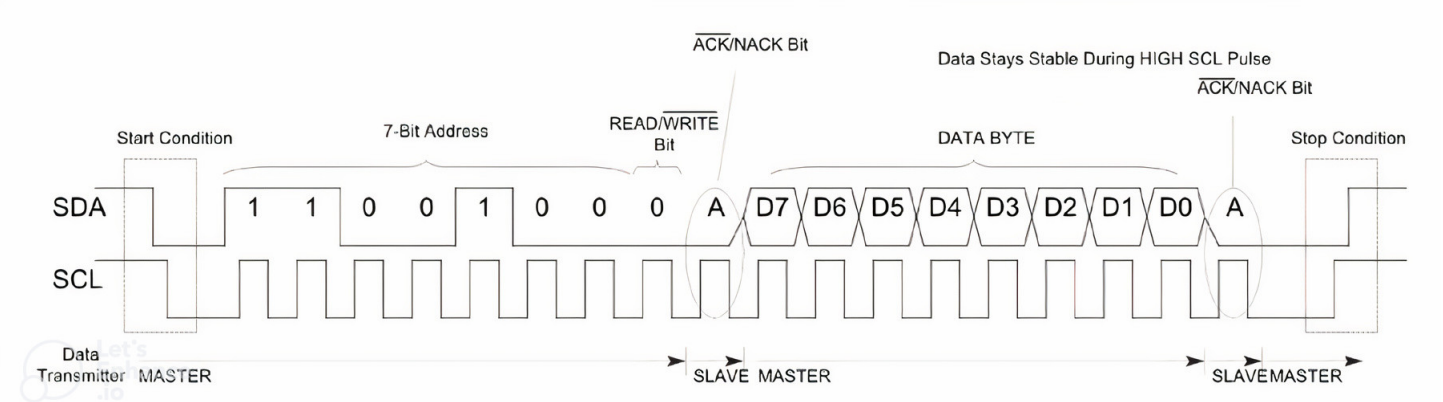


Figur 3.3: Overordnet oversikt for kommunikasjon over I^2C [50]

.

Startbittet markerer start på overføring, og stoppbittet marker slutt på overføring. Etter startbittet kommer adresserammen markert som en blå boks i figur 3.3. Adressen til en slave blir satt av masteren som vil kommunisere med den gitte slaven. Etter adresserammen kommer det et "*Read/Write*"-bit som markerer om masteren vil lese eller skrive til slaven. Deretter kommer datapakkene som skal sendes, og på hver side av disse er det et "*Acknowledge*"-bit for å markere om data er motatt uten feil eller

ikke. I figur 3.4 nedenfor er en mer detaljert illustrasjon over hvordan I^2C-kommunikasjon kan se ut.



Figur 3.4: Eksempelillustrasjon av kommunikasjon over I^2C. Data sendes fra master til slave [51]

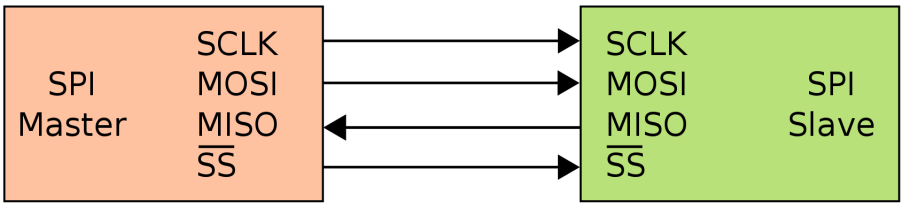
.

Når det skal foregå kommunikasjon på I^2C-bussen, vil masteren starte klokken som slaven skal synkroniseres etter på SCL-linjen. Denne klokken vil ha en bestemt klokkefrekvens gjennom hele overføringen. Startbittet markeres ved at SDA er lav mens SCL er høy, etterfølgt av 7-bits adressering de neste syv klokkepulsene. I dette eksemplet er "*Read/Write*" lik 0, som markerer at masteren ønsker å skrive til slaven. Slaven markerer at adresseringen og bittet for skriving/lesing er mottatt ved å sette det aktivt lave "*Acknowledge*"-bittet lavt. Deretter begynner masteren sending av en byte med data. Etter en byte er sendt, markerer slaven at byten er mottat ved å sette "*Acknowledge*"-bittet lavt. Dette gjentas for hver byte frem til all data er sendt fra master til slaven. Slutten av overføringen markeres ved at både SDA og SCL går høy.

3.4.2 SPI

Dette delkapittelet er inspirert av boken *Discovering the STM32 Microcontroller* av Geoffrey Brown [52].

SPI (*Serial Peripheral Interface*) er en synkron seriell kommunikasjonsstandard for kommunikasjon over korte avstander[53]. SPI finnes i de fleste mikrokontrollere som en perifermodul. SPI-protokollen bruker en master/slave-arkitektur, hvor SPI-master styrer SPI-slaven. Et master-slave SPI-oppsett baserer seg på fire ledere mellom slaven og masteren. Dette er vist i figur 3.5 nedenfor. Det er også mulig å koble flere slaver til master-modulen.



Figur 3.5: Oppsett for single-master to single-slave SPI.[53]

.

De fire digitale signalene mellom master og slave er:

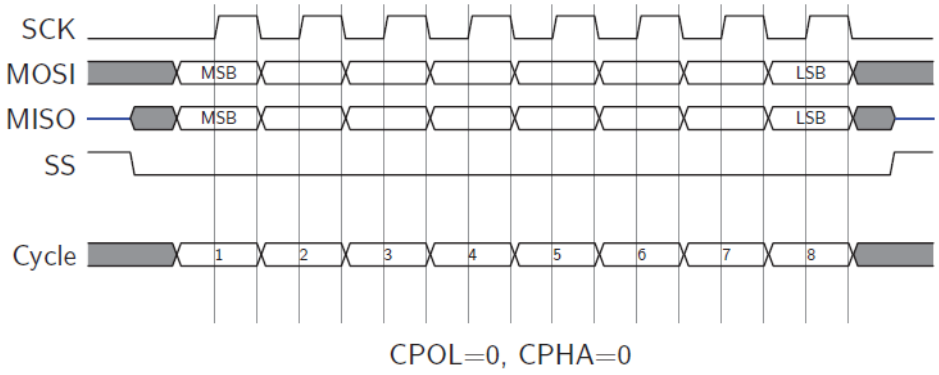
SCLK: Dette er det serielle klokkesignalet. Bitstrømmen i datapakkene er synkronisert med klokkesignalet.

MOSI: Forkortelse for "Master Out Slave In". Den serielle datastrømmen går fra master til slave.

MISO: Forkortelse for "Master In Slave Out". Den serielle datastrømmen går fra slave til master.

SS: Forkortelse for "Slave Select", men erstattes av og til med CS som står for "Chip Select". SS blir satt av master, og brukes for at slaven skal vite når det blir sendt data. SS/CS er ofte aktivt lave. Brukes det en vanlig mikrokontroller, kan CS-inngangen på en slave styres ved å bruke en av GPIO-utgangene på mikrokontrolleren.

SPI-standarden har i seg selv ingen begrensninger når det kommer til overføringshastighet[54]. Oppdateringsfrekvens opp mot 100MHz har blant annet blit brukt i noen tilfeller. Det som setter begrensningen for overføringshastigheten, er hver enkelt komponent som skal kommunisere over SPI og dens klokkefrekvens. For eksempel kan det være en master som kjører en 16 MHz klokke på SPI-bussen, men da må slavene også være laget for å operere med denne klokkefrekvensen. I figur 3.6 nedenfor, er et tidsdiagram for en typisk SPI-overføring i "full-duplex"[1].



Figur 3.6: Tidsdiagram for overføring over SPI-buss i full duplex.[52]

.

Ved SPI-kommunikasjon i "full-duplex" sendes dataen over linjene, MISO og MOSI. CPOL (*Clock Polarity*) bestemmer klokkepolariteten, og er i dette eksempelet satt til 0 som betyr at klokkesignalet er lavt når klokken ikke kjører. En kan se på dette som et slags tomgangsnivå[2]. CPOL lik 1 gir oss det motsatte ved at "tomgang" er representert av et høyt klokkesignal.

CPHA (*Clock Phase*) bestemmer hvordan sending og lesing av data skal synkroniseres opp mot klokken. For CPHA lik 1, sendes data på stigende flanke, mens data leses av mottaker på fallende flanke. Motsatt for CPHA lik 0, hvor sending av data skjer på fallende flanke, og mottak av data skjer på stigende flanke. I denne figuren har vi CPOL og CPHA lik 0. Det vil si at klokkesignalet er lavt når klokken ikke kjører [52]. Data sendes på fallende flanke av klokkesignalet SCK, og data mottas på stigende flanke av klokkesignalet.

Dataoverføringen av en byte markeres først ved at signalet SS går lavt. Hver bit leses av ved stigende flanke på klokkepulsen. Sendingen avsluttes ved at SS går høyt igjen.

[1] Full duplex betyr at sending og mottak av data foregår samtidig over to linjer.
[2] Når klokken på SPI-bussen ikke er aktiv, kaller vi dette for tomgangsnivå

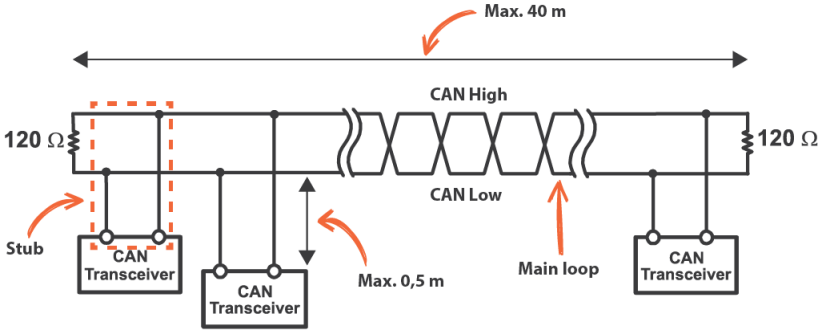
3.4.3 CAN-Buss

Dette kapittelet er delvis basert på teori/informasjon hentet fra Wikipedia([55]) og tidligere UiS Subsea Bachelor([31]).

CAN-buss (**Controller Area Network**) er en meldingsbasert kommunikasjonsprotokoll utviklet av Bosch, orginalt designet for å redusere antall ledere brukt til kommunikasjon i kjøretøy. Protokollen gjorde det mulig for mikrokontrollere og andre enheter å kommunisere seg i mellom over to ledere uten en overordnet datamaskin for styring av kommunikasjonen. Bussen baserer seg på et prioritetssystem hvor enheten med lavest *ID* har den høyeste prioriteten. Hvis flere enheter prøver å sende på bussen samtidig, vil den enheten med høyest prioritet få lov å fortsette sendingen. Enhetene på CAN-bussen mottar alle pakkene som blir sendt over bussen, men ved hjelp av et digitalt filter kan man velge hvilke *ID-er* som aktiverer avbruddsrutinen til mikrokontrolleren[31]. I de neste seksjonene vil CAN-bussstandarden gjennomgås i detalj.

3.4.3.1 Elektrisk oppsett og virkemåte

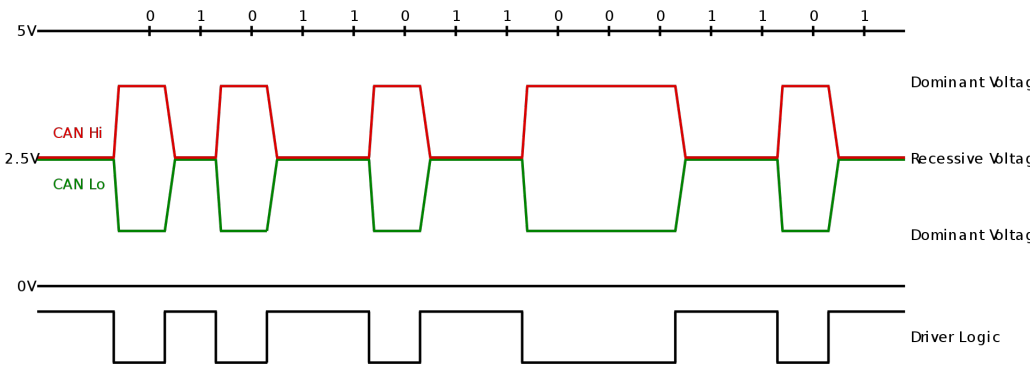
CAN-bussen består av to ledere - CAN-H og CAN-L - som hver enkelt modul kobles opp mot. Bussen termineres med en motstand på 120Ω i hver ende som vist i figur 3.7 nedenfor. Hensikten med å terminere bussen er å unngå refleksjoner av signalene som kan oppstå på en to-veis buss. Motstanden i hver ende absorberer signalenergien. Makslengden på bussen for å opprettholde en overføringshastighet på 1Mbps er 40m. Hastigheter mellom 125Kbps og 1Mbps er definert som *High-Speed CAN*[55].



Figur 3.7: Standard oppsett av CAN-buss[56]

.

CAN-bussen er en differensialbuss. Det betyr at signalene, CAN-H og CAN-L i de to lederene, er inverterte. Dette er vist i figur 3.8 nedenfor. Et logisk høyt driversignal representeres ved at CAN-H er logisk lav, og at CAN-L er logisk høy. Det logisk høye driversignalet hvor det blir sendt en ener blir også sett på som den resessive tilstanden til CAN-bussen. Et logisk lavt driversignal representeres ved at CAN-H er logisk høy, og at CAN-L er logisk lav. Det logisk lave driversignalet hvor det blir sendt en null blir også sett på som den dominante tilstanden til CAN-bussen. Hvis ingen av enhetene på CAN-bussen sender en dominant null, vil termineringsmotstandene dra bussnivået ned til en resessiv tilstand med nominell differensialspenning på 0V [55].



Figur 3.8: Spenningsnivå på CAN-bussen. CAN-H er representert av det røde signalet. CAN-L av det grønne.[55]

.

Nominelt sett vil CAN-H og CAN-L ligge på rundt 2,5V, altså med en nominell differensialspenning på 0V i resessiv tilstand. Og i den dominante tilstanden vil nominelt sett CAN-H ligge på 3,5V og CAN-L ligge på 1,5V, med en nominell differensialspenning på 2,0V [55]. Disse spenningsnivåene er nominelle, og kan i praksis avvike litt alt etter hvilken CAN-driver som brukes på bussen.

En CAN-driver hvor spenningsnivåene ville avvike fra den nominelle spenningen, er SN65HVD230[57]. Dette er en CAN-driver som kan drives på 3,3V. Et utklipp fra databladet er vist i figur 3.9 nedenfor.

PARAMETER			TEST CONDITIONS		MIN	TYP(1)	MAX	UNIT
V_{OH}	Bus output voltage	Dominant	V_I = 0 V, See Figure 18 and Figure 20	CANH	2.45		V_{CC}	V
				CANL	0.5		1.25	
V_{OL}		Recessive	V_I = 3 V, See Figure 18 and Figure 20	CANH		2.3		
				CANL		2.3		
$V_{OD(D)}$	Differential output voltage	Dominant	V_I = 0 V,	See Figure 18	1.5	2	3	V
			V_I = 0 V,	See Figure 19	1.2	2	3	
$V_{OD(R)}$		Recessive	V_I = 3 V,	See Figure 18	–120	0	12	mV
			V_I = 3 V,	No load	–0.5	–0.2	0.05	V

Figur 3.9: Utklipp fra databladet til SN65HVD230.[57]

.

Fra databladet kommer det fram at SN65HVD230 vil avvike fra nominelle spenningsnivå nevnt tidligere. De ulike spenningsnivåene i dominant og resessiv tilstand er:

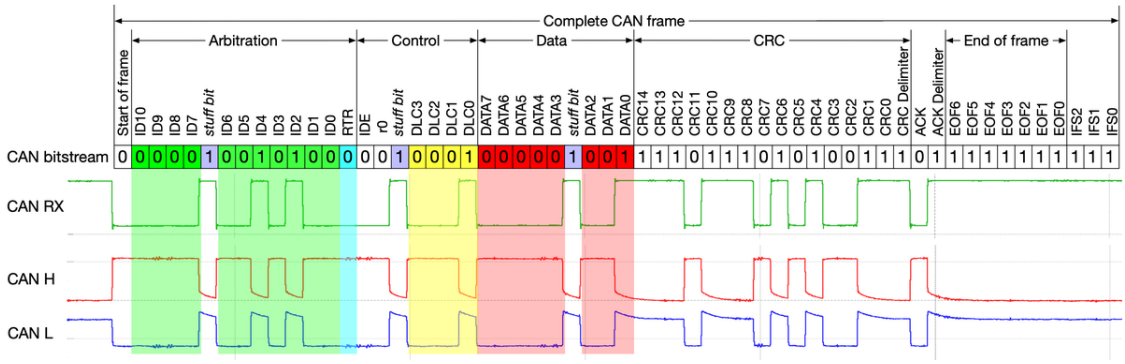
Dominant: I dominant tilstand drives linjene aktivt. Da vil V_{OH-CAN_H} og V_{OH-CAN_Lv} ligge i intervallene: $\boxed{2,45V \leq V_{OH-CAN\_H} \leq 3,3V}$ og $\boxed{0,5V \leq V_{OH-CAN\_L} \leq 1,25V}$. Differensialspenningen ligger typsik på $V_{OD(D)} = 2V$. Dette er verifisert i testrapport A.1.

Resessiv: I resessiv tilstand drives ikke linjene aktivt, og er typisk representert av $\boxed{V_{OL-CAN\_H} = V_{OL-CAN\_L} = 2,3V}$. Med andre ord en nominell differansialspenning på $V_{OD(R)} = 0V$.

3.4.3.2 CAN-busspakke

Som vist i figur 3.10 nedenfor, består en CAN-busspakke av fem hovedsegmenter [31]. Disse er gitt i listen nedenfor, og blir mer forklart etter figuren.

- 11-bit pakke-ID med 1 kontrollbit
- 6-bit med styresignal
- Datasegment på 1 til 8 byter
- 15-bit CRC sjekksum med skilletegn
- 7-bit pakkeslutt hvor alle bit skal være 1



Figur 3.10: Komplett CAN-busspakke med 1 byte datasegment[55]

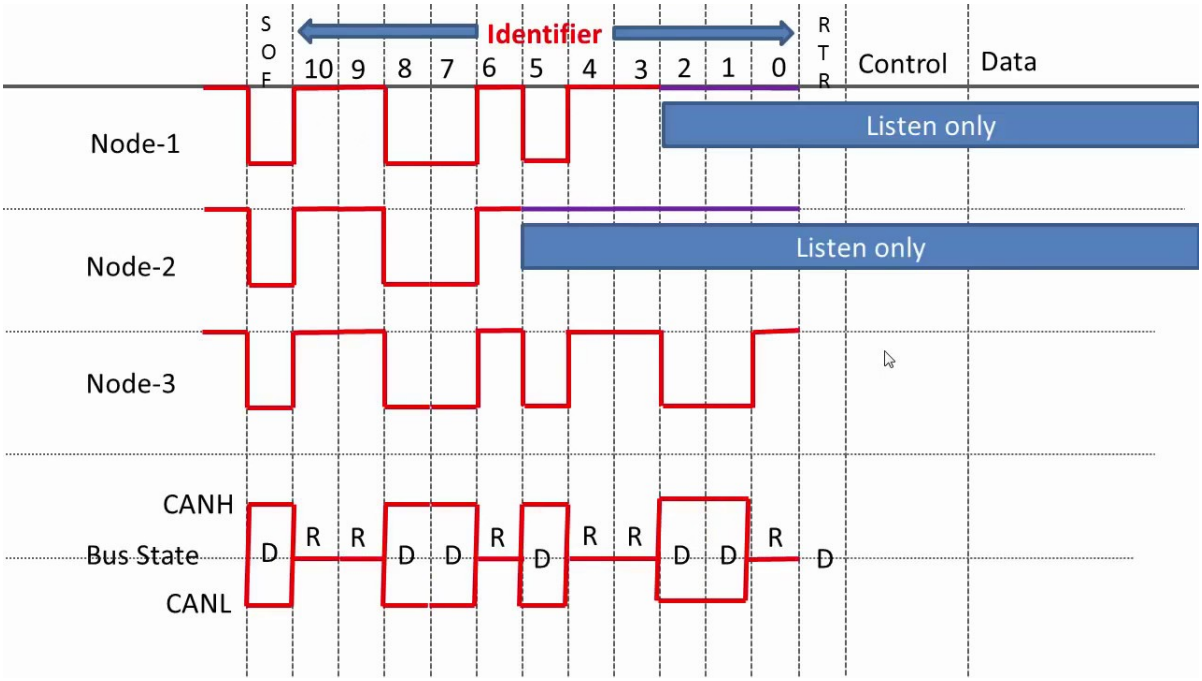
.

Slutten på en pakke (*End of frame*) signaliseres med syv bit som er 1 (ressesive). Spenningsnivået er ressesivt i normalstilstand på CAN-bussen ($CANH = CANL = 2,5V$). Når spenningsnivået da blir satt til dominant tilstand ($CANH = 3,5V$ og $CANL = 1,5V$), vil nodene på CAN-bussen lese et ressesivt bit (0) på RX som markerer starten på en ny melding (*Start of frame*). Et bit er brukt for å bekrefte sending av melding (*ACK*) med et etteføgende skilletegn. 3-bit med IFS (*Inter-frame spacing*) er brukt til å lage et skille mellom flere pakker. Og *stuff-bit* som blir satt inn i pakken der det blir 5 like bit etter hverandre. Dette bittet vil da ha motsatt polaritet[31]. Både sender og mottaker vet at det kommer et *stuff-bit* av motsatt polaritet etter 5 like bit, og klarer dermed skille mellom et *stuff-bit* og faktisk data. Grunnen til at det brukes *stuff-bits* er fordi at 6 eller flere like bit etter hverandre mellom *Start of frame* og CRC, blir sett på som en feil i datapakken[58]. Ved å sette inn *stuff-bits* i bestemte posisjoner i datapakken unngås dette. Plasseringen av *stuff-bit* er bestemt av CAN-bussprotokollen, så alle enhetene på CAN-bussen vet hvor det forventes et ekstra bit.

Pakke-ID: Pakke-ID, markert i grønt, brukes til å avgjør hvilken modul datapakken er tenkt til. Som nevnt tidligere i introduksjonen (3.4.3), er prioriteten angitt ved stigende ID hvor den laveste ID-en har høyest prioritet. Dette er gjort for å unngå at flere noder sender på bussen samtidig.

Når nivået på linjen er i den dominante tilstanden, betyr det at en eller flere noder aktivt driver linjene. I resessiv tilstand er det ingen noder som driver linjene. Nodene på CAN-bussen leser aktivt nivået på linjene. Når to eller flere noder prøver å sende samtidig, vil de begynne med å sende deres respektive pakke-ID. Hvis en node prøver å sende en resessiv 1, men linjenivået forblir i dominant tilstand, vil denne noden registrere at den ikke sender på den laveste ID-en og

dermed avbryte sendingen. Dette skjer helt til noden med lavest ID har vunnet frem. I figur 3.11 er det vist et eksempel på hva som skjer når tre noder prøver å sende på CAN-bussen samtidig.



Figur 3.11: Eksempel på når to noder sender melding samtidig.[59]

Alle nodene starter ved å sende deres respektive pakke-ID. Ved bit 5 registrer node 2 at linjenivået er dominant selv om noden prøver å sende en resessiv 1, og avbryter dermed sendingen. Node 1 avbryter ikke sendingen før ved bit 2 der den leser et dominant linjenivå, mens den prøver å sende resessiv 1. Til slutt vinner node 3 som har lavest ID frem, og får fortsette sendingen av datapakken. Dette illustrerer hvordan noder med lavest ID har den høyeste prioriteten.

RTR-bittet markert lyseblått i figur 3.10, avgjør om det er en innkommende datapakke, eller om det er en forespørsel om data. Et dominant bit (0) er innkommende datapakke, og et ressesivt bit (1) er forespørsel om data.

Styresignal: Bittene markert i gult i figur 3.10 brukes for å sette DLC, og angir hvor mange byter som kommer i etterfølgende datasegment. I figuren ovenfor er det satt til 0001 som indikerer 1 byte med data. IDE brukes for å avgjør om det er ID på 11-bits (*Standard ID*)[3] eller 29-bits (*Extended ID*)[4] som brukes. For å indikere 11-bits adresse, er IDE-bittet 0. Er IDE-bittet 1, indikerer dette 29-bits adresse og de resterende 18 bittene i 29-bits adressen blir sendt. r0-bittet er reservert, og skal være 0 [55].

Datasegment: Inneholder databytene som blir sendt. Kan være 1 til 8 byter, men i dette eksempelet er det som nenvt, indikert 1 byte i styresignalet med 0001.

CRC: CRC (*Syklisk redundanssjekk*) er en kode laget for å detektere bitfeil i datapakkene enn mottar[60] ved hjelp av en sjekksum som blir beregnet gjennom polynomdivisjon. Dette gås grundig igjennom i delkapittel 3.4.3.3 som omhandler syklisk redundanssjekk.

Pakkeslutt: 7 bit hvor alle skal være ressesiv (1). Dette indikerer pakkeslutt, og bussen går i "hvilemodus".

[3]Bruker vi standard ID, sier vi at vi har en CAN 2.0A buss
[4]Bruker vi extended ID, sier vi at vi har en CAN 2.0B buss

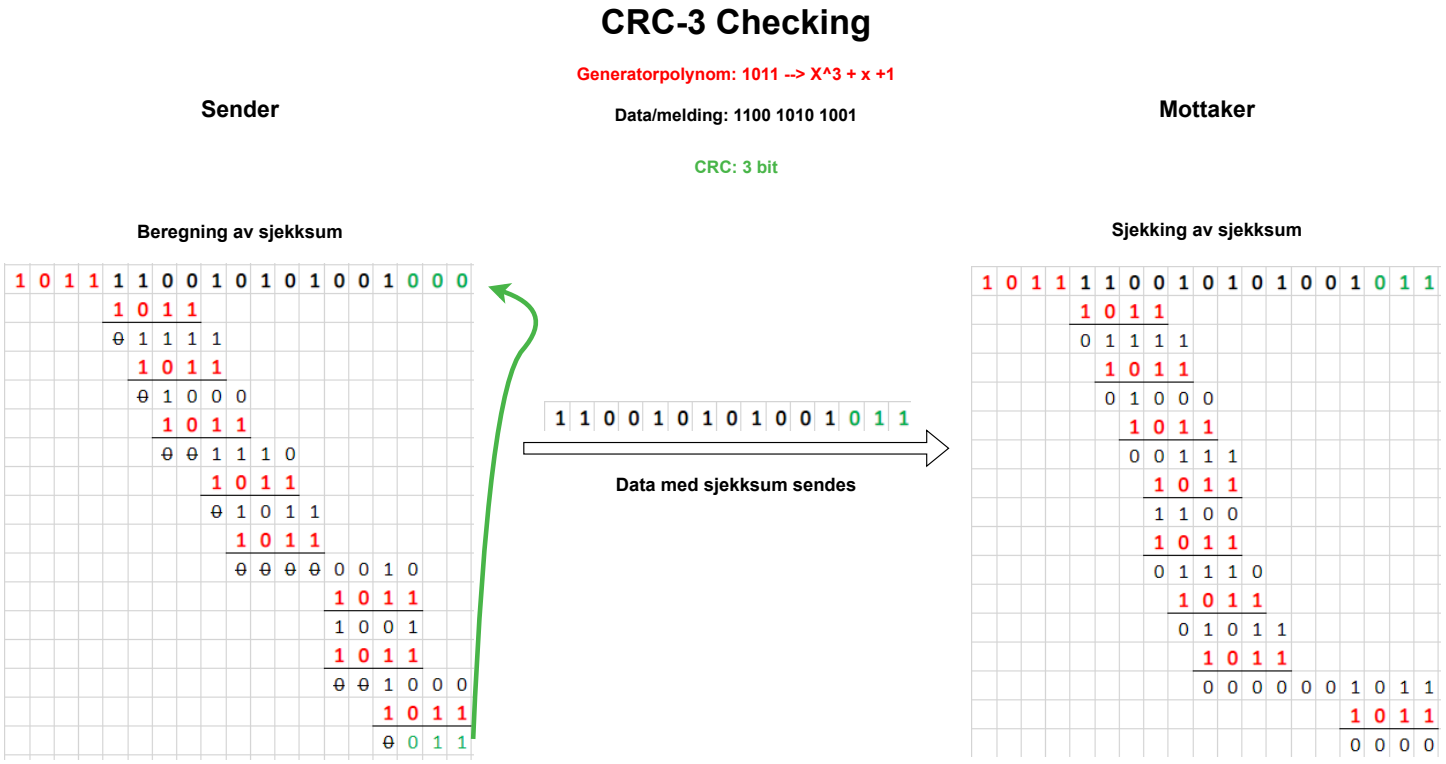
3.4.3.3 CRC - syklisk redundanssjekk

Som nevnt i delkapittel 3.4.3.2, baserer syklisk redundanssjekk seg på polynomdivisjon og en sjekksum, og gås grundigere igjennom i dette delkapittelet.

Før en CAN-busspakke sendes, utvides datapakken med 15 bit som er 0. Disse bittene kommer rett etter datasegmentet som vist i figur 3.10. Antall bits som kommer i CRC-segmentet er gitt av CRC-protokollen som blir brukt. For CAN-buss brukes det en CRC-15-protokoll, altså en 15-bits syklisk redundanssjekk. Polynomet for CAN-buss kan representeres binært av to byte, eller 16 bit. På binær form blir polynomet: 1100 0101 1001 1001, hvor hvert bit som er høyt representerer x^n, og n er posisjonen til bittet. Polynomet for CAN-buss blir dermed: $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$. Dette polynomet brukes både av sender og mottaker av meldingen og er standard for CAN-buss.

Før en melding blir sendt på CAN-bussen, vil senderen utføre en polynomdivisjon på meldingen ved hjelp av standardpolynomet for CAN-buss. Denne polynomdivisjonen løses ved hjelp av bitskiftninger og flere eksklusive-eller-opereasjoner[5]. Resultatet av polynomdivisjonen sendes til mottaker som en sjekksum sammen med datapakken. Når mottakeren mottar denne sjekksummen og datapakken, vil mottakeren kjøre en ny polynomdivisjon på datapakken kombinert med sjekksummen. Hvis resultatet av denne polynomdivisjonen blir 0, betyr det at meldingen ble mottat uten feil.

Et eksempel på hvordan polynomdivisjonen gjennomføres er illustrert i figur 3.12 nedenfor. Her er det brukt CRC-3-protokoll. Dette fordi en polynomdivisjonen med en CRC-15-protokoll er lang og tidkrevende. Forskjellen er at sjekksummen representeres av 3 bit i stedet for 15. Prinsippet er det samme for begge protokollene.



Figur 3.12: Eksempel på 3-bits syklisk redundanssjekk. Generatorpolynomet sendes ikke, men er en del av CRC-protokollen for sender og mottaker.

[5]Ein bitvis eksklusiv ELLER-operator, dvs. XELLER-operator kan brukast til å snu bit ("toggle"), dvs. invertera bitverdiar. Viss ein bit er 1, blir han 0 og omvendt[61]

Eksempelet i figur 3.12 ovenfor for CRC-sjekking bruker CRC-3-protokoll som gir, et tredjegradspolynom og en 3 bits sjekksum. Binært representeres dette polynomet som $\boxed{1011}$. Hvert bit som er høyt representerer x^n hvor n er posisjonen til bittet. Dette gir da tredjegraspolynomet: $\boxed{x^3 + x + 1}$[6]. Dette polynomet blir kalt for generatorpolynomet, og er kjent av både sender og mottaker. Med andre ord blir ikke dette polynomet sendt over CAN-bussen. Datameldingen utvides med 000 på grunn av CRC på 3 bit (CRC-3-protokoll).

Polynomdivisjonen løses ved hjelp av bitskiftninger og flere eksklusive-eller-opereasjoner. For hver horisontale linje i figuren, gjennomføres en eksklusiv-eller-operasjon av polynomet og datameldingen. Resultatet 011 av polynomdivisjonen for senderen erstatter de tre nullerene før pakken blir sendt over bussen til mottakeren. Mottakeren gjennomfører en ny polynomdivisjon, og hvis denne går opp, betyr det at meldingen ble motatt uten feil. Dette er også gjengitt i figuren.

Syklisk redundanssjekk blir brukt til å oppdage bitfeil. Ulike CRC-protokoller og polynom gir ulike muligheter til å oppdage antall bitfeil og hvor langt det er mellom bitfeilene. Med CRC-15-protokollen som er standardprotokoll og polynom for CAN-bussen, kan det oppdages opp til fem uavhengige bitfeil, eller sprangfeil på 15 bit. Sprangfeil vil si at det er feil på to bit, og avstanden mellom disse blir kalt et sprang[31]. Finnes det feil mellom disse to bittene, vil denne feilen også bli oppdaget. Hvis det finnes feil i datapakken, vil mottakeren sende ut en feilmelding og be om at datapakken sendes på ny.

3.4.3.4 Bit-timing på CAN-buss

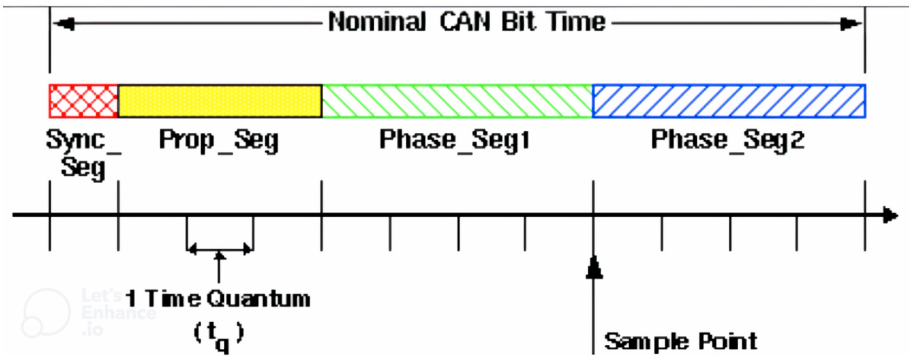
Dette delkapittelet omhandler konfigurasjon av bit-timing, og er basert på dokumentet laget av Florian Hartwich og Armin Bassemir for Robert Bosch GmbH [62]. Det var Robert Bosch GmbH, ofte forkortet til Bosch, som startet utviklingen av CAN i 1983[55].

Bit-timing for modulene på CAN-bussen er vesentlig for at kommunikasjonen skal fungere bra og uten feil. CAN-protokollen støtter hastigheter på opptil 1 Mbps, og alle modulene på nettverket må kjøre på samme bitrate for å kunne kommunisere med hverandre. Hver modul på CAN-bussen har sin egen klokkegenerator, ofte i form av en krystalloscillator (forklares i kapittel 4.4.4). Oscilleringsfrekvensen på disse klokkegeneratorene kan være ulike fra modul til modul, og det er her bit-synkronisering kommer inn i bildet. Ved hjelp av bit-timing for hver modul på CAN-bussen, kan modulene synkroniseres til samme bitraten selv om de har ulik klokkefrekvens.

Bitraten som kjøres avgjør hvor mange bit som sendes per sekund. Hvert av disse bittene har en gitt lengde og bruker en bestemt tid for å sendes. Tiden det tar å sende et bit kalles for den nominelle bit-tien, t_{NBT}, og kan finnes som den inverse av ønsket bitrate, $\boxed{t_{NBT} = CAN_{bitrate}^{-1}}$. En bitrate på $500kbps$ vil gi en nominell bit-tid på $t_{NBT} = 500000^{-1} = 2000ns$. Dette gjelder for alle modulene på CAN-bussen som kjører på $500kbps$.

Tiden det tar å sende et bit kan også deles inn i flere mindre intervall, kalt tidskvantum eller t_q. Dette er vist i figur 3.13 nedenfor. Med andre ord er lengden av alle tidskvantaene summert, det samme som lengden på et bit, altså $\boxed{t_{NBT} = t_{Sync\_Seg} + t_{Prop\_Seg} + t_{Phase\_Seg1} + t_{Phase\_Seg2}}$.

[6] 1111 binært ville blitt representert som polynomet: $x^3 + x^2 + x + 1$



Figur 3.13: Illustrasjon for bit-timing [62]

For å oppnå ønsket bit-tid og bitrate, er det en rekke parameter som må konfigureres riktig, hvor hvert av parametrene påvirker antall tidskvantum og tiden til et tidskvantum. I henhold til CAN-standarden er den nominelle bit-tiden delt inn i fire tidssegmenter, også vist i figur 3.13 ovenfor: "*Syncronization Segment*", "*Propogation Time Segment*", "*Phase Buffer Segment 1*" og "*Phase Buffer Segment 2*". Bittet leses av mellom "*Phase Buffer Segment 1*" og "*Phase Buffer Segment 2*". Hvert tidssegment består av et programmerbart antall tidskvantum, og er 4 av de konfigurerbare parametrene. Summen av de fire tidssegmentene blir den nominelle bit-tiden. De to siste parametrene er "Baud Rate Prescaler" (BRP) og "Synchronization Jump Width" (SJW). Alle parametrene er gjengitt i tabell 3.14 nedenfor, og forklares under tabellen.

Parameter	Range	Remark
BRP	[1 .. 32]	defines the length of the time quantum t_q
Sync_Seg	1 t_q	fixed length, synchronization of bus input to system clock
Prop_Seg	[1 .. 8] t_q	compensates for the physical delay times
Phase_Seg1	[1 .. 8] t_q	may be lengthened temporarily by synchronization
Phase_Seg2	[1 .. 8] t_q	may be shortened temporarily by synchronization
SJW	[1 .. 4] t_q	may not be longer than either Phase Buffer Segment
This table describes the minimum programmable ranges required by the CAN protocol		

Figur 3.14: Viser minimums- og maksverdier for de ulike parametrene ved bit-timing. Tabell hentet fra *"The Configuration of the CAN Bit Timing"* [62]

BRP: "*Baud Rate Prescaler*" avgjør lengden på et tidskvantum. Formelen for dette er: $\boxed{t_q = BRP/f_{sys}}$ hvor f_{sys} er systemklokken. BRP brukes for å trappe ned klokken som kjøres ut på perifermodulen hvor CAN-bussen er. Med en systemklokke på 72Mhz og BRP på 9, vil man få en frekvens på 8Mhz og et tidskvantum på $t_q = 125ns$.

Sync_Seg: "*Syncronization Segment*" brukes for å synkronisere klokken til mottakeren opp mot klokken til senderen. Flanken på bittet er forventet å forekomme innenfor dette segmentet. Oppstår flanken utenfor Sync_Seg, blir avstanden mellom disse to en fasefeil. Dette segmentet er alltid satt til å være ett tidskvantum langt.

Prop_Seg: "*Propogation Time Segment*" brukes for å kompensere for de fysiske forsinkelsene på CAN-bussen. Det vil si at selv om det er forsinkelser på bussen, vil ikke en node kunne lese av et bit før denne tiden har gått, og dermed rekker bitverdier fra alle noder som sender ID-er å nå frem til alle nodene på nettverket før avlesningspunktet. Dette gjør at det er mulig å avgjøre hvilken node som har høyest prioritet selv med fysiske forsinkelser på nettverket.

Phase_Seg1: "*Phase Buffer Segment 1*" varer frem til punktet hvor bittet leses av. Hvis fasefeilen er positiv, økes lengden på dette segmentet[62]. Det vil si, hvis flanken som er forventet å forekomme i

synkroniseringssegmentet kommer for sent, vil lengden på "*Phase Buffer Segment 1*" økes med like mye som fasefeilen.

Phase_Seg2: "*Phase Buffer Segment 2*" varer fra det bittet leses av til slutten på bittet. Hvis fasefeilen er negativ, vil lengden på dette segmentet bli kortere[62]. Det vil si, hvis flanken som er forventet å forekomme i synkroniseringssegmentet kommer for tidling, vil lengden på "*Phase Buffer Segment 2*" kortes ned med like mye som fasefeilen.

SJW: "*Synchronization Jump Width*" avgjør hvor langt avlesningspunktet mellom "*Phase Buffer Segment 1*" og "*Phase Buffer Segment 2*" maksimalt kan forskyves for å kompensere for en fasefeil. Er fasefeilen større enn SJW, kan ikke hele feilen kompenseres for[62].

I noen tilfeller hvor det skal konfigureres og programmeres en CAN-buss, er noen av segmentene kombinert og gjengitt som ett parameter[62]. Dette er sant for CAN-buss på Nucleo32-STM32G431KB[63] og Nucleo64-STM32G431RB som programmeres i STM32CubeIDE. Her er parametrene gjengitt som:

- "Prescaler" som er det samme som BRP
- "Syns Jump Width" som er det samme som SJW
- "Time Seg1" som er "*Propogation Time Segment*" og "*Phase Buffer Segment 1*" kombinert
- "TimemSeg2" som er "*Phase Buffer Segment 2*"

"*Syncronization Segment*" er ikke et parameter som kan konfigureres, da denne alltid skal være ett tidskvantum. I figur 3.15 nedenfor er det vist hvordan et bit er bygget opp av segmentene: "*Syncronization Segment*", "*TSEG1*" og "*TSEG2*". Det er også vist hvordan BRP kan brukes for å redusere systemklokken som brukes på CAN-buss.

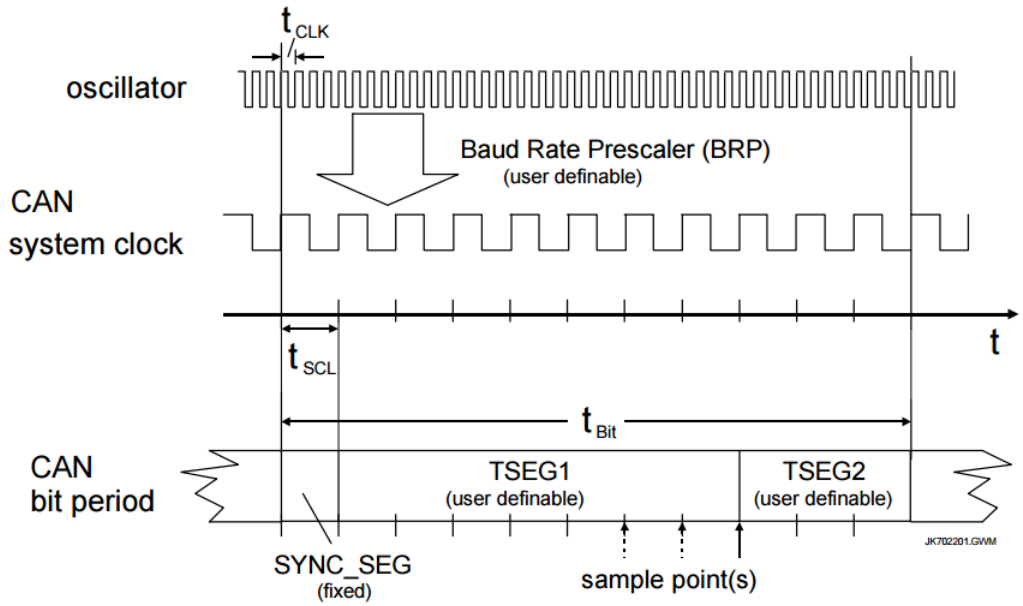


Fig. 2 Principle of deriving the bit period as implemented in Philips CAN controllers

Figur 3.15: Figuren viser hvordan f_{sys} (oscillator) blir delt på BRP for redusere systemklokken som brukes på CAN-bussen.[64]

Fra figuren ovenfor, kan det vises at den nominelle bit-tiden også er gitt av følgende formel: $\boxed{t_{NBT} = t_{Sync\_Seg} + t_{Seg1} + t_{Seg2}}$. I dette delkapittelet har det blitt gitt tre ulike formler for å beregne den nominell bit-tiden, og en formel for å beregne tiden til et tidskvantum. Formlene er oppsummert nedenfor:

$$Nominell\ Bittid : t_{NBT} = CAN_{bitrate}^{-1} \tag{3.1}$$

$$t_{NBT} = t_{Sync\_Seg} + (t_{Prop\_Seg} + t_{Phase\_Seg1}) + t_{Phase_Seg2} \tag{3.2}$$

$$t_{NBT} = t_{Sync\_Seg} + t_{Seg1} + t_{Seg2} \tag{3.3}$$

$$Tidskvantum : t_q = \frac{BRP}{f_{sys}} \tag{3.4}$$

Ved å bruke formel 3.1 og 3.3 for beregning av den nominelle bit-tiden og formel 3.4 for beregning av tiden til et tidskvantum, kan følgende forhold utledes for antall tidskvantum, $N_{tidskvantum}$ (3.6):

$$t_{NBT} = \frac{1}{CAN_{bitrate}} = t_{Sync\_Seg} + t_{Seg1} + t_{Seg2} = t_q(Sync\_Seg + Seg1 + Seg2)$$

$$\frac{1}{CAN_{bitrate}} = t_q(Sync\_Seg + Seg1 + Seg2) = \frac{BRP}{f_{sys}}(Sync\_Seg + Seg1 + Seg2)$$

$$CAN_{bitrate} = \frac{f_{sys}}{BRP(Sync\_Seg + Seg1 + Seg2)} \tag{3.5}$$

$$N_{tidskvantum} = (Sync\_Seg + Seg1 + Seg2) = \frac{f_{sys}}{BRP \cdot CAN_{bitrate}} \tag{3.6}$$

Parametrene som brukes for å konfigurere ønsket bit-timing kan beregnes ved hjelp av formlene som er blitt gitt i dette kapittelet. I neste delkapittel vil det gis et praktisk eksempel på hvordan formlene kan brukes til nettopp dette.

3.4.3.5 Eksempel på beregning av bit-timing

I forrige delkapittel ble det presentert en tabell med de ulike parametrene som brukes når bit-tiden skal bestemmes. Denne tabellen er gjengitt i figur 3.16 nedenfor igjen for enkelhets skyld. I dette delkapittelet vil det gjennomgås en praktisk beregning av disse parametrene ved hjelp av formlene som ble utledet i forrige delkapittel.

Parameter	Range	Remark
BRP	[1 .. 32]	defines the length of the time quantum t_q
Sync_Seg	1 t_q	fixed length, synchronization of bus input to system clock
Prop_Seg	[1 .. 8] t_q	compensates for the physical delay times
Phase_Seg1	[1 .. 8] t_q	may be lengthened temporarily by synchronization
Phase_Seg2	[1 .. 8] t_q	may be shortened temporarily by synchronization
SJW	[1 .. 4] t_q	may not be longer than either Phase Buffer Segment
This table describes the minimum programmable ranges required by the CAN protocol		

Figur 3.16: Viser minimums- og maksverdier for de ulike parametrene ved bit-timing. Tabell hentet fra *"The Configuration of the CAN Bit Timing"* [62]

Tabellen viser det minste og største antallet tidskvantum de ulike parametrene kan ha. Fra denne tabellen kommer det frem at $\boxed{1 \leq BRP \leq 32}$ og at $\boxed{Sync\_Seg = 1}$. Antall tidskvantum skal være mellom 8 og 25[62]. Det gir følgende grenser: $\boxed{8 \leq t_{Sync\_Seg} + t_{Seg1} + t_{Seg2} \leq 25}$. Dette er grenser som er med og påvirker beregningen av bit-timing.

Det skal beregnes parameter for to ulike noder med ulike systemklokker, og som skal ha samme bitraten. Node A har en systemklokke på $\boxed{f_{A\_sys} = 144MHz}$ og node B har en systemklokke på $\boxed{f_{B\_sys} = 100MHz}$. Begge nodene skal ha en bitrate på $\boxed{CAN_{bitrate} = 500kbps}$, og dermed en nominell bit-tid på $\boxed{t_{NBT} = 2000ns}$. Avlesningspunktet på bittet skal være på omtrent 60% av hele bittet. Parametrene beregnes først for node A, så for node B, før de til slutt oppsummeres og verifiseres.

Ved hjelp av formel 3.4 fra forrige delkapittel, kan antall tidskvantum for node A beregnes. Antall tidskvantum skal som nevnt tidligere være innenfor 8 og 25. I tillegg til dette, må antallet være et heltall. Dette gir flere ulike valg av BRP, da BRP f.eks. kan være 8, 16, 18 og 24. I dette tilfellet velges det $BRP = 18$.

$$N_{tidskvantum} = (Sync\_Seg + Seg1 + Seg2) = \frac{f_{A\_sys}}{BRP \cdot CAN_{bitrate}} = \frac{144 \cdot 10^6 Hz}{18 \cdot 500 \cdot 10^3} = 16 \tag{3.7}$$

Fra utregningen ovenfor, ender det opp med 16 tidskvantum. For å finne lengden på hvert tidskvantum, kan den nominelle bit-tiden deles på antall tidskvantum. Med en nominell bit-tid på $t_{NBT} = 2000ns$, blir lengden på hvert tidskvantum $t_q = 125ns$. Et bit blir avlest mellom $Seg1$ og $Seg2$. Avlesningspunktet skal være på omtrent 60% av bittet. Formelen nedenfor kan brukes for å beregne antall tidskvantum for $Seg1$ slik at enn oppnår ønsket avlesningspunkt. Som nevnt tidligere, har $Sync\_Seg$ en fast lengde på et tidskvantum, og det totale antallet tidskvantum er 16.

$$Avlesning[\%] = \frac{(Sync\_Seg + Seg1)}{N_{tidskvantum}} \cdot 100 = 60\%$$

$$Seg1 = 16\frac{60}{100} - 1 = 8,6 \text{–> Ikke OK}$$

For å oppnå et avlesningspunkt på nøyaktig 60%, må $Seg1 = 8,6$. Alle parametrene som brukes til å konfigurere bit-timingen må være et heltall. Runder derfor $Seg1$ opp til 9, og ser om avlesningspunktet fortsatt ligger på omtrent 60%:

$$Avlesning[\%] = \frac{(Sync\_Seg + Seg1)}{N_{tidskvantum}} \cdot 100 = \frac{(1+9)}{16} \cdot 100 = 62,5\% \approx 60\% \quad (3.8)$$
$$-> \text{OK}$$

Avlesningspunktet blir på 62,5% ved 16 tidskvantum og $Seg1 = 9$. Dette er nært nok 60% og dermed akseptabelt. Fra det faktum at $Seg1 = 9$ og $Sync\_Seg = 1$, følger det at de resterende tidskvantumne plasseres i $Seg2$.

$$(1 + Seg1 + Seg2) = 16$$
$$Seg2 = 16 - 1 - Seg1 = 16 - 1 - 9 = 6 \quad (3.9)$$

Ved å bruke parametrene: $BRP = 18$, $Seg1 = 9$ og $Seg2 = 6$, vil node A få en bitrate på 500kbps og et avlesningspunkt på 62,5% når det brukes en systemklokke på 144MHz.

De samme beregningene gjøres for å konfigurere node B som har en systemklokke på $f_{B\_sys} = 100MHz$. Målet er å konfigurere node B slik at den får en bitrate på 500kbps og et avlesningspunkt på rundt 60%. Også her er det flere verdier for BRP som vil fungere, men det velges $BRP = 10$.

$$N_{tidskvantum} = (Sync\_Seg + Seg1 + Seg2) = \frac{f_{B\_sys}}{BRP \cdot CAN_{bitrate}} = \frac{100 \cdot 10^6 Hz}{10 \cdot 500 \cdot 10^3} = 20 \quad (3.10)$$

Fra utregningen ovenfor, blir antallet tidskvantum for node B lik 20, og lengden på hvert tidskvantum blir da $t_q = 100ns$. Neste steg er å beregne hvor mange tidskvantum $Seg1$ skal være. Dette gjøres likt som for node A:

$$Avlesning[\%] = \frac{(Sync\_Seg + Seg1)}{N_{tidskvantum}} \cdot 100 = 60\%$$
$$Seg1 = \frac{60}{100} N_{tidskvantum} - Sync\_Seg = \frac{60}{100} 20 - 1 = 11 \quad (3.11)$$
$$-> \text{OK}$$

Ved å bruke $Seg1 = 11$, blir avlesningspunktet nøyaktig på 60%. Som for node A, kan $Seg2$ for node B beregnes slik:

$$(1 + Seg1 + Seg2) = 20$$
$$Seg2 = 20 - 1 - Seg1 = 20 - 1 - 11 = 8 \quad (3.12)$$

Ved å bruke parametrene: $BRP = 10$, $Seg1 = 11$ og $Seg2 = 8$, vil node B få en bitrate på 500kbps og et avlesningspunkt på 60% når det brukes en systemklokke på 100MHz. Resultatet av beregningene for node A og node B er oppsummert i tabell 3.1 nedenfor.

Noder	**Klokke** [f_{sys}]	**BRP**	**Seg1**	**Seg2**	**Avlesning** [%]	t_q [ns]
Node A	144MHz	18	9	6	62,5	125
Node B	100MHz	10	11	8	60	100

Tabell 3.1: Konfigurerbare parameter for å oppnå en bitrate på 500kbps

For å verifisere at de parametrene som er beregnet for node A og node B faktisk gir en bitrate på 500kbps, kan CAN-bussen i STM32CubeIDE settes opp med de ulike parametrene. Resultatet for node A er vist i figur 3.17a nedenfor og resultatet for node B er vist i figur 3.17b nedenfor.

Bit Timings Parameters	
Nominal Prescaler	18
Nominal Time Quantum	125.0 ns
Nominal Time Seg1	9
Nominal Time Seg2	6
* Nominal Time for one Bit	2000 ns
* Nominal Baud Rate	500000 bit/s

(a) Konfigurasjon av node A i STM32CubeIDE. Systemklokke på 144MHz

Bit Timings Parameters	
Nominal Prescaler	10
Nominal Time Quantum	100.0 ns
Nominal Time Seg1	11
Nominal Time Seg2	8
* Nominal Time for one Bit	2000 ns
* Nominal Baud Rate	500000 bit/s

(b) Konfigurasjon av node B i STM32CubeIDE. Systemklokke på 100MHz

Resultatet i figurene ovenfor verifiserer utregningen av bit-timing for de ulike nodene. Begge nodene får en bitrate på 500kbps, og vil dermed være i stand til å kommunisere med hverandre over CAN-bussen. SJW er i begge tilfellene satt til 1. Det vil si at et bit kan maksimalt forlenges eller forkortes med 1 tidskvantum ved fasefeil og synkronisering av klokkene. Dette er ikke urimelig for dette prosjektet hvor linjene på CAN-bussen typisk er maksimalt 0,5m, men teoretisk sett kunne strekt seg til 40m og likevel opprettholdt en bitrate på 1Mbps. Den fysiske forsinkelsen på bussen vil dermed ikke bli særlig stor, noe som fører til en lav fasefeil.

3.4.4 CAN-FD-Buss

Dette delkapittelet er basert på informasjon fra delkapittelet om CAN (3.4.3) og wikipediasiden for CAN-FD [65].

CAN-FD (*Flexible Data*) er en videreutviklet CAN-bussprotokoll som er raskere og mer fleksibel enn standard CAN-protokoll, selv om det er en del likheter. Her er en liste med hovedforskjellene mellom CAN-FD og den klassiske CAN-bussen:

- Overføringshastighet av data opp mot 5Mbps til 8Mbps, men og dynamisk bytte mellom overføringshastighetene. Det vil si 5-8 ganger hurtigere enn vanlig CAN.
- Kan overføre datapakker på 1 til 64 byter, i motsetning til vanlig CAN som kan overføre 8 byter på det meste.

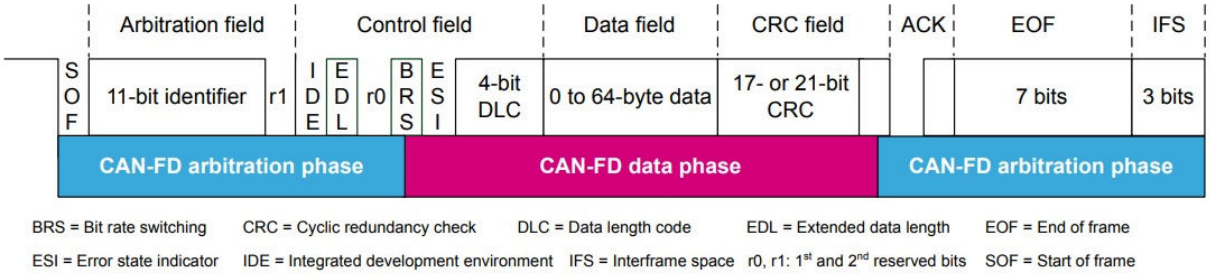
- CAN-FD har mulighet til å kjøre 21-bits syklisk redundanssjekk (CRC). Dette brukes når pakkestørrelsen er mellom 16 og 64 byter.

Selv om CAN-FD har en høyere overføringshastighet enn vanlig CAN, er dette kun ved sending av selve dataen, markert "*data phase*" i figur 3.18 nedenfor. I de to fasene markert "*arbitration phase*" kjører den på maksimalt 1 Mbps som vanlig CAN. Dette blir gjort for at arbitreringsfasen skal gå korrekt for seg slik at modulen med lavest ID skal få prioritet. Hadde modulene kjørt på ulik hastighet, hadde det vært en fare for at den raskeste modulen fikk prioritet på tross av høyere ID.

The data sent is packaged into a message as shown in the figure below. A CAN-FD message can be divided into three phases:

1. a first arbitration phase
2. a data phase
3. a second arbitration phase

Figure 1. Standard CAN-FD frame



Figur 3.18: Illustrasjon av CAN-FD-format ved sending av data [66]

CAN-FD-protokoll er kompatibelt med en klassisk CAN-protokoll som CAN 2.0A/B. Dette gjelder ikke andre veien. Dermed kan en modul med CAN-FD brukes i et nettverk med moduler som kjører vanlig CAN, men da mister man alle fordelene med CAN-FD.

Også for CAN-FD må det settes parameter for bit-timing som nevnt i delkapittel 3.4.3.4, og ved høyere overføringshastigheter blir naturligvis tiden for et tidskvantum mindre. Dette betyr dermed at en CAN-FD-buss ikke er i stand til å takle signalforsinkelser like bra som en vanlig CAN-buss. Alle nodene på bussen må oppleve samme nivå på bussen, og dermed kan ikke den fysiske lengden på bussen være for lang. Dette for at arbitreringen skal gå rett for seg og at datapakken ikke skal inneholde feil. Ved 40 meter er overføringshastigheten maksimalt 1Mbps, som er det samme for den vanlige CAN-bussen, og ved 250 meter synker hastigheten til 250 Kbps[67]

3.4.5 Nettverk

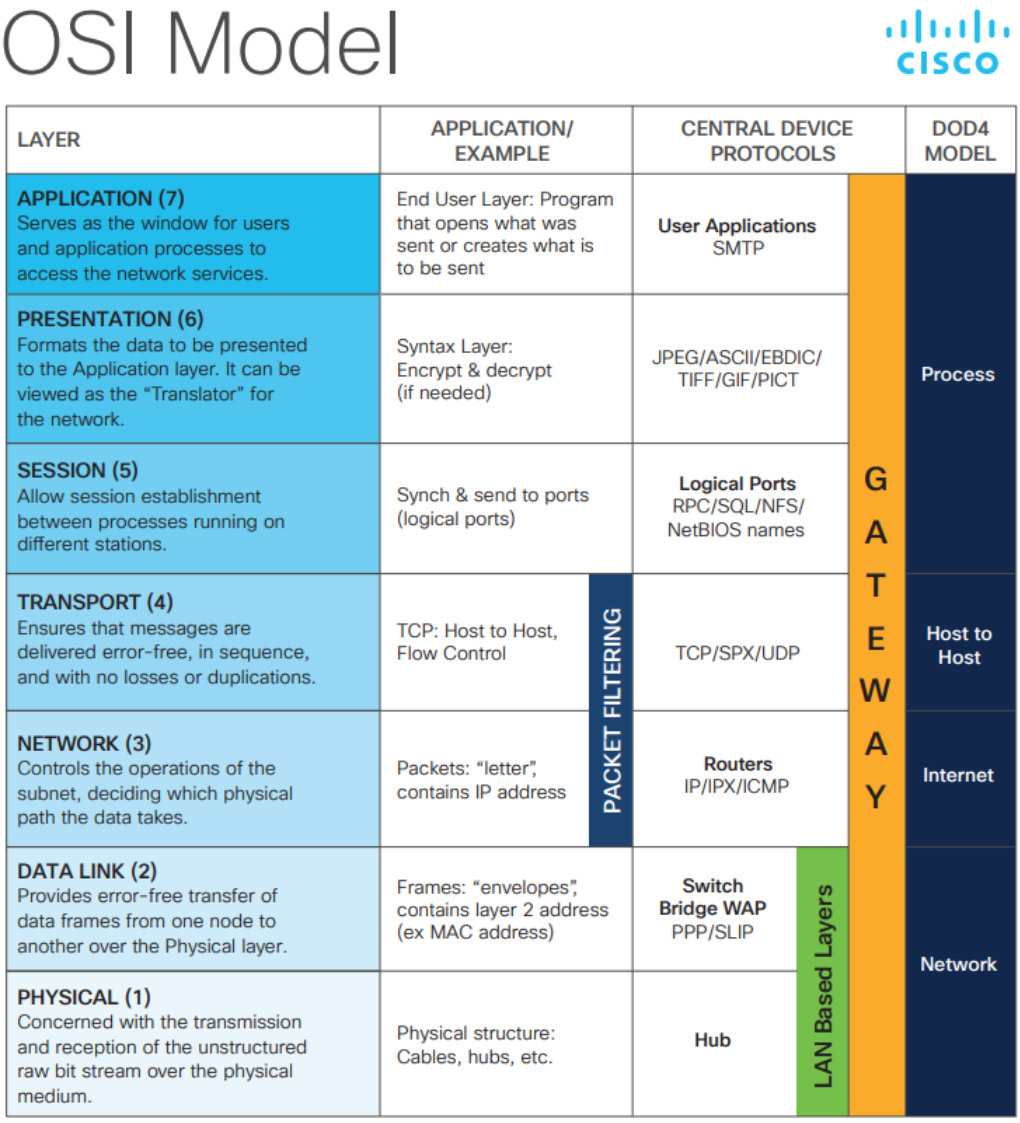
I dette delkapittelet vil diverse standarder og protokoller for nettverkskommunikasjon over Ethernet, og teorien bak dette bli gjennomgått.

3.4.5.1 OSI-Modell

Teorien i dette delkapittelet er basert på Wikipedia sin side om OSI [68].

Open Systems Interconnenction model (OSI-modellen) er en referansemodell definert av organisasjonen ISO. Modellen består av syv lag som beskriver den logiske oppbygningen av et nettverk.

Nedenfor i figur 3.19 er et utdrag fra leverandøren av nettverksutstyr - *Cisco* - sin forklaring av OSI-modellen.



Figur 3.19: Grafisk presentasjon av OSI modell[69]

Hvert lag i OSI modellen er beskrevet i mer detalj nedenfor:

- **Lag en** er definert som det fysiske laget. Det fysiske laget tar hånd om å sende datastrømmen som en bitstrøm over kabler.
- **Lag to** er definert som datalinklaget. Dette laget håndterer overføringen av datapakker mellom enheter via lag en.
- **Lag tre** er definert som nettverkslaget. Nettverkslaget kontrollerer hvor dataen sendes ved hjelp av adresser, typisk ved bruk av en IP-adresse. Nettverkslaget har også jobben med å dele opp segmenter hvis de er for store, samt sette dem sammen igjen hos mottaker.
- **Lag fire** er definert som transportlaget. Transportlagets oppgave er å håndtere at: Pakker som blir sendt kommer frem i den tilstanden de ble sendt i, at store pakker deles opp i mindre pakker for sending og at pakkene settes sammen igjen riktig ut fra de oppdelte pakkene.
- **Lag fem** er definert som sesjonslaget. Sesjonslaget setter opp og kontrollerer tilkoblinger mellom enheter kalt *sesjoner*. Sesjonslaget håndtererer også *duplex*-operasjoner som *full-duplex* som er sending og mottak på likt, *half-duplex* som er at en enten sender eller mottar men ikke på samme tid, eller *simplex* som er enveis kommunikasjon.
- **Lag seks** er definert som presentasjonslaget. Presentasjonslaget håndterer enkoding, dekoding, kryptering og dekryptering av data slik at den kan brukes i applikasjonslaget.
- **Lag syv** er definert som applikasjonslaget. Applikasjonslaget er hvor brukerer kan programmere eller bruke en applikasjon til å kommunisere nedover laget ved bruk av protokoller som *HTTPS*, *websockets*, *Modbus-TCP/IP*.

3.4.5.2 Ethernet

Ethernet er en kommunikasjonsstandard utviklet for å muliggjøre kommunikasjon mellom enheter på et lokalt nettverk, også kjent som LAN. Ethernet ble først utviklet med bruk av koaksialkabler som overføringsmedium mellom enhetene, men i dag brukes hovedsakelig fiberoptiske kabler eller tvunne parkabler (*Twisted Pair Cable*) bestående av 4 par (8 ledere). Ethernetstandarden har utviklet seg over tid for å tilby og imøtekomme kravene om høyere båndbredde og hastighet, samt bedre skjerming for å opprettholde signalintegriteten i overføringen.

Båndbredden er et teoretisk mål for hvor mye data som maksimalt kan overføres gjennom en kommunikasjonskanal, og måles vanligvis i frekvensområdet. Mens overføringshastighet refererer til hvor raskt dataen faktisk kan overføres, og måles vanligvis i bit per sekund (bps).

Ethernet brukes også i nettverksformene; MAN[7] *(Metropolitan area networks)* og WAN (*Wide area networks*). Ethernet er hva man kan beskrive som en del av det fysiske laget i OSI modellen (delkapittel 3.4.5.1). I tabell 3.2 nedenfor kan man se hvordan båndbredden og hastigheten har utviklet seg, samt behovet for skjermet kabler.

Kategori	Skjerming	Overføringshastighet [100m]	Båndbredde
Cat-3	Uskjermet	10 Mbps	16 MHz
Cat-5	Uskjermet	100 Mbps	100 MHz
Cat-5e	Uskjermet	1 Gbps	100 MHz
Cat-6	Skjermet/Uskjermet	1 Gbps	250 MHz
Cat-6a	Skjermet	10 Gbps	500 MHz
Cat-7	Skjermet	10 Gbps	600 MHz

Tabell 3.2: Tabell med spesifikasjoner for noen av kategorikablene[71]

[7]MAN kan være flere LAN-nettverk knyttet sammen i et spesifikt geografisk område [70]

Listen med kategorikabler[71] ovenfor er ikke fullstendig da den ikke tar med alle kablene, men viser hvordan utviklingen har gått. Koaksialkabelen kommer i mange ulike varianter og har en stor variasjon i både overføringshastighet og båndbredde, og brukes stort sett av telekommunikasjonsselskap. En del av grunnen til dette er at koaksialkabelen er avhengig av bra utstyr for mottak og sending i begge ender. Derfor er det mer vanlig å bruke kategorikabler og fiberkabler i dagens nettverk. Fiberkabelen baserer seg på en eller flere optiske ledere som kan overføre informasjon i form av lys. Dette gjør at vi idag kan overføre rundt 100Gbps til vanlig. Men i 2012 demonstrerte et Japansk firma at de klarte sende data over 50 kilometer med en hastighet på 1 Pbps (Peta bit per second)[72].

IEEE 802.3 - [31]

"1000BASE-T er en Ethernet standard som beskriver 1000Mbit/s overføring over 4 tvinna par med kober kabel (Kat 5 eller opp) over avstander på 100 m, mens 1000BASE-LX10 Ethernet beskriver 1000Mbit/s overføring over fiber (typisk enkelmodus $9\mu m$ kjerne $125\mu m$ beskyttelse) med lys som har ein bølgelengde på ca 1310nm og maksimalavstander på 10 km."

3.4.5.3 TCP - Transmission Control Protocol

Teorien i dette delkapittelet er basert på Wikipedia sin side om TCP [73].

TCP er en nettverksprotokoll som opererer i transportlaget på datanettet etter *OSI-modellen* [74] og definerer hvordan data skal pakkes, sendes og mottas over et nettverk. TCP-protokollen tar hånd om at dataen blir overført på en pålitelig måte ved flere metoder. En av metodene er ved å segmentere dataen i pakker med en bestemt størrelse. Denne størrelsen kan være mellom 512 og 1500 byter. En standard verdi på dette er 1024 byter. Hvert segment som sendes startes med en "segmentheader" som består av 64 bytes. Som vist i figur 3.20 nedenfor, inneholder "headeren" forskjellig data som kontrollerer og passer på at kommunikasjonen foregår på tiltenkt vis.

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if *data offset* > 5. Padded at the end with "0" bits if necessary.)																															
⋮	⋮																																
60	480																																

Figur 3.20: TCP IPv4 segment header [73]

Source Port: Et 16-bits ord som indikerer hvilken port senderen sender pakken fra.

Destination Port: Et 16-bits ord som indikerer hvilken port som skal motta pakken.

Sequence number: Har to funksjoner basert på *SYN*-flagget i byte 13. Hvis *SYN* er logisk høy så er dette det initielle sekvensnummeret, og hvis *SYN*-flagget er logisk lavt, så forteller det hvilken akkumulerte segment som blir sendt.

Acknowledgment number: Blir brukt hvis *ACK*-flagget er logisk høyt i byte 13. 32-bits ordet innholder neste sekvensnummer som mottakeren forventer å motta fra senderen. Dette blir gjort for å bekrefte at dataen blir mottatt i riktig rekkefølge.

Data offset: Et 4-bits ord som spesifiserer lengden på TCP-headeren. TCP-headeren har en minimumslengde på fem 32-bits ord og en maksimumslengde på 15 ord.

Reserved: Er 4 bit som er reserverte til fremtidig bruk og skal alltid være satt til 0.

Flags: Er et 9-bits kontrollord som ligger på byte 12 og 13 som nevnt tidligere. Disse flaggene har forskjellige funksjoner og egenskaper, og er beskrevet i listen under:

- **CWR**: *Congestion window reduced* flagget blir satt av sender for å indikere at den har mottatt et TCP-segment med ECE-flagget på og har respondert med en mekanisme for å kontrollere opphopning.
- **ECE**: *ECN-Echo*-flagget har to roller basert på om *SYN*-flagget er logisk høyt eller lavt. Ved logisk høyt, indikerer flagget at enheten man kommuniserer med er *ECN*-kapabel. Er *SYN* logisk lavt så indikerer *ECN*-flagget at en pakke med *Congestion Experienced flag (ECN=11)* er mottatt. Dette indikerer hos senderen at pakkene har opphopet seg hos mottakeren.
- **URG**: Dette flagget indikerer at *Urgent pointer* 16-bits ordet blir brukt og om verdiene i ordet er reelle.
- **ACK**: Dette flagget indikerer at *Acknowledgement number* 32-bits ordet blir brukt.
- **PSH**: *Push*-flagget etterspør om å flytte alt dataen i bufferet til mottakerens applikasjon.
- **RST**: Dette flagget starter nettverkstilkoblingen på ny.
- **SYN**: *Synchronize sequence numbers*-flagget etterspør om å synkronisere sekvensnummeret mellom sender og mottaker. Bare den første pakken som blir sendt skal ha dette flagget på.
- **FIN**: Dette flagget indikerer at det er den siste pakken som blir sendt.

Window size: Er et 16-bits ord som blir brukt av mottakeren til å gi beskjed til senderen hvor mange byte som kan sendes før mottakeren sender en *ACK*-bekreftelse tilbake. Dette er en vital del av flytkontrollen til TCP-protokollen. Verdien er dynamisk og justeres for å optimalisere hastigheten.

Checksum: Er et 16-bits ord som blir brukt for feilsjekking av TCP-headeren og datapakkene. Sjekksumfeltets innhold blir regnet ut med en algoritme som først legger sammen alle 16-bitsordene fra alle feltene i TCP-headeren der man setter sjekksumfeltet til 0. Hvis antall byte i headeren er et oddetall legges det til en ekstra byte med nuller for å gjøre antallet til en partallsverdi. Deretter legges datafeltet til i sekvensen. Sekvensen blir så delt inn i 16-bits ord og summert. Hvis summen av

antall byte er et oddetall legges det til en ekstra byte med nuller før den siste summen blir bergenet. Resultatet av summeringen blir delt i to 16-bits ord for så å bli invertert og sendt som sjekksummen. Mottakeren gjør samme beregning for å sjekke validiteten til mottatt segment. Eksempel på utregning av sjekksum vises i figur 3.21 nedenfor, der datapakken "This is a test" blir sendt fra port 12345 til port 80 med sekvensnummer 1000.

TCP Datapakke

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	12345																80															
4	32	1000																															
8	64	0																															
12	96	5				0				0								1024															
16	128	0																0															
20	160	0																															
Data		This is a test																															
Utregning TCP sjekksum																																	
Summering header		0x3039 + 0x0050 + 0x03E8 + 0x0000 + 0x5000 + 0x0000 + 0x0000 + 0x4000 + 0x0000 + 0x0000																															
Resultat header		0x9FB1																															
Summering data		0x5468 + 0x6973 + 0x2069 + 0x6120 + 0x7465 + 0x7374 + 0x0000																															
Resultat data		0x35B0																															
Sum		0x9FB1 + 0x35B0 = 0xD561																															
Delt i to ord		0xD5 + 0x61 = 0x136																															
Invertert (Resultat)		0xECA9																															

Figur 3.21: TCP IPv4 sjekksum

Urgent pointer: Hvis *URG*-flagget er på, viser dette 16-bits ordet til den posisjonen hvor kritisk data som kommer ligger, og at dataen skal prioriteres av mottaker.

Options: Dette feltet har en valgfri lengde og brukes til å gi ekstra informasjon eller konfigurasjonsmuligheter for TCP-tilkoblingen. Feltet kan brukes av både sender og mottaker for å tilpasse TCP-tilkoblingen etter behov eller krav. Feltet har en justerbar lengde som stilles med *Data offset*-feltet. Hver opsjon har en egen 8-bits opsjonskode og en verdi for konfigurering av opsjonen. Noen vanlige opsjoner er for eksempel "Maximum segment size" som angir den største datamengden senderen kan sende i et segment.

Når en skal etablere og drive kommunikasjon over TCP-protokollen blir det utført en treveis "handshake". De tre stegenene i "handshaken" er:

- **Initieringssteget** av tilkoblingen gjøres ved at klienten sender en forspørsel om å opprette en TCP-kobling til serveren. Dette gjøres med at segmentet har "**SYN**"-flagget på og har en tilfeldig verdi i "**Sequence number**"-ordet.
- **Bekreftelsessteget** av tilkoblingen gjøres ved at serveren svarer på forespørselen ved å send tilbake et segment med "**ACK**" og "**SYN**"-flaggene på, samt at "**Acknowledgment number**"-ordet inneholder verdien fra initieringssegmentet +1. Serveren velger også sitt eget nummer i "**Sequence number**"-ordet.
- **Avtalesteget** av tilkoblingen gjøres ved at klienten sender et segment med "**ACK**"-flagget på der "**Sequence number**" er verdien fra initieringssegmentet +1 og "**Acknowledgment number**" er verdien i fra "**Acknowledgment number**" fra bekreftelsessegmentet +1.

Initierings- og bekreftelsesstegene etablerer bekreftelses- og sekvensnummer fra klient til server. Bekreftelses- og avtalestegene etablerer bekreftelses- og sekvensnummer fra server til klient. Når denne "handsha-

ken" er fullført har klienten og serveren mottatt den nødvendige bekreftelsen om at "full-duplex"-kommunikasjon er etablert.

Når kommunikasjonen er etablert er det flere egenskaper TCP-protokollen har for at overføring av dataen skjer på en pålitelig måte. Noen av disse er:

- **Ordered data transfer:** Som nevnt tidligere blir dataen delt opp i segmenter som får hvert sitt sekvensnummer. Mottakeren vil da kunne sette sammen dataen i rett rekkefølge.
- **Retransmission of lost packets:** For hvert segment som blir sendt svarer mottakeren med en bekreftelse av segmentet. Hvis et segment ikke mottas, vil senderen sende segmentet på ny.
- **Error-free data transfer:** Hvis "**CRC**"-sjekken av pakken som mottas ikke stemmer, blir pakken håndtert som en tapt pakke og mottakeren svarer ikke med en bekreftelse av meldingen.
- **Flow-control:** Flytkontrollen i TCP-protokollen er en mekanisme for å justere overføringshastigheten når kommunikasjonen er etablert. Den gjør det ved å justere "**Window size**" etter hva nettverket og maskinvaren takler.
- **Congestion-control:** Opphopningsmekanismen i TCP-protokollen har i oppgave å unngå og overbelaste nettverket. Den gjør det ved hjelp av mekanismer som flytkontroll og "Slow start", som er at den begynner overføringen med en lav hastighet og øker hastigheten basert på responsen fra nettverket. "Congestion avoidance" som reduserer hastigheten hvis det oppdages tegn på overbelastning. "Fast retransmit" brukes til å gjenopprette tapte pakker raskere. Når senderen mottar tre duplikate bekreftelser, sendes pakken på ny med nye justerete verdier for "Window size" og "Slow start threshold".

Når en skal avslutte tilkoblingen blir det utført en fireveis "handshake". Initiering av frakoblingen skjer ved at klienten eller serveren sender en melding med "**FIN**"-flagget på. Mottakeren av denne meldingen sender så to meldinger tilbake, en med "**ACK**"flagget på for å bekrefte at frakoblingen er i gang og en med "**FIN**" for å si at den er klar for frakoblingen. Serveren eller klienten som initierte frakoblingen sender til slutt en bekreftelse for at tilkoblingen er avsluttet.

3.4.5.4 UDP - User Datagram Protocol

Teorien i dette delkapittelet er basert på Wikipedia sin side om UDP[75].

UDP er en annen nettverksprotokoll som også opererer i transportlaget i *OSI-modellen* (3.4.5.1). I motsetning til TCP har UDP mye mindre "overhead" (ekstra data som er nødvendig for å sende en melding) med en kortere header på totalt 64 bit. UDP er designet for enklere kommunikasjon og har ingen "handshake". Derfor egner UDP seg godt for applikasjoner der en ikke trenger: feilsjekking av meldinger, man håndterer feilsjekking av datapakker på applikasjonsnivå eller at nytteformålet er tidssensitivt. "Overheaden" til UDP er mindre hovedsaklig på grunn av at man slipper tre meldinger for å starte tilkoblingen, to meldinger for hver segment som sendes og fire meldinger for å avslutte. Hver melding som blir sendt med UDP, blir kalt et datagram og består av en header og dataen som sendes. Strukturen til headeren er vist under i figur 3.22.

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

Figur 3.22: UDP datagram header[75]

Source Port: Er et 16-bits ord som indikerer hvilken port senderen sender pakken fra.

Destination Port: Er et 16-bits ord som indikerer hvilken port som skal motta pakken.

Length: Er et 16-bits ord som indikerer lengden på datagrammet. Maksimums lengden er 65535 og gir 65527 byter med data siden headeren består av 8 byter. Bruker man IPv4 så har en 65507 byter man kan bruke til å sende data da IP-headeren tar 20 byte.

Checksum: Er et 16-bits ord som blir brukt for feilsjekking av headeren og data. Sjekksummen blir utregnet på samme måte som i TCP-protokollen som vist i delkapittel 3.4.5.3. Forskjellen på UDP og TCP er at i UDP-protokollen er sjekksummen valgfri å bruke. Under er et eksempel av en sjekksumutregning i figur 3.23.

UDP Datapakke

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	12345																80															
4	32	32																0															
Data		This is a test																															
Utregning UDP sjekksum																																	
Summering header		0x3039 + 0x0050 + 0x0020 + 0x0000																															
Resultat header		0x60A9																															
Summering data		0x5468 + 0x6973 + 0x2069 + 0x6120 + 0x7465 + 0x7374 + 0x0000																															
Resultat data		0x35B0																															
Sum		0x60A9 + 0x35B0 = 0x9629																															
Delt i to ord		0x96 + 0x29 = 0xBF																															
Invertert (Resultat)		0x40																															

Figur 3.23: UDP sjekksum eksempel utregning

Som nevnt tidligere har UDP, i motsetning til TCP, ingen til- og frakoblingssekvens eller bekreftelse av meldinger. Den virker istedenfor med at man sender datagrammene til en spesifikk port. Dette gjør at UDP-protokollen har mulighet for flere overføringsmetoder. To av disse er "Multicast" og "Unicast".

Med sending av data over "Unicast" sendes data til en spesifikk IP-adresse og portnummer. Med sending over "Multicast" sendes data til en multicast-gruppeadresse og port der hver mottaker som har koblet seg opp til denne gruppeadressen og porten mottar meldingen. "Multicast" egner seg derfor godt til multimedia-applikasjoner.

3.4.5.5 RTP - Real-time Transport Protocol

Teorien i dette delkapittelet er basert på Wikipedia sin side om RTP[76]

RTP er en nettverksprotokoll som opererer i applikasjonslaget i OSI-modellen. RTP er designet for multimediastrømming over nettverk i sanntid. RTP kjøres vanligvis over UDP men kan også kjøres over TCP. UDP er den foretrukne metoden, hovedsaklig på grunn av lav forsinkelse. RTP-protokollen inneholder funksjonalitet for håndtering av pakketap og usynkroniserte pakker. Hver pakke har en "header" som er minimum 12 byte, men kan være lengre hvis man legger til opsjoner. Headeren er vist i figur 3.24 under.

RTP packet header

Offsets	Octet	0								1								2								3							
Octet	Bit [a]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version		P	X	CC				M	PT							Sequence number															
4	32	Timestamp																															
8	64	SSRC identifier																															
12	96	CSRC identifiers …																															
12+4×CC	96+32×CC	Profile-specific extension header ID																Extension header length															
16+4×CC	128+32×CC	Extension header …																															

Figur 3.24: RTP header[76]

Flagg og ord i headeren er beskrevet i mer detalj under.

- **Version** Er et 2-bits ord som indikerer hvilken versjon av RTP-protokollen det er. Versjonen som er i bruk i dag er versjon 2.
- **P** Er et bit som står for "padding" og indikerer om det skal fylles med bytes på slutten av RTP-pakken slik at den er av en fast lengde.
- **X** Er et bit som står for "extension" og indikerer om det skal en applikasjons-header mellom RTP-headeren og dataen. Denne headeren vil være plassert der hvor "Extension header" er plassert.
- **CC** Er et 4-bits ord som står for "CSRC count" og forteller hvor mange "CSRC Identifiers" er i bruk.
- **M** Er et bit som står for "Marker" og indikerer til applikasjonen som mottar pakken at dataen som kommer er av en spesiell betydning.
- **PT** Er et 7-bits ord som står for "payload type" og forteller hva formatet på dataen er.
- **Sequence number** Er et 16-bits ord som forteller hvilken pakke det er som blir sendt. Dette blir inkrementert for hver RTP-pakke, og mottakeren bruker det for å detektere tap av pakker og håndtere hvis pakkene kommer frem i feil rekkefølge.
- **Timestamp** Er et 32-bits ord som mottakeren bruker til å spille av mediastrømmen til rett tid og intervall.
- **SSRC identifier** Er et 32-bits ord som står for "Synchronization source identifier" og indikerer hva som er kilden til mediastrømmen. Verdien til denne er unik for hver RTP-sesjon.
- **CSRC identifers** Er et 32-bits ord som står for "Contributing source identifiers" som brukes når man har flere kilder for mediastrømmen. Dette er en unik ID for hver kilde som bidrar til mediastrømmen, og lengden av denne blir som nevnt tidligere satt av "CC".

- **Profile specific extension header ID** Er et 16-bits ord som er frivillig å bruke. Dette ordet forteller hvilken "extension header" som blir brukt.
- **Extension header length** Er et 16-bits ord er frivillig å bruke. Dette ordet indikerer lengden på "Extension headeren"
- **Extension header** Dette er område en eventuelt ekstra applikasjons-header vil være plassert.

Fra headeren ser man at, hvis man har en kilde og ikke legger til "extension headers", har man en header på 12 bytes som allikevel kan ta seg av å spille mediastrømmen i rett rekkefølge og håndtere tapte meldinger.

3.4.6 USB - Universal Serial Bus

USB er en seriell databuss for å koble til enheter og utstyr til en datamaskin[77]. Industristandarden definerer kabler, kontakter, kommunikasjonsprotokoller og strømforsyning mellom datamaskiner og elektronisk utstyr[77]. USB-standarden baserer seg på en skjermet kabel med fire ledninger, hvor to av lederene er til strøm (+5V og jord), og to ledere er tvunnet og er for dataoverføring. Overføringshastigheten på USB forbedres stadig ved utgivelser av nye USB-versjoner. Listen nedenfor som er hentet fra [78] er en oversikt over de forskjellige USB-standardene.

Chart 1: USB Cable Types, Standards and Speeds

Standard	Also Known As	Logo	Year Introduced	Connector Types	Max. Data Transfer Speed	Cable Length**
USB 1.1	Full Speed USB		1998	USB-A USB-B	12 Mbps	3 m
USB 2.0	Hi-Speed USB		2000	USB-A USB-B USB Micro A USB Micro B USB Mini A USB Mini B USB-C*	480 Mbps	5 m
USB 3.2 Gen 1	USB 3.0 USB 3.1 Gen 1 SuperSpeed		2008 (USB 3.0) 2013 (USB 3.1)	USB-A USB-B USB Micro B USB-C*	5 Gbps	3 m
USB 3.2 Gen 2	USB 3.1 USB 3.1 Gen 2 SuperSpeed+ SuperSpeed 10Gbps		2013 (USB 3.1)	USB-A USB-B USB Micro B USB-C*	10 Gbps	3 m
USB 3.2 Gen 2x2	USB 3.2 SuperSpeed 20Gbps		2017 (USB 3.2)	USB-C*	20 Gbps	3 m
USB 4	USB4 Gen 2×2 USB4 20Gbps		2019	USB-C*	20 Gbps	0.8 m
USB 4	USB4 Gen 3×2 USB4 40Gbps		2019	USB-C*	40 Gbps	0.8 m

* USB-C is more accurately known as Type C or USB Type C

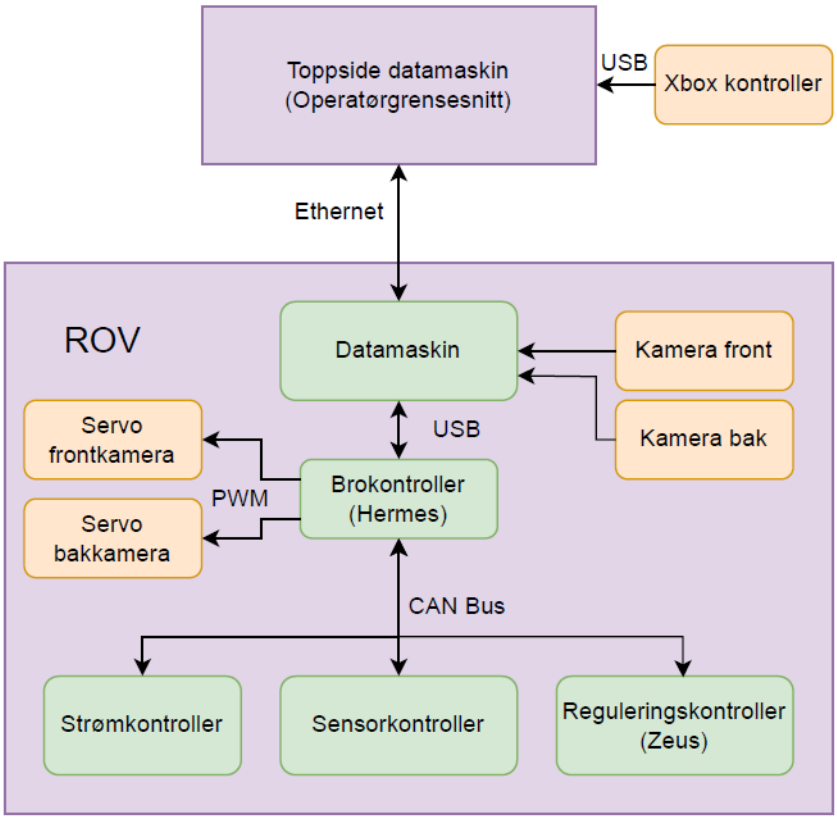
** Cable length is the length covered by the specification. Longer lengths can be achieved using active cables and in some cases, longer passive cables

Figur 3.25: Utklipp av tabell fra:[78]

I de neste delkapittelene vil først relevante erfaringer som ble gjort i fjor bli gjennomgått i delkapittel 3.5. Deretter vil valgene av kommunikasjonsmetoder som er blitt tatt og begrunnelser for dette komme frem i delkapittel 3.6.

3.5 Erfaringer fra tidligere

I dette delkapittelet vil det bli sett på hvilke kommunikasjonsmetoder som er blitt brukt av grupper fra tidligere år, og hvilke erfaringer disse gruppene gjorde seg. Dette tas med for å få læredom om valg som kanskje ikke fungerte så bra og bør endres på, men også for å se hva som fungerte bra og kan tas med videre. Delkapittelet er basert på lignende bachelor fra 2022 [31]. En oversikt over hvilke kommunikasjonsprotokoller som er brukt hvor i fjorårets design er gjengitt i figur 3.26 nedenfor.



Figur 3.26: Blokkskjema over kommunikasjonsflyt fra tidligere bachelor. Hentet fra:[31]

3.5.1 Kommunikasjon internt ombord i ROV

I fjor gikk kommunikasjonen mellom de ulike kretskortene over CAN-buss som vist i figur 3.26 ovenfor, mens kommunikasjonen mellom brokontrolleren og datamaskinen gikk over USB. Dette ble gjort da datamaskinen ikke hadde en egen CAN-modul, og derfor måtte en mikrokontroller med CAN-modul kommunisere med datamaskinen over USB.

Fra testrapport A.2 i bacheloren fra ifjor som omhandler "Kommandoforsinkelse og kommandointegritet" blir det konstantert med at meldingene som ble sendt kom frem uten endringer eller feil, med en snittid på 1,5ms en vei mellom kretskort og kontrollstasjonen. Snittiden på 1,5ms var forventet da USB-kommunikasjonen kun klarte sende en melding omtrent hvert millisekund [31]. Forsinkelsen ved bruk av USB mellom datamaskin og brokontroller vil være en "flaskehals" for kommunikasjonshastigheten, og vil være ønskelig å endre på.

Det ble laget en I/O-liste for alle ID-ene som ble brukt på CAN-bussen i fjor. Denne listen bestod av 24 forskjellige pakker, altså 24 forskjellige ID-er på CAN-bussen. Det ble bestemt at hastigheten på CAN-bussen skulle tåle at disse 24 pakkene ble sendt med en frekvens på 20 Hz, altså 480 pakker i

sekundet.

3.5.2 Kommunikasjon mellom kontrollstasjon og ROV

I fjor ble det brukt TCP-protokoll mellom ROV og kontrollstasjonen. For videostrømmen ble det utviklet en Python-funksjon som kunne sende bilder over UDP-protokollen ved bruk av "OpenCV". Forsinkelsen på bildestrømmen var da på rundt 200ms [31]. TCP-protokollen sørget for at all dataen som ble sendt, kom frem feilfritt. Sendingen av dataen hadde en forsinkelse på 1,5ms[31].

Fjorårets prosjekt brukte fiber mellom ROV og kontrollstasjonen med adaptere for konvertering til CAT6-Ethernet i hver ende. Under MATE-konkurransen var det en person som tråkket på tjoren, som gjor at fiberen ble ødelagt. Det ble derfor byttet ut med en vanlig Ethernetkabel etter konkurransen.

3.6 Valg av kommunikasjonsstandarder

I dette delkapittelet kommer valgene av kommunikasjonsmetoder frem, samt begrunnelser for disse valgene. Valgene baserer seg på gjennomgått teori i de foregående delkapittlene, og erfaringer fra tidligere oppgaver fra UiS Subsea.

3.6.1 Valg av intern kommunikasjonsmetode ombord i ROV-en

I delkapittel 3.4 som omhandler ulike kommunikasjonsstandarder, er det gått gjennom flere kommunikasjonsmetoder som er aktuelle å bruke ombord i ROV-en. Fordeler og ulemper ved de forskjellige kommunikasjonsmetodene vil oppsummerest før det gjøres et valg.

SPI er en vanlig kommunikasjonsmetode som baseres seg på et master/slave-oppsett og bruker 4 ledere. Sending og mottak av data kan foregår samtidig da det er en standard som kjører i "full duplex". Overføringshastigheten begrenses av hver enhet som skal kommunisere over bussen, og dens evne til å kjøre ønsket klokkefrekvens. En SPI-standarden kan påvirkes av støy og tilbyr ikke feil-deteksjon ved sending og mottak av data.

I^2C er en vanlig kommunikasjonsmetode som baseres seg på et master-slave-oppsett og bruker 2 ledere med "pull-up"-motstander. Sending og mottak av data foregår ikke samtidig da det er en standard som kjører i "half duplex". Overføringshastigheten kan varierere mellom 100Kbps og 5Mbps. En I^2C-standarden kan påvirkes av støy og tilbyr ikke feil-deteksjon ved sending og mottak av data.

USB er en kommunikasjonsmetode som baserer seg på master-slave-oppsett og bruker 4 ledere. Ved bruk av USB-versjon USB 3.0 og opp kjøres kommunikasjonen i "full duplex", mens for lavere versjoner kjøres den i "half duplex"[79]. USB 3.0 tilbyr en overføringshastighet på 5Gbps. USB-standarden kan påvirkes av støy, men inneholder feildeteksjon ved sending og mottak av data.

CAN er en kommunikasjonsmetode hvor det ikke trengs en master for å styre kommunikasjonen, men det brukes et ID-basert prioritetssystem for å avgjøre hvilke node som får sende på bussen ("half duplex"). Bussen er bygget opp av 2 ledere og bruker et differensialsignal. Overføringshastigheten kan strekke seg opp mot 1Mpbs. CAN-standarden tilbyr feildeteksjon, og feil på en node fører ikke til at bussen streiker. På grunn av differensialsignalet er også bussen robust mot støy.

Av de fire kommunikasjonsmetodene nevnt ovenfor, er CAN den mest robuste og stabile når det kommer til støy og påvirkninger utenifra. Kommunikasjon over en CAN-buss vil også gi tilstrekkelig overføringshastighet for mengden data som skal sendes ombord i ROV-en. Det er også fordelaktig at bussen ikke er avhengig av at alle nodene fungerer som de skal, slik at kommunikasjonen kan opprettholdes selv om en mikrokontroller slutter å kommunisere. CAN er standard å bruke i biler og andre kjøretøy, og det er derfor tenkt at den er egnet i en ROV. Da CAN-bussen ifjor fungerte bra, og det var USB-kommunikasjonen som var flaskehalsen, taler også dette for CAN-buss. Det er mye god informasjon på nett om hvordan CAN-buss kan implementeres, men også mye god hjelp fra tidligere oppgaver fra UiS Subsea. CAN-buss velges derfor som kommunikasjonmetode internt ombord i ROV-en mellom de ulike kretskortene.

Gruppen som har ansvar for sensorkortet har også delt ut temperatursensorer som hver gruppe må montere på sitt kort. Denne sensoren kommuniserer over I^2C. Dermed vil det også bli brukt I^2C-kommunikasjon på kommunikasjonskortet.

3.6.2 Valg av kommunikasjonsmetode mellom ROV og kontrollstasjon

3.6.2.1 Ethernet

Det ble tidlig i prosjektet bestemt at man i år skulle gå bort fra fiber mellom ROV og kontrollstasjonen. Hovedsaklig på grunn av kostnaden, samt leveringstid hvis man opplever et uhell og kabelen blir ødelagt. Dette skjedde i fjor under MATE-konkurransen hvor en person tråkket på tjoren og fiberen ble ødelagt. I tillegg til dette ville det ikke oppnås noe ekstra overføringshastighet, da fiberlinjen måtte ha gått gjennom en adapter i begge ender som konverterer mellom fiber og Ethernet. Derfor ville overføringshastigheten fortsatt være begrenset til 1Gbit/s inntil 100 meter. Målsetningen for årets ROV er å ha mulighet til å gå ned til en dybde på 100 meter. På denne lengden vil en Cat6 Ethernetkabel være tilstrekkelig da den har en oppgitt hastighet på 1 Gbit/s.

3.6.2.2 Protokoll for kommunikasjon av styrekommandoer og informasjon

Valg av kommunikasjonsprotokoll for styrekommandoer og informasjon ble bestemt i fellesskap med datagruppen. Dette fordi det er datagruppen som mottar datastrømmen og behandler den. På grunn av den lave datamengden som blir sendt, vil det ikke være en betydningsfull ulempe å kjøre en tyngre protokoll som TCP med tanke på forsinkelser. Velger vi TCP får vi feilsjekking, flytkontrol, og opphopingensmekanisme på protokollnivå og kan være trygge på at dataen som sendes mellom ROV-en og kontrollstasjonen kommer frem og er feilfri. Velger vi UDP må vi selv utvikle en driver for dette som må inneholde logikk for feilsjekking av data, om kommunikasjonen er brutt og etterspør en ny melding hvis det blir funnet en feil. Det er sannsynlig at denne driveren legger til mer forsinkelse samt at vi mister flytkontroll og opphopnings mekanikk som allerede er en del av TCP-protokollen.

Datagruppen hadde også et ønske om å bruke TCP-driveren som ble utviklet i fjor om igjen. Viktigheten av at dataen kommer frem feilfritt, og ønsket til datagruppen tatt i betraktning. Velges det derfor TCP-protokoll for flyten av styrekommandoer og informasjon mellom ROV-en og kontrollstasjon.

3.6.2.3 Protokoll for videostrøm

For valg av protokoll og pakketype for videostrøm ble disse protokollene vurdert:

- TCP
- UDP
- RTP over UDP

TCP-protokollen ble vurdert som for tung for datamengden som skal overføres. For hvert segment som skal sendes med TCP, må vi vente på en bekreftelse før neste segment kan sendes. Headeren er også 64 bytes, som skal sendes i hvert segment. Hvert segment har også en maksimum lengde på 1500 byte. Dette gjør at det blir en betydelig mengde headere og "protokoll-snakk" for hvert bilde eller bildeserie som skal sendes.

UDP-protokollen er betydelig lettere å drive og headeren krever kun 8 bytes for hvert datagram. Lengden på datagrammet kan være opp til 65535 bytes, som gir oss 65527 bytes tilgjengelig til data. Dette gjør at vi kan sende kun en eller noen få headere for hvert bilde eller bildeserie som må sendes. UDP-protokollen har også muligheten for "Multicast", som gjør at flere enheter kan lese av den samme videostrømmen uten at vi trenger å sette opp en ekstra strøm for sending.

Vi vurderte noe pakketap som akseptabelt så lenge forsinkelsen er lav. For å prøve og minimere dette, ønsket vi også å ha en protokoll for synkron avspilling og håndtering av pakketap. Vi valgte derfor RTP som gir denne funksjonaliteten. RTP gir 12 byte ekstra data for hvert segment. Kombinasjonen med RTP over UDP gir da en total header-størrelse på 20 byte, som er 44 byte mindre en TCP.

3.7 Resultat og oppsummering

Resultatet av valgene som er blitt tatt angående kommunikasjonsmetoder blir gjennomgått i dette delkapittelet. Dette inkluderer kommunikasjonen internt og med kontrollstasjon, samt videostrømmen. Diskusjon rundt hva som fungerte bra, og hva som fungerte dårlige angående kommunikasjonen - vil bli diskutert videre i kapittel 8.

3.7.1 Kommunikasjon internt og med kontrollstasjon

For å verifisere at kommunikasjonen internt og med kontrollstasjonen fungerte, ble det gjort flere tester. Disse testene er dokumenter i testrapport A.1. Testene gikk ut på å finne forsinkelsen på meldinger og integriteten av meldingene.

Fra test A.1, kom det frem at forsinkelsen på en melding tur-retur mellom en mikrokontroller ombord i ROV-en og kontrollstasjon hadde en forsinkelse på 2,13ms. En melding en vei mellom to mikrokontrollere intert ombord i ROV-en hadde en forsinkelse på $231,83\mu s$. Fra disse to resultatene, er det åpenbart at flaskehalsen i systemet er Nvidia Jetson Nano når den skal videreformidle meldinger. På tross av dette, ble det beregnet at kommunikasjonslinjen mellom mikrokontroller og kontrollstasjonen vil være i stand til å overføre omtrent 939 pakker i sekundet. Dette er mye mer enn kravet som er satt for kommunikasjonslinjen på 160 pakker hvert sekund.

Internt ombord i ROV-en var kravet at det skulle kunne sendes 16 pakker med frekvens på 20Hz, altså 320 pakker hvert sekund. Med en gjennomsnittlig forsinkelse på $231,83\mu s$ mellom to mikrokontrollere, kan denne linjen maksimalt ta 4313 pakker i sekundet. Kravet er med andre ord ingen problem og oppfylle.

Under igangkjøring og testing av ROV, er alle meldingene som skal sendes og mottas over CAN-bussen blitt sjekket. Alle meldingene kom frem der de skulle uten feil, og med en akseptabel forsinkelse. Dette viste at CAN-bussen og TCP-linjen opp til land opererte som ønsket.

3.7.2 Videostrøm

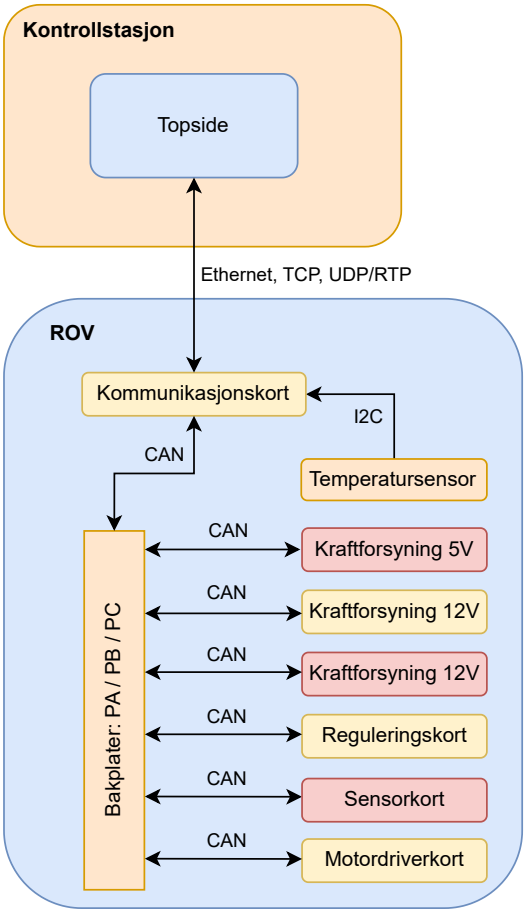
I testrapport A.3 er sending av videostrøm dokumentert og testet. Metodene som ble testet var:

- TCP
- UDP
- RTP over UDP

Testen viste at TCP-forsinkelsen var betydelig. Den ble målt til å være rundt 150ms senere en UDP sending. I testen brukte vi samme metoden for komprimering som ble brukt i fjorårets prosjekt, der vi oppnår samme forsinkelse som i fjor ved bruk av OpenCV. Deretter testet vi en alternativ metode for komprimering ved bruk av programvare gStreamer. Denne metoden ga oss en lavere forsinkelse og var på omtrent 100ms. Dette er et resultat som er godt innenfor rammene vi har satt som krav til forsinkelse. Dette viser at valget av programvare som blir brukt for komprimering har mye å si for forsinkelsen som oppleves på videostrømmen.

3.7.3 Oppsummering

I kapittelet som omhandler kommunikasjon, er det blitt gjennomgått flere kommunikasjonsmetoder for å realisere kommunikasjon internt ombord i ROV-en og mellom kontrollstasjon og ROV. Valgene av kommunikasjonsmetoder som ble tatt er gjengitt i figur 3.27 nedenfor.



Figur 3.27: Blokkskjema som illustrerer kommunikasjonsmetodene som ble valgt

Mellom ROV-en og kontrollstasjon, ble det brukt Ethernet, hvor styredata og informasjon ble sendt ved hjelp av TCP-protokollen, mens videostrømmen ble overført ved hjelp av UDP/RTP. Internt ombord i ROV-en ble CAN-buss brukt for å formidle informasjon og styredata mellom kretskortene. Temperatursensoren sender data til kommunikasjonskontroller over I^2C.

Kapittel 4

Maskinvare

Kapitteloversikt

For å realisere ROV-en i henhold til behovs- og funksjonsbeskrivelsen ble flere alternativ til maskinvare på kommunikasjonskortet vurdert. I dette kapittelet legges de forskjellige vurderingene frem, samt argumentasjon for valgene som ble tatt. Blokkskjemaet i figur 4.1 er utgangspunktet før valg av maskinvare, og i slutten av kapittelet i figur 4.15 er valgene representert.

4.1 Problemstilling

Kommunikasjonskortet med maskinvare har to hovedoppgaver:

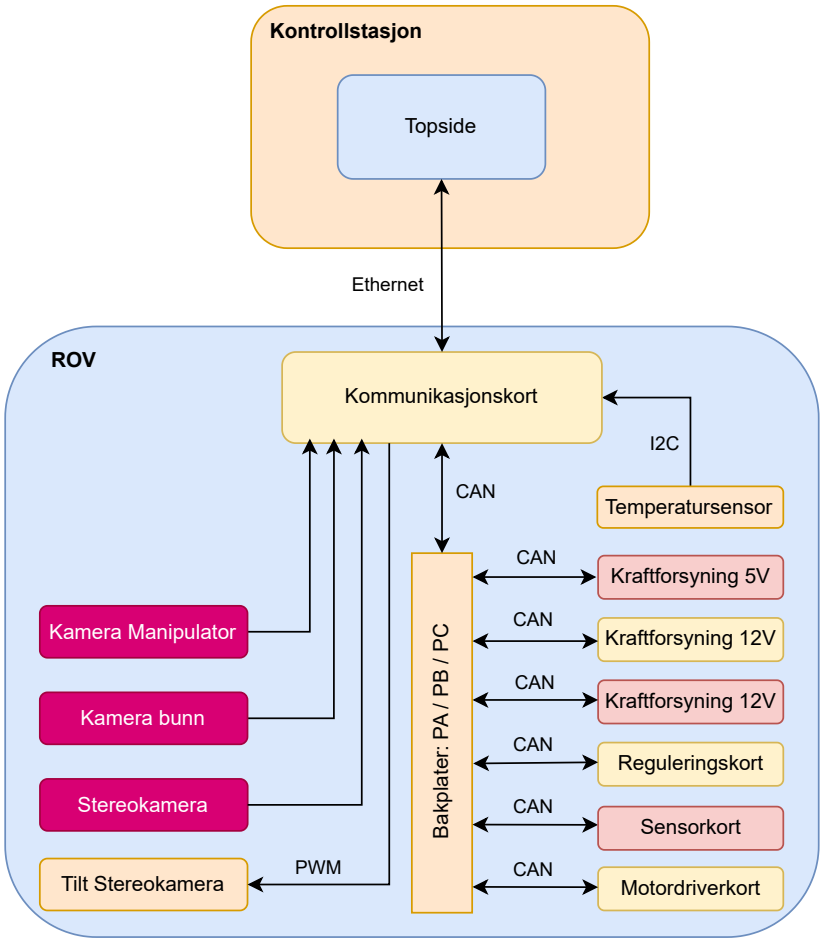
- Fungere som en kommunikasjonsbro mellom kontrollstasjon og de ulike systemene nede i ROV-en
- Ta inn bildestrømmer fra kamera og sende bildestrøm opp til kontrollstasjon

Kommunikasjonsbro - Informasjon skal kunne sendes og mottas mellom de ulike kretskortene og kontrollstasjonen. I kapittelet som omhandler valg av kommunikasjonsstandarder (3.6) ombord på ROV-en ble det bestemt at kommuniknformasjon skal kunne sendes og mottas mellom de ulike kretskortene og kontrollstasjonasjonen mellom de ulike kretskortene skal gå over CAN-buss, og kommunikasjonen mellom ROV-en og kontrollstasjon skal gå over Ethernet. Dermed må det velges maskinvare som muliggjør dette.

Sending av bildestrøm - På ROV-en skal det monteres tre ulike kamera. To enkle kamera skal monteres, et på på undersiden av ROV-en og et på manipulatoren. Det siste kameraet er et stereokamera med tilt som skal monteres i front for å kunne ha dybdesyn på ROV-en. Dette gir oss totalt 4 videostrømmer som skal sendes fra ROV-en opp til kontrollstasjonen. Maskinvaren må være kraftig nok til å behandle fire bildestrømmer, samt drive kommunikasjonen. Denne må kunne sendes og ankomme kontrollstasjonen uten at det blir for store forsinkelser. For å tilte kameraet må det også være en liten servomotor som styres ved hjelp av et PWM-signal[1], maskinvaren må dermed ha mulighet til å sende et slikt signal.

Blokkskjemaet nedenfor viser utgangspunktet før valg av maskinvare tas, og består foreløpig kun av valgene som ble tatt i forrige kapittel.

[1] Pulsbreddemodulering



Figur 4.1: Blokkskjema for å illustrere utgangspunkt før valg av maskinvare

Kort oppsummert skal det velges maskinvare som realiserer muligheten for kommunikasjon over CAN-buss, I^2C og Ethernet (TCP, UDP/RTP). Samtidig må kommunikasjonskontrolleren ha PWM utgang og mulighet for tilkobling av et stereokamera og to enkle kamera. Maskinvaren som ble valgt vil komme frem i et blokkskjema på slutten av kapittelet (figur 4.15) hvor "kommunikasjonskort" i blokkskjemaet erstattes med maskinvare.

4.2 Behovsspesifikasjon

- Kommunikasjonskontroller som kan realisere kommunikasjon mellom kontrollstasjon og de ulike kortene i ROV-en
- Kommunikasjonskontroller med program for kommunikasjon mellom kontrollstasjon og kretskort i ROV.
- Kommunikasjonskort eller kontroller som kan hente bildestrøm fra kamera å skal sendes opp til kontrollstasjon
- Lavest mulig forsinkelse på kommunikasjon og bilde
- Må ikke bli for varmt under drift
- Rimelig priset i henhold til budsjett
- Lav leveringstid
- Må kunne monteres på elektronikkortet eller i elektronikkhuset

4.3 Funksjonsspesifikasjon

- Forsinkelse på bildestrøm skal være lavere enn 400ms, men bør vær under 200ms.
- Må kunne kommunisere over CAN-buss og skal kunne sende/motta 16 pakker med en frekvens på 20Hz, det blir 320 pakker hvert sekund
- Må ha PWM-signal
- Må ha kamerainnganger til to enkle kamera og et stereokamera. Tre innganger gjør at det må velges et stereokamera med integrerte videolinjer, og fire gir mulighet for stereokamera med to seperate videolinjer.
- Ethernet med 1Gbit/s overføringshastighet
- Må ha I2C for kommunikasjon med temperatursensor (gitt av sensorgruppe)

4.4 Alternativ til løsninger

Det er flere alternativ for maskinvare å velge mellom for realisering av kommunikasjon og prosessering av bildedata. I denne seksjonen vil det bli diskutert hvilke alternativ som ble vurdert, og hvordan en slik løsning kunne blitt realisert. Selve valget av maskinvare kommer ikke før i seksjon 4.5.

4.4.1 Kommunikasjonkontrollere

I seksjonene nedenfor, vil de ulike kommunikasjonskontrollerene som er blitt vurdert gjennomgås. Det vil bli sett på spesifikasjoner, fordeler og ulemper ved hver kontroller. I siste seksjon (4.4.1.5), vil kommunikasjonskontrollerene bli oppsummert og sammenlignet i en tabell. Denne tabellen vil bli brukt når valg av kommunikasjonskontroller blir gjennomgått.

4.4.1.1 Nucleo-32 STM32G431KB

Nucleo-32 STM32G431KB (se figur 4.2) er et utviklerkort fra STMicroelectronics[63] basert på mikrokontrolleren STM32G431KB. Dette er en 32.pinners mikrokontroller som kan kjøres på 1,71V til 3,6V. Mikrokontrolleren bruker en ARM Cortex-M4-prosessor som kan kjøres på opptil 170 MHz, har 128 Kbyte flashminne[2] og 22 Kbyte SRAM[3].

I faget Datamaskinarkitektur[80] (ELE210) ble det brukt et Nucleo-64 STM32F103RB utviklerkort basert på STM32F103RB mikrokontrolleren. Denne mikrokontrolleren ble programmert i utviklingsmiljøet STM32CubeIDE hvor alle mikrokontrollere av typen STM32 kan programmeres. Dette er et utviklingsmiljø hvor vi har litt erfaring fra før av, og dermed blir det lettere og programmere en mikrokontroller av type STM32G431KB.

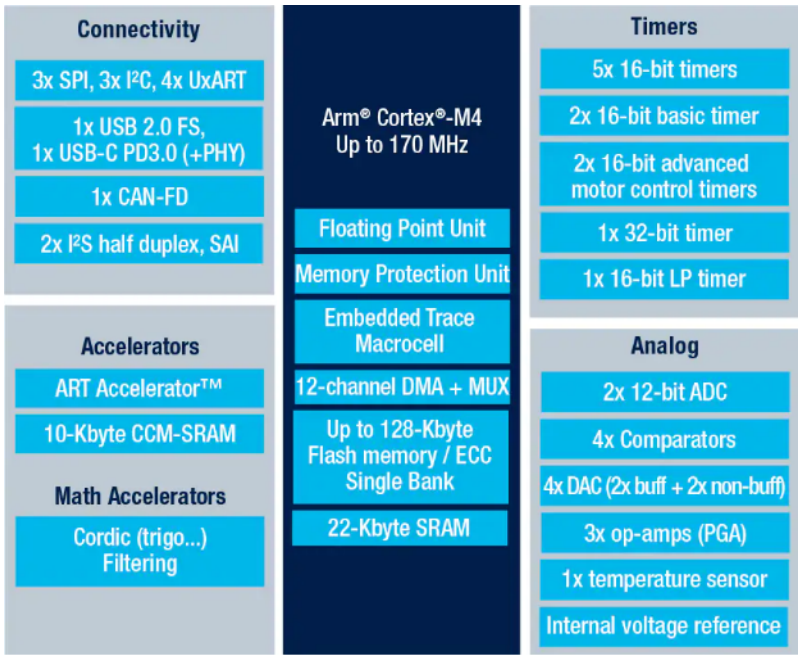
Innad i prosjektet er det blitt bestemt at de andre gruppene skal bruke denne mikrokontrolleren og STM32G431RB. Dette er to veldig like mikrokontrollere, hvor hovedforskjellen mello disse mikrokontrollerene er at KB-versjonen har 32 pinner, mens RB-versjonen har 64 og er litt kraftigere. Ved valg av en slik mikrokontroller, sikres lettere samarbeid blant gruppene, kompatibilitet og felles reservekort dersom noen må erstattes.

Figur 4.2: Nucleo-32 STM32G431KB fra Digi-Key.[63]

[2]Tett minnetype bygget opp av en spesiell transistor som er treg ved skriving men rask ved lesing.[61]

[3]Statisk RAM basert på "flip-flopper" beholder minnet så lenge det er strøm tilgjengelig som er rask både ved lesing og skriving.[61]

I figur 4.3 er det en overordnet oversikt over spesifikasjonene til STM32G431KB mikrokontrolleren. Her kommer det frem hvilke type perifermoduler GPIO-ene kan programmeres opp mot.



Figur 4.3: Spesifikasjoner for STM32G431KB.[81]

Mikrokontrolleren har flere perifermoduler på GPIO-ene med støtte for blant annet tre stykk SPI, tre stykk I2C, fire stykk UART/USART og en stykk CAN-FD. CAN-FD er som nevnt i 3.4.4 kompatibelt med CAN, og kan dermed brukes på en vanlig CAN-buss. I tillegg til dette, har mikrokontrolleren også to 16-bits PWM-timere dedikert til motorkontroll. Disse spesifikasjonene gjør mikrokontrolleren i stand til å løse mange oppgaver, men den er uegnet til å drive bildebehandling fra kamera og sende det opp til kontrollstasjon.

Fordeler:

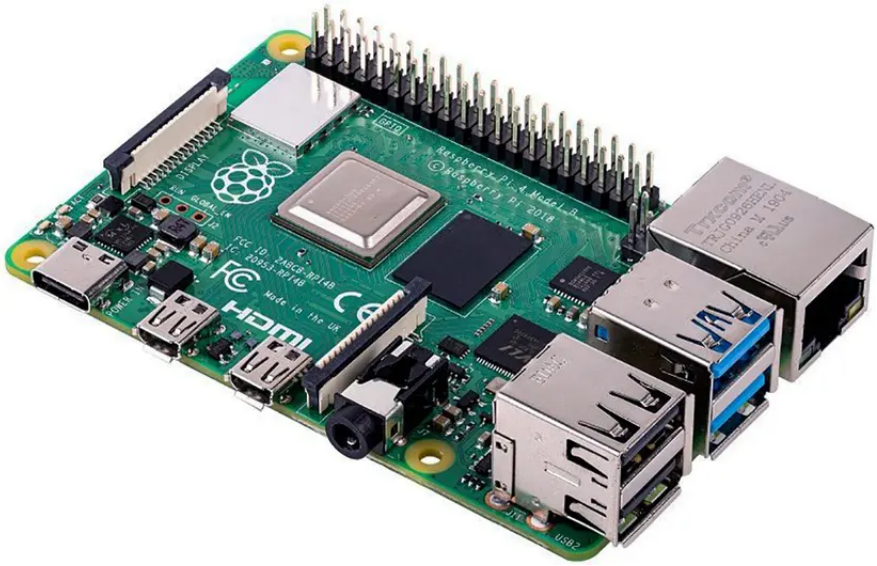
- Billig alternativ (rundt 130kr)
- Har mulighet for SPI, I2C, PWM, UART og CAN (FDCAN)
- Lavt effektforbruk
- Programmeres i STM32CubeIDE hvor gruppen har tidligere erfaring
- Tar relativt lite plass på elektronikkort
- Andre grupper i prosjektet bruker denne (og STM32G431RB)

Ulemper:

- Ikke kraftig nok til å drive bildebehandling
- Har ingen Ethernetport

4.4.1.2 Raspberry Pi 4 Model B

Det ble vurdert og bruke en Raspberry Pi 4 Model B som kommunikasjonskontroller. Dette er en liten ettkortsdatamaskin som kan programmeres i blant annet Python. Denne er meget populær blant *hobbymiljøet* for hjemmeprosjekt, og det finnes mye dokumentasjon tilgjengelig om ulike prosjekter hvor denne er blitt brukt. Denne har en 1Gbit/s Ethernetport for overføring av data, fire USB-innganger og en GPIO som kan konfigureres for å få SPI, I2C, UART og PWM. Raspberry Pi 4 Model B kan også drive videokomprimering ved hjelp av H.264-kodeken. Kodeken er mer forklart i delkapittel 5.6.2.



Figur 4.4: Raspberry Pi 4 Model B.[82]

Fordeler:

- Har 1Gbps Ethernetport for sending av data.
- Relativt rimelig i pris (900-1100kr)
- Har mulighet for SPI, I2C, PWM og UART
- Lavt effektforbruk
- Kan programmeres i Python
- Stort hobbymiljø som utvikler til Pi
- Har 2-linjers MIPI CSI-2
- H.264-kodek for videokomprimering

Ulemper:

- Lang leveringstid og vanskelig å få tak i
- Har ikke egen CAN-modul
- Har kun en 2-linjers MIPI CSI-2

4.4.1.3 NVIDIA Jetson Nano 4GB Developer Kit

Nvidia Jetson Nano er utviklet av Nvidia som er en verdensledende produsent av skjermkort. Kortet er basert på Nvidia sin Tegra brikke som for Jetson Nano har en GPU basert på Maxwell-arkitekturen og prosessor basert på ARM64 [83]. Kortet er utviklet for AI-applikasjoner i integrerte system og har god ytelse spesielt innen bildebehandling. Nvidia opplyser at GPU[4]-en har 572 GFLOPS [5]. Kortet har også innebygde multimedia-enkodere og dekodere for populære formater som H.264, H.265 og JPEG. Utviklerkortet som ble vurdert kommer med:

- fire USB-porter
- en gigabit Ethernetport
- to MIPI CSI-2 porter
- 40 pin header til GPIO

Gjennom GPIO headeren har man tilgang til følgende periferimoduler:

- to SPI
- tre UART
- fire I2C
- to I2S

Kamera kan kobles til de to CSI-portene. Utviklerkortet har egen dedikert maskinvare for å prosessere videostrømmen fra disse CSI-portene.

[4]Graphics processing unit
[5]Floating point operartions per second (FLOPS)

Figur 4.5: Nvidia Jetson Nano 4GB.[84]

Fordeler:

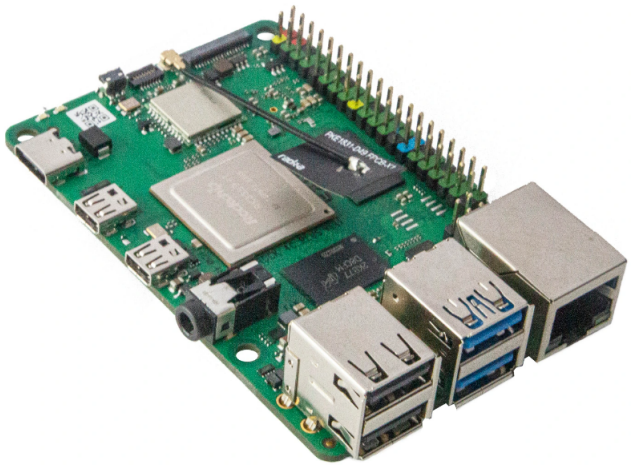
- Har 1Gbps Ethernetport for sending av data.
- maskinvare enkoder og dekoder for video komprimering
- Relativt rimelig pris i forhold til ytelse (1500-2000kr)
- Har mulighet for SPI, I2C, PWM og UART
- To innganger for MIPI CSI-2 (4 linjer)
- Lavt effektforbruk
- Kan programmeres i Python
- Stort profesjonelt miljø som utvikler til Jetson. Er også brukt i hobbymiljøet.
- H.264-kodek for videokomprimering

Ulemper:

- Har ikke en egen CAN-modul
- Tar relativt stor plass på elektronikkort
- Relativt dyr i forhold til andre alternativ

4.4.1.4 OKdo ROCK 4 Model C+ 4GB

OKdo Rock 4 Model C+ er enda en Single-Board Computer på markedet som er blitt vurdert. Denne har kraftig maskinvare som tåler å brukes til bildestrømmer. Den er et mindre kjent kort som ikke har nådd de store markedene enda. Den er basert på ARM prossesor, og har et innebygd grafikkort. Den markedsfører seg som et alternativ til Raspberry Pi, og har mye av de samme funksjonalitetene. Som for Raspberry Pi, har også OKdo ROCK støtte for videokomprimering ved hjelp av H.264-kodeken.



Figur 4.6: OKdo ROCK 4 Model C+ 4GB.[85]

Fordeler:

- GPIO med mulighet for SPI, PWM og I^2C
- 1 Gbps Ethernetport
- Relativt billig (700-800kr)
- H.264-kodek for videokomprimering

Ulemper:

- Har ikke egen CAN-modul
- Har kun en 2-linjers MIPI CSI-2

I den neste seksjonen (4.4.1.5), vil de ulike alternativene som kunne blitt brukt til kommunikasjonskontroller bli oppsummert og sammenlignet i en tabell. Denne tabellen brukes når selve valget tas i seksjon 4.5.

4.4.1.5 Sammenligning av kommunikasjonskontrollere

I tabellen nedenfor er de ulike nøkkelspesifikasjonene for de vurderte kommunikasjonskontrollerene blitt oppsummert.

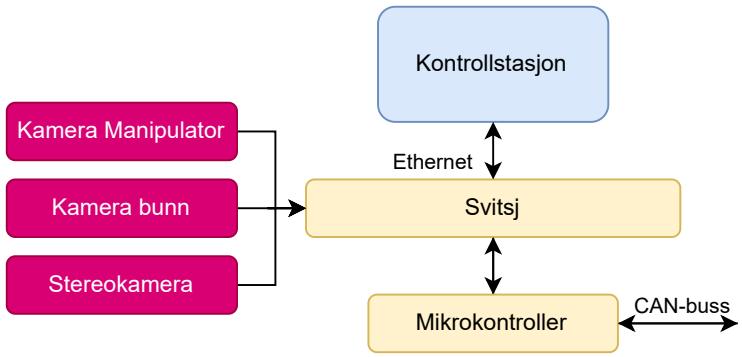
Denne tabellen vil bli benyttet senere i seksjon 4.5 hvor kommunikasjonskontrolleren blir valgt.

	Nucleo-32 STM32G431KB	Raspberry Pi 4 Model B	Nvidia Jetson Nano 4GB	OKdo ROCK 4 Model C+
Pris	ca. 130kr	ca. 1000kr	ca. 1500kr	ca. 750kr
Tilgjengelighet	Stor	Lav	Middels	Stor
Prossesorkraft	Lav	Middels	Høy	Middels
Antall USB	0	4	4	4
Antall MIPI CSI-2	0	1	2	1
H264 Enkoder	Nei	Ja	Ja	Ja
CAN	Ja	Nei	Nei	Nei
SPI	Ja	Ja	Ja	Ja
I2C	Ja	Ja	Ja	Ja
PWM	Ja	Ja	Ja	Ja
1Gbps Ethernet	Nei	Ja	Ja	Ja

Tabell 4.1: Sammenligning av ulike kommunikasjonskontrollere

4.4.2 Svitsj med IP-kamera

Et alternativ for å realisere kommunikasjon og prosessering av bildedata, er svitsj med IP-kamera. Dette vil være et mindre maskinvarekrevende alternativ, og gir muligheten til å velge en mindre kommunikasjonskontrollere som ikke vil være kraftige nok til å ta inn fire bildestrømmer, og så å sende de opp til kontrollstasjonen. Da kan alle bildestrømmene fra IP-kamera og kommunikasjonen fra kommunikasjonskontrolleren gå gjennom svitsjen og opp til kontrollstasjonen. Et blokkskjema av dette kan vises på figur 4.7.



Figur 4.7: Blokkskjema for å illustrere funksjon ved IP-kamera og svitsj

Fordeler ved å bruke svitsj:

- Slipper programvare for å ta seg av bildestrømmen da svitsjen tar seg av dette automatisk

- Kan gi mindre forsinkelser på bildestrøm da kommunikasjonskontrolleren ikke trenger å behandle bildestrømmen.
- Potensielt effektsparende da det ikke trengs mye kraft fra kommunikasjonskontroller for å håndere gjenstående oppgaver

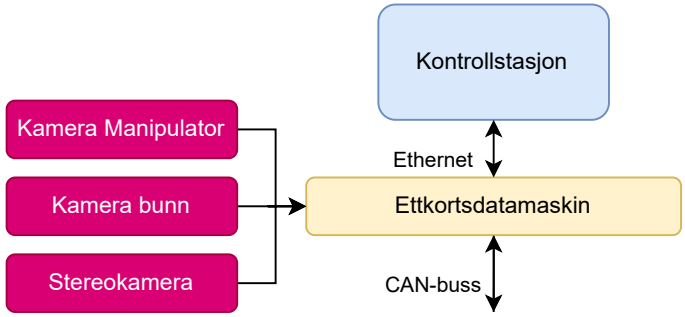
Ulemper ved å bruke svitsj:

- Tar opp mer av plassen på kommunikasjonskortet da det fortsatt er nødvending med en kommunikasjonskontroller i tillegg til svitsjen
- Mest sannsynlig dyrere enn en løsning uten svitsj da IP-kamera generelt er dyrere enn USB-kamera, spesielt IP-stereokamera

4.4.3 Bildedata gjennom kommunikasjonskontroller

Et alternativ for å prosessere bildedata og drive kommunikasjon er å ha en ettkortsdatamaskin[6]. En ettkortsdatamaskin vil ha prosessorkraften til å ta inn bildedata i tillegg til å drive CAN-bus.

Dersom en ettkortsdatamaskin skal ta inn bildedata, må den ha nok prosesseringkraft til å behandle dataen. For å sende bildedata videre, er det vanlig å komprimere data-en fra rådata til et annet format. Denne prosessen er krevende, og ikke alle ettkortsdatamaskiner klarer dette med liten forsinkelse. Det vil være en mye mindre plasskrevende løsnings, da en ettkortsdatamaskin vil ta mindre plass enn en svitsj og en mikrokontroller. Flere ettkortsdatamaksiner har også tilkobling for ethernet, ofte opp til 1Gb/s. Det er også vanlig med andre tilkoblinger som USB, og det gjør det mulig å koble opp rimeligere USB-kamera. Et blokkskjema av dette kan vises på figur 4.7.



Figur 4.8: Blokkskjema for å illustrere funksjon ettkortsdatamaskin

Fordeler ved å bruke ettkortsdatamaskin:

- Plassbesparende i forhold til svitsj
- Kan bruke rimelige USB-kamera
- Har ethernet tilkobling for 1Gb/s

Ulemper ved å bruke ettkortsdatamaskin:

[6]Single-board computer (SBC) er en komplett datamaskin bygget et på et kretskort.[86]

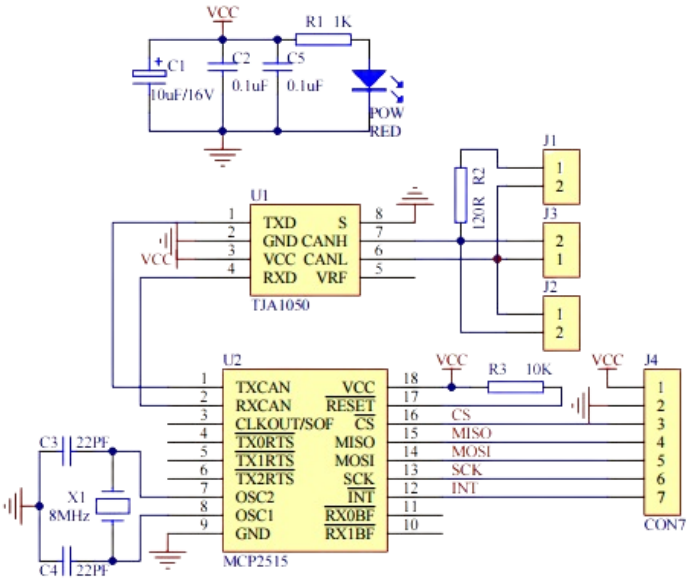
- Må lages/implementere programvare for å sende bildedata til kontrollstasjon
- Mer kraftkrevende enn svitsj og IP-kamera
- Mulig lengre tidsforsinkelse for bildestrøm

4.4.4 SPI til CAN

Selv om det er bestemt at kommunikasjonen ombord i ROV-en skal gå over CAN-buss, er det ikke gitt at kommunikasjonskontrolleren må ha en egen perifermodul med CAN-buss for å kommunisere med de andre kretskortene. Ved hjelp av en perifermodul med SPI-grensesnitt, kan en CAN-kontroller med SPI-grensesnitt styres. Det kan leses mer om CAN-kontrolleren i delkapittel 4.4.4.1 nedenfor. Dette er altså en løsning som kan implementeres hvis kommunikasjonskontrolleren ikke har en egen perifermodul for CAN-buss, men for SPI.

I tillegg til en CAN-kontroller, er det også nødvendig med en CAN-driver for å kunne kommunisere over CAN-bussen. En CAN-driver er nødvendig i alle tilfeller, selv om kontrolleren har et innebygd CAN-grensesnitt. Det kan leses mer om CAN-driverene i delkapittel 4.4.4.1 nedenfor.

I figuren nedenfor er et eksempel på hvordan en MCP2515-CAN-kontroller og en TJA1040-CAN-driver kan kobles med tilhørende komponenter. Med dette oppsettet, kan en kontroller med SPI-grensesnitt kommunisere på CAN-bussen ved å koble seg til CS, MISO, MOSI, SCK og INT på MCP2515.



Figur 4.9: Skjema for oppkobling av MCP2515 CAN-modul. CAN-kontroller (MCP2515) og CAN-driver (TJA1050)[87]

.

CAN-kontrolleren har to hovedoppgaver, og det er å sende og motta data. Sending gjøres ved å sende bits serielt på bussen til CAN-driveren når bussen er ledig. Ved mottak lagrer CAN-kontrolleren hver bit som kommer serielt på bussen frem til den har mottatt en hel melding. Når en hel melding er motatt, settes et avbrudd for å varsle at en melding kan hentes av kontrolleren.

Ved sending konverterer CAN-driveren bitstrømmen som kommer fra CAN-kontrolleren til korrekt spenningsnivå for CAN-bussen. Og ved mottak konverterer den spenningsnivåene på CAN-bussen om til en bitstrøm som sendes til CAN-kontrolleren.

CAN-kontrolleren MCP2515 som er brukt i oppkoblingensskjemaet i figur 4.9, er designet for bruk av en ekstern keramisk- eller krystallresonator som oscillerer med en bestemt frekvens, og i dette tilfellet er det en 8MHz krystalloscillator. Det vil si at klokkefrekvensen MCP2515 i figuren er på 8MHz. En keramisk resonator kan brukes ved busshastigheter opp til 125kbps, mens for hastigheter over må det brukes en krystalloscillator[88]. I databladet er det også oppgitt at den maksimale node-til-node osillatorvariasjonen kan være maksimum 1,7%. Krystalloscillatoren på 8MHz er brukt for å oppnå bedre presisjon enn ved bruken av en intern klokke på kontrolleren[7].

Selv om en krets for SPI til CAN er lagd med CAN-kontrollere og CAN-drivere hvor det oppgis at det kan kjøres en bitrate på opptil 1 Mbps, er ikke dette alltid tilfellet. Den oppgitte bitraten er teoretisk, og den faktiske begrensningen settes i hovedsak av størrelsen på krystalloscillatoren som brukes inn på CAN-kontrolleren. En CAN-kontroller med en 16 MHz krystalloscillator kan i teorien oppnå en høyere bitrate enn en CAN-kontroller med 8 MHz krystalloscillator. Det kan leses mer om teorien bak dette i delkapittel 3.4.3.4 som omhandler bit-timing og tidssegment for CAN-bussen.

4.4.4.1 CAN-kontrollere og CAN-drivere

Det fins en rekke forskjellige CAN-kontrollere med forskjellige spesifikasjoner. Noen av de som har blitt vurdert i dette tilfellet er MCP2515 [89], MCP2517FD [90], MCP2510 [91] og HI-3111PSIF [92]. I tabell 4.2 nedenfor kommer de viktigste spesifikasjonene frem.

CAN-kontroller	Protokoll	Maks bitrate	Operasjonsspenning	SPI-makshastighet
MCP2515	CAN 2.0B	1 Mbps	2,7V - 5,5V	10 MHz
MCP2517FD	CAN-FD (CAN 2.0B)	8 Mbps	2,7V - 5,5V	20 MHz
MCP2510	CAN 2.0B	1 Mbps	3,0V - 5,5V	5 MHz
HI-3111PSIF	CAN 2.0B	1 Mbps	5V	20 MHz

Tabell 4.2: En liste over CAN-kontroller og deres nøkkelspesifikasjoner

Den eksterne CAN-kontrolleren MCP2517FD skiller seg ut fra de andre kontrollerene i tabellen ved at den kan håndtere CAN-FD. CAN-kontrollere som kan håndere CAN-FD-protokollen blir mer og mer populært da de er mer fleksible og støtter høyere bitrater. En ulempe med en slik kontroller er at det finnes begrenset med informasjon på nett fra andre som har brukt den fordi den er relativt ny i forhold til de andre kontrollerene på listen.

En CAN-kontroller som har blitt brukt hyppig til hobbyprosjekt, er derimot MCP2515. På nettet kan man finne mye informasjon om hvordan denne har blitt brukt i ulike applikasjoner for å kommunisere på CAN-buss. Denne har derimot kun mulighet for å kommunisere med CAN2.0B-protokollen, noe som begrenser maks bitrate til 1 MBps. Dette gjelder også for de to andre kontrollerene, MCP2510 og HI-3111PSIF.

Akkurat som for CAN-kontrollere, finnes det en rekke forskjellige CAN-drivere med forskjellige spesifikasjoner. Noen av de som har blitt vurdert i dette tilfellet er MCP2551 [93], TJA1044GT/3Z [94], SN65HVD230DR [95] og TJA1050 [96]. I tabell 4.3 nedenfor kommer de viktigste spesifikasjonene frem.

[7] En presis krystalloscillator vil ha mindre variasjon i oscillatorfrekvens ved temperaturendring enn en intern klokke

CAN-driver	CAN2.0 eller CAN-FD	Maks bitrate	Operasjonsspenning
TJA1044GT/3Z	CAN-FD / CAN2.0	5 Mbps	4,75V - 5,25V
MCP2551	CAN2.0	1 Mbps	4,5V - 5,5V
SN65HVD230DR	CAN2.0	1 Mbps	3,0V - 3,6V
TJA1050	CAN2.0	1 Mbps	4,75V - 5,25V

Tabell 4.3: En liste over CAN-drivere og deres nøkkelspesifikasjoner

I ROV-en som UiS Subsea bygget ifjor ble det brukt en TJA1044GT/3Z CAN-drivere. Denne driveren kan brukes på CAN-FD med en bitrate på 5 Mbps, men ble brukt til vanlig CAN hvor de hadde en bitrate på 750 kbps. Selv om operasjonsspenningen på denne er på 5V, er det mulig å gjøre det slik at spenningen på TX/RX ikke er høyere enn 3,3V som er det GPIO-en på mikrokontrolleren tåler. Dette gjøres ved å koble 3,3V-forsyningen inn på pinnen merka V_{IO} på driveren. På MCP2551 og TJA1050 derimot, er det vanskeligere å få til dette, da begge har en operasjonsspenning på 5V, men ingen V_{IO}.

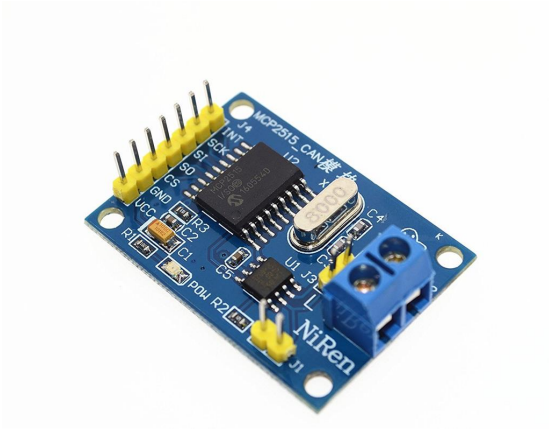
SN65HVD230DR CAN-driveren, hvor DR indikerer at den er overflatmontert, virker å være populær å bruke i hobbyprosjekt hvor det skal lages en CAN-buss. Dette kommer mest sannsynlig av at den har en operasjonsspenning på 3,3V noe som gjør den utmerket å kombinere med kommunikasjonskontrollere som maks tåler 3,3V inn på GPIO-en.

Før valgene av maskinvare kommer, vil det først i neste seksjon bli sett på en annen løsning for å realisere CAN-kommunikasjon. Dette er en løsning hvor man ikke trenger å velge en CAN-kontroller og -driver individuelt.

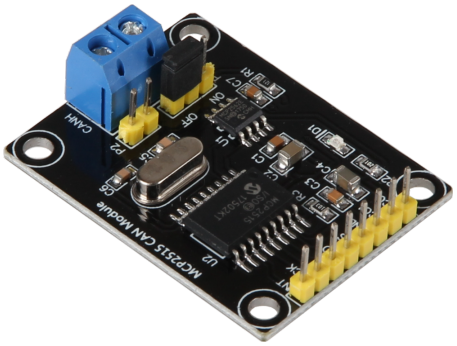
4.4.4.2 CAN-moduler

For å kunne implementere SPI til CAN trengs det altså en CAN-kontroller med SPI-grensesnitt, og en CAN-driver som kobles til kontrolleren. Rundt disse kommer det tilleggskomponenter som motstander, kondensatorer og en krystalloscillator av et slag (eksempel i figur 4.9). Det finnes ferdiglagde CAN-moduler hvor hele kretsen er implementert på en PCB-modul, og denne kan settes på et kretskort for å kunne kommunisere over CAN via SPI. Et par eksempler på slike moduler er vist nedenfor.

(a) CAN SPI Click 3.3V [97]



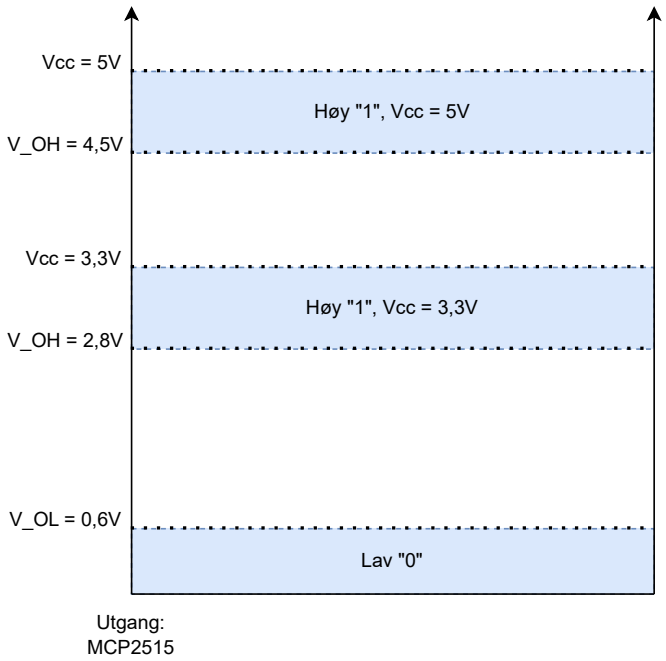
(b) MCP2515 CAN Bus Modul [98]



(c) CAN modul fra Joy-It [99]

Alle CAN-modulene ovenfor er basert på den samme CAN-kontrolleren som er MCP2515 og har en lask for terminering av en 120 Ohm motstand, men bruker ulike CAN-drivere. Størrelsen på krystalloscillatorene som er brukt er også forskjellig, noe som gir ulik klokkefrekvens på modulene.

CAN SPI Click i figur 4.10a er en CAN-modul laget av Mikroe som er basert på MCP2515 og en SN65HVD230[95] CAN-driver. CAN-modulen bruker en 10 Mhz krystalloscillator og har en teoretisk maks bitrate på 1 Mbps (CAN2.0B protokoll). Mikroe leverer både CAN-moduler med en operasjonsspenning på 3,3V og 5V da MCP2515 kan operere på begge spenningene. Denne modulen er laget for en operasjonsspenning på 3,3V. Dette er en fordel da flere av kommunikasjonskontrollerene som er vurdert ikke tåler mer enn $V_{IHmax} = 3,3V$ på GPIO-en[100]. I databladet til MCP2515 er det oppgitt at $V_{OHmin} = V_{DD} - 0,5V$ på slave ut (SO) noe som gir oss $V_{OHmin} = 4,5V$ ved 5V og $V_{OHmin} = 2,8V$ ved 3,3V. Dette er illustrert i figur 4.11 nedenfor.



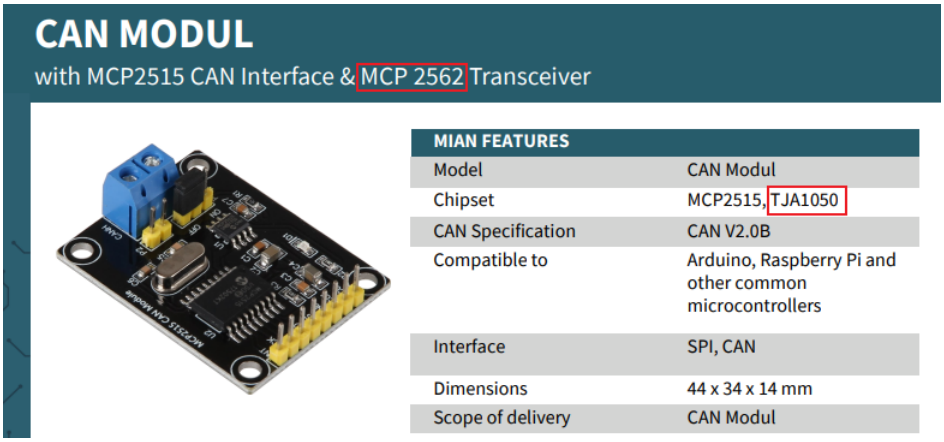
Figur 4.11: Spennigsnivå på utgang for MCP2515 med 3,3V og 5V forsyning

.

Figuren ovenfor illustrerer de ulike spenningsnivåene på utgangen til MCP2515 med ulik tilført spenning. En kommunikasjonskontroller som maksimalt tar 3,3V inn på GPIO-en, vil ikke være kompatibel med MCP2515 når den forsynes med 5V. Skal kravet om maksimalt 3,3V inn på GPIO-en overholdes, må MCP2515 forsynes med 3,3V.

MCP2515 CAN-buss modul i figur 4.10b er kanskje den mest vanlige CAN-modulen brukt i hobbyprosjekt. Denne baserer seg på en MCP2515 CAN-kontroller og en TJA1050 CAN-driver sammen med en 8 MHz krystalloscillator. Som for CAN SPI Click, har denne modulen også en teoretisk maks bitrate på 1 Mbps. På grunn av TJA1050, må denne modulen forsynes med 5V, noe som gir en minimumsspenningen inn på GPIO-en på 4,5V. For å gjøre denne modulen kompatibel med for eksempel en Raspberry Pi, er det mulig å erstatte TJA1050 med en CAN-driver som kan forsynes med 3,3V, eventuelt forsyne MCP2515 med 3,3V og TJA1050 med 5V hver for seg[100].

CAN-modulen fra Joy-It i figur 4.10c virker å vær et populært valg når Nvidia Jetson Nano (4.4.1.3) skal kommunisere over CAN-bussen via SPI. Denne baserer seg på en MCP2515 CAN-kontroller, en 16 MHz krystalloscillator og har en teoretisk maks bitrate på 1 Mbps. Databladet til denne CAN-modulen er noe uklart, så det er litt tvil om hvorvidt denne modulen har en TJA1050 eller en MCP2562 CAN-driver (se figur 4.12). Men i begge tilfellene må CAN-driveren forsynes med 5V, noe som gjør at det må gjøres de samme modifikasjonene på denne CAN-modulen som på MCP2515 CAN-buss modulen (4.10b) for at den skal vær kompatibel med kommunikasjonskontrollere med en 3,3V toleranse på GPIO-en.



Figur 4.12: Utklipp av databladet til CAN-modulen fra Joy-It [101]

.

Som nevnt i innledningen til dette delkapittelet (4.4.4), har CAN-bussen lav toleranse for oscilleringsvariasjoner før det oppstår feil. Skal det lages en krets for SPI til CAN må disse toleransene tas hensyn til ved valg av komponenter til kretsen. Den største fordelen ved å bruke en ferdig CAN-modul er nettopp det at den er ferdig testet og at det har blitt tatt hensyn til toleransene for CAN-buss. Det er derfor enklere og mindre bekymrende å implementere en CAN-modul på kretskortet. En annen fordel er det faktum at det er tidsbesparende å bruke en CAN-modul i plassen for å måtte bruke mye tid på å designe en egen CAN-krets.

En av ulempene ved å gå for en ferdig CAN-modul er at det blir mindre fleksibilitet ved valg av komponenter. Designes kretsen fra bunnen av, kan man velge de komponentene man selv ønsker i forhold til ønsket spesifikasjoner på CAN-bussen. En annen ulempe er det at den ferdige CAN-modulen mest sannsynlig må modifiseres litt for å passe inn på et kretskort der hvor målet er å unngå bruk av ledninger. Tilkoblingspunktet for CAN-L og CAN-H må fjernes, og så må det settes på en header som passer ned i kretskortet.

4.5 Valg av løsning og komponenter

I denne seksjonen blir det først utredet hvorfor det ikke ble gått for en løsning med svitsj og IP-kamera. Etter dette blir det bestemt valg av hvilken ettkortsdatamaskin som skal fungere som en bildeprosesserende kommunikasjonskontroller. Maskinvaren for realisering av CAN-buss og tilting av kamera vil også bli bestemt her.

4.5.1 Kommunikasjonskontroller med svitsj og IP-kamera

Dette var en løsning som ble vurdert på et tidlig stadium av prosjektet. Ved å bruke en svitsj for å håndtere bildedataen fra kamera, kunne det implementeres en mindre kommunikasjonskontroller som Nucleo-32 STM32G431KB. Denne mikrokontrolleren og Nucleo-32 STM32G431RB blir brukt på alle de andre kretskortene i ROV-en, noe som ville gjort implementering av CAN-buss enkelt da disse mikrokontrollerene har en innebygget CAN-modu, som nevnt tidligere i tabell 4.1.

Det ble tidlig testet om en Raspberry Pi 4 Model B var i stand til å håndtere flere bildestrømmer med tilfredsstillende forsinkelse og bilder per sekund, noe den ikke var. Bruken av en Raspberry Pi ville dermed medføre et behov for en svitsj og bruk av IP-kamera. Da den heller ikke har en innebygget CAN-modul, ville dette alternativet bli en kombinasjon med en ekstern CAN-modul og svitsj. I tillegg er det nært sagt umulig å få tak i en Raspberry Pi da det er svært stor etterspørsel for tiden. Dette gjorde bruken av en Raspberry Pi uaktuell.

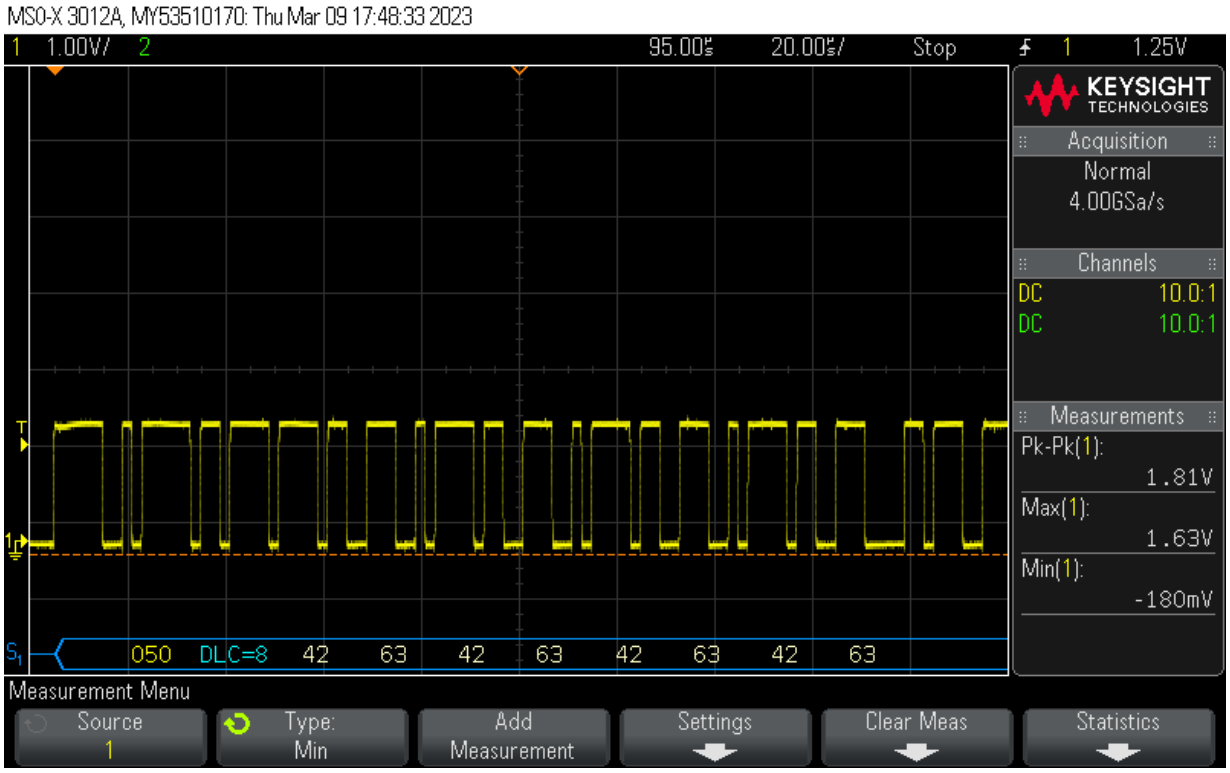
Det å finne en svitsj som var liten og i tillegg hadde ønsket spesifikasjoner til en rimelig pris var vanskelig. Brukbare IP-kamera til en rimelig pris var også vanskelig å finne. Alt i alt viste det seg at kostnaden av denne løsningen ville være større enn fordelene. Dermed ble det bestemt å ikke gå for denne løsningen.

4.5.2 Bildeprosesserende kommunikasjonskontroller

4.5.2.1 Ettkortsdatamaskin

De resterende alternativene til kommunikasjonskontroller etter at Nucleo-32 STM32G431RB og Raspberry Pi 4 Model B falt bort, var Nvidia Jetson Nano og OKdo ROCK 4 Model C+. Disse må kombineres med en ekstern CAN-modul for å være i stand til å kommunisere over CAN-bussen. Dette fordi de ikke har en egen perifermodul for CAN-buss, som nevnt i tabell 4.1.

Bruken av en ekstern CAN-modul ble testet tidlig i prosjektet. Det var tilgang på en Raspberry Pi 4 Model B og en CAN-modul av typen CAN SPI Click 3,3V4.10a. Denne kombinasjonen ble testet opp mot en mikrokontroller av typen Nucleo-64 STM32F103 kombinert med SN65HVD230 CAN-driver. Kommunikasjonen mellom disse nodene fungerte feilfritt. Senere ble Raspberry Pi byttet ut med en Nvidia Jetson Nano 2GB som er en eldre versjon av den som er vurdert. Selv om dette oppsettet var vanskeligere å sette opp, fungerte også dette feilfritt til slutt på testbenken. En melding som er sendt på dette testoppsettet er vist nedenfor i figur 4.13.



Figur 4.13: Melding på CAN-bussen til testoppsettet tatt opp på scope. Målt differensialspenning mellom CAN-H og CAN-L.

Nvidia Jetson Nano 2GB ble også testet på hvordan den håndterte bildedata fra et stereokamera og et vanlig USB-kamera samtidig. Denne testen gikk bra, og viste at Nvidia Jetson Nano 4GB ville være i stand til å håndtere bildedata og kommunikasjon.

Etter testene som ble utført med Nvidia Jetson Nano 2GB, ble det klart at Nvidia Jetson Nano 4GB ville klare løse oppgavene og være et godt alternativ som kommunikasjonskontroller. I tillegg til testene, var det mer informasjon på nett om hvordan Nvidia Jetson kunne brukes, enn for OKdo ROCK. Dermed ble ikke OKdo ROCK 4C+ Model B vurdert noe videre. Valget falt dermed på Nvidia Jetson Nano 4GB.

4.5.2.2 CAN-buss

Det var mye informasjon om hvordan realisere CAN-buss ved hjelp av Nvidia Jetson Nano kombinert med CAN-modulen Joy-It (4.10c) på nett. Men valget falt fortsatt på CAN-modulen SPI CAN Click 3,3V (4.10a) fordi denne allerede var testet. Dette på bekostning av mindre informasjon på nett og en mindre krystall på 10 MHz kontra en på 16 MHz.

Etter at valget ble tatt om å bruke CAN-modulen SPI CAN Click 3,3V, var det nødvendig å finne en passende CAN-driver til de andre kretskortene i ROV-en. CAN-driveren som er brukt på CAN-modulen er en SN65HVD230 som kan drives på 3,3V. Mikrokontrollerene brukt i prosjektet tåler ikke mer enn 3,3V på GPIO-en, og derfor ble det bestemt at CAN-driveren SN65HVD230DR[95] skulle brukes. Det å ha en overflatemontert CAN-driver var også et ønsket, og SN65HVD230DR er det (gitt av DR"). CAN-driveren som ble valgt var av samme type som CAN-modulen, og gir fordelere med å ha deler i reserve.

4.5.3 Servomotor for tilt av kamera

Kamera som skal festes er av relativ lite vekt. Det er derfor enkelt nok å få tak i en servo som er sterk nok til å justere vinkelen på denne. Servoen trenger derfor ikke å være så kraftig, men det ønskes at servoen er lett og rimelig. Styresignalet som servoen mottar vil være **PWM**, men dette vil ikke være sterkt nok til å fungere som et drivesignal. Servoen må derfor ta inn kraften sin fra en annen kilde.

For dette prosjektet er det mulig å bruke en stadard servo som brukes til hobbyprosjekt som er rimelige og lette. Servoene som er sett på er vist på figur 4.14a, 4.14b, 4.14c og 4.14d.

(a) MG90S [102]

(b) MG99R [103]

(c) HS65-MG [104]

(d) SG90 [105]

Alle servoene som er vist, har relativt like funksjoner. Alle servoene:

- Kan bruke 5V som V_{CC}
- Veier under 20g
- Har over 1 kg/cm
- Har vinkelhastighet over 0,1 sekund per 60 grader

Prisene på servoene er også relativt lik, samenligning av dett vist på tabell

Vare	Pris per stk
MG90S	69,00 kr
MG99R	101,86 kr
HS65-MG	392,00 kr
SG90	45,00 kr

Tabell 4.4: Samenligning av servoer

`HS65-MG` er mye dyrere enn de andre, da denne har *Karbonite gears*[8]. Siden det er lite slitasje og kraft som servoen må tåle, gikk vi for det billigeste alternativet med **SG90**. Denne har nylongir, som er tilstrekkelig for bruket. Den kommer med en oppløsning på 10 bits, som tilsvarer 1024 ulike posisjoner mellom 0 og 180 grader. Den har driftsspenning mellom 4,8V til 6V.

Denne har tre tilkoblinger:

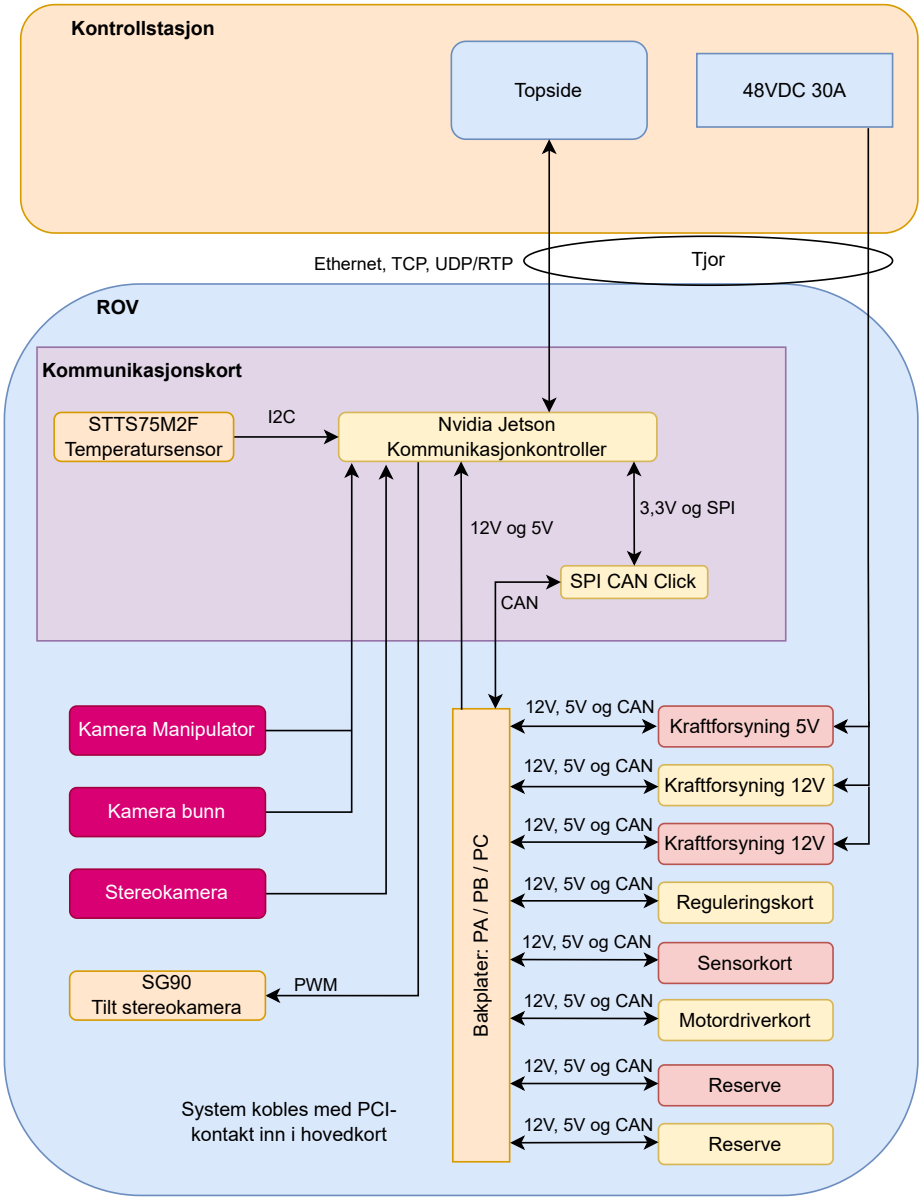
- V_{CC}
- GND
- PWM

Denne vil trekke strømmen sin mellom V_{CC} og GND, ikke mellom PWM signalet og GND. Denne må derfor kobles opp til felles 5V fra kraftforsyningen.

[8] Gir laget av Hitec som viser nesten fem ganger styrken til nylon [106].

4.6 Resultat og oppsummering

For å realisere behovs- og funksjonsspesifikasjonene gitt i begynnelsen av kapittelet, ble det i forrige seksjon valgt maskinvare som kunne gjøre dette. Som kommunikasjonskontroller ble det valgt en ettkorts datamaskin av typen Nvidia Jetson Nano 4GB, gjennomgått i seksjon 4.4.1.3. Kommunikasjonskontrolleren kombineres med en CAN-modul av typen SPI CAN Click 3,3V (4.10a) for å realisere CAN-kommunikasjon. Tilting av stereokamera realiseres ved hjelp av servomotoren, SG90 (4.14d). Disse valgene er gjenspeilet i blokkskjema 4.15 nedenfor.



Figur 4.15: Blokkskjema for å illustrere utgangspunkt før valg av maskinvare

I figuren ovenfor kommer også temperatursensoren STTS75M2F som er gitt av sensorgruppen. Denne skal kommunisere med kommunikasjonskontrolleren over I^2C. CAN-driveren som skal brukes på de andre kretskortene ombord i ROV-en er den samme som på CAN-modulen, altså av typen SN65HVD230DR.

Kapittel 5

Video og kamera

Kapitteloversikt

Dette kapittelet omhandler kamera som skal plasseres på ROV-en og bildestrømmen som skal sendes til kontrollstasjonen. Det vil gås gjennom begrunnelser for valgene som er blitt tatt med litt relevant bakgrunnsteori. Til slutt oppsummerest resultatet. Det er tatt inspirasjon fra en tidligere UiS-Subsea bachelor fra 2018 [107].

5.1 Problemstilling

I kapittelet som omhandler maskinvare 4, ble det bestemt at det skulle brukes en Nvidia Jetson Nano 4GB som kommunikasjonskontroller. Denne skal operere som en kommunikasjonsbro mellom ROV og kontrollstasjon, samt ta inn bildedata fra kamera på ROV og sende opp til kontrollstasjon. Valget av Nvidia Jetson Nano setter noen begrensninger til hvilke type kamera som kan brukes. I dette kapittelet vil det først bli gjennomgått litt teori angående ulike typer kamera, linser og videokomprimering. Til slutt vil valgene av kamera og begrunnelser for dette komme frem.

5.2 Behovsspesifikasjon

- Skal ha et kamera i front av elektronikkhus
- Skal ha et kamera under ROV
- Skal ha et kamera for oversikt over manipulator

5.3 Funksjonsspesifikasjon

- Kamera i front skal være et stereokamera
- Stereokamera skal minimum ha 1920x1080p oppløsning med 30 fps
- Kamera i bunn og med manipulator skal minimum ha 1280x720p oppløsning med 30 fps
- Stereokameraet i front bør kunne se et objekt som er 24cm x 13,5cm (27,5cm diagonalt) når objektet er 20cm fra kamera
- Kamera i bunn og med manipulator bør kunne se et objekt som er 48cm x 27cm (55,1cm diagonalt) når objektet er 10cm fra kamera
- Stereokamera må passe i en kuppel (halvkule) med diameter på 160mm

5.4 Litt generelt om kamera

5.4.1 Analoge videokamera

Analoge videokamera baserer seg på å omforme lys til analoge spenninger som brukes for overføring av bildestrømmen. Fordelen med analoge videokamera er den lave forsinkelsen på overføringen av bildestrømmer. Ofte er denne kun på et par millisekund. En av ulempene med et analogt videokamera er bildekvaliteten som generelt er dårligere enn bildekvaliteten til et digitalt kamera. Det analoge signalet er også sårbart for støy, som kan føre til at bildekvaliteten reduseres ytterlige ved at bildet blir utydelig.

5.4.2 USB-kamera

Et USB-kamera er et webkamera som er laget for å kobles til en datamaskin eller andre enheter via en USB-port. USB-kamera brukes i mange ulike sammenhenger. Det blir ofte brukt som webkamera og dokumentkamera, men det brukes også industrielt i produksjon og automasjonssammenheng for å nevne noen.

USB-kamera er relativt billige og enkle å bruke. USB-standarden gjør at man ikke er avhengig av noen spesielle program eller drivere for å få inn bildestrømmen på en datamaskin.

Siden USB-standarden har så rask overføringshastighet, kan bildedataen sendes uten for stor forsinkelse. Avhengig av hvilket kamera som velges er overføringshastigheten til USB:

Standard	Overføringshastighet	Lengde
USB 2.0	480 Mbit/s	5 meter
USB 3.0	5 Gbit/s	3 meter
USB 3.1	10 Gbit/s	3 meter
USB 3.2	20 Gbit/s	3 meter

Tabell 5.1: Samenligning av USB-standarder [108].

Etter funksjonsbeskrivelsen blir det nevnt at stereokameraet skal ta inn bildedata med 1920x1080p og 30 bilder i sekundet. Dersom det brukes RGB med 24 bit for fargealternativer[109], får vi $\boxed{1920 \cdot 1080 \cdot 30 \cdot 24 \approx 1493 \text{ Mbit/s}}$. Dersom man samenligner med enklere kamera som har mindre oppløsning, får man $\boxed{1280 \cdot 720 \cdot 30 \cdot 24 \approx 664 \text{ Mbit/s}}$. Selv om dette er mer enn bandbredden USB2.0-standarden, er det vanlig å komprimere datapakken før den sendes, noe som gjør at datapakken blir mindre. Mer om dette i delkapittel 5.6 som omhandler komprimering av video.

5.4.3 IP-kamera

Et IP-kamera (***Internet Protocol Camera***) er et digitalt kamera som sender bildestrømmen via en IP-protokoll som for eksempel TCP(3.4.5.3), og kan dermed sendes på en Ethernetkabel over internett eller et lokalt nettverk. IP-kamera blir ofte brukt til overvåking[110]. IP-kamera kan gi bildekvalitet med høy oppløsning, ofte "HD"[1] eller "full-HD"[2], og i noen tilfeller så høyt som "ultra-HD"[3]. Med så

[1] HD gir en oppløsning på 1280x720 piksler
[2] Full-HD gir en oppløsning på 1920x1080 piksler
[3] Ultra-HD gir en oppløsning på 3840x2160 piksler

høye oppløsninger blir det lettere å gjennkjenne og skille ulike gjenstander fra hverandre. Et IP-kamera kan drives ved hjelp av "PoE"[4], noe som gjør at det ikke trenger en dedikert strømkilde.

Høye oppløsninger krever også en høyere båndbredde på overføringen. Båndbredden som kreves påvirkes av flere ulike faktorer. For eksempel oppløsning, bilder i sekundet, komprimering og antall kamera. Som referanse, vil et IP-kamera med "full-HD"-oppløsning, 30 bilder i sekundet og H.264[5]-enkoding kreve en båndbredde på 4-5Mbps[112].

5.4.4 Stereokamera

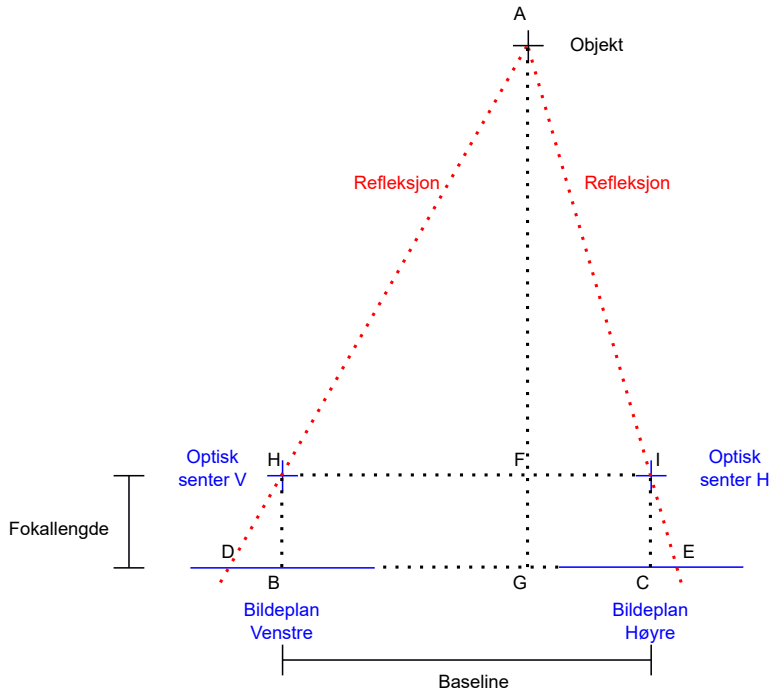
Et stereokamera består på sett og vis av to individuelle kamera plassert med litt avstand mellom hverandre. Dette gjør at stereokameraet kan vise tredimensjonale bilder av objekter slik et menneske ville sett det. Med andre ord, har et stereokamera dybdesyn. For å oppnå dette, bør avstanden mellom linsene være omtrent lik avstanden mellom øynene på et menneske, som er rundt 6,35cm[113]. Denne avstanden blir ofte kalt "baseline length". Stereokamera kan fås med overføring gjennom blant annet USB, IP og CSI[6].

Stereokamera er helt nødvendig i noen autonome systemer som for eksempel en robot som baserer seg på informasjon fra et kamera. Ved hjelp av stereosyn, kan roboten finne den relative posisjonen og avstanden til et objekt, og skille to objekter fra hverandre. Dette blir grundigere forklart i neste delkapittel 5.4.4.1.

5.4.4.1 Stereosyn til avstandsmåling

Med stereosyn kan avstanden til et objekt beregnes ved å sammenligne hvor objektet befinner seg på de to bildene fra hvert kamera[114]. Teorien bak dette bruker trigonometri for å finne avstanden til et objekt. Et eksempel på dette er gitt i figur 5.1 nedenfor der avstanden **AF** skal finnes. Linsen på hvert kamera blir kalt *optisk senter*. Avstanden mellom linsen og *bildeplanet* er kjent og kalles *fokallengden*. Her blir det lengden **HB**. *Baseline* mellom kameraene er også kjent og vises som lengden **BC**.

[4]PoE (Power over Ethernet) supplerer kraft gjennom en Ethernetkabel
[5]Utbredt standard for videokompremering[111]
[6]CSI (Camera Serial Interface) er et grensesnitt mellom kamera og prosessor



Figur 5.1: Illustrasjon for stereosyn

Ved å bruke formlikhet og trigonometri, er det mulig å bruke de mindre trekantene mellom *optisk senter* og *bildeplanet* (**HBD og ICE**), for å finne ut størrelsen på trekantene mellom *objektet* og *optisk senter*(**AFH og AFI**). Ved å telle antall piksler mellom hvor refleksjonen fra objektet treffer bildeplanet, og senter i bildeplanet - kan avstandene **BD og CE** kalkuleres. Når man har målene for dette, kan avstanden **AF** utledes slik:

$$\frac{AF}{AF + FG} = \frac{BC}{BC + CE + BD}$$

$$AF(BC + CE + BD) = AF \cdot BC + BC \cdot FG$$

$$AF(CE + BD) = BC \cdot FG$$

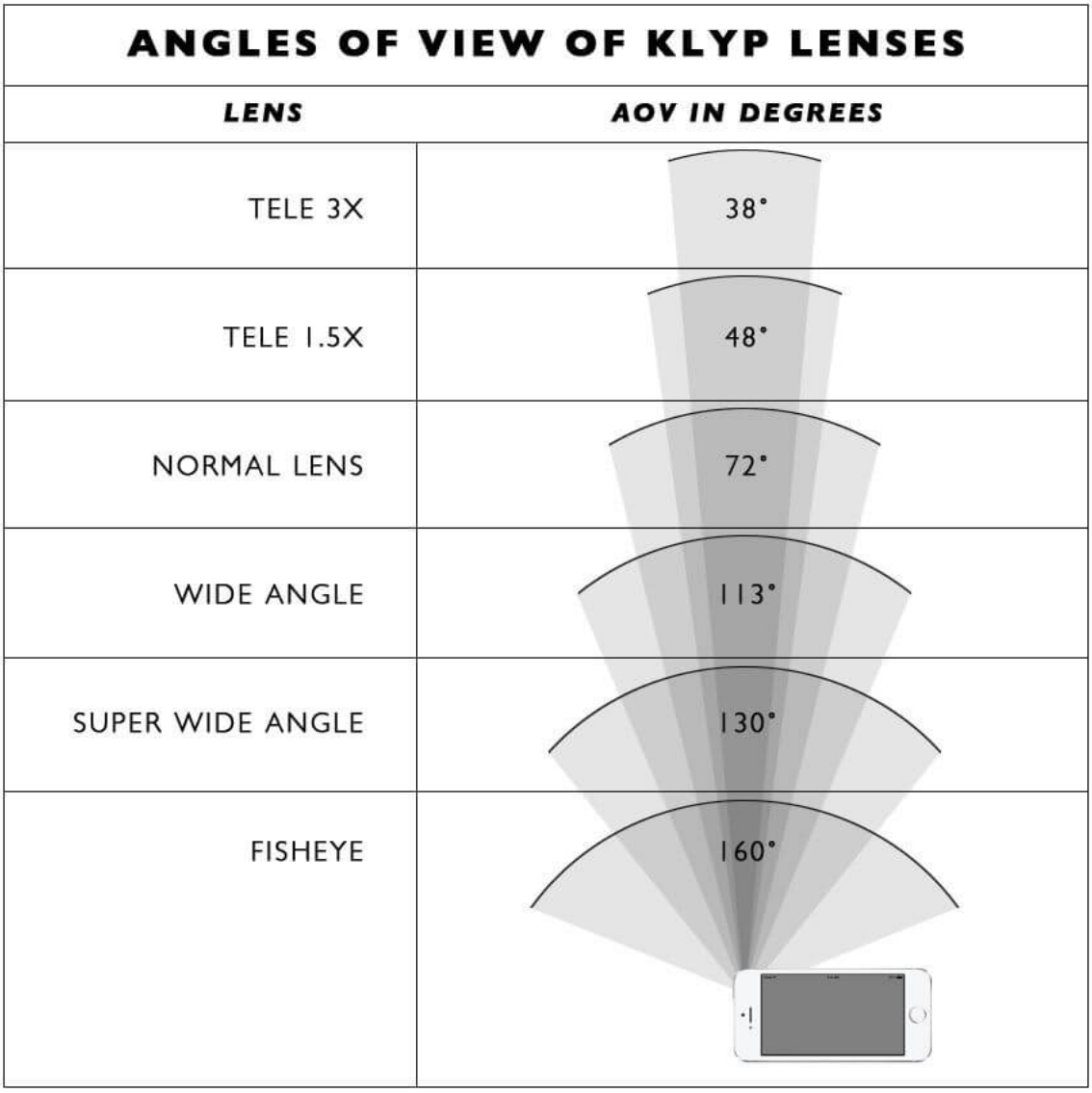
$$AF = \frac{BC \cdot FG}{CE + BD} \tag{5.1}$$

Fra utledet formel kan det merkes at ***avstanden*** mellom kamera og et objekt er gitt ved **AF**, ***baseline*** er gitt ved **BC** og ***fokallengden*** er gitt ved **FG**. Avstandene X_v og X_h som er hvor mye refleksjonen er forskjøvet fra optisk senter, er gitt ved **BD** og **CE**. Dette gir da følgende teoretiske formel for avstandsberegning med stereosyn:

$$Avstand = \frac{base\ line \cdot fokallengde}{x_h + x_v} \tag{5.2}$$

5.5 Teori om linser og synsvinkel

Linsen er en kritisk del i alle kamera. Det er linsen som fokuserer lyset inn på kamerasensoren/bildeplanet, som igjen lar oss fange et bildet. Valget av linse har stor betydning for hvordan bildet blir seende ut. Linsen påvirker blant annet synsvinkelen (***FOV - Field Of View***), og dybdeskarpheten (***DOF - Depth Of Field***). Synsvinkelen påvirkes av fokallengden, også kalt brennvidden, som er avstanden mellom der lyset går inn i linsen til der det treffer kamerasensoren. En kort fokallengde gir større synsvinkel, og mindre synsvinkel med en lang fokallengde. Dybdesynet påvirkes av fokallengden og hvor mye lys som slipper inn i linsen, altså bredden på kameraåpningen. Figur 5.2 nedenfor illustrerer hvordan ulike linser påvirker synsvinkelen.



Figur 5.2: Oversikt over forskjellige linser og deres synsvinkel.[115]

Det finnes mange forskjellige typer linser. En linse med smal synsvinkel brukes ofte for å se et objekt på avstand klart, mens en linse med bred synsvinkel brukes ofte for å klare se et objekt selv på nært hold. Ved bruk av en bred synsvinkel kan periferen av bildet virke uskarpt eller forvrengt. Dette kan sammenlignes med menneskesynet hvor objekt langt ute i synsfeltet virker uklare. En kikkert baserer

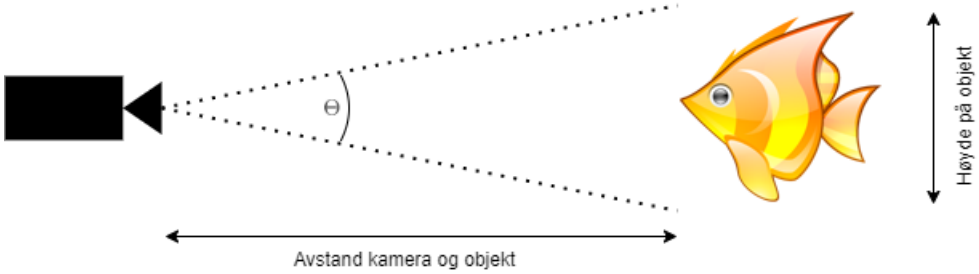
seg på en smal synsvinkel og snevrer inn synsfeltet for å fokusere på et objekt på avstand. I figur 5.3a er det brukt en zoom-linse hvor objektene i fokus er klare og skarpe, mens periferen er uklar. I figur 5.3b er det brukt en såkalt "fish eye"-linse med en bred synsvinkel hvor man kan se at bildet er forvrengt.

(a) Bilde tatt med 300mm fokallengde zoom-linse. [116]

(b) Bilde tatt med 4mm fokallengde "fish eye"-linse. [117]

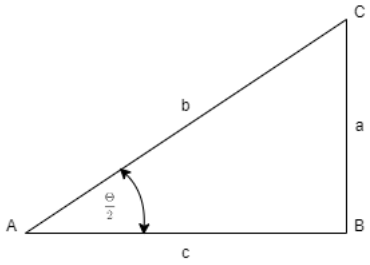
5.5.1 Formler for beregning av synsvinkel

Det er viktig å vite hvor stor synsvinkelen er for å forstå hvor nært et objekt kan befinne seg ROV-en før objektet enten blir uklart, eller at hele objektet ikke kan sees. Forholdet mellom avstanden fra kamera til objekt, synsvinkelen og størrelsen på objektet er vist i figur 5.4 nedenfor.



Figur 5.4: Illustrasjon for forholdet mellom avstand og synsvikel.

For å finne synsvinkelen i figuren ovenfor basert på avstand til objekt og høyden på objektet, kan synsvinkelen først deles inn i to rettvinklede trekanter. En av disse to rettvinklede trekantene er illustrert i figur 5.5 nedenfor.



Figur 5.5: Illustrasjon av halve synsvinkelen som en rettvinklet trekant

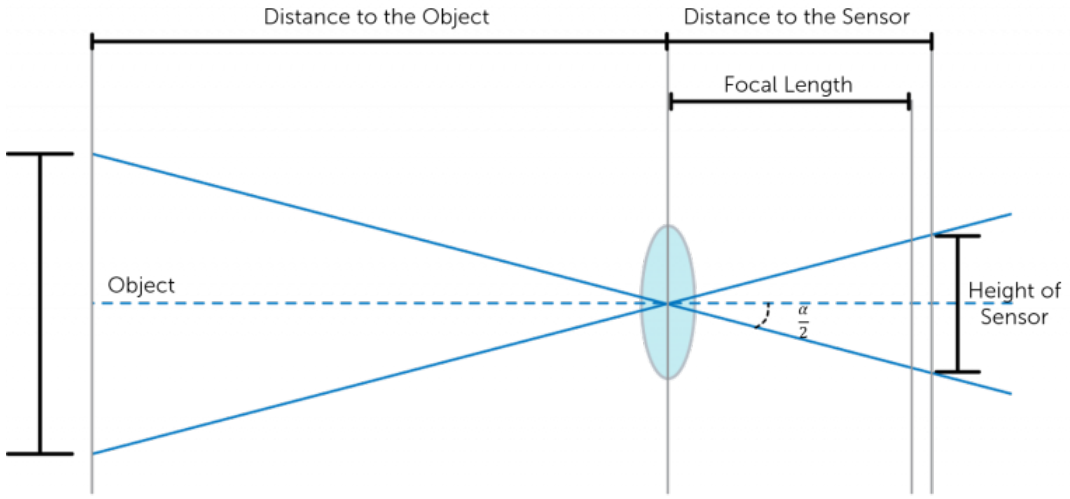
Vinkelen $\frac{\theta}{2}$ i figuren ovenfor kan finnes ved hjelp av trigonometri og linjestykkene a og c. Følgende formel for hele synsvinkelen kan utledes som:

$$tan(\frac{\theta}{2}) = \frac{a}{c}$$
$$synsvinkel = \theta = 2arctan(\frac{a}{c}) \tag{5.3}$$

Hvor:

- $\boldsymbol{a}$ er halve høyden,bredden eller diagonalen til objektet
- $\boldsymbol{c}$ er avstanden mellom kamera og objektet
- θ er hele synsvinkelen

Formelen ovenfor kan brukes for å beregne den minimale synsvinkelen som skal til for å se hele objektet ved en ønsket avstand til objektet. Det må derfor sjekkes at valgt linse på kameraet oppfyller ønsket synsvinkel. I figur 5.6 nedenfor vises det hvordan avstanden mellom linsen og kamerasensoren, fokallengden, påvirker synsvinkelen[118]. I tillegg til denne avstanden, så vil også størrelsen på kamerasensoren påvirke synsvinkelen.



Figur 5.6: Illustrasjon av hvordan avstanden mellom kamerasensor og linse (fokallengden) påvirker synsvinkelen. [119]

Fra figuren ovenfor, kan formelen for synsvinkelen til kameraet utledes. Formelen baserer seg på dimensjonene til kamerasensoren og fokallengden, og kan utledes slik:

$$tan(\frac{\theta}{2}) = \frac{d}{2f}$$
$$\theta = 2arctan(\frac{d}{2f}) \tag{5.4}$$

Hvor:

- θ er den totale synsvinkelen
- d er dimensjonen til kamerasensoren. Enten høyden, bredden eller diagonalen.
- f er fokallengden

Ligningene 5.3 og 5.5 som er utledet gir forholdet $\boxed{2arctan(\frac{a}{c}) = 2arctan(\frac{d}{2f})}$ som viser en direkte relasjon mellom størrelsen og avstanden til et objekt, og størrelsen på kamerasensor og fokallengden til kameraet. $\boldsymbol{a}$ er gitt som enten halve bredden, høyden eller diagonalen til et objekt, som vil si at $\boldsymbol{2a}$ vil være hele bredden, høyden eller diagonalen på objektet. I ligningen nedenfor utledes et utrykk for $\boldsymbol{2a}$ (hbd_{objekt}) og et utrykk for fokallengden.

$$\frac{a}{c} = \frac{d}{2f}$$
$$hbd_{objekt} = 2a = \frac{dc}{f} \tag{5.5}$$
$$f = \frac{dc}{hbd_{objekt}} \tag{5.6}$$

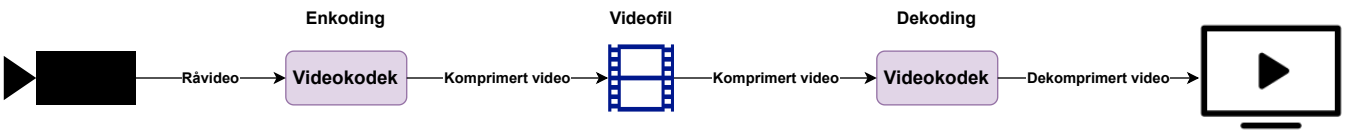
Hvor:

- c er den avstanden mellom kamera og objekt
- d er dimensjonen til kamerasensoren. Enten høyden, bredden eller diagonalen
- hbd_{objekt} høyden, bredden eller diagonalen på synsfeltet
- f er fokallengden

Med formlene gitt i dette delkapittelet, kan synsvinkelen, synsfeltet og nødvendig fokallengde for en linse beregnes. Formlene brukes delkapittel 5.7.1.1 og 5.7.2.3 for å beregne og verifisere synsfeltet på linsene til de valgte kameraene basert på funksjonsspesifikasjonene.

5.6 Komprimering

Komprimering brukes til å representere informasjon ved hjelp av færre bits enn utgangspunktet[120], og gjøres for redusere nødvendig minneområdet ved lagring eller overføringshastighet ved overføring av informasjonen. Noen vanlige komprimeringsstandarder er blant annet JPEG for bildekomprimering og MP3 for lydkomprimering. I figur 5.7 nedenfor er flyten for komprimering og dekomprimering av en video vist.



Figur 5.7: Flytskjema for enkoding og dekoding av video uten lyd.

Ved overføring av bildestrømmer (video), kan bildene komprimeres før sending (videokomprimering), og dekomprimeres ved mottak av bildene for å gjennopprette bildekvaliteten. En komprimeringsstandard som både komprimerer og dekomprimerer blir kalt en kodek, og komprimeringen blir ofte kalt enkoding, mens dekomprimering blir ofte kalt dekoding. Et eksempel på en kodek er H.264 (*AVC - Advanced Video Coding*) som brukes til videokomprimering. H.264 er en av de mest brukte standardene for videokomprimering i dag, og er hyppig brukt innen streaming av video. Nvidia Jetson Nano som er den valgte kommunikasjonskontrolleren, har støtte for H.264-videokomprimering. I dette prosjektet er det blitt bestemt at videoen skal komprimeres ved hjelp av H.264, og i de neste delkapittelene vil videokomprimering og H.264-standarden bli gjennomgått.

5.6.1 Motivasjon for videokomprimering

Som nevnt i innledningen (5.6), brukes komprimering for å redusere størrelsen på digital data ved å fjerne redundant informasjon, noe som gjør det mulig å lagre og sende dataen mer effektivt. Ved sending av videostrømmer, er det svært nyttig å komprimere videoen. Dette fordi ukomprimert video krever en veldig høy båndbredde. Et eksempel for å illustrere dette er gitt i neste avsnitt, og er selve motivasjonen for bruken av videokomprimering.

En videostrøm i Full-HD, som nevnt i delkapittel 5.4.3 har en oppløsning på 1920x1080 piksler, hvor hver piksel er representert av 24 bits[121] i RGB-modellen[7]. Videostrømmen består ofte av 60 bilder i sekundet (fps). Med denne informasjonen kan nødvending overføringshastighet for å overføre den ukomprimerte videostrømmen beregnes som:

$$\begin{aligned} Bitrate &= Opploesning \cdot fps \cdot 24bits \\ Bitrate &= 1920 \cdot 1080 \cdot 60fps \cdot 24bits \\ Bitrate &= 2985984000bits/s \approx 3Gbps \end{aligned} \tag{5.7}$$

Fra beregningen ovenfor, kommer det frem at en ukomprimert videostrøm i Full-HD krever en overføringshastighet på omtrent 3Gbps for å kunne overføres, mens en lavere overføringshastighet vil føre til

[7]RGB-modellen er en fargemodell som er bygget opp av primærfagene i lys som er rød, grønn, blå. Hver farge er representert av 8 bits, noe som gir $2^8 = 256$ ulike toner av hver farge.[121]

at det oppleves hakking ved avspilling av videostrømmen. Som nevnt i delkapittel 3.4.5.2, har kategorikablene, Cat-5e og Cat-6, som er de vanligste kategorikablene i dagens husinstallasjoner, en maksimal overføringshastighet på 1Gbps. Dette gjør det vanskelig å spille av ukomprimerte videostrømmer i Full-HD.

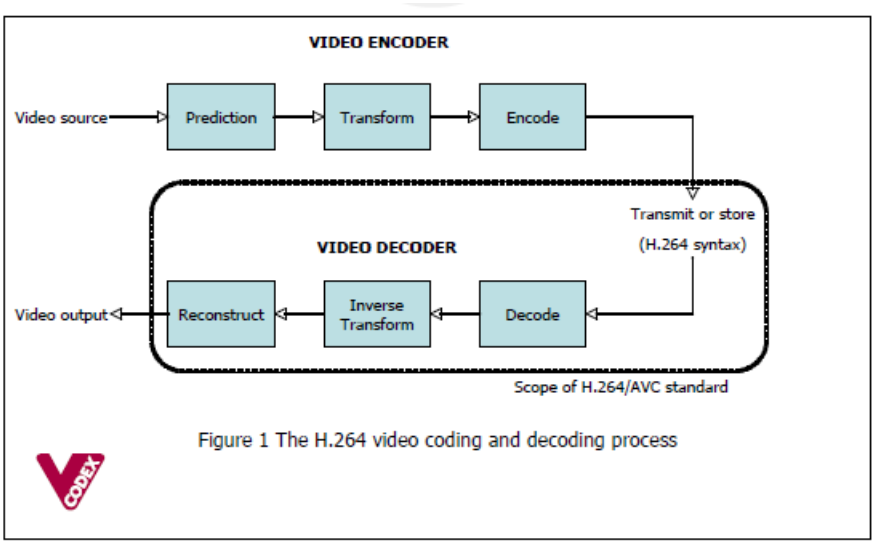
For å redusere nødvending overføringshastighet, brukes videokomprimering, ofte i form av H.264-standarden. Dette er en videokodek som på det meste har en komprimeringsrate på 2000:1[122], noe som vil si at en 3Gbps videostrøm kan komprimeres ned til en videostrøm på 1,5Mbps. En så kraftig komprimering kommer selvfølgelig med noen ulemper. Fordelene og ulempene ved bruk av H.264-standarden, samt en overordnet forklaring av standarden i seg selv vil bli gjennomgått i det neste delkapittelet.

5.6.2 Kort om H.264-videokodeken

Dette delkapittelet er hovedsaklig basert på wikipediasidene for H.264 [123] og datakompresjon [120].

H.264-kodeken er en av de mest brukte og populære videokodekene per dags dato. I september 2019 ble det anslått at omtrent 91% av videoindustrien brukte H.264 for videokomprimering[123]. Standarden ble utviklet for å kunne levere like bra bildekvalitet som tidligere standarder, men ved hjelp av en mye lavere bitrate. Den er mye brukt til blant annet opptak av høyoppløsnings video, streaming av video på nett og kringkasting av TV-programmer. Standarden støtter komprimering av videoer med oppløsning opp til og med 8K (*Ultra High Definition*).

Videokomprimering ved hjelp av H.264-kodeken bruker flere teknikker og metoder som kan redusere filstørrelsen drastisk, samtidig som tapet i bildekvalitet er lavt. I figur 5.8 nedenfor, er flyten ved enkoding og dekoding vist. Videre vil de ulike boksene i denne figuren bli forklart.

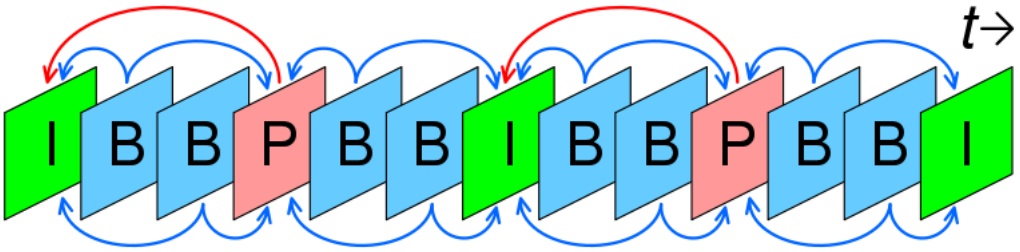


Figur 5.8: Enkoding- og dekodingsprosessen for H.264-kodeken illustrert.[124]

Flyten over prosessen i figuren ovenfor kan deles inn i to separate prosesser: enkodingsprosessen og dekodingsprosessen. enkodingsprosessen består av prediksjon, transformasjon og enkoding, mens dekodingsprosessen består av dekoding, invers-transformasjon og rekonstruksjon. Disse prosessene blir gjennomgått i de neste delkapittelene, noen grundigere en andre, da en grundig gjennomgang av hele H.264-kodeken ville vært verdig en bachelor i seg selv.

5.6.2.1 Enkodingsprosessen

Som vist i figur 5.8 i forrige delkapittel, er predikering det første enkoderen gjør. Det vil si, enkoderen bruker prediktiv koding for å redusere mengden data som trengs for å beskrive videoen. I denne prosessen blir det brukt en kombinasjon av to typer predikering, nemlig intra- og inter-predikering. For å forstå disse to begrepene, er det vesentlig å forstå hvordan bildene i en videostrøm er strukturert. De ulike bildene i en videostrøm kan deles inn i tre ulike "frames", også kalt rammer. Disse er I-ramme, P-ramme og B-ramme, og en videostrøm består enten av kun I-rammer eller som en kombinasjon av alle. I figur 5.9 nedenfor er det gjengitt en videosekvens som en kombinasjon av alle rammene.



Figur 5.9: Oversikt over en vilkårlig videostrøm også kalt GOP.[125]

En vilkårlig videostrøm, som den vist i figuren ovenfor, vil alltid starte med en I-ramme. Det totalet antallet ulike rammer kan variere, og er gitt av GOP-strukturen hvor GOP står for "Group of pictures".

I-ramme: Denne rammen blir kalt for en nøkkelramme eller et nøkkelbilde. Det er et komplett bilde som ikke er avhengig eller refererer til noen av de andre bildene i videostrømmen[125]. Rammen inneholder alt av informasjon som trengs for å rekonstruere det enkeltet bildet i dekodingsprosessen. I-rammen er stort sett større enn både P-rammer og B-rammer, og krever dermed større båndbredde eller lagringsplass. Dette fordi I-rammen inneholder all bildeinformasjonen, noe de andre rammene ikke gjør. En I-ramme brukes som regel i starten av en ny bildesekvens eller ved store endringer i scenen som blir filmet.

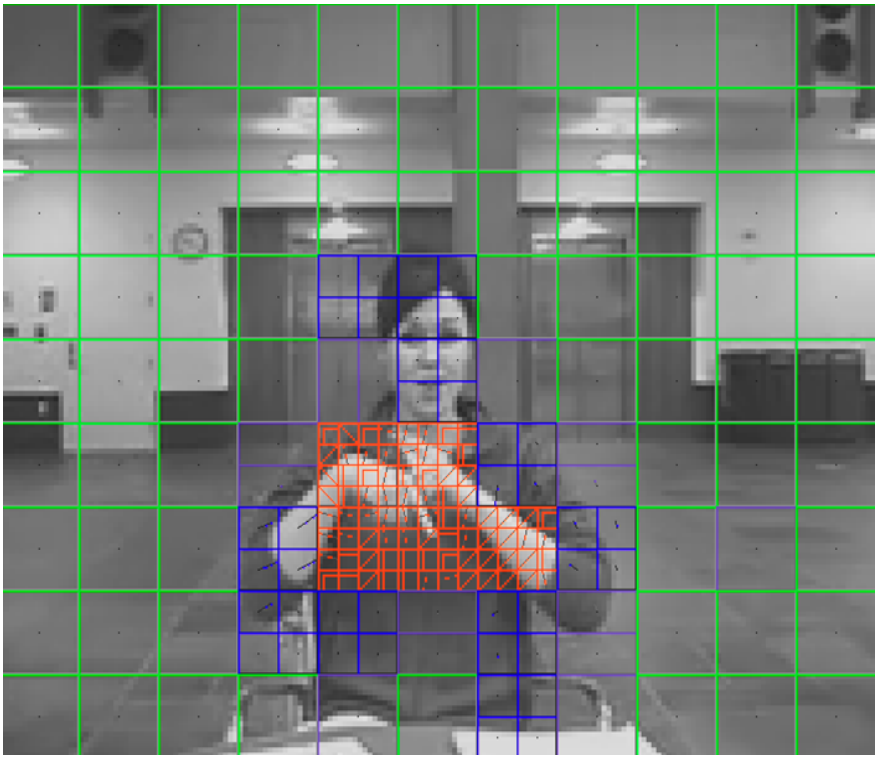
P-ramme: P-rammen (prediktiv ramme) er et bilde som refererer til og er avhengig av forrige I- eller P-ramme i bildesekvensen. Rammen blir beregnet ved å bruke bevegelseskompensasjon[8] fra forrige ramme (I eller P) for å predikere bildedataen i nåværende ramme. Forskjellen mellom den predikerte rammen og den faktiske rammen lagres som et residual[124] som kan komprimeres ytterligere ved hjelp av transformasjons- og kvantiseringsteknikker, som blir forklart senere i delkapittel 5.6.2.1.3. Dette gjør at P-rammer krever mindre båndbredde eller lagringsplass enn I-rammer.

B-ramme: En B-ramme, også kalt en bi-prediktiv ramme, er et bilde som refererer til og er avhengig av både forrige og neste I- eller P-ramme i bildesekvensen[125]. Det vil si at fremtidige I- eller P-rammer må regnes ut før informasjonen i rammene kan sendes over videolinjen. B-rammen bruker bevegelseskompensasjon både fremover og bakover for å predikere nåværende ramme. Som for P-rammen, blir også her forskjellen mellom den predikerte rammen og den faktiske rammen lagret som et residual som kan komprimeres ytterligere. B-rammen krever stort sett mindre båndbredde eller lagringsplass enn P-rammer.[125]

---

[8] Motion compensation in computing, is an algorithmic technique used to predict a frame in a video, given the previous

Før predikeringen av en ramme kan skje, deler enkoderen rammen opp i flere mindre seksjoner, kalt makroblokker på typisk 16x16 piksler[124]. Hver av disse makroblokkene kan behandles og kodes separat, noe som resulterer i mer effektiv videokomprimering. Når rammen er delt opp i makroblokker, avgjør enkoderen hvilken type predikasjon, intra eller inter, som er mest effektiv for hver enkelt makroblokk. Hver makroblokk kan også deles ned i mindre blokker, helt ned til 4x4 piksler, noe som gjør det enklere å kode og komprimere detaljerte områder eller bevegelser. I figur 5.10 nedenfor, er det vist hvordan et bilde i en video kan deles opp i makroblokker.



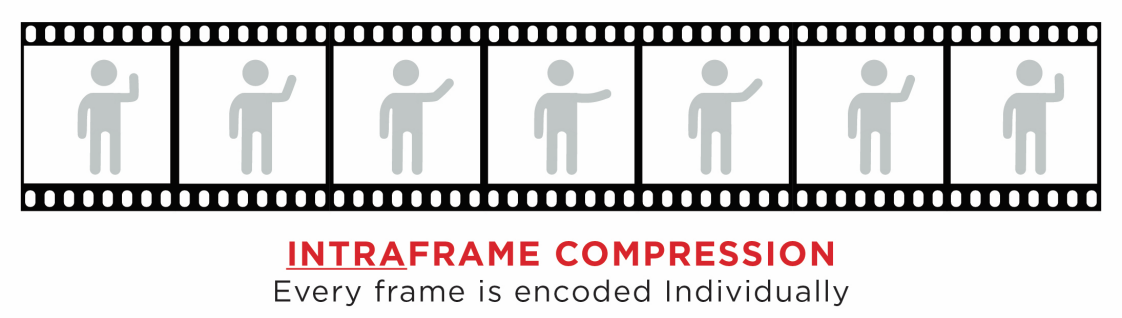
Figur 5.10: Et bilde i en video delt opp i hele og delte makroblokker.[127]

De grønne kvadratene i figuren representerer makroblokker, og de andre kvadratene er makroblokker delt opp i mindre deler. Registrer hvordan bakgrunnen er representert av hele makroblokker, mens personen i bildet er representert av oppdelte makroblokker. Dette er fordi bakgrunner endrer seg minimalt fra bilde til bilde, mens personen beveger seg, og endrer seg fra bilde til bilde. Det kan også registreres at noen av boksene som omringer personen har noen linjer inni seg. Dette er bevegelsesvektorer utledet fra bevegelseskompensasjon som ble nevnt tidligere for P- og B-rammer. Bevegelsesvektorene og bevegelseskomepnsasjonen vil bli litt mer forklart kombinert med inter-predikering i delkapittel 5.6.2.1.2.

5.6.2.1.1 Intra-predikering

Intra-predikering er en teknikk som blant annet brukes kombinert med inter-predikering (5.6.2.1.2) av H.264-kodeken til bildekomprimering. Predikasjonen brukes for å redusere datamengden for et enkelt bilde eller ramme, som igjen øker effektiviteten av komprimeringen. Ordet "intra" betyr innenfor, og betyr at intra-predikering kun skjer på enkeltrammer uten å være avhengig av andre rammer. Det vil si at I-rammene i figur 5.9 fra tidligere, blir predikert og komprimert ved hjelp av intra-predikering. I video komprimert ved hjelp av intra-predikering, vil hvert bilde komprimeres individuelt og inneholde all informasjonen[128]. Dette er illustrert i figur 5.11 nedenfor.

and/or future frames by accounting for motion of the camera and/or objects in the video. Motion compensation describes a picture in terms of the transformation of a reference picture to the current picture. The reference picture may be previous in time or even from the future.[126]



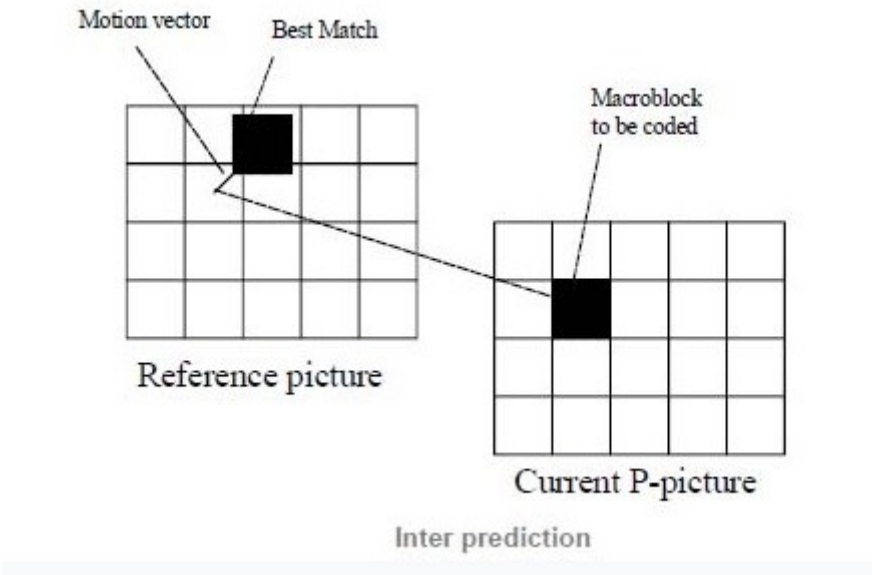
Figur 5.11: Illustrasjon som viser kompresjon ved hjelp av intra-frame[129]

Nærliggende piksler i et bilde er ofte veldig like i lysstyrke og tone, som for eksempel gulvet i figur 5.10 i forrige delkapittel. I stedet for å lagre hver piksel seperat, kan hele eller oppdelte makroblokker representere flere av pikslene som er omtrent like[128]. Makroblokkene eller de mindre blokkene inneholder da differansen mellom den faktiske pikselverdien og den predikerte verdien for hver piksel[128]. Dette reduserer antallet bits nødvendig for å enkode dataen og sende den.

5.6.2.1.2 Inter-predikering

Inter-predikering er en annen teknikk som blant annet brukes kombinert med intra-predikering (5.6.2.1.1) av H.264-kodeken. Predikasjonen reduserer datamengden nødvendig for å representere en ramme ved å predikere den nye rammen basert på forrige og/eller neste ramme. Med andre ord, brukes inter-predikering for å predikere P- og B-rammene i figur 5.9 som ble nevnt tidligere[125].

Hver ramme deles også opp i makroblokker ved inter-predikering. Når en makroblokk skal enkodes, vil enkoderen prøve å finne en lignende blokk i en referanseramme fra tidligere. Et eksempel som gjengir dette er vist i figur 5.12 nedenfor.

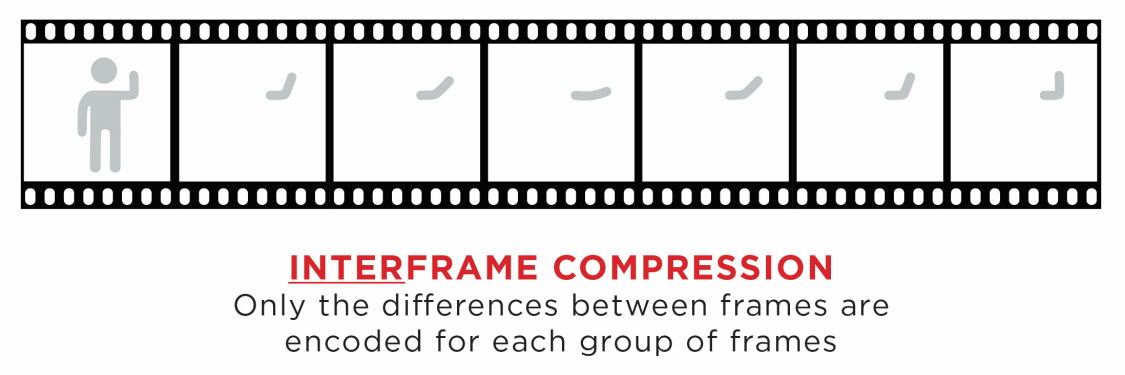


Figur 5.12: Illustrasjon for hvordan enkoderen prøver å matche en makroblokk med en tidligere referanseramme.[130]

I figur ovenfor kan man se at enkoderen finner en blokk i referanserammen som ligner på makroblokken i rammen som skal enkodes, men blokken har forskjøvet seg. Denne forskyvningen blir enkodet som en bevegelsesvektor som peker til den lignende blokken i referanserammen[125]. Denne prosessen blir

også kalt for bevegelsesestimasjon.

Enkoderen vil i de fleste tilfellene klare finne en lignende blokk, men blokkene er sjeldent identiske. Dermed vil differansen mellom blokkene bli beregnet som residualet eller feilpredikering[125]. Ved hjelp av dette residualet og bevegelsesvektoren, vil dekoderen være i stand til gjenskape bildet. I figuren nedenfor er det gitt et eksempel på hvordan inter-predikering fungerer. Her er den første rammen et nøkkelbilde eller I-ramme, mens de resterende er P- og B-rammer.



Figur 5.13: Illustrasjon som viser kompresjon ved hjelp av inter-frame.[129]

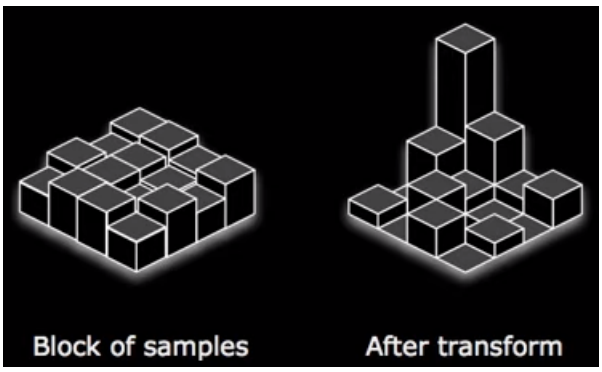
Tar man de ulike P- og B-rammene i figuren ovenfor og kombinerer dem individuelt med I-rammen, vil man være i stand til å dekode og gjenskape de orginale bildene og videoen. Det er også tydelig fra figuren hvorfor informasjonen nødvendig for å representere en video bli mindre ved hjelp av inter-predikering.

H.264-kodeken kombinerer både inter- og intra-predikering, og oppnår derfor en svært høy komprimeringseffektivitet av video. Inter-prediksjonen brukes for å predikere forskjellen mellom de ulike bildene i videostrømmen, mens intra-predikasjon reduserer redundansen i hvert bilde. Kombinert kan filstørrelsen reduseres kraftig uten at det går noe særlig merkbart ut over bildekvaliteten.

5.6.2.1.3 Transformasjon og kvantisering

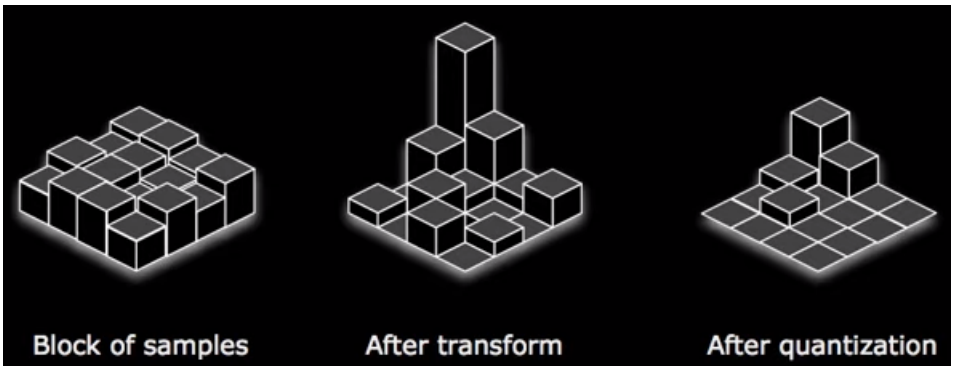
Når enkoderen har predikert alle makroblokkene i rammene, må alle makroblokkene transoformeres og kvantiseres før de kan enkodes som en bitstrøm. Dette er to trinn som er med på å komprimere informasjonen i videoen enda mer.

Et bildet er gitt av blokker av piksler, hvor dataen i blokkene representerer lysstyrke og toner for pikslene som et nummer. Hvert av disse nummerene er viktige, men totalt sett blir dette mye data. Dette kan løses med transformasjon. Transform refererer til prosessen med å konvertere blokker av piksler i et bilde og dataen i dem, til et sett med transformasjonskoeffisienter som representerer blokkene i frekvensdomenet[131]. Blokkene som transformeres kan ha størrelsene 4x4 eller 8x8, og transformeres ved hjelp av heltallstransformasjon som er en tilnærming av DCT ("Discrete Cosine Transform")[124]. Dette er en vanlig transformteknikk for bildekomprimering som gir en representasjon av blokkene hvor frekvensene som bidrar med mest energi, som i lysstyrke, vektes mest. Grunnen til at det er lysstyrken som vektes, og ikke fargene eller tonen, er fordi menneskesynet er mer sensitivt til endringer i lysstyrke enn til endringer i toner[132]. Figur 5.14 nedenfor viser hvordan en typisk blokk av bildedata kan se ut etter transformasjonen.



Figur 5.14: Blokk av bildedata som transformeres til frekvensdomenet.[131]

Etter at blokken av piksler er blitt transformert, kan representasjonen i frekvensdomenet se ut som i figur 5.14 ovenfor. Ved å kvantisere transformasjonen, kan små eller ubetydelige verdier fjernes. Dette gjøres ved å dele hver transformasjonskoeffisient på et heltall (høyere tall reduserer bildekvalitet), selv om det reduserer presisjonen til transformasjonskoeffisientene[124]. Dette reduserer med andre ord antall bits nødvendig for å representere transformasjonkoeffisientene. Figur 5.15 nedenfor illustrerer hvordan den orginale blokken av data kan komprimeres til en mye mindre blokk ved hjelp av transformasjon for så og kvantiseres.



Figur 5.15: Illustrerer hvordan en blokk kan se ut etter transformasjons- og kvantiseringsprosessen.[131]

Etter kvantiseringen, gjenstår de signifikante transformkoeffisientene, mens de resterende mindre signifikante er satt til null[124]. Dette betyr at denne blokken vil kreve drastisk mye mindre båndbredde eller lagringsplass enn den orginale blokken. Gjøres transformasjonen og kvantiseringen på riktig vis, kan denne prosessen reverseres av dekoderen slik at den reverserte blokken er omtrent lik den orginale[131].

H.264-kodeken har adaptiv kvantisering, som justerer kvantiseringmatrisen for å oppnå ønsket bildekvalitet på videoen[123]. Det vil si at den kan vurdere den visuelle kompleksiteten i hver makroblokk og tilpasse kvantiseringsnivået for hver blokk. Høy kompleksitet betyr at det bør beholdes mer av informasjonen, og kvantiseringen justeres deretter for å redusere tapet av informasjon.

5.6.2.1.4 Enkoding

Etter bildedataen er predikert, transformert og kvantisert, må den enkodes før den kan sendes eller lagres som en komprimert bitstrøm. Her må alt informasjonen som skal til for at dekoderen skal klare dekode bitstrømmen kodes inn[124]. Det vil blant annet si:

- Informasjon fra predikeringsprosessen som lar dekoderen gjenskape predikeringen
- Informasjon fra transformasjonen og kvantiseringen, som for eksempel transformasjonskoeffisientene
- Informasjon om strukturen til den komprimerte dataen og verktøyene brukt for å komprimere den [124]
- Informasjon om hele bildesekvensen [124]

Kodingen av denne binære bitstrømmen kalles for "Entropikoding"[131], og har en fast struktur som beskriver ulik informasjon for dataen. Bitstrømmen kan overføres eller lagres, og er lesbar for en H.264-dekoder. Dette er i grove trekk slik enkodingsprosessen fungerer.

5.6.2.2 Dekodingsprosessen

Når H.264-kodeken skal dekode en komprimert video, trenger dekoderen all informasjonen som ble oppsummert ovenfor i delkapittel 5.6.2.1.4. Den faste strukturen i bitstrømmen gjør at dekoderen kan hente ut informasjonen som trengs for å begynne dekodingen av videoen og gjennskape den.

Først blir de kvantiserte transformasjonskoeffisientene skalert ved å gange hver koeffisient med et heltall for å gjennskape den orginale verdien på koeffisienten[124]. Deretter blir det gjennomført en inverstransformasjon av alle blokkene, før de blir satt sammen til makroblokker for å gjennskape residualet.

For hver av makroblokkene, gjennomfører dekoderen en identisk predikasjon slik som enkoderen gjor. Deretter kombinerer dekoderen prediksjonen med residualet for å gjennskapet det "orginale" bildet, og kan så vise det på en skjerm. Som nevnt innledningsvis i delkapittelet om predikering, finnes residualet ved å ta differansen mellom prediksjonen og det orginale bildet. Med andre ord kan det orginale bildet gjennskapes ved å summere prediksjonen og residualet. Dette er i grove trekk slik dekodingsprosessen fungerer.

5.6.2.3 Lossy og lossless

H.264-kodeken er som regel en "lossy"-komprimeringsmetode, men kan i noen tilfeller også være "lossless"[123]. "Lossy" vil si at for å redusere filstørrelsen, fjernes det permanent informasjon fra den orginale filen. Dette fører til en reduksjon i bildekvaliteten. Men ved hjelp av smarte komprimeringsalgoritmer, er det hovedsaklig redundant og mindre viktig informasjon som fjernes. Dette er stort sett informasjon som ikke påvirker i særlig stor grad for hvordan det menneskelige syn oppfatter bildet. Med andre ord, oppnår man en mindre filstørrelse på bekostning av bildekvalitet, men som har liten betydning for hvordan et menneske oppfatter bildet. "Lossy"-komprimeringsmetoder har blitt mer og mer populært ettersom komprimeringsalgoritmene har utviklet seg med årene[120].

Motparten til "lossy"-komprimering, er "lossless"-komprimering. "Lossless"-komprimering gjør det mulig å rekonstruere den komprimerte dataen uten tap av informasjon. I en video bestående av for eksempel

60 bilder i sekundet, er ofte et område i bildet representert av flere piksler med samme farge og lysstyrke (blå himmell for eksempel). Da kan denne dataen representeres som "100 blå piksler" i stedet for 100 individuelle "blå piksel"[120].

Som nevnt, vil ikke bildekvaliteten reduseres ved "lossless"-komprimering slik den vil med "lossy"-komprimering. Men dette gjør også at det er begrenset for hvor mye et bilde kan komprimeres før bildekvaliteten reduseres. Med andre ord, kan et bilde komprimeres mye mer ved hjelp av "lossy"-komprimering. Gode komprimeringsalgoritmer har gjort det vanlig å bruke "lossy"-komprimering av video, da mye av bildekvaliteten beholdes samtidig som filstørrelsen reduseres drastisk.

5.6.2.4 Profiler

Funksjonene til H.264-kodeken som er nevnt i dette kapittelet er bare en liten del av alle de tilgjengelige funksjonene kodeken kan operere med. Kodeken har en rekke profiler eller moduser som kan kjøres, og som avgjør hvilke funksjonaliteter, teknikker og algoritmer som blir brukt av kodeken for å komprimere[122]. Hovedsaklig opereres det med tre ulike profiler, hvor mange av de andre profilene er utgreininger fra disse[123]. Disse hovdeprofilene er: "Baseline"-, "Main"- og "High"-profile[122]. De ulike profilene er forklart i boksene nedenfor. Forklaringene er hentet fra [122].

Baseline-profile - [122]

"This is the simplest profile used mostly for low-power, low cost devices, including some video-conferencing and mobile applications. Baseline profiles can achieve a compression ratio of about 1000:1 — i.e. a stream of 1 Gbps can be compressed to about 1 Mbps. They uses 4:2:0 chrominance sampling, which means that color information is sampled at half the vertical and half the horizontal resolution of the black and white information. Other important features of the Baseline Profile are the use of Universal Variable Length Coding (UVLC) and Context Adaptive Variable Length Coding (CAVLC) entropy coding techniques."

Main-profile - [122]

"Main Profile includes all of the functionality of Baseline, but with improvements to frame prediction algorithms. It is used for SD digital TV broadcasts that use the MPEG-4 format, but not for HD broadcasts."

High-profile - [122]

"H.264 High Profile is the most efficient and powerful profile in the H.264 family, and is the primary profile for broadcast and disc storage, particularly for HDTV and Bluray disc storage formats. It can achieve a compression ratio of about 2000:1. The High Profile also uses an adaptive transform that can select between 4x4 or 8x8-pixel blocks. For example, 4x4 blocks are used for portions of the picture that are dense with detail, while portions that have little detail are compressed using 8x8 blocks. The result is the preservation of video image quality while reducing network bandwidth requirements by up to 50 percent. By applying H.264 High Profile compression, a 1 Gbps stream can be compressed to about 512 Kbps."

For å forstå det som står i boksene ovenfor, krever det en del innsikt og forståelse av H.264-kodeken. I dette prosjektet blir det brukt en "High-profile" for H.264-videokomprimering på Nvidia Jetson Nano. Denne profilen bruker flere teknikker for komprimering enn de to andre, og er i stand til å oppnå et komprimeringsforhold på 2000:1. Som det står i boksen ovenfor, kan en videostrøm på 1Gbps bli komprimert ned til en videostrøm på 512kbps.

Etter å ha valgt en profil som skal kjøres, er man nødt til å sette noen grenser. Disse grensene avgjør blant annet hvor mye datakraft som trengs for enkoding og dekoding, bildeoppløsning og høyeste tilatte bitrate. Dette sørger for at dekoderen er i stand til å dekode informasjonen som er blitt enkodet.

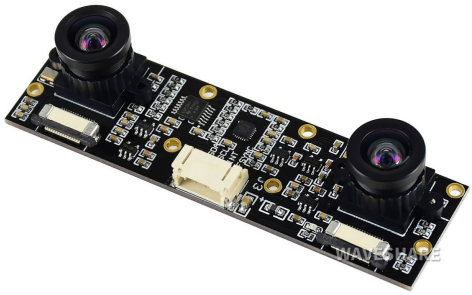
Nvidia Jetson Nano har egen dedikert enkoder og dekoder for videostrømming for støtte opp til 4k-oppløsning. Både enkoderen og dekoderen er designet for å være effektive når det kommer til effektforbruket, uten at det skal gå på bekostning av ytelsen. Den har støtte for alle hovedprofilene som er nevnt ovenfor, noe som gjør at den er godt egnet for å ta inn bildedata, komprimere den og sende komprimert bildedata videre til kontrollstasjon.

5.7 Valg av kamera

I dette delkapittelet vil valg av de ulike kameraene komme frem basert på spesifikasjoner, beregninger, fordeler og ulemper. Det skal velges et stereokamera som skal brukes i fronten av ROV, og to enkle kamera. Ett som ser under ROV og ett for oversikt over manipulator.

5.7.1 Valg av stereokamera og linser

Da det skulle velges stereokamera som skulle stå i fronten av ROV-en, var det to ulike kamera som ble vurdert. Det første stereokameraet av typen IMX219-83, vist i figur 5.16a nedenfor, og det andre av typen ELP 960p, vist i figur 5.16b nedenfor. Sistnevnte var samme type stereokamera som ble brukt for å løse oppgavene i fjorårets bachelor [31].



(a) IMX219-83 Stereo Camera [133]

(b) ELP 960p Synchronized Dual Lens Stereo USB Camera [134]

Stereokameraet fra ELP bruker MJPEG[9] som komprimeringsformat, og sender begge bildestrømmene over en integrert videolinje via USB. Det at bildene må komprimeres før de kan sendes fra kameraet kan bidra til uønsket forsinkelse på bildestrømmen. I fjor var dette en av erfaringene som ble gjort med stereokameraet fra ELP.

Et av målene som ble satt ifjor, var å redusere forsinkelsen på sending av video fra ROV og opp til kontrollstasjonen. Men under testing kom det frem at stereokameraet fra ELP introduserte mye

[9] "Motion JPEG" er et komprimeringsformat som komprimerer hvert bilde i en videostrøm som et eget bilde i JPEG-format

forsinkelse på videostrømmen. Testene viste at videostrømmen fra ROV opp til kontrollstasjonen hadde en forsinkelse på 195ms[31]. I denne forsinkelsen ble det målt at det var nettopp innhenting av video fra kamerat som tilførte den største forsinkelsen (målt til ca 100ms). Løsningen for å redusere denne forsinkelsen var en reduksjon i oppløsning. Dette ble ikke gjort da en forsinkelse lavere enn 200ms ble sett på som greit. Erfaringene som ble gjort ifjor med tanke på stereokameraet fra ELP, gjor at dette kameraet ble mindre aktuelt å bruke om igjen, selv om det oppfylte flere av nøkkelspesifikasjonene. En av løsningene som ble foreslått i fjor angående stereokamera, var å bruke en NVIDIA Jetson hvor stereokameraet kunne kobles direkte ved bruk av MIPI[31].

Kommunikasjonskontrolleren som blir brukt i år er av typen NVIDIA Jetson Nano som har to porter for MIPI CSI-2, noe som gjør at forslaget angående kamera over MIPI fra i fjor kan realiseres. MIPI CSI-2 er en protokoll som i dag er svært vanlig når bildedata skal overføres fra et kamera til en prosessor[135]. Dette på grunn av blant annet den høye båndbredden på maksimalt 10Gbps[136], som er det dobbelte av det man får med USB3.0. Protokollen støtter flere videoformat, men den viktigste i dette tilfellet er "RAW"-formatet. Som navnet på formatet angir, er dette overføring av ukomprimert rådata direkte fra kameraet ved hjelp av den høye båndbredden. Dette gjør at det unngås forsinkelse på videostrømmen grunnet at kameraet må komprimere bildedataen før den sendes.

Stereokameraet IMX219-83 (figur 5.16a) er et kamera som er kompatibelt med NVIDIA Jetson Nano, hvor det finnes mye informasjon hvordan de to kan konfigureres sammen. Dette er et stereokamera hvor hvert kamera sender sin egen bildestrøm over CSI, noe som gjør det godt egnet å kombinere med NVIDIA Jetson Nano og dens porter for MIPI CSI-2. Selv om det ikke vil bli brukt, kommer også kameraet med en egen IMU. Noen av de viktigste spesifikasjonene til stereokameraet er gjengitt i listen nedenfor[133]:

- **Dimensjon:** 24mm x 85mm
- **Oppløsning:** 3280 x 2464p (8MP per kamera)
- **Kamerasensor:** Sony IMX219
- **Linsespesifikasjoner:**
  - **CMOS-størrelse:** 1/4"(4,60mm diagonalt)
  - **Fokallengde:** 2,6mm
  - **FOV:** 83/73/50 grader (diagonalt/horisontalt/vertikalt)
  - **Baseline-lengde:** 60mm

Kamerasensoren IMX219 fra Sony har en maksimal overføringshastighet på 755Mbps per linje ved bruk av 4-linjers overføring og en overføringshastighet på 912Mbps per linje ved 2-linjers overføring[137]. Nvidia Jetson Nano har støtte for 4-linjers overføring på 1,5Gbps per linje. Kamerasensoren er blant annet i stand til å levere video med en oppløsning på 1920 × 1080p (Full-HD) med 60 bilder i sekundet.

Ved å kombinere IMX219-83 og Nvidia Jetson Nano, vil de fleste av de ønskede funksjonene til systemet la seg realisere. Kameraet er i stand til å levere minstekravet til oppløsning på 1920 x 1080p med 30 bilder i sekundet, som da blir en ukomprimert bildestrøm på rundt 0,7Gbps fra hvert kamera, og MIPI CSI-2 har en båndbredde som er større enn dette. Ved å sende ukomprimert bildedata fra stereokamera til Jetson, fjernes også et unødvendig komprimeringsledd i kameraet som kan bidra til forsinkelse. Stereokameraet har en størrelse på 24mm x 85mm, noe som gjør at det passer bra i kuppelen med diameter på 160mm. Prisen på kameraet er på omtrent 480kr, mens stereokameraet fra ELP koster rundt 890kr. Stereokameraet IMX219-83 har både god funksjonalitet med Jetson Nano og gode spesifikasjoner, og i tillegg er det nesten halvparten av prisen på ELP.

Stereokameraet IMX219-83 passer i kuppelen på ROV-en og leverere minimumsoppløsningen på 1920 x 1080p oppløsning med 30 bilder i sekundet uten problem. Ukomprimert bildedata direkte til Jetson Nano, fjerner unødvendig ledd med forsinkelse. Og kamearaet kan fås til en veldig rimelig pris. Derfor er det blitt bestemt at det er IMX219-83 som skal brukes som stereokamera på ROV-en.

5.7.1.1 Synsvinkelen til IMX219-83 Stereo Camera

Det er blitt bestemt at det skal brukes et stereokamera av typen IMX219-83, som oppfyller flere av kravene som er satt. Det som gjenstår er å sjekke om linsen som følger med på IMX219-83 gir de ønskede synsvinklene og synsfelt. I dette delkapittelet vil beregninger av synsvinklene bli gjennomført for å sjekke om stereokameraet oppfyller ønskede krav.

Stereokameraet har en maksimal oppløsning på 3280 x 2464 (8 megapiksler) per kamera og en "base line"-lengde på 60mm. Fokallengden på linsen er 2,6mm og kamerasensoren har en CMOS-størrelse på 1/4". Den oppgitte synsvinkelen er på 83/73/50 grader (diagonalt/horisontalt/vertikalt). Kravet som er satt, er at stereokameraet bør være i stand til å se et objekt med størrelsen 24cm x 13,5cm (27,5cm diagonalt) når avstanden mellom kamera og objekt er på 20cm.

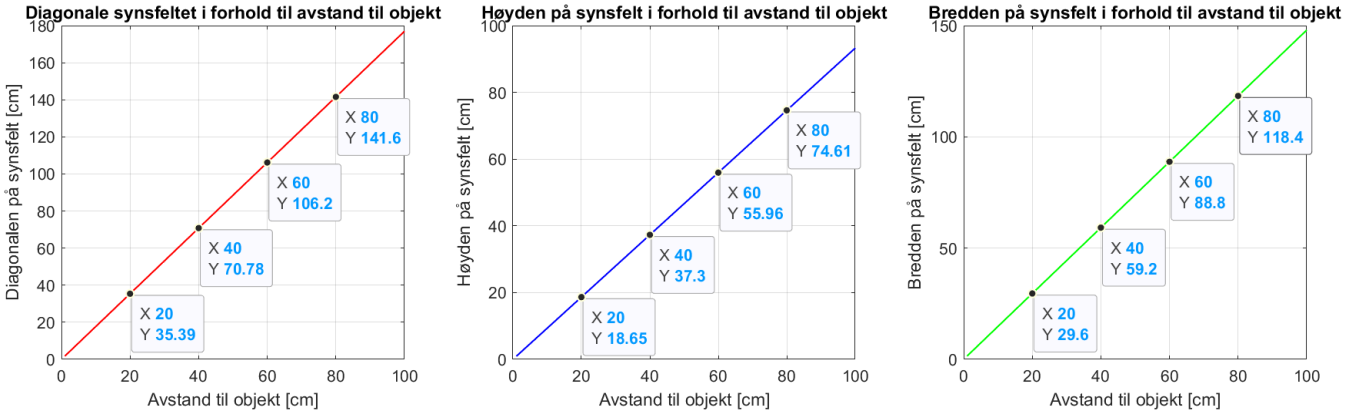
Når synsvinklene til kameraet er oppgitt, kan formel 5.3 fra delkapittel 5.5.1 snus på og løses for $\boldsymbol{a}$. Da dette kun er halve synsfeltet i en retning, kan den ganges med to for å finne hele synsfeltet i en retning, og blir slik:

$$synsfelt = 2tan(\frac{\theta}{2})c \tag{5.8}$$

Hvor:

- θ synsvinkelen i diagonal-,høyde- eller bredderetning
- c avstanden mellom kamera og objekt

Formel 5.8 ovenfor er blitt brukt i Matlab for å plotte sysnfeltet i de ulike retningene. Dette er vist i figur 5.17 nedenfor.



Figur 5.17: Lengden på synsfeltene til IMX219-83 i forhold til avstanden til et objekt. Utgangspunkt i synsvinkler på 83/73/50

Fra plottene i figur 5.17 ovenfor, kan det ses at IMX219-83 med synsvinklene 83/73/50, vil være i stand til å se et objekt på 35,39cm x 18,65cm (29,6cm diagonalt) når objektet er 20cm fra kameraet. Med andre ord, vil stereokameraet oppfylle kravet til å se et objekt med størrelsen 24cm x 13,5cm (27,5cm diagonalt) når avstanden mellom kamera og objekt er på 20cm. Av denne grunn, vil ikke standardlinsen på IMX219-83 som følger med bli byttet ut.

5.7.2 Valg av enkle kamera og linser

Da det skulle velges to enkle kamera, hvor et skulle plasseres under ROV-en, og et skulle plasseres for å ha oversikt over manipulator. Var det tre kamera som ble vurdert. Alle kameraene som ble vurdert er av typen USB-kamera. Grunnen til at det ble bestemt at det ikke skulle brukes analoge kamera, var behovet for å sende 4 videostrømmer til land og minimere antall kabler i tjoren. Hvis analoge kamera hadde blitt valgt, hadde dette gitt en koaksialkabel for hvert kamera. Eventuelt måtte muligheten ved å kjøpe inn eller utvikle en multiplekser[10] blitt undersøkt, slik at alle videostrømmene kunne blitt sendt over samme kabel.

Tanken på å bruke IP-kamera ble lagt bort da det ble bestemt at det skulle brukes en Nvidia Jetson Nano, i plassen for en løsning med en nettverkssvitsj, som ble diskutert i delkapittel 4.4.2 under valg av maskinvare. En løsning med både Nvidia Jetson Nano og en nettverkssvitsj ville gitt mangel på plass i elektronikkhuset.

For valg av USB/CSI-kamera som skal brukes på ROV-en, er det to ulike kameramoduler som er blitt vurdert. De ulike kameramodulene er av typene: IMX219 og OV2710. I de neste seksjonene vil de ulike kameraene bli diskutert, og til slutt vil det komme frem hvorfor kameramodulen OV2710 ble valgt.

5.7.2.1 IMX219 Kameramodul

Den første kameramodulen som ble vurdert var av typen IMX219, vist i figur 5.18 nedenfor. Dette er et kamera som bygger på samme kamerasensor som stereokameraet IMX219-83 som ble valgt. Også dette kameraet er bra egnet å kombinere sammen med Nvidia Jetson Nano på grunn av den ukomprimerte bildedataen som kan sendes over MIPI CSI-2.

Figur 5.18: IMX219 Camera Module [138]

[10]En mulitplekser tar to eller flere informasjons-/datakanaler og kombinerer dem slik de kan sendes over en felles datalinje.

Dette kameraet har en diagonal synsvinkel på 160 grader. Ved å bruke formel 5.3 fra delkapittel 5.5.1, kan synsfeltet til dette kameraet når det er 10cm fra objektet beregnes til omtrent 110cm. Et synsfelt så stort er noe ekstremt for det kameraene skal brukes til. To kamera av typen IMX219 vil også kreve to dedikerte linjer med MIPI CSI-2. Da Nvidia Jetson Nano kun har to slike linjer, og det er blitt valgt et stereokamera som tar opp begge disse, vil to IMX219 være uegnet å bruke. Av denne grunn ble det bestemt det måtte finnes to kamera hvor overføringen av bilde gikk over USB.

5.7.2.2 OV2710 USB-kamera

Den andre kameramodulen som ble vurdert og valgt var et USB-kamera basert på sensoren OV2710, vist i figur 5.19 nedenfor. I denne seksjonen vil nøkkelspesifikasjoner til kameraet gjennomgås, samt begrunnelser for at akkurat dette kameraet ble valgt.

Figur 5.19: OV2710 2MP USB-Kamera [139]

OV2710 USB-kameraet (figur 5.19) er et kamera som er kompatibelt med NVIDIA Jetson Nano[139]. Sendingen av videstrømmen går over en USB2.0-linje. Noen av de viktigste spesifikasjonene til kameraet er gjengitt i listen nedenfor[139]:

- **Dimensjon:** 38mm x 38mm
- **Oppløsning:** 1920 x 1080p (2MP) 30fps
- **Kamerasensor:** OV2710
- **Driftstemperatur:** 0℃ + 50℃
- **Bildeformat:** MJPEG eller YUY2
- **Linsespesifikasjoner:**
  - **CCD-størrelse:** 1/2,7"(4,60mm diagonalt)
  - **Fokallengde:** 2,9mm
  - **FOV:** 145 grader (diagonalt)

Fra listen av spesifikasjonen gitt ovenfor, kan det ses at kameraet er lite og kompakt. Dette gjør at det vil være greit å lage små, fleksible kamerahus til kameraene som kan monteres på ROV-en. Hvordan kamerahuset ble seende ut er vist i kapittel 2.7.1.2. Kameraet kan levere 30 bilder i sekundet med en oppløsning på 1920 x 1080p, som betyr at det er over minimumskravet til 1280 x 720p. Bildeoverføringen går over USB2.0, noe som gjør at kameraet enkelt kan kobles til en av USB-porten på Nvidia Jetson Nano. Det er også oppgitt på produktsiden[139], at dette skal være et kamera som egner seg godt i mørke omgivelser med lite lys.

En potensiell ulempe med dette kameraet, er det faktum at bildene ikke overføres som rådata. Kameraet kan ta i bruk komprimeringsformatene MJPEG eller YUY2. Som nevnt når stereokameraet ble diskutert, kan komprimering av bildet før det sendes til Nvidia Jetson Nano, føre til uønsket og unødvendig forsinkelse på bildestrømmen opp til land. Hvor mye dette har å si vil komme frem i delkapittel 5.8 hvor resultatet blir presentert.

Det siste så må gjennomgås for å verifisere at kameraet er egnet til denne oppgaven, er å sjekke synsfeltet oppfyller funksjonsspesifikasjonen. Synsvinkelen og beregning av synsfeltet for linsen til USB-kameraet OV2710 vil bli gjennomgått i neste delkapittel.

5.7.2.3 Synsvinkelen til OV2710 USB-kamera

Det skal plasseres to USB-kamera av typen OV2710[139] på ROV-en. Et kamera i bunnen, og et kamera skal plasseres slik at det gir et overblikk av gripekloen. Kravet som er satt for synsfeltet til USB-kameraene, er at de skal være i stand til å se et objekt som er 48cm x 27cm når objektet er 10cm fra kameraet. Dette USB-kameraet har en maksimal oppløsning på 1920 x 1080 (2 megapiksler) og en oppgitt synsvinkel diagonalt på 145 grader. Fokallengden er på 2,9mm og kamerasensoren har en CCD-størrelse på 1/2,7". Aperaturen, altså åpningen på linsen er 2,8. Med andre ord er åpningen på linsen relativt stor, noe som gjør at mer lys slipper inn til kamerasensoren[140]. Dette gjør at kameraet er bra egnet i miljø hvor det ikke er fult så mye lys, som for eksempel under vann.

I motsetning til stereokameraet IMX219-83 som har CMOS-kamerasensor, har OV2710 en CCD-kamerasensor. Listen nedenfor hentet fra [141] oppsummerer kort forskjellen på de to typene kamerasensorer.

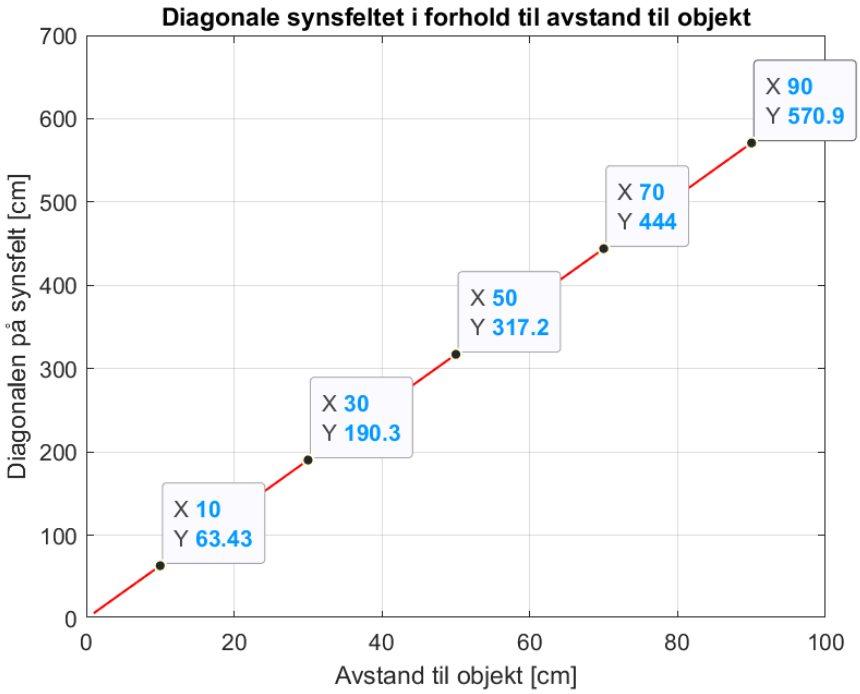
Key differences between CCD and CMOS imaging sensors - [141]

This all adds up to several main differences between CMOS and CCD sensors:

- CCD sensors create high-quality, low-noise images. CMOS sensors are usually more susceptible to noise.
- Because each photosite on a CMOS sensor has several transistors located next to it, the light sensitivity of a CMOS chip tends to be lower, as many of the photons hit the transistors instead of the photosite.
- CCD sensors consume as much as 100 times more power than an equivalent CMOS sensor.
- CMOS sensors can be manufactured on most standard silicon production lines, so are inexpensive to produce compared to CCD sensors.

Overall, CMOS sensors are much less expensive to manufacture than CCD sensors and are rapidly improving in performance, but CCD sensors may still be required for some demanding applications.

Den oppgitte synsvinkelen på diagonalen er på 145°. Ved å bruke ligning 5.8 slik som i beregningene for stereokameraet, kan grafen for det diagonale synsfeltet lages. Denne grafen er vist i figur 5.20 nedenfor.



Figur 5.20: Lengden på det diagonale synsfeltet til OV2710 ved ulike avstander og synsvinkel på 145 grader

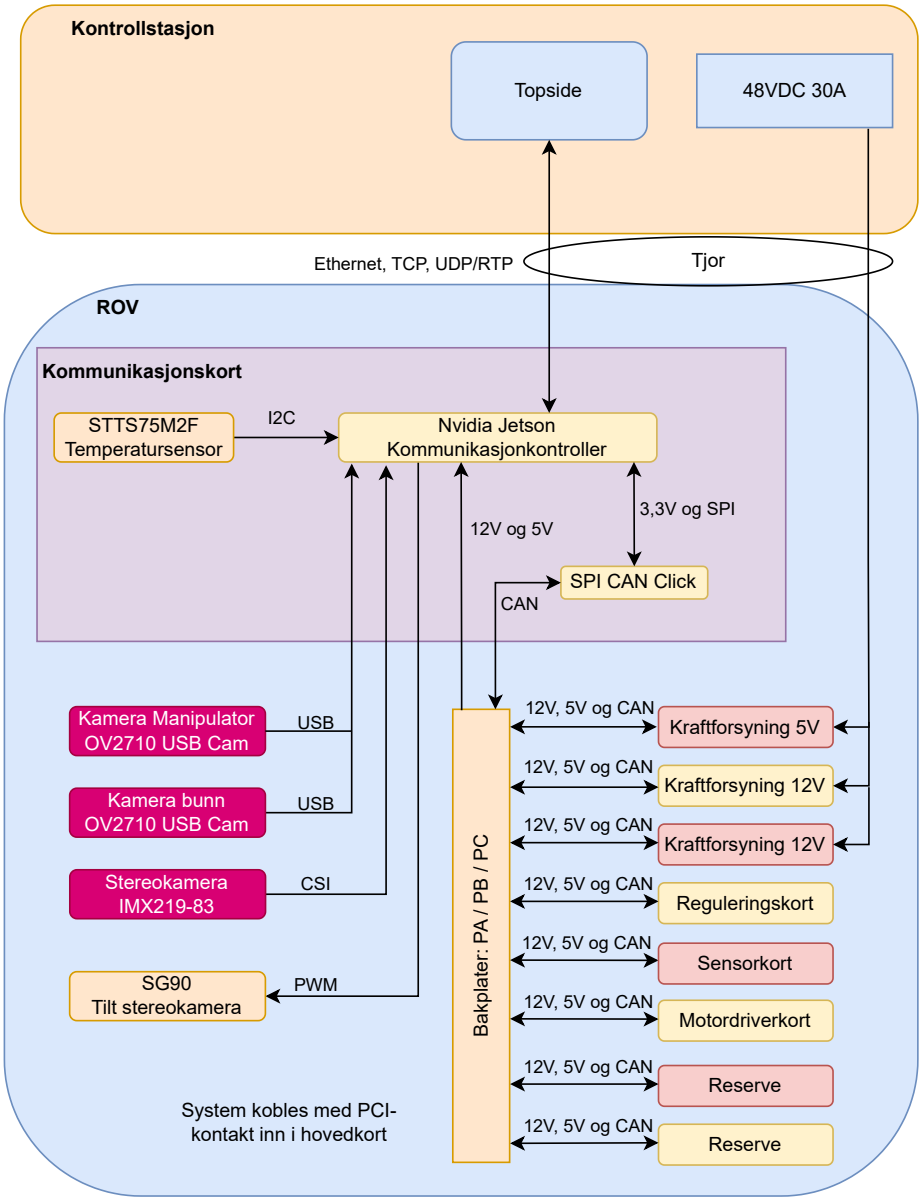
Fra plottene i figur 5.20 ovenfor, kan det ses at USB-kameraet OV2710, vil være i stand til å se et objekt med en diagonal lengde på 63,43cm når objektet er 10cm fra kameraet. Videostrømmen har et bildeformat hvor størrelsesforholdet mellom bredden og høyden på bildet er 16:9. Dette forholdet kan utnyttes kombinert med Pytagoras og rettvinklede trekanter for å finne høyden og bredden på synsfeltet. Ved å sette den ene kateten lik $9x$ og den andre lik $16x$, kan pytagoras brukes for å finne en verdi for "x" slik at hypotenusen blir 63,43cm. Utledningen av dette er gitt nedenfor:

$$\begin{aligned} \sqrt{(16x)^2 + (9x)^2} &= \sqrt{256x^2 + 81x^2} = \sqrt{337x^2} = 63,43 \\ 337x^2 &= 63,43^2 \\ x &= \sqrt{\frac{63,43^2}{337}} = 3,4553 \end{aligned} \tag{5.9}$$

I utregningen ovenfor, er det blitt funnet en verdi for "x" på $x = 3,4553$. Ved å gange denne verdien med 16, kan bredden på synsfeltet finnes, og ved å gange den med 9, kan høyden på synsfeltet finnes. Bredden blir da på 55,28cm, og høyden blir på 31,1cm. Kravet til synsfeltet for USB-kameraet var å kunne se et objekt på 48cm x 27cm (55,1cm diagonalt) når objektet er 10cm fra kameraet. Det er blitt verifisert at dette kravet er oppfylt. Av denne grunn, vil ikke standardlinsen på OV2710 som følger med bli byttet ut.

5.8 Resultat og oppsummering

Blokkskjemaet i figur 5.21 nedenfor er blitt oppdatert med valgene som er blitt tatt i dette kapittelet som omhandler bilde, kamera og linser. Diskusjon rundt hva som fungerte bra, og hva som fungerte dårlige angående kamera og videostrøm - vil bli diskutert videre i kapittel 8.



Figur 5.21: Blokkskjema for ROV og kommunikasjonskort etter kamera er blitt valgt

Fra dette kapittelet som omhandler bilde, kamera og linser, er det blitt valgt tre kamera av to ulike typer for å realisere innhenting av video på ROV. Stereokameraet som ble valgt er av typen IMX219-83[133] og sender video over MIPI CSI-2. Mens de to andre kameraene er av typen OV2710, og sender video over USB.

Målet som var satt når det kom til forsinkelse på videostrømmene, var at forsinkelsen ikke burde vær på mer enn 200ms opp til kontrollstasjonen. Fra testene gjort i testrapport A.3, kommer det frem at alle 4 videostrømmene har en forsinkelse på rundt 100ms. Da bildekvaliteten er god og forsinkelsen er akseptabelt lav, blir dette sett på som et bra resultat.

Etter fem tester, fikk det ene USB-kameraet den laveste gjennomsnittstiden på 82,2ms forsinkelse, mens det andre USB-kameraet fikk den høyeste på 108,4ms. Dette tyder på at selv om bildene i videostrømmen fra USB-kameraene komprimeres før de blir sendt, i motsetning til stereokameraet som sender rådata, får ikke dette noen merkbare konsekvenser på forsinkelsen. I fjor ble det brukt OpenCV for behandling av video, mens i år blir gStreamer (7.3.4.1) brukt for å behandle videostrømmen. Dette tyder på at måten videostrømmen leses av og behandles på relatert til maskinvare og programvare er av mer betydning, enn om kameraene sender råvideo eller komprimert video.

Kapittel 6

Kretskort

Kapitteloversikt

Dette kapittelet omhandler kretskortene som ble laget av vår gruppe for ROV-en. Det vil gås igjennom grunngivelser for valg som ble gjort, ta for seg design og konstruksjon - samt en oppsummering og resultat. I seksjonen *Kretskortdesign* vil det forklares hvilke teoretiske grunnlag som blir vektlagt ved seksjonen *Valg av utforming og komponenter*. I seksjonen *Erfaringer fra tidligere*, ses det på hvilke erfaringer som tidligere grupper har gjort ved lignende prosjekt.

6.1 Problemstilling

I dette kapittelet skal det løses to forskjellige problemstillinger, der de kan deles inn i to kategorier: **Bakplate** og **Kommunikasjonskort**.

Bakplaten sine oppgaver:

- Modulær tilkobling for alle kretskort
- Kobling mellom kort
- Holde kretskort trygt på plass

Kommunikasjonskortet sine oppgaver:

- Drive kommunikasjon mellom kretskortene
- Kommunikasjon mellom ROV og kontrollstasjon

Modulær tilkobling - Alle kretskortene ombord på ROV-en skal knyttes sammen gjennom bakplaten. Formålet med dette er at det skal være mulig å ha flere modulære kort som utfører forskjellige oppgaver. Det må være mulig å fjerne kretskortene og bytte de ut. Dette krever at designet er modulært og bruker en tilkoblingsmetode som ikke er permanent. Den må likevell være driftssikker nok til at det ikke oppstår feil med tilkoblingene.

Kobling mellom kort - Kretskortene skal forsynes og kommunisere med hverandre gjennom bakplaten. Dette skal gjøres gjennom elektriske kontakter og baner i bakplaten. Bakplaten skal ha egne baner for de ulike signalene og driftsspenningene som kortene skal kobles til. Det skal også være mulig å sende digitale signal til et annet kort direkte ved å bruke baner på bakplaten. Siden det er en rekke ulike signal og driftsspenninger som skal gå på tvers av ROV-en, er det nødvendig å ha tilstrekkelig med baner og kontaktpunkt.

Holde kretskort - Den mekaniske styrken i kontaktpunktene der kretskortene kobles til er viktig, da det vil være bevegelse i ROV-en. Den må være sterk nok til å tåle slag og brå bevegelser. Dersom dette ikke er oppfylt, er det fare for at kretskortene kan ta skade og at ROV-en svikter under drift.

Kommunikasjon mellom kretskort - Kretskortene ombord ROV-en skal kommunisere sammen over CAN-buss. I tillegg skal temperaturdata tas inn over I2C. Det er derfor nødvendig at kortet har de komponenter som trengs for at den skal drive denne kommunikasjonen.

Kommunikasjon med kontrollstasjon - Kommunikasjon med kontrollstasjonen for styring og datasending skal foregå gjennom tjoren. Det skal være mulig å koble tjoren til kommunikasjonskortet for å kommunisere med kontrollstasjonen.

6.2 Behovsspesifikasjon

Bakplate

- Skal holde kretskort
- Skal ha mulighet for utvidelse av flere kort
- Skal være modulær slik at de ulike kortene kan flyttes på eller fjernes
- Skal ha baner for overføring av signal
- Bør kunne monteres på vifter

Kommunikasjonskort

- Skal drive en kommunikasjonsprotokoll for kretskort
- Skal drive en kommunikasjonsprotokoll med kontrollstasjon
- Skal ta inn bildedata
- Skal styre tilt-funksjonen til kamera

6.3 Funksjonsspesifikasjon

Bakplate

- Skal ha plugger for tilkobling av kretskort
- Skal være lik tilkobling slik at det kan monteres et kort hvor som helst på platen
- Skal ha plass til minst et kort til for utvidelse
- Skal ha baner fra kraftforsyningskortet ut til alle kretskortene
- Skal ha lavt spenningsfall for kraftbaner
- Bør ha ledige baner for eventuell utvidelse for andre prosjekt
- Skal føre CAN-busskommunikasjon gjennom baner
- Skal ha plugger i enden for å redusere ledninger koblet til kort
- Bør kunne monteres på to 5V-vifter
- Under 35℃ temperaturøke i baner

Kommunikasjonskort

- Skal ha en krets for å drive CAN-buss
- Skal ta inn to USB-kamera

- Skal ta inn et stereokamera gjennom CSI
- Skal ha temperatursensor
- Skal ha en mulighet for sende direkte til kontrollstasjon gjennom Ethernet
- Skal drive PWM-signal ut til servoen for tilting av kamera

6.4 Kretskortdesign

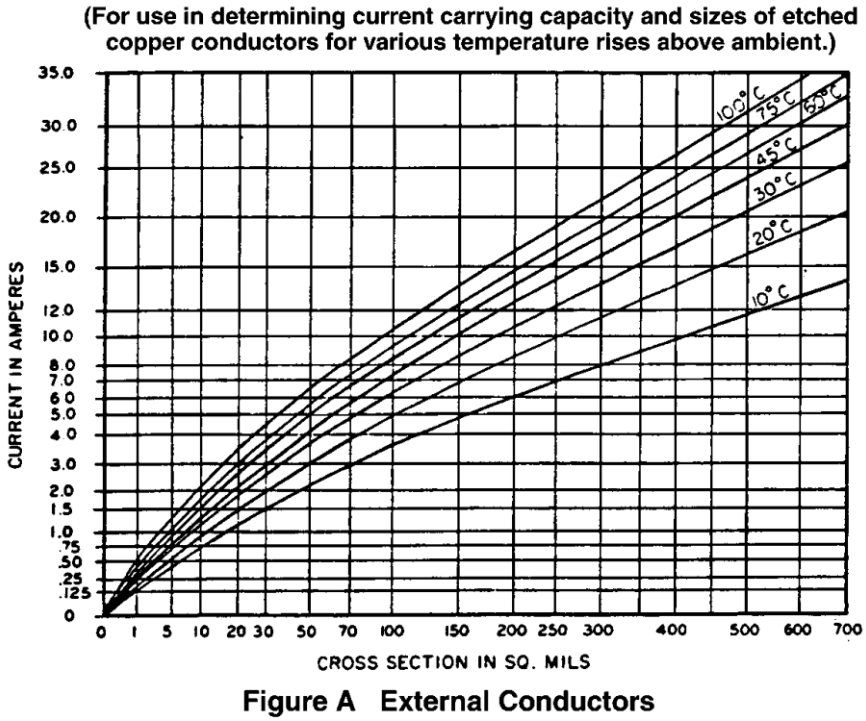
I denne seksjonen utdypes det hva som er lagt vekt på ved bestemmelsen av utforming og metode. Det belyses hvilke faktorer som ligger bak begrunnelsen og hvorfor de er viktige. Hver vurdering vil først bli fremstilt teoretisk, før alle bestemmelser vises i en senere seksjon.

Bakgrunnen til kapittelet er at kretskortdesign ikke var en tidligere kunnskap for denne gruppen. Det er derfor gjort et arbeid ved å sette seg inn i kretskortdesign for å utføre nødvendige oppgaver.

- Banebredde
- Differansialepar
- Lagtykkelse og antall lag
- Jord
- Komponenter
- Overflatebehandling
- PWM

6.4.1 Banebredde

Dimensjonene til banene på kretskortet bestemmer det maksimale strømforbruket som kan tolereres av hver enkelt krets. Ved denne vurderingen er det avgjørende å ta hensyn til temperaturendringen som kan oppstå ved maksimalt strømforbruk. IPC-2221-standarden brukes til å fastsette det maksimale strømforbruket som kan tåles av banene ved ulike temperaturer. Figur 6.1 viser flere linjer som representerer temperaturøkningen som tilsvarer ulike nivåer av strømforbruk i banene.



Figur 6.1: Banebredde for ulike temperaturer tatt fra IPC-2221 standarden [142].

IPC-2152-standarden gir en retningslinje for å velge riktig banebredde basert på strømforbruket i banene. Det finnes også andre metoder for å bestemme banebredden som bruker formler, slik som den som vises i ligning (6.1).

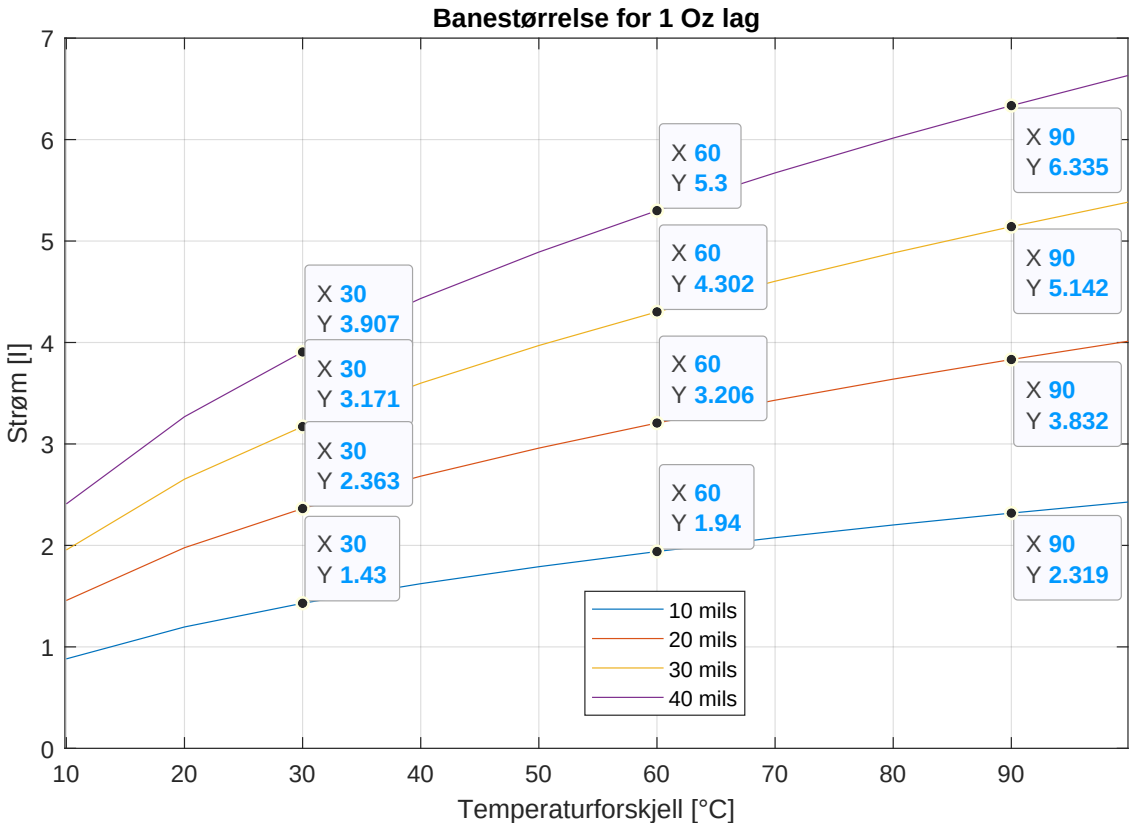
Formel for maksimalt strømtrekk gjennom baner. Tatt fra IPC-2221 [142].

$$I = k \cdot \Delta T^{0.44} \cdot A^{0.725} \tag{6.1}$$

Hvor:

- $I[A]$: Maksimalt strømtrekk
- k : Konstant. Verdi 0.024 for beregninger i indre lag og 0.048 for beregninger i ytre lag
- $\Delta T[℃]$: Temperaturendring i banen
- $A[mils^2]$: Tverrsnittet på banen

For å finne ut tversnittet for banen, må **tykkelsen** på overflatelaget tas med. Det kan variere hvilken tykkelse leverandører tilbyr. En vanlig tykkelse er *0.035mm*, som ofte blir oppgitt som *1 Oz* kobberlag. Tallet oppgir hvor mange "unser" med kopper som blir fordelt på en kvadratfot. I figur 6.2 er det vist hvor mye strøm som kan gå igjennom et 1 Oz lag med banebredder på opp til 40 mils ved ulike temperaturendringer.



Figur 6.2: Banebredde for 1 Oz med ulike banebredder for ytre lag.

Banebredde vil også påvirke spenningsfallet i en bane. I formelen for spenningsfall (6.2), må resistansen være lavest mulig for å få minst mulig spenningsfall.

$$\Delta U = R_{bane} \cdot I \tag{6.2}$$

Se formel for resistans i ledene material (6.3). Redusert lengde og økt tversnitt vil gi mindre resistans i baner.

$$R_{bane} = \rho \cdot \frac{2 \cdot l}{A} \tag{6.3}$$

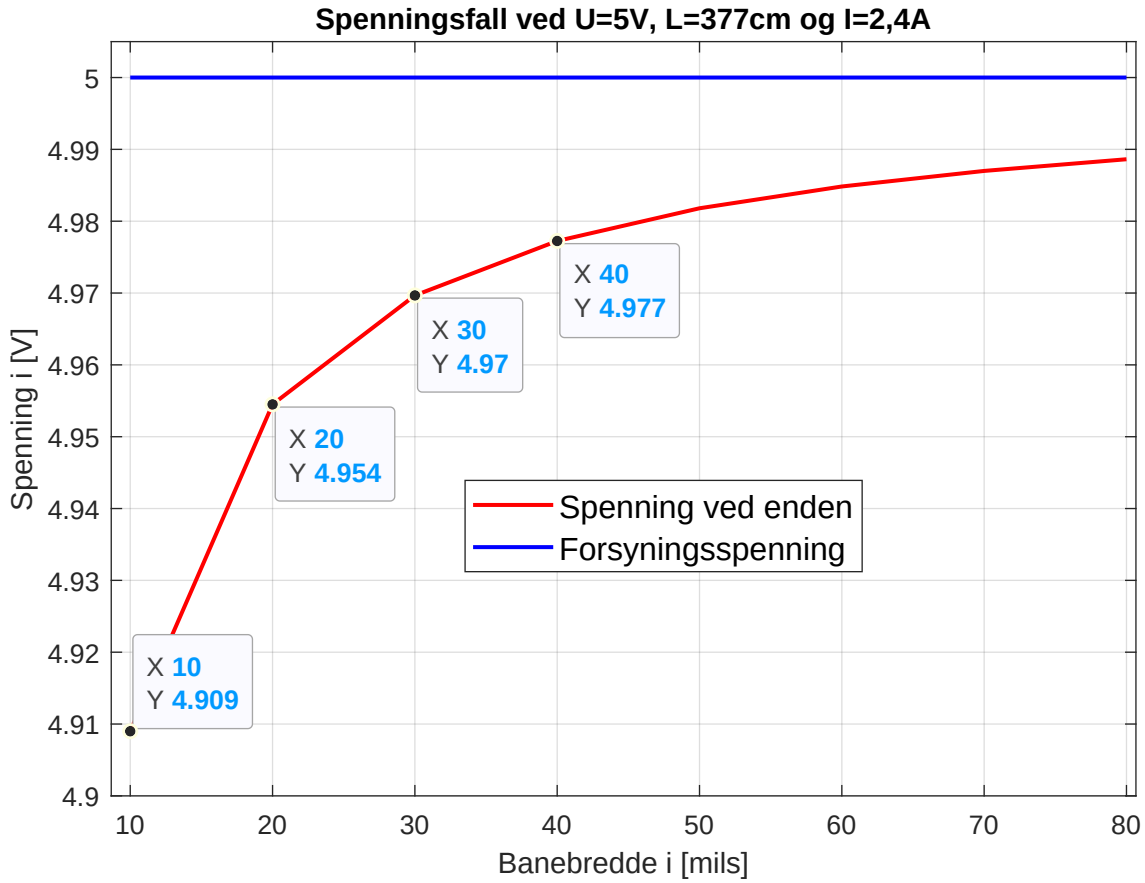
Hvor:

- ρ : Resistivitet til kopper
- l : Lengde på bane
- $A[mils^2]$: Tverrsnittet på banen

Satt ilag til en formel kan denne brukes for å finne spenningsfall ved ulike banebredder (6.4).

$$\Delta U = I \cdot (\rho \cdot \frac{2 \cdot l}{A}) \tag{6.4}$$

På figur 6.3, er det vist hva spenningsfallet vil bli med ulik banestørrelse på et *1 Oz* lag. Det er brukt tall fra fjorårets ROV for utregninger. Der lengden på elektronikkortene er 37,7cm og høyeste strømtrekk på en 5V bane er 2,4A.



Figur 6.3: Spenningsfall ved en 37,7cm bane med 5V ved ulike banestørrelser (*1 Oz*)

De angitte verdiene i grafen brukes som referanseverdier og gir innsikt i hvordan kurven oppfører seg når banebredden øker. Grafen viser at en liten økning i banestørrelse vil føre til betydelig reduksjon i spenningsfall i enden av banen. Dette skyldes at arealet av banetverrsnittet øker eksponentielt, og dermed vil spenningsfallet avta eksponentielt.

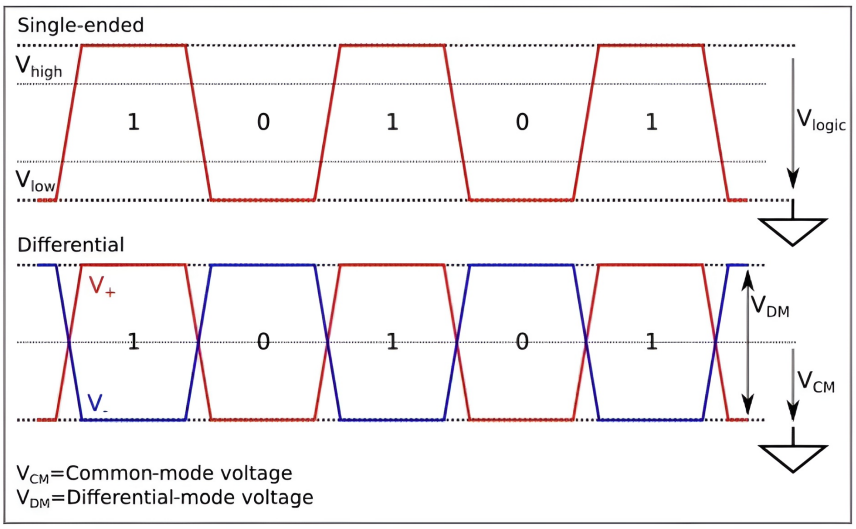
Selv med en banebredde på *10 mils* vil imidlertid spenningsfallet være ubetydelig for de fleste komponenter. Det kan likevel være viktig å være klar over dette for å unngå problemer når det gjelder spenningsfall, spesielt når det gjelder sensitive eller høyytelses-applikasjoner.

For å beregne spenningsfallet mer nøyaktig, kan man også ta hensyn til faktorer som temperaturøkning og termisk motstand, som kan påvirke spenningsfallet ytterligere.

6.4.2 Differensialepar

Litt info om denne seksjonen er tatt fra kilder [143], [144], [145], [146].

Fellesjordoverføring, er en måte å sende et signal ved å bruke et logisk høyt og et logisk lavt spenningsnivå. På denne måten vil det kunne sendes signal over banen. Overføring ved bruk av differensialpar, vil ha to signal som er den inverterte av hverandre. Der differansen mellom spenningsnivået til signalene vil gi et logisk nivå. En illustrasjon av dette vises på figur 6.4.



Figur 6.4: Fellesjord og differensialpar-overføring [143]

Figuren viser at der fellesjordoverføring har en V_{High} og en V_{Low} som blir målt i forhold til jordreferansen. Til motsetning er differensialpar definert av en V_{DM} som viser spenningsdifferansen mellom parene.

En av fordelene ved å bruke differensialpar, er at ved EMI vil det legges til på det inverterte og ikke-inverterte signalet. Spenningsdifferansen mellom de to signalene vil derfor være den samme. I motsetning, kan spenningsreferansen ved fellesjordoverføring forskyves nok til at V_{OL} eller V_{OH} ikke blir aktivert. Typisk brukes det høyere spenning for å unngå dette problemet for fellesjordoverføring, da ved høyere spenninger må signalet påvirkes av mer EMI for å forskyve spenningsreferansen forbi grensene for det logiske nivået. Differensialpar kan i motsetning holde et lavere spenningsnivå uten å bli like påvirket.

Differensialpar kan også bruke redundans på signalet. Slik at ved brudd i en av signalbanene, vil den andre signalbanen sitt logiske nivå brukes for overføringen. CAN-buss som ble valgt i kapittel om kommunikasjon 3, benyttter redundans på signalparet.

For høgfrekvente signal er det viktig å redusere refleksjonen av signalet. Refleksjon av signal kan svekke styrken av signalet. Refleksjonen av signalet er bestemt av den karakteristiske impedansen og lastimpedansen i enden av overføringslinjen. Formelen for refleksjonskoeffiseinten kan vises på ligning (6.5).

$$\Gamma = \frac{Z_L - Z_0}{Z_L + Z_0} \tag{6.5}$$

- Γ : Refleksjonskoeffisient
- Z_L : Lastimpedans
- Z_0 : Karakteristisk impedans

Den karakteristiske impedansen kan variere avhengig av geometri eller ytre påvirkninger, og derfor kan det være nødvendig å ta i bruk metoder for å redusere denne variasjonen. En effektiv metode er å rute banene med samme lengde og tversnitt.

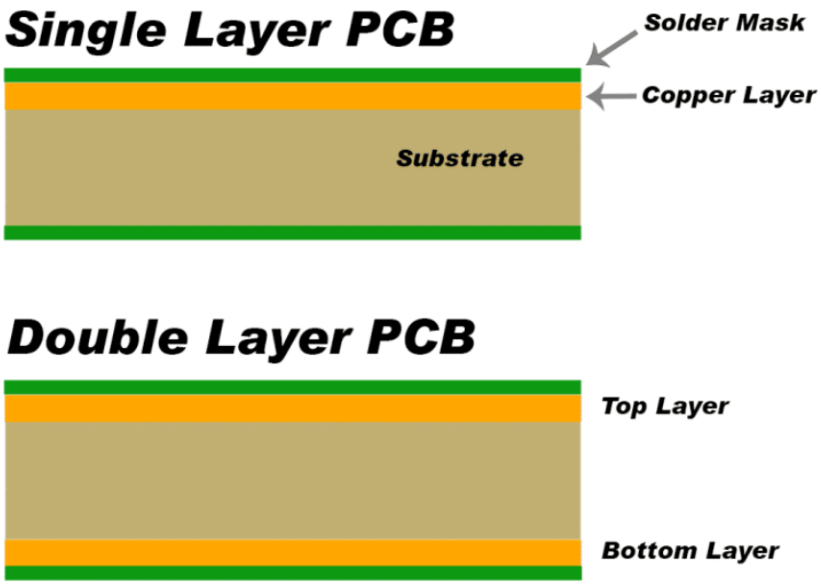
Hvis den karakteristiske impedansen er lik impedansen i lastmotstanden, vil refleksjonene bli absorbert av motstanden og forhindres fra å forårsake skadelige signalforstyrrelser. Fra formelen om refleksjonskoeffisienten i ligning 6.5, ser man at refleksjonskoeffisienten er null når den karakteristiske impedansen er lik lastmotstanden.[147].

6.4.3 Lagtykkelse og antall

Lagtykkelsen i et kretskort spiller en viktig rolle i å oppnå ønskede egenskaper, inkludert mekanisk styrke, signalintegritet, varmeavledning og kostnad. Tykkere lag gir høyere mekanisk styrke, noe som resulterer i mindre sannsynlighet for fysiske skader på kortet. Videre kan tykkere lag også redusere innkobling av støy fra andre signaler og forbedre varmeavledning.

Imidlertid er valget av lagtykkelse avhengig av kortets behov. Kretskort med høy effekt eller høy grad av støy krever generelt tykkere lag for å opprettholde ønsket signalintegritet og varmeavledning. Vanlige måleenheter for lagtykkelse inkluderer "unser per kvadratfot"og "oz", med varianter på *1 oz*, *0,5 oz* og *2 oz*.

Kretskort med høy effekt kan også kreve flere lag for å oppnå ønsket ledningsevne. Dette muliggjør bruk av flere signal- og jordlag for å opprettholde signalintegritet og varmeavledning. Generelt har mer avanserte kretskort mer enn fire lag, mens enkle kretskort vanligvis har fire lag eller mindre. Mer enn fire lag vil øke produksjonskostnadene betydelig. Figur 6.5 viser eksempler på en- og to-lags kretskort.

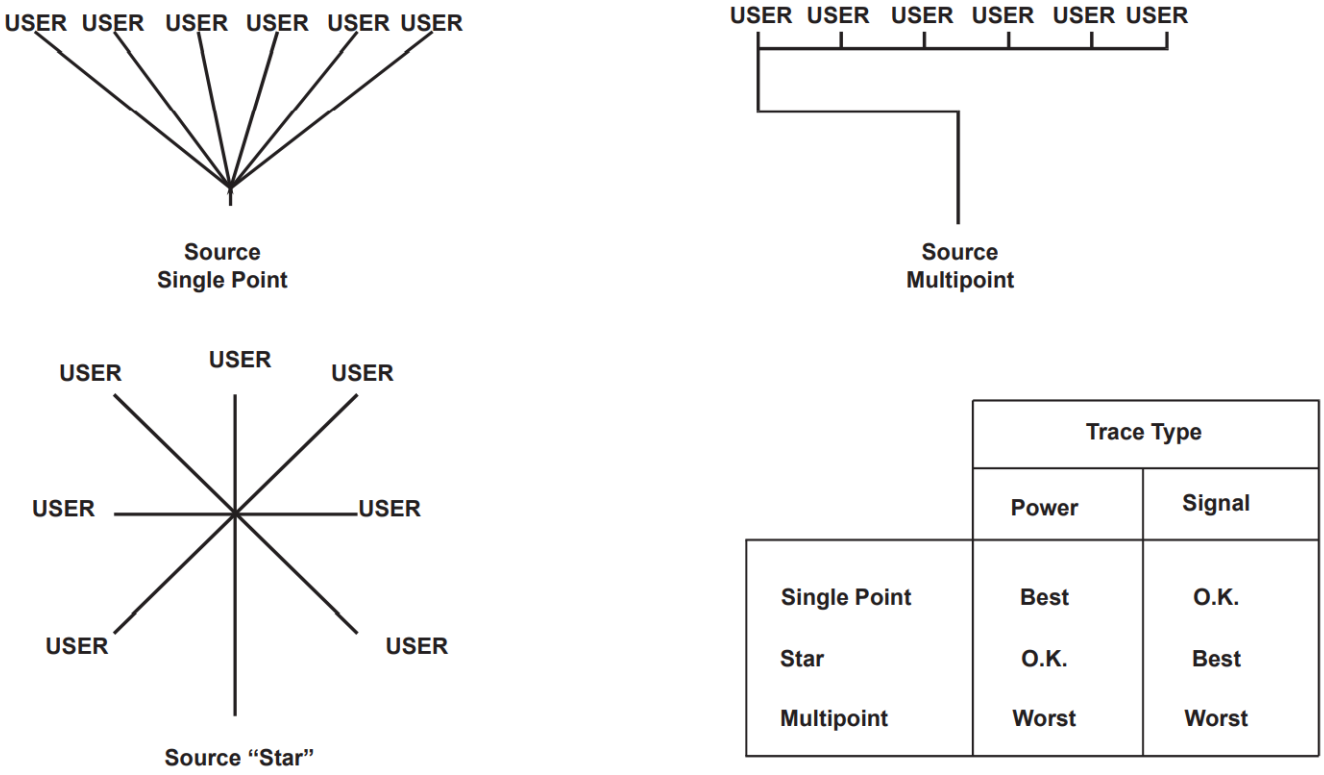


Figur 6.5: Enkelt eller dobbeltlag [148].

6.4.4 Jord

I PCB-design er det viktig å ha en god jordforbindelse for kretskortet. Det er flere måter å koble jord på i et elektronikksystem, og det er derfor nødvendig å tenke på hvordan denne skal implementeres for det aktuelle behovet. Noe som må unngås er jordsløyfer. En jordsløyfe vil oppstå om det er en spenningsdifferanse mellom to punkt som er koblet til samme jordreferanse[149]. For å redusere dette

er det viktig å tenke på hvordan jordpunkt skal rutest til jordreferansen. Figuren 6.6 viser ulike anbefalte topologier for tilkobling av jordplan.



	Trace Type	
	Power	Signal
Single Point	Best	O.K.
Star	O.K.	Best
Multipoint	Worst	Worst

Figur 6.6: Enkelt punkt, flerpunkt og stjernekobling [150].

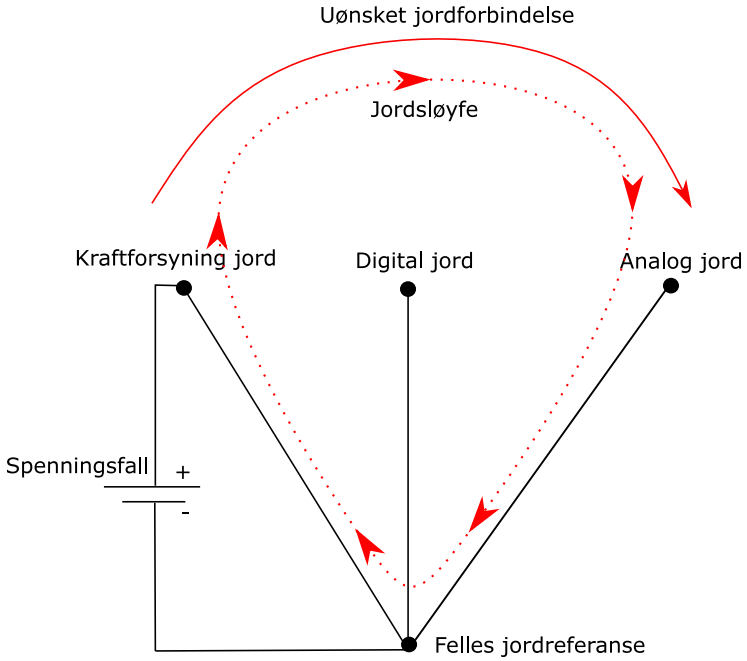
Stjernekobling:	Jord kobles i en stjerneformasjon til et felles punkt som er jordreferansen.

Enkelt punkt:	Likt som stjerne går det fra et felles punkt, men rutes i ulike retning kun på ene siden.

Flerpunkts:	Seriekobling av jordpunkt, som til slutt ender i en jordreferanse.

En annen metode for å koble jord er å koble til et felles **jordplan**. Et felles jordplan vil strekke seg over hele kretskortet, og bruke en direkte kobling til jord ved hver komponent. Ved å bruke et slikt plan vil man redusere plassen for individuelle baner som kan rutes. Et slikt plan vil likevel være meget effektivt for å redusere uønskede jordsløyfer, da den lager en direkte kobling til en felles jordreferanse rett ved lasten. Siden jordplanet danner en kobling med lavere resistans enn de andre tilkoblingsmetodene, vil det potensialet som kan dannes med elektromagnetisk induserte strømmer være mindre.

En jordsløyfe kan normalt oppstå om det er et spenningsfall over en last og dens jordreferanse[151]. Ved høyere returstrømmer vil dette spenningsfallet være større. Dersom lasten da har en uønsket forbindelse til en annen last sin jordtilkobling, vil noe av returstrømmene gå denne veien og lage EMI i kretsen til den andre lasten. Denne jordsløyfen er illustrert på figur 6.7.



Figur 6.7: Stjernekobling med uønsket forbindelse.

Figuren viser et spenningsfall mellom jord på kraftforsyning, og en ønsket kobling mellom dette punktet og et punkt for et analogt signal sin jord. Returstrømmer vil da gå fra det ene til det andre punktet gjennom den felles jordreferansen. Dette kan spesielt være problematisk for analoge signaler, da de er mer følsomme for støy enn digitale signaler. Digitale signaler er vanligvis mer robuste mot kryssforbindelser, da det kreves betydelig mer støy før signalets verdi endres.

Tilkoblingen til kraftforsyningen er like viktig for returstrømmer, da potensial mellom ulike punkt vil lage en lik sløyfe. De samme prinsipp gjelder derfor også for ruting av denne.

6.4.5 Komponenter

Det er to typer monteringsmetoder av komponenter å velge mellom i PCB-design:

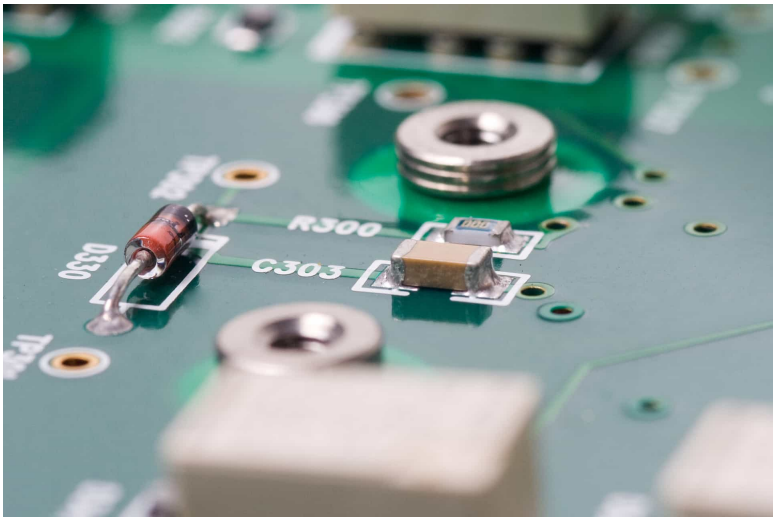
- **Overflatemontering** - *Surface-mount technology*[1]
- **Hullmontering** - *Through-hole technology*

Overflatemonterte komponenter er komponenter som blir montert kun på den ene siden av kretskortet. Denne vil være isolert på den ene siden og kan monteres uavhengig av hva som er på andre siden. Hullmonterte komponenter er montert med pinner som går igjennom kretskortet. De tillater kobling på begge sider, og gjør det lettere dersom du skal nytte signalet på begge sider av kretskortet. Det gjør det også lettere å koble komponenten til et lag som er mellom topp- og bunnlaget. På figur 6.8 er det vist en overflatemontert og en hullmontert komponent.

Komponenter har også ulikt *fotavtrykk*. Fotavtrykk refererer til størrelsen på en overflatemontert komponent, og det kan variere fra komponent til komponent, selv om funksjonen deres kan være den samme. Størrelsen på fotavtrykket påvirker komponentens overflateareal, noe som igjen påvirker varmefordelingen. Komponenter har en angitt effektbegrensning som er bestemt av størrelse og andre faktorer, og

[1] Ofte refert til som SMT

hvis effekten som går gjennom komponenten overskrider denne begrensningen, kan varmen ødelegge komponenten. Derfor er det viktig å velge riktig størrelse og type komponenter for å sikre at de tåler den forventede belastningen og ikke blir ødelagt av overoppheting.



Figur 6.8: Overflatemontert og hullmontert [152].

Overflatemonterte komponenter er generelt rimeligere å montere enn hullmonterte komponenter. Dette skyldes at hullmonterte komponenter ofte må håndloddes, noe som kan føre til metallurgiske feil[153]. Overflatemonterte komponenter kan også være sårbare for feil hvis de håndloddes, men det er vanlig å bruke *Reflow soldering*[2], som gir pålitelig og god metallurgisk lodding. Til tross for dette har hullmonterte komponenter bedre mekanisk styrke. Selv om overflatemonterte komponenter har bedre metallurgisk kobling enn håndloddede hullmonterte komponenter, vil limet som holder kobberlaget på plass være det svakeste leddet i overflatemonterte komponenter.

6.4.6 Overflatebehandling

Fra prosjektet er det bestemt at kort skal kjøpes gjennom leverandøren ***JLCPCB***. Dette er en rimelig produsent som er blitt brukt tidligere år og som har levert bra kvalitet i resultatet.

De har to alternativ for overflatebehandling:

HASL : *Hot Air Solder Leveling* er en rimelig overflatebehandling som bruker loddetinn

ENIG : *Electroless Nickel Immersion Gold* Mer kostbar overflatebehandling som er gullbelagt

Informasjon for tabell tatt fra kilde [155].

Gullfingre får man bare ved å bruke overflatebehandlingen ENIG. Likevel er gullfingre bare en type *kantkontakt*[3], som også kan lages med tinn. Siden gull har mye bedre kontakt enn tinn, vil ENIG være bedre enn HASL. Likevel er det fullt mulig å bruke tinnbelagt overflate dersom det er lavere krav til driftsikkerhet, levetid, og kontaktevne. Det er likevel viktig at det er samme metall mellom to kontakter, slik at det ikke oppstår elektrisk potensial mellom metallene[156].

[2]Reflow soldering er en prosess som bruker loddepasta for å feste komponenter som deretter blir behandlet med varme [154]

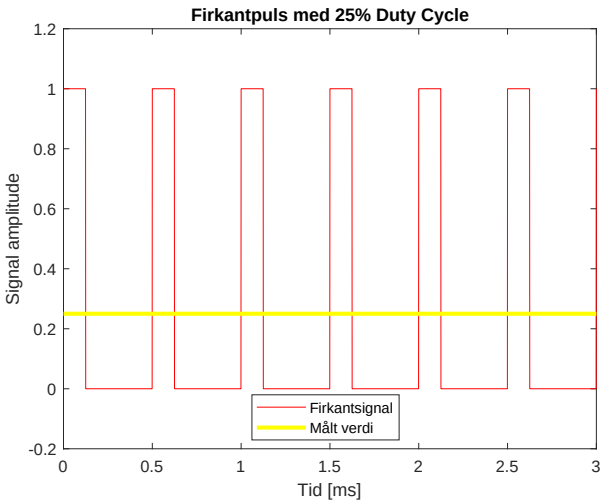
[3]Kontakt som bruker enden av et kretskort som kontaktfeste

	HASL	ENIG
Oppfyller RoHs krav	Nei	Ja
Flat overflate	Nei	Ja
Aliminium Wire Bond	Nei	Ja
Loddbarhet	Best	God
Holdbarhet	Over et år	Over et år
Kost	Lav	Høy
Popularitet	Høy	Høy

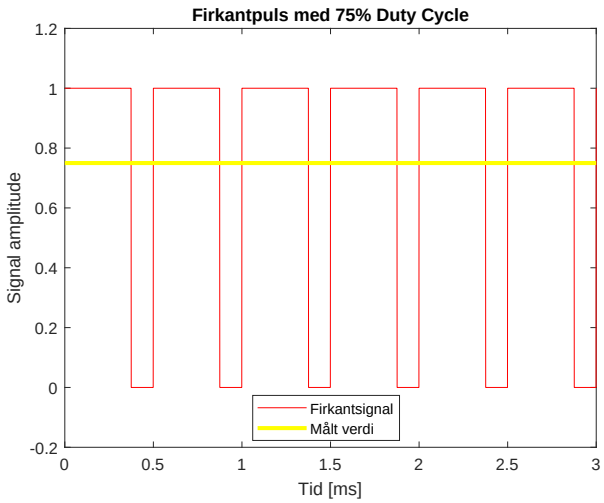
6.4.7 PWM

Pulse-width modulation[157] er en enkel metode å sende et digitalt styresignal. I motsetning til analoge signal der signalet endres linært i forhold til verdien, brukes det breddemodulering av digitale pulser for å lage et analogt signal. Pulsene har logisk verdi *1* eller *0*.

Et eksempel på PWM-signal vises på figurene 6.9a og 6.9b. Her er det brukt to ulike "*Duty Cycle*". Dette begrepet betyr hvor lenge signalet er høyt i en gitt syklus. For eksempelene med 25 og 75 prosent på-tid, vil verdien av signalet som sendes også bli henholdsvis 25 og 75 prosent.



(a) 25% duty cycle



(b) 75% duty cycle

Verdien av signalet vil være gjennomsnittet av verdien som blir sendt. I formel (6.6) er det vist en formel som regner verdien av et pulsbredde-modulært signal.

$$\bar{y} = \frac{1}{T}\int_0^T f(t)dt \tag{6.6}$$

Hvor:

- $\bar{y}$: Verdi
- T : Tidsperiode
- $f(t)$: Pulsbredde-modulært signal

Tidsperioden er ikke nødvendigvis den samme som syklusen til signalet. På denne måten er det mulig å luke ut ulike feil ved kun flatt signal eller kun høyt signal.

6.5 Erfaringer fra tidligere

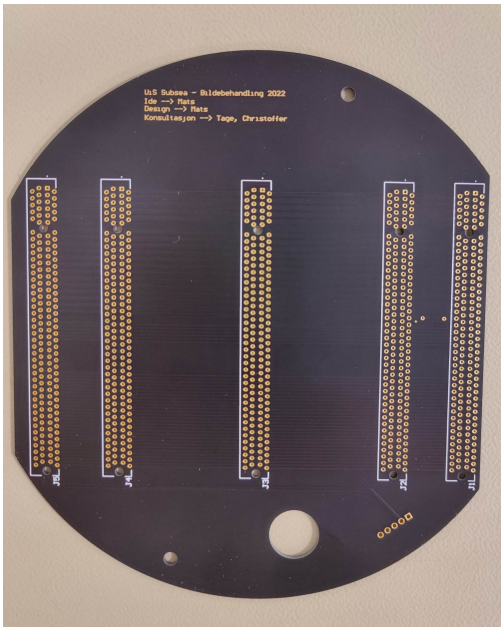
Her vil det bli sett på erfaringer som tidligere grupper har rapportert fra tidligere år. Dette tas med for å få lærdom fra tidligere feil og for å videreføre valg som fungerte. Hele denne seksjonen er basert på rapporten med lignende oppgave i fjor [31].

6.5.1 Design av bakplate

Designet i fjor baserte seg på en bakplate som alle kretskort ble montert i og som så skulle distribuere signal og kraft mellom kortene. Dette var en måte å få et modulært design som muliggjorde at det kunne utvides og endres. Det er også lagt til flere kontakter som er ledige for å sette på ekstra kort.

Det er brukt et liggende design som strekker seg over hele ROV-en. Dette gjør at noen kort ble større enn de trengte å være, og at noen kort ble kombinert til ett. Dette kan gjøre det vanskeligere å få et modulært design, da kortet må erstatte to oppgaver istedenfor en.

Som vist i figur 6.10 er avstanden mellom pluggene forskjellig. Dette er for å gi plass til kort som har større komponenter som bygger mer ut fra kretskortet. To av pluggene blir likevel veldig nærme hverandre og det er vanskelig å få plass til flere kort selv om det er kontakter ledig til de. Dersom designet av bakplate går fra vertikalt til horisontalt, vil det være plass til **flere** kort - selv om de ikke blir **større**.



Figur 6.10: Bakplate brukt i fjorårets design [31].

Banene som er dedikert til kraftforsyningene 5V og 12V er trukket separat. Det er mulig å ha to *skinner* for disse som gir bedre kontakt. Dette ble begrunnet med at det da er lettere å bruke bakplaten til noe annet dersom en trenger det.

Tas med videre:

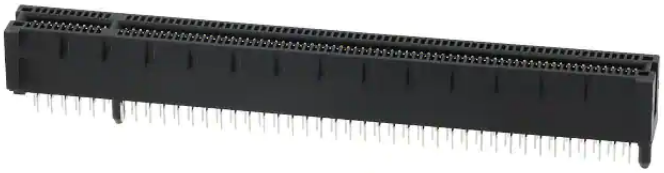
- Modulært design med like tilkoblinger for alle kortene
- Ledige baner for mulig utvidelse
- Flere avstander mellom kortene

Endres på:

- Antall kortholdere økes for å muliggjøre flere ulike mindre kort
- Tilkoblingspunkt med 5V, 12V og jord blir koblet ilag i større kopperlag

6.5.2 Kontakter

Gruppen i fjor brukte *164-Way PCI-kontakter* for å få et modulært design [158]. PCI-kontakter bruker *gullfingre* som kontaktpunkt. Det vil si at det da er en del av kortet som er ledende inn til baner videre på kortet. De passer så ned i en fjærbelastet kontakt som kobler pinnene til baner på et annet kort. Et bilde av denne kontakten er vist på figur 6.11



Figur 6.11: Samtec PCI-kontakt [159]

PCI er en solid måte å feste kretskort uten å ha loddede plugger montert på kretskortet. Ved at selve kretskortet som blir plassert i kontakten, reduseres antall ledd som kan svikte i forhold til en loddet kontakt. Dette er også gunstig da det reduserer behovet for plugger på kretskortet som må loddes.

Designet fra i fjor er mekanisk sterkt, og holder kretskortene tilstrekkelig fast. Det har rikelig med kontaktpunkt for overføring av signal og kraft. Det er også mye ledig som kan brukes en senere gang av andre kort som kan settes inn ved en utvidelse.

Kontakten er likevel relativt stor i størrelse, og har lite naturlig oppdeling av ulike typer signal. PCI-kontaktene finnes også i andre størrelser som har færre pinner.

Tas med videre:

- PCI-kontakter for solid mekanisk feste
- Ledige pinner for mulig utvidelse

Endres på:

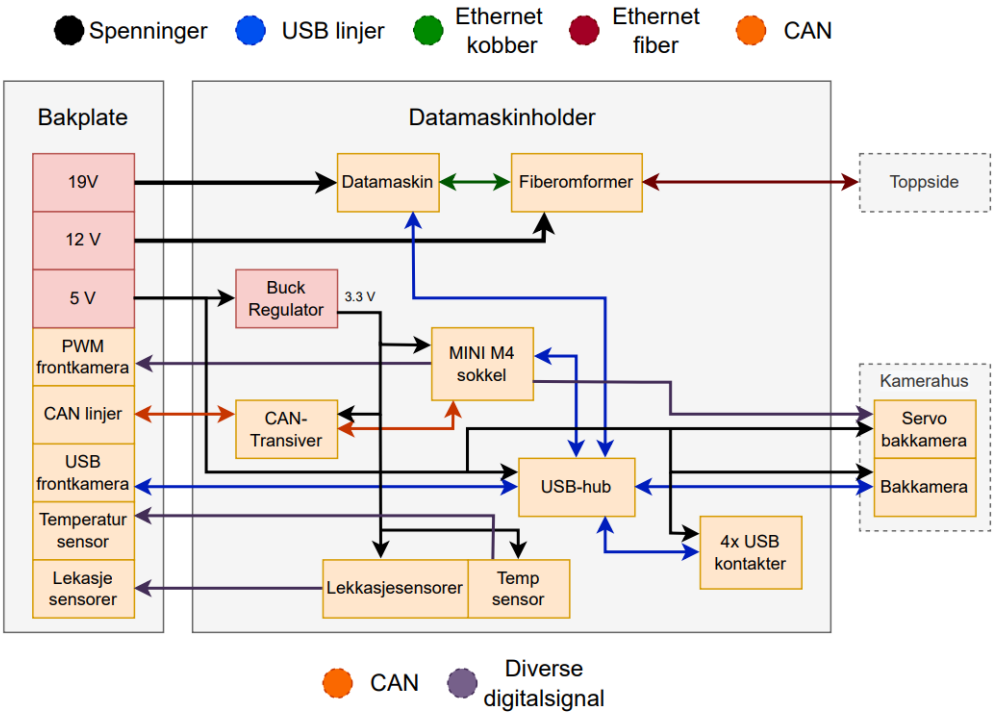
- Størrelse av kontakt
- Flere kontakter for bedre feste

6.5.3 Design av kommunikasjonskort

Fra rapporten i fjor ble denne navngitt *datamaskinholder*, men for å redusere forvirring, blir den referert til som kommunikasjonskort fra nå av.

Dette kortet hadde en mini-PC montert på seg som tok opp store deler av kortet. Denne datamaskinen var for å drive med bildebehandling, noe som ikke er en del av denne oppgaven. I tillegg var det montert en mikrokontroller som ble brukt for å kommunisere på CAN-bussen. Denne kommuniserte så med datamaskinen for å sende data mellom kontrollstasjonen og ROV-en.

Det var også en USB-hub for utvidelse av USB-tilkoblinger til datamaskinen. Denne var montert for å kunne ha mange USB-kamera og programmeringskretser til mikrokontrollere. Et blokkskjema av dette er vist på figur 6.12.



Figur 6.12: Blokkskjema brukt i fjorårets design [31].

Siden det ikke var noe 3.3V på kortet, brukte de også en *Buck Regulator*[4]. Denne ble brukt til å drive CAN-kretsen, lekkasjesensoren og temperatursensoren. Dersom en velger en datamaskin som har 3.3V kraftforsyning ut, trenger en ikke en regulator.

For å sende signal opp gjennom fiber, var det montert en fiberomformer som tok ethernet inn. Fiber er en meget rask overføringsmetode som bruker lys. Denne kan overføre signaler over store avstander med minimal dempning[5]. Det er likevel en relativt dyr teknologi i forhold til alternativet, og over korte avstander har man lite igjen for implementering.

[4] En DC til DC regulator som reduserer spenningen til et lavere nivå. [160]
[5] Tatt fra SNL [161]

Tas med videre:

- Montering av datamaskin på kortet
- Datamaskin med mulighet for ethernet
- CAN-transiver montert på kortet

Endres på:

- 3.3V tilførsel tas fra datamaskin
- Datamaskin driver CAN-buss slik at mikrokontroller droppes
- Fiberomformer monteres ikke
- USB-hub monteres ikke

6.6 Valg av utforming og komponenter

6.6.1 Bakplater

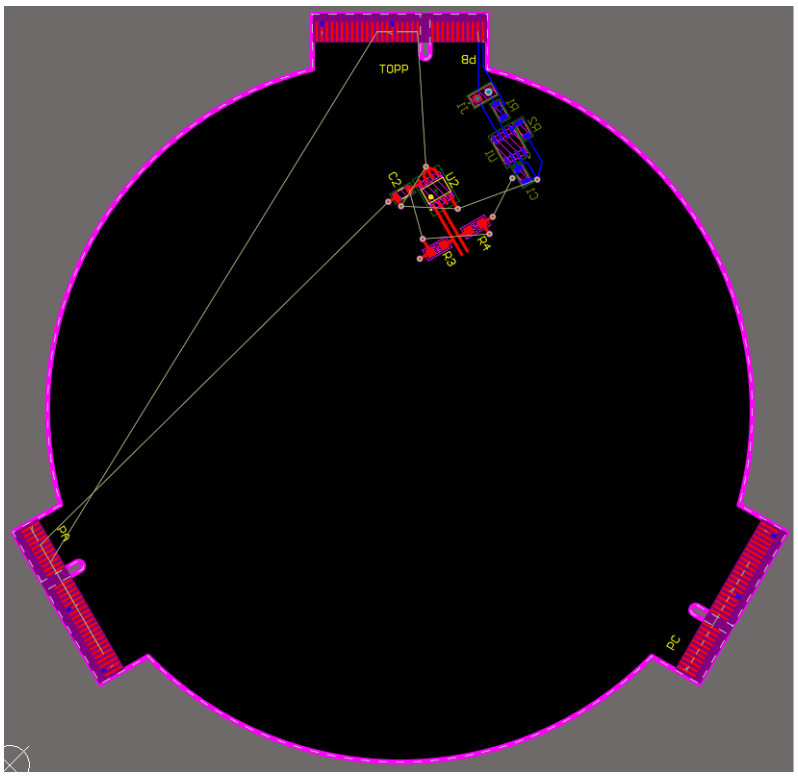
6.6.1.1 Design

Siden kortene skal monteres stående istedenfor liggende, vil en rektangulær form ha lav utnyttingsgrad inne i en sirkulær form. Derfor ble designet laget sirkulært for alle kortene. Bakplatene vil da være liggende, og det er derfor viktig å ha de minst mulig for å få nyttet mest mulig areal til kretskortene. For å ha tilstrekkelig styrke i konstruksjonen, blir det brukt tre bakplater. Dette ble gjort etter konsultasjon med Kristian Thorsen.

Fordelene ved dette designvalget er:

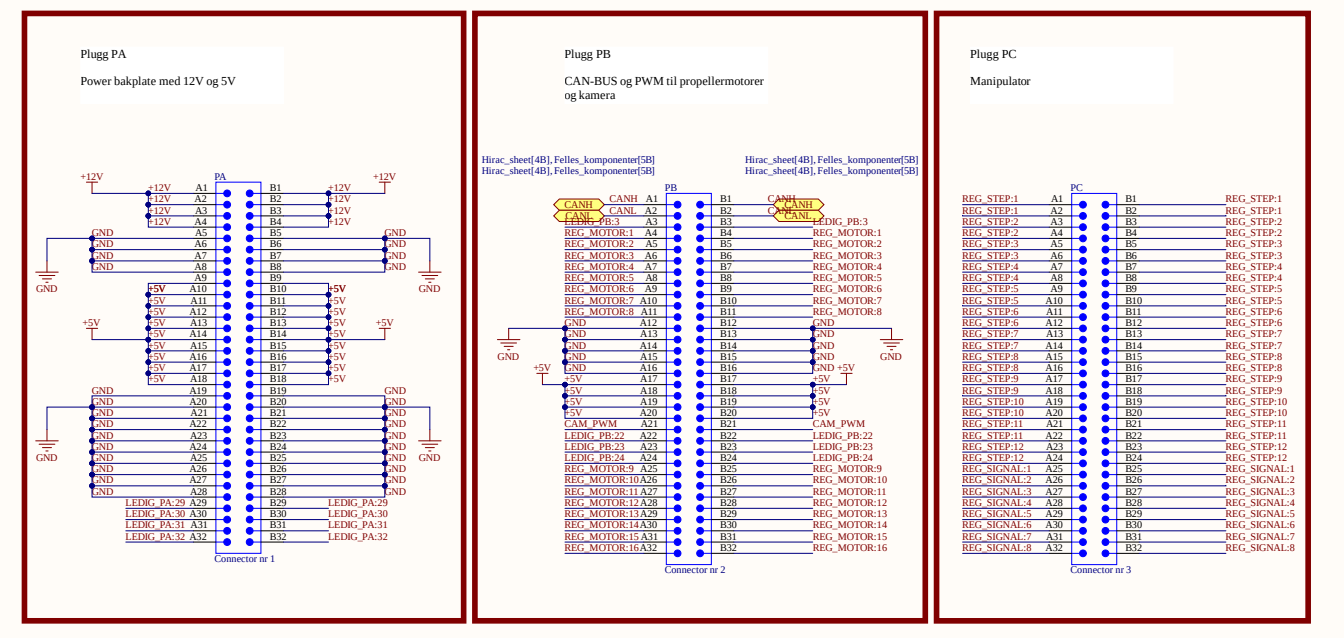
Formasjon	Likesidet trekant er en av de mest stabile formene som er, og den er effektiv på å fordele belastning. Denne vil gi utformingen et robust design som tåler belastning bra. Med tre bakplater vil det også være flere mulige kontaktpunkt.
Funksjoner	Ved å ha ulike funksjoner for hver bakplate, er det lettere å skille signal fra hverandre. Dette vil være støyreduserende samtidig som det gjør det mer oversiktlig.
Koblinger	Ved å føre ned signaler gjennom bakplatene vil det redusere behovet for kabler som går fra kretskortene og gjennom elektronikkhuset. Færre kabler gjør at det blir bedre plass til de kablene som faktisk må gå gjennom elektronikkhuset, og utformingen vil være ryddigere.

Designet av den sirkulære formen på kortene er vist på figur 6.13. Her er det også tegnet inn krets for CAN-buss og temperatursensor på I^2C. Denne ble gitt ut i god tid til grupper for å sikre at formen på kortene ble lik. Den sirkulære delen av kortet har en diameter på **140mm**.



Figur 6.13: Mal av kretskort som var gitt ut til grupper

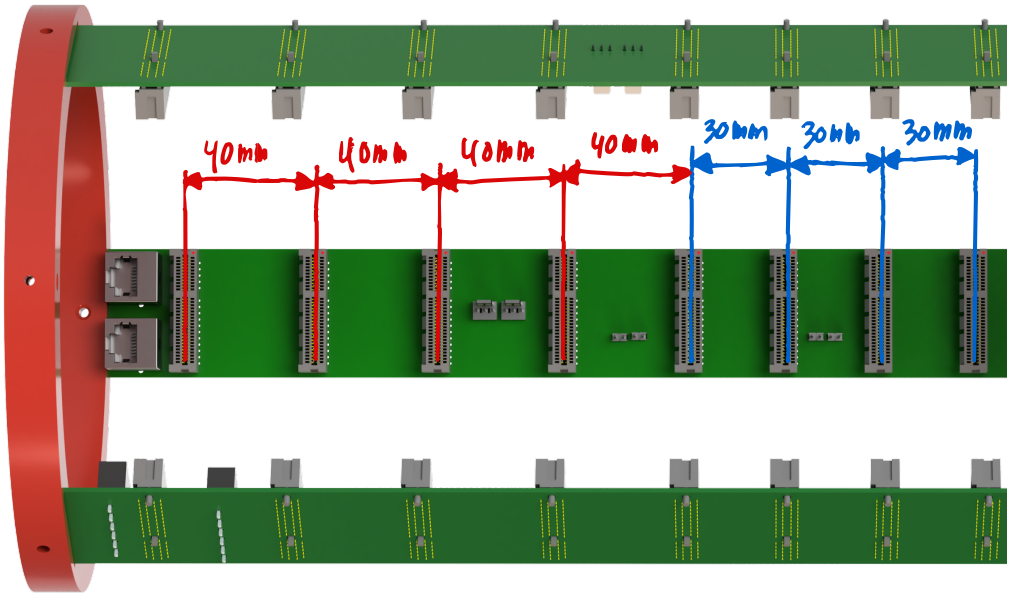
Kortet inneholder også merkinger for hvor de ulike bakplatene skal stå, der de er navngitt *PA, PB og PC*. Gullfingrene er koblet opp mot det samme som PCI-e-kontaktene er, og er tilgjengelige på begge sider av kretskortet. Som det vises i figur 6.14, er det koblet til samme signal på begge sider av kortet sine gullfingre.



Figur 6.14: Tilkoblinger for gullfingre

Som vist hadde de forskjellige bakplatene og gullfinger-koblingene ulike formål. Det er også noe ledig, som er trukket på de ulike bakplatene for å muliggjøre at de kan brukes senere.

Det er lagt opp til to ulike avstander mellom PCI-pluggene for å tilpasse for ulike høyde på kretskortene. For de fire første plassene er det lagt opp til **40mm** mellom senter til senter på hver plugg. For de fem neste plassene er det lagt opp **30mm** mellom senter til senter på hver plugg. De første plassene med mer avstand er for kort som enten har stor varmgang, eller som har behov for komponenter som bygger mye i høyden. Se figur 6.15.



Figur 6.15: Avstand mellom PCI-kontakter

6.6.1.2 Banebredde

Bakplatene har PCI-kontakter montert. Kontaktene har en maksbegrensing på **2,4 A** per pinnepar. Siden 5V skal forsynes gjennom bakplatene må det dedikeres flere pinnepar til dette. Kraftforsyningen som kommer for 5V er på 100W, og er ikke regulert ned for å gi mindre effekt. Det dimensjoneres derfor for at det trygt skal kunne trekkes full effekt ut av tilkoblingene. Utregningen er vist på ligning (6.7). Et pinnepar er på bakplatene koblet gjennomgående mellom kontakten. Det er på denne måten fordelt på pinneparet.

$$
\begin{aligned}
Antall\ par &= \frac{I_{totalt}}{I_{maks}} \\
Antall\ par &= \frac{\frac{100W}{5V}}{2,4A} \\
Antall\ par &= 8.3\ \approx \mathbf{9\ pinnepar}
\end{aligned}
\tag{6.7}
$$

I figur 6.14 viser tilkoblingene til PA. På denne bakplaten er det satt av 9 stk pinnepar for 5V. Det er lagt opp til at denne skal forsyne all 5V ut til kretskort. Det er også lagt inn 4 stk pinnepar for 12V, som er til generelt bruk av 12V for laveffekt-elektronikk. Motorene som skal drives på 12V er koblet direkte med egen kabel og går ikke gjennom bakplatene.

Både 12V, 5V og jord er koblet på pinner som har kontakt i større overflater, kalt "polygoner" i Altium[6]. Dette gjør at det er en bedre overflate som bidrar til bedre ledningsevne enn med enkeltbaner. Ulempen med dette, er at det er mindre mulighet for fremtidige grupper å bruke denne platen til noe annet videre. Det antas likevel at framtidige grupper ønsker å ha to strømførende baner for forsyning til kort, og at det kjøres tilbake i en felles jord.

[6]Programmet brukt for kretskortdesign.

Platene PB og PC er derimot laget med egne baner for alt annet enn felles jord. Dette er av to grunner:

Stjernejord	Som nevnt i seksjonen om kretskortdesign 6.4 er det bedre å ha en stjernejord enn det er å ha en enkelt punkt. Ved å bruke flere bakplater får man en stjernejord med mer enn et enkelt punkt. Samlet vil også tilkoblingen generelt til jord bli bedre.
Modulært	Baner som ikke er koblet sammen, gir bedre muligheter for å utvide bruken av bakplatene for framtidige prosjekt.

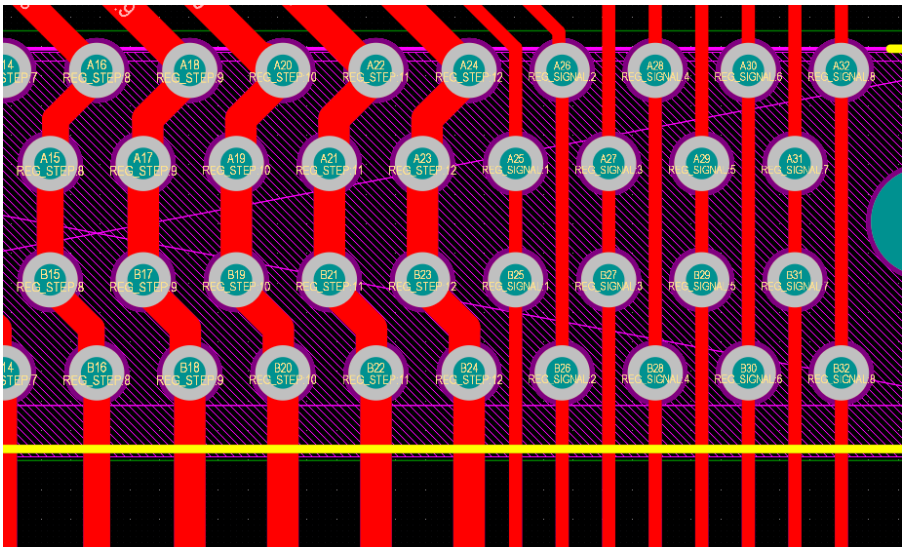
Det er også lagt til målepunkt på kretskortene for de ulike banene. Dette muliggjør enkel måling på banene selv etter montering av kretskort i bakplater.

Siden PCI-kontaktene sin begrensning per pinnepar er **2,4 A**, vil det ikke være mulig å trekke mer enn dette ved et enkelt signal. Likevel vil noen signal trekke mer enn dette, noe som gjør at det må settes opp flere pinnepar. Avstanden mellom pinnene er for liten for at en bane kan trekkes med nødvendig dimensjoner. Som man visuelt kan se ut fra grafen om banestørrelse ut fra et *1 Oz* kort (figur 6.2), vil man trenge 30 mils bane for å holde oss på en temperaturøke lavere enn de 35℃ som ble spesifisert i funksjonsspesifikkasjonen. Se utregning for ønsket banestørrelse ved 35 ℃ temperaturøke på ligning (6.8).

$$
\begin{aligned}
A &= \left(\frac{I}{k \cdot \Delta T^{0,44}}\right)^{1,39} \\
A &= \left(\frac{2,4A}{0,048 \cdot 35^{0,44\text{℃}}}\right)^{1,39} \\
A &= 35,8\ mils^2
\end{aligned}
$$

$$
\text{A}[mils] \text{ justert for 1 Oz} = 26{,}7 \text{ mils} \tag{6.8}
$$

Mellom PCI-kontaktene sine pinner er det ikke plass til baner som er bredere enn 12 mils. Alternativet til dette er å koble to pinnepar ilag for å få plass til en bredere bane. Banen kan også rutes på både oppside og nedside av kretskortet. I figur 6.16 vises det hvordan flere baner kobles ilag for å få plass til bredere baner.



Figur 6.16: Brede og smale baner for et eller to pinnepar, her tatt fra bakplate PC

Ved å bruke brede baner, reduseres antall ulike signal som kan sendes gjennon en bakplate. Dette merkes fort der signal som skal utenfor elektronikkhuset sendes i baner gjennom bakplaten. De andre banene som er 12 mils, har en mindre begrensning på strømtrekk før de går over ønsket temperatur. Alle er rutet på begge sider av kortet, noe som gjør at det samlet kan trekke omtrent 2 A ved 20°C. For de fleste signal som trekker under 100 mA, vil dette være langt innenfor kravet.

Jordlaget er koblet over hele platen som et dekkende lag på både top- og bunnlaget. Både for å redusere EMI[7], men også for å få et større tverrsnitt for jordtilkoblingen.

Det er da mulig å regne hvor mye spenningsfall det vil bli over banene på platen. Dersom man tar utgangspunkt i den lengste banen, er det **295mm** fra ende til ende. Siden maks strømtrekk til PCI-kontakten sine pinner er 2,4A, brukes dette som verdi for utregningene. Tabellen nedenfor viser tapet fra 5V på maksimal lengde i bakplaten:

Banebredde [mils]	**Spenning ved baneslutt[V]**	**Tapsprosent [%]**
12	4,970	0,61
24	4,985	0,30
55	4,993	0,13

Banebreddene er valgt for de tre ulike størrelsene: **12, 24 og 55 mils**. Dette er fordi alle baner er rutet på **hver side** av bakplaten, med minst *12 mils*. Unntaket er kraftbanene, og baner til manipulator som er rutet med *26,5 mils* på hver side. Ut fra tabellen ser man at dersom det ble rutet med bare *12 mils* på den ene siden, ville det også vært et akseptabelt spenningsfall. Det faktiske spenningsfallet til kraftforsyningen vil være større, da det vil være motstand i kontaktflatene mellom banene på kretskortet.

Siden alle baner er trukket på begge sider av kretskortet, vil alle være minst *24 mils* i effektivt tverrsnitt.

[7] Elektromagnetisk forstyrrelse

Den mest kraftkrevende komponenten i elektronikkhuset er Jetson Nano, denne har et makstrekk på 3A og en minimum VDD_{IN} på 4,75V. Dette vil si at det er en god sikkerhetsfaktor på banebredden.

6.6.1.3 Komponenter

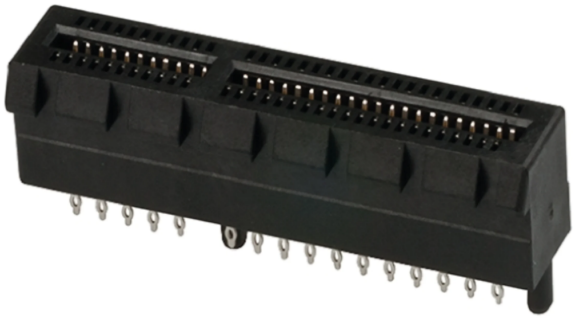
Av komponenter til bakplatene skal det velges:

- Plugger til PCI-kontakt
- Vifter for kjøling mellom kort
- Plugger til vifter
- Plugg til PWM for tilting av kamera
- Plugger for thrustere sitt PWM-signal
- Plugger for steppermotorer til manipulator

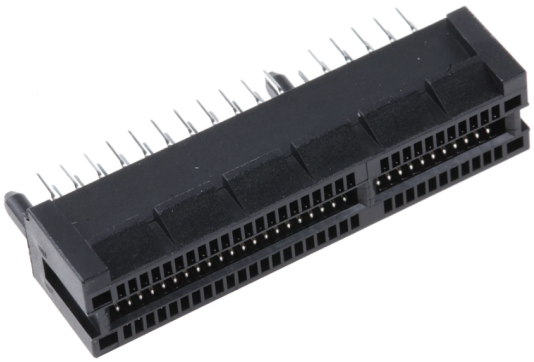
Kriteriene ved valg var:

Pris:	Siden gruppen har et budsjett, så ønsker vi å opprettholde dette så fremt det er mulig. Varene som blir sett på er da av en rimelig natur så fremt dette ikke går ut over kvalitet.
Leveringstid:	Prosjektet er høyst avhengig av at det er rask leveringstid på varene som kommer. Dersom deler er i fare for å komme for sent vil ikke prosjektet kunne gjennomføres. Siden såpass mye er avhengig av tidlig testing, vil også det å få utstyret på plass tidlig gi oss en stor fordel videre i prosjektet.
Kvalifikasjoner:	Produktet må ha de kvalifikasjoner som er nødvendig for oppgaven, og helst med en god sikkerhetsmargin. Siden prosjektet har mange elementer med seg, kan det fort skje at noe er feilberegnet. En god sikkerhetsmargin vil hjelpe å beskytte mot dette.
Leverandør:	Kjente leverandører har ofte bedre datablad som gjør at planleggingen av hvordan komponentene skal brukes kan skje før testing blir gjort. Det hjelper å vite alt om komponentene som blir levert.

Av PCI-plugger, var det disse kandidatene som ble sett på:



(a) Kontakt fra Amphenol



(b) Kontakt fra Samtec

Vare	Pris per stk	3D-modell	Kjent leverandør
Ampenol	30,84 kr	Nei	Nei
Samtec	36,46 kr	Ja	Ja

Tabell 6.1: Samenligning av PCI-E kontakter.

Selv om Amphenol sin var billigere, gikk vi for **Samtec PCIe 64 Way** da de var mer kjent leverandør, og hadde 3D-modell. Denne kunne da monteres i Fusion-modellen på forhånd. Dette gjør det lettere å vite at vi har rette dimensjoner på komponentene. Kontaktene er gullbelagt for å sikre god kontakt og lite slitasje.

Viftene skal kunne plasseres mellom PCI-kontaktene for å hjelpe med luftsirkulasjon inne i elektronikkhuset. Økt luftsirkulasjon hjelper for å kjøle ned elektronikken, mer om dette er forklart i tidligere delkapittel om varmeoverføring fra elektronikkhus 2.4.1.4. Viftenes dimensjoner er derfor begrenset til under **30mm** i bredde. Selv om små aksial-vifter til elektronikk er relativt billige, vil så små vifter være beregnet som ekstraordinære og derfor ha en større pris.

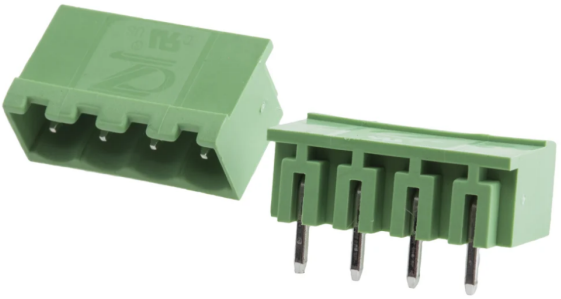
Vi ønskte en vifte som gikk på 5V, da denne enkelt kan kobles i både PA og PB som allerede har 5V- og jordtilkobling. Viftene vi endte opp med var **Nidec Copal** sin **F17HA-05HC**. Denne har dimensjonene **17mmx17mm** i bredde og lengde, og er **8mm** høy. Denne vil derfor passe godt mellom PCI-kontaktene. Den har en effekt på 0,15W og en luftstrøm på $0,018 m^3/min$, noe som vil tilsvare 0.3 liter per sekund. Effekten av viften er meget usikker, og det er antatt at denne ikke er veldig stor da den beveger mindre volum enn viftene brukt tidligere år. Denne rettes likevel slik at den blåser vekk fra eventuelle varmekilder som kretskortet har, og det anses at retningen vil hjelpe på virkningsgraden. Bilde vises på figur 6.18d.

Av plugger til vifter valgte vi en **Molex KK254** 3-pin kontakt. Dette er en veldig vanlig kontakt for PCB som er dimensjonert for 3 A. For laveffekt-viftene på 0,06 A er dette særlig overdimensjonert. Likevel er det nyttig at deler er store nok slik at kobling og lodding ikke blir for vanskelig, så fremt dette ikke går på bekostning av pris eller plass. Bilde vises på figur 6.18e.

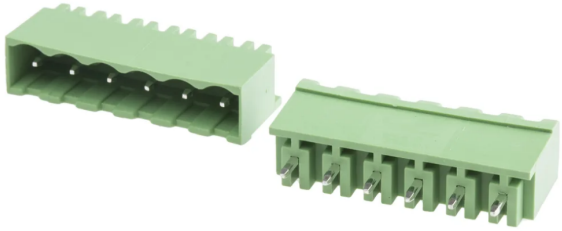
PWM-signalet til servoen for styring av kameratilt sin plugg ble **RS PRO PCB Terminal Block Header** 4-pins kontakt. Dette er en rimelig type som blir levert av RS PRO. Kontakten er oppgitt til 30A i strømtrekk, noe som er overdimensjonert for behovet. Likevel ble det lagt vekt på størrelse for å få en lettere kontakt å koble til. Siden det bare brukes tre av pinnene, vil den gjenstående bli brukt som ekstra jordtilkobling. Bilde vises på figur 6.18a.

PWM-signalet til thrusterene har mange utganger som ikke har stort strømtrekk, det er derfor mulig å bruke en kontakt som har flere kontaktpunkt som er nærme hverandre. Derfor ble det brukt en RJ45-kontakt, da denne er har gode mekaniske egenskaper for strekkavlastning. Valget endte på **TE Connectivity RJ-45** hunnplugg-kontakt. Denne har en beskyttende metallskjerm som er jordet, noe som avleder statisk elektrisitet ved berøring. Den er oppgitt til 2A per pin, som er tilstrekkelig da PWM-signalet bare er et styresignal. Bilde vises på figur 6.18c.

Plugger til manipulator trekker for mye strøm til at det kan brukes en RJ45 som for PWM-signalet til thrustere. Her er det derfor brukt en plugg av samme type som til PWM-signalet for kameratilt. Forskjellen er at det er valgt en 6-pins kontakt istedenfor 4-pins. Selv om det totalt skal gå 12 signal, var det rimeligere å bruke to stk 6-pin for å få ønsket mengde istedenfor å finne en 12-pins. Bilde vises på figur 6.18b.



(a) 4-pins fra RS PRO



(b) 6-pins fra RS PRO

(c) RJ-45 TE Conennctivity

(d) Nidac Copal F17HA-05HC

(e) Molex KK254

Ved valget av slike rimelige komponenter, var det sett på færre alternativ som var tilgjengelige på markedet før bestemmelsen ble tatt. Dette er fordi det er prioritert å finne kontakter som er driftssikre og gjør den påkrevde jobben, samtidig som de kommer med lav leveringstid. Siden alle kontaktene var rimelige og innenfor budsjettet, var det derfor ikke nødvendig å lete mer for å finne kontakter som var enda rimeligere.

6.6.1.4 Overflatebehandling

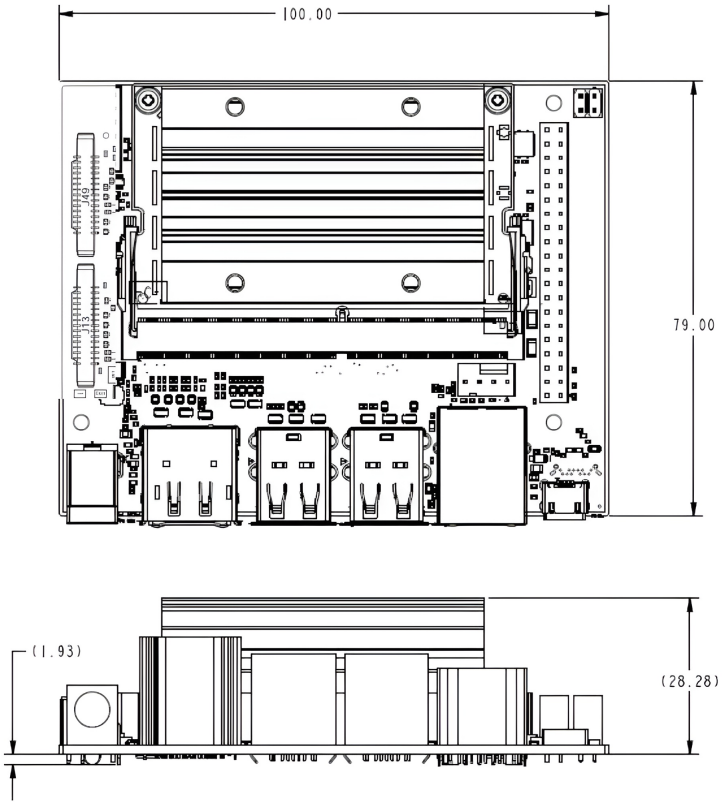
Selv om PCI-kontaktene er laget av gullbelagt nikkel, falt valget av overflatebehandling fortsatt på **HASL**. Dette er uoptimalt, da det vil være et elektrisk potensial mellom gull og tinnet(forklart mer i seksjon om overflatelag 6.4.6). Virkingen av dette vil likevel være relativt liten, og ha mest å si over tid. Siden det ikke er store krav til hvor lenge dette skal driftes, gikk vi for det rimeligeste alternativet for å holde oss under budsjettet.

6.6.2 Kommunikasjonskort

6.6.2.1 Design

Utviklingskortet som er valgt *(Jetson Nano)* i tidligere kapittel 4 om maskinvare skal monteres på kommunikasjonskortet. Kortet er relativt stort og følger med en 40-pins Header som skal kobles til kommunikasjonskortet.

Siden kortene står på rekke ovenfor hverandre med begrenset plass, er det ikke plass til at kortet bygger for mye opp på verken av sidene. Som utledet i kapittel 2 om elektronikkhus, er det 40mm mellom senter til senter mellom hvert kort. Dimensjonene på Jetson Nano er vist på figur 6.19.

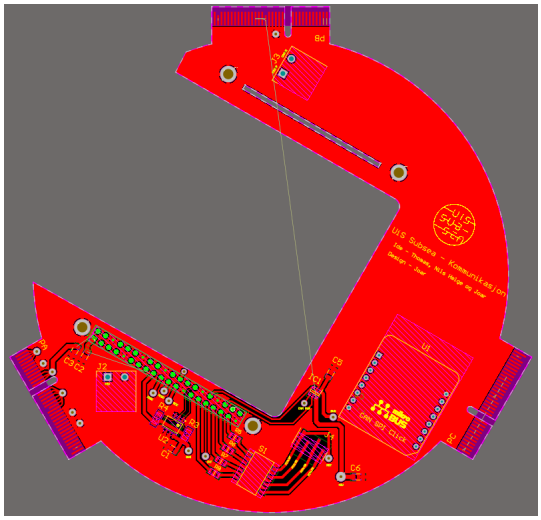


Figur 6.19: Nvidia Jetson mekaniske tegninger

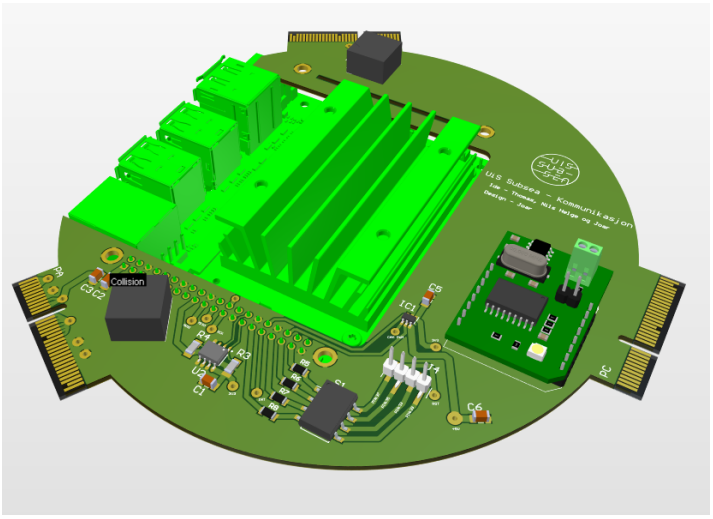
Siden Jetson bygger totalt over **30mm** i høyden, valgte vi å montere Jetson sin 40 pin GPIO via en header inn i kretskortet. Dette gjorde at Jetson da kan bruke ledig plass på begge sider av kortet til å bygge i høyden. Det vil også gi bedre muligheter for luftgjennomstrømming da Jetson er en kraftkrevende komponent. Dette gjorde også at Jetson enkelt kan demonteres eller byttes. Det tillater å teste funksjoner utenfor kretskortet også, noe som gir større rom for feil i planleggingsfasen.

Jetson sin bredde på **100mm** reduserer betydelig plassen til andre komponenter på kortet. Likevel var det tilstrekkelig plass til de resterende komponentene og modulkortet `SPI CAN Click 3,3V`.

Den ferdige utformingen av kortet kan vises på figur 6.20a og 6.20b. Tilkoblingene til Jetson som skal brukes for USB, strøm og Ethernet er vendt ut fra kortet slik at de ikke kolliderer med kommunikasjonskortet.



(a) Utlegg av PCB



(b) 3D visning

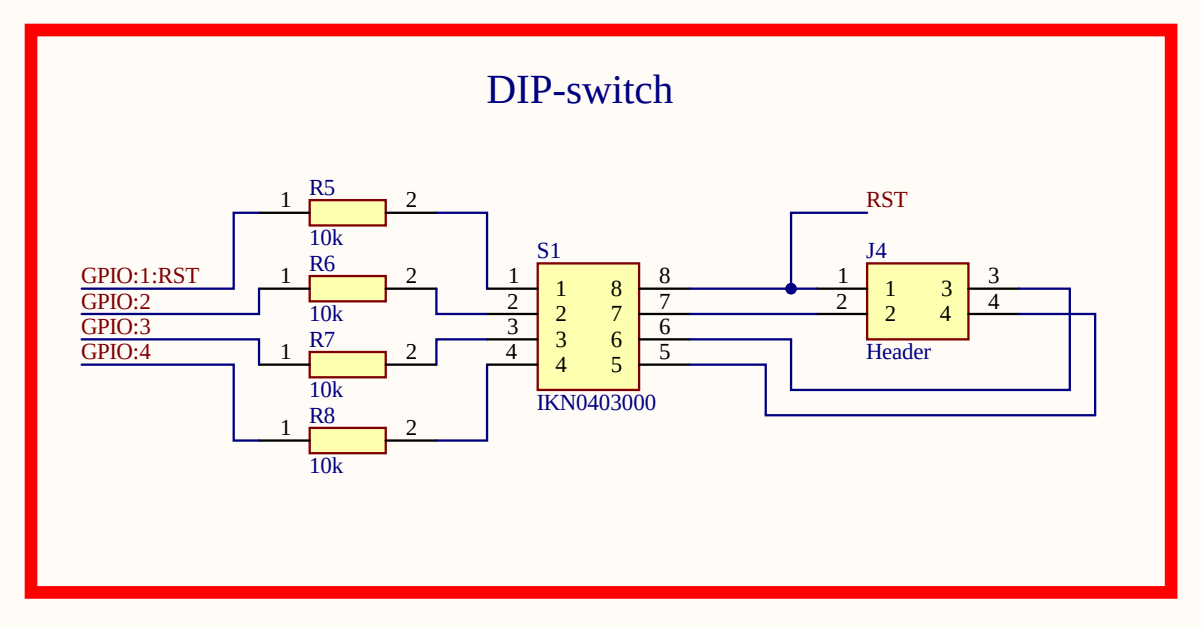
For å feste Jetson til kretskortet, bruker vi fire avstandsholdere i de tilpassede monteringshullene. Disse hullene går over CSI-kamerainngangene, og derfor har vi laget et rektangulært hull for flatkablene slik at de kan kobles til inngangene uten å bøyes. Jetson er montert innover i kretskortet for å unngå at USB-tilkoblinger og Ethernet kommer i konflikt med elektronikkhuset.

Utformingen og rutingen av kortet er gjort på en slik måte at det kan lages lokalt på UiS. Dette var et valg vi gjorde tidlig, da det ble gitt et forsøk på å få et fungerende kort bestilt fra leverandør. Med et prosjekt som er begrenset på tid, og en leveringstid på omtrent to uker - ønskte vi å ha muligheten til å lage et nytt kort uten å gjøre store endringer på design.

Selv om toleransene til leverandør *JLCPCB* er **5 mils** mellom baner eller loddeplater, er toleransen for den lokale fresen **13 mils**. Det er derfor prøvd å opprettholde denne klaringen slik at kortet kan lages i fresen. I tillegg er kortet designet som et to-lagskort. Dette er delvis fordi det ikke er mulig å lage flere lag lokalt, men dette ble også valgt da kretskortet ikke har noe grunnlag for å ha flere enn to lag. Kortet har såpass få komponenter at dette ikke er et problem for rutingen. Topp- og bunnlaget har et jordlag der det ikke er koblet baner for å hjelpe mot EMI og gi lettere tilkobling av komponenter som trenger jordtilkobling. Kortet har verken behov for høyt strømtrekk i et indre lag, analoge signal - eller inneholder støygenererende komponenter.

For å teste funksjoner fra kortet eller å ha tilgang til ekstra utganger, ble det lagt opp tilkobling for flere GPIO-utganger. Siden det er ønskelig å ha funksjoner som kan kobles av eller på uten å endre kode, ble det koblet opp fysiske brytere på kretskortet. Utgangene ble derfor koblet opp mot en header som gjorde at det enkelt kunne kobles opp ledninger ut fra eller mellom portene. Det er for eksempel mulig å aktivere en kodesnutt ved å koble en ledning mellom to ulike porter. Dersom den ene er satt som utgang, og den andre inngang som venter på signal - kan det aktiveres som et flagg som derretter aktiverer en kodesnutt.

Siden det skal bestilles kort tidlig i prosessen, er det er vanskelig å forutse alle utfordringer som kommer. Det var derfor et fokus å ha mulighet for å utvide eller endre virkemåten til kortet. På figur 6.21 vises tegningene for koblingen av dette.



Figur 6.21: Tegninger av kretsen for DIP-switch

RST er omstartskommandoen til CAN-modulen. Denne ønsker vi å ha mulighet til å fysisk koble vekk fra modulen for å unngå at det kommer uønsket omstart av modulen. Den er derfor koblet via bryteren som er montert på kortet. Alle GPIO-pinnene er koblet rett i en header, noe som forenkler testing eller viderekobling. En seriemotstand på 10 kΩ er montert for å redusere strømtrekk fra GPIO-utgangene.

6.6.2.2 Banebredde

Oversikt over kraftkrevende komponenter på kommunikasjonskortet:

Komponenter	**Effekt[W]**	**Strøm[mA]**
Jetson Nano 4GB	15W	3000mA
PWM servo kamera	3W	600mA
SPI CAN Click	0,035W	10mA
Temperatursensor	0,05W	10mA

Kortets klart mest kraftkrevende komponent er Jetson Nano utviklerkortet. Dette kan trekke opp til 15 W. Siden forsyningsspenningen som Jetson krever må være 5V, må banen ha lite spenningsfall fram til Jetson. Ut ifra formel (6.4) for spenningsfall og figur 6.3 for spenningsfall ved 5V, er det egentlig ikke nødvendig med mer enn **10 mils** i banestørrelse. Likevel er det god praksis å ha mer enn dette når det er plass og mulighet.

Banestørrelsen er derfor satt til minimum **0,35mm**(*13,7 mils*), men som foretrukket størrelser har vi brukt:

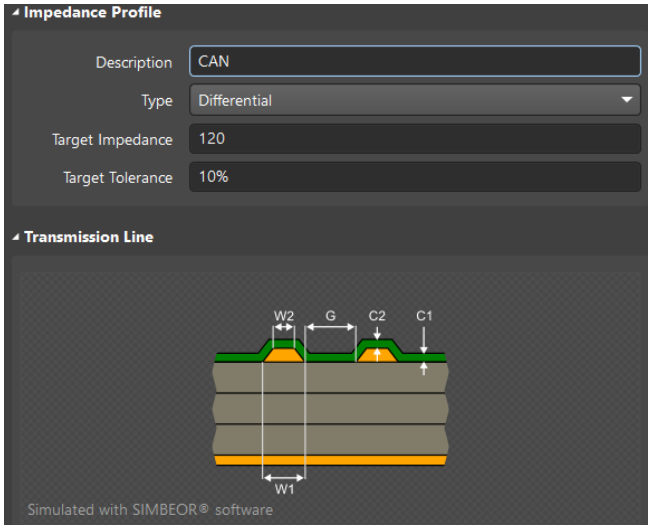
- 0,35mm

- 0,65mm
- 1,00mm
- 3,00mm

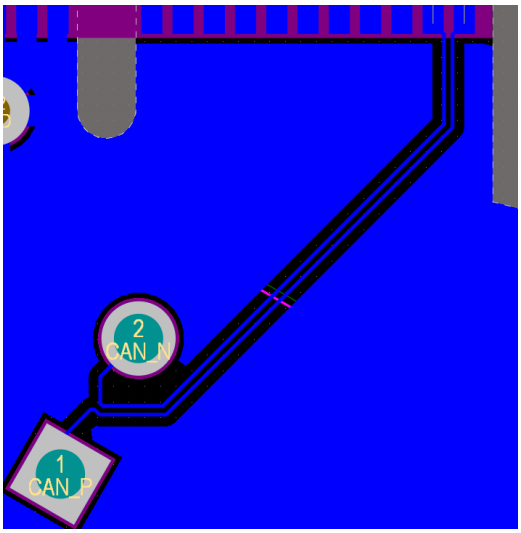
Der 3mm ble brukt for kraftbanene på 5V tilførselen til Jetson.

Differensialparet fra CAN-bussen skal som nevnt i seksjonen om krestkortdesign 6.4.2, rutes med en karakteristisk impedanse som er lik termineringsimpedansen. Termineringsimpedansen i enden av CAN-bussen er på 120 Ω. For å få den karakteristiske impedansen lik dette er det enklest å bruke det innebygde verktøyet som er i **Altium**. Et utklipp av dette kan vises på figur 6.22a.

Ved å velge den karakteristiske impedansen, vil utregninger automatisk bli gjort for banebredde og avstand mellom baner. Det er også en funksjon for ruting av baner som gjør at banene til differensialparet blir like lange. På figur 6.22b vises det hvordan CAN-bussparet er rutet hos oss.



(a) Altium meny for valg av karakteristisk impedans



(b) Ruting av CAN-bussen

Differensialparet på CAN-buss går også igjennom en plugg og videre til tvinnede ledninger før den så termineres i *CAN SPI Click* modulen. For å få et par med tvinnede ledninger til å ha rett karakteristisk impedans, må faktorer som permittivitet, diameter av ledere og avstand mellom lederne [162] være korrekt. Se ligning (6.9).

$$Z_0 = \frac{120}{\sqrt{\epsilon_r}} \times \ln \frac{2S}{D} \tag{6.9}$$

Hvor:

- Z_0 : Karakteristisk impedans
- ϵ_r Den relative permittiviteten til isolasjonen mellom lederne
- D : Diameter til ledere
- S : Avstand mellom tvinnede ledere

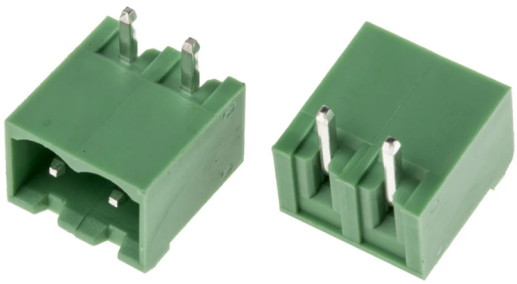
Dersom vi setter inn verdier for relativ permittivitet og avstand mellom ledere, kan vi finne hvilket tverrsnitt lederne må ha, basert på hva den karakteristiske impedansen er.

6.6.2.3 Komponenter

Av komponenter til kommunikasjonskortet skal det velges:

- Plugg for forsyning til Jetson
- Plugger til CAN-modul
- Bryter for test/videreutvikling
- Transformator fra 3.3V til 5V for PWM-signal til tilting av kamera
- Temperatursensor

6.6.2.3.1 Kraft Jetson Etter beregningene for banestørrelse for Jetson, trengs det plugger som er beregnet på over *3A* strømtrekk. Plugger med to tilkoblinger er kjøpt felles for flere grupper til å bruke. Typen fra *RS PRO PCB Terminal Block Header* 2 pins er derfor ledig for oss. Et bilde av denne vises på figur 6.23.



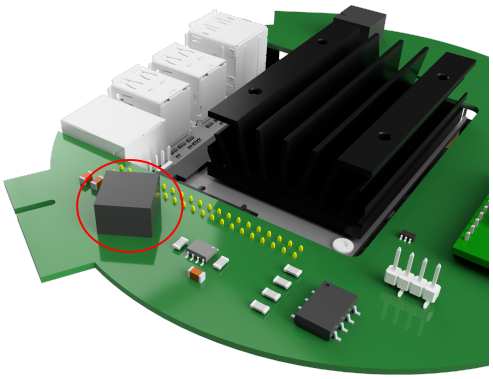
Figur 6.23: RS PRO PCB Terminal Block Header 2-Way

I databladet er denne beregnet på et makstrekk på *15A*, og en spenning på *300V*. Pluggen er også montert i en vinkel fra kretskortet, slik at pluggen vil ha en kontakt langs kretskortet. Dette er mer plassbesparende i høyden enn en vertikal montert plugg. Siden komponenten oppfyller våre behov og ikke er til ekstra belastning, er det derfor ikke nødvendig å se på andre alternativ for plugger. Valget ender derfor på **RS PRO PCB Terminal Block Header 2-way**.

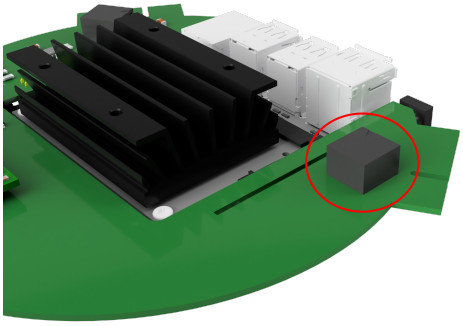
6.6.2.3.2 CAN
CAN-bussen sin plugg skal termineres fra modulen til kretskortet. Gjennom pluggen er det vanskelig å rute denne som et differensialpar. Det å rute et differensialpar gjennom en via eller plugg er generelt ønskelig. Siden CAN-bussparet kommer ut fra CAN-modulen gjennom en plugg som er koblet i en via, er det vanskelig å gjøre noe med denne. Den stabile driften av CAN-bussen gjorde likevel at det kunne velges å bruke en modul med slike plugger og koblinger. Dette var testet med ønskelig overføringshastighet på forhånd, og det var dermed mulig å velge denne løsningen.

Gjennom pluggen vil det derfor ikke bli tatt hensyn til at CAN-bussparet burde rutes som et differensialpar. Valget av plugg for CAN-buss falt derfor også på **RS PRO PCB Terminal Block Header**

2-way, ettersom denne var tilgjengelig for prosjektet uten ekstra kostnad. De to kontaktene kan sees montert på kortet i figur 6.24a og 6.24b.



(a) Plugg for Jetson Nano 5V



(b) Plugg for CAN-bus

6.6.2.3.3 DIP-switch

Det stilles få krav til DIP-switchen annet enn at den må fungere ved 5V og kunne tåle et makstrekk på over 50mA. Det er sett ut en som har fire innganger, og den må være av relativt liten karakter. DIP-switchen som ble valgt er `IKN0403000` av `APEM`. Denne har fire innganger, tåler makstrekk på 100mA ved 24VDC. Den har et lite fotavtrykk på 14x7.5mm, slik at denne ikke tar opp for mye plass på kretskortet.

6.6.2.3.4 PWM

GPIO-pinenne fra Jetson er på 3,3V. Servomotoren som er brukt til kameratilting, går på 5V. Siden vi ønsker å bruke en PWM-utgang fra Jetson, må logikken fra utgangen konverteres til 5V. Den boolske logikken er rimelig enkel, og vises på ligning (6.10).

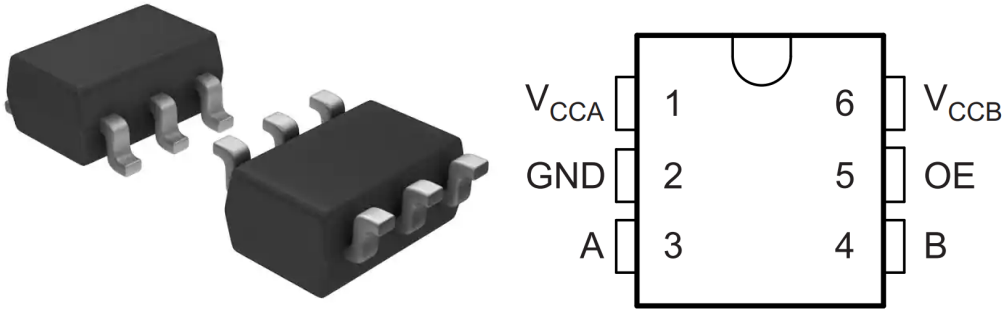
$$A = B \tag{6.10}$$

Hvor:

- A : PWM-signal fra Jetson på 3,3V
- B : PWM-signal til Servo på 5V

For å få til dette kan det brukes flere komponenter. Vår prioritering var at det skulle være plassbesparende, og helst laget som en IC[8]. Valget av logikk falt på en *Level Shifter* etter konsultasjon med Jon Fidjeland. Ved leting etter komponenter fant vi til slutt *TXS0101DBVT* fra *Texas Instruments*. Denne tar inn to referansespenninger, en for V_{CCA} og en for V_{CCB}. Se figur 6.25a og 6.25b for illustrasjon og tilkoblinger.

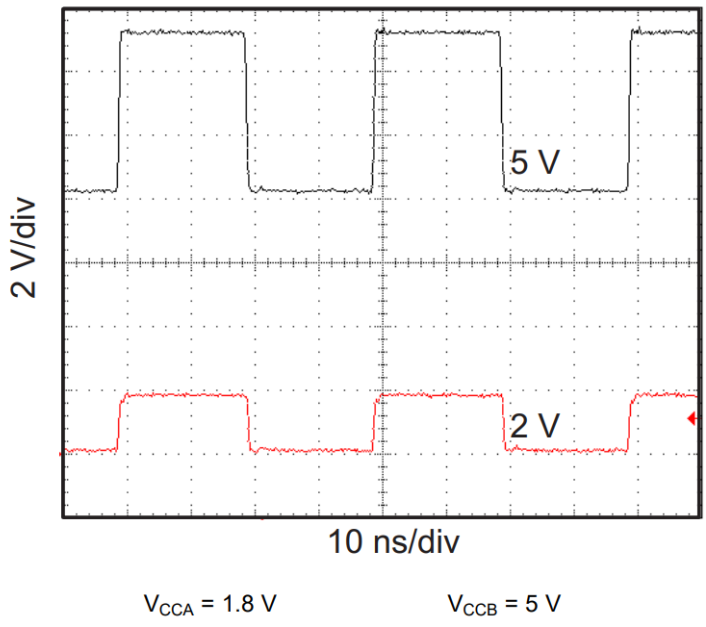
[8]Integrated Controller



(a) TXS0101DBVT **(b)** Tilkoblinger [163]

- A : Inngangsignal
- B : Utgangsignal
- V_{CCA} : Referansespenning for A logikk
- V_{CCB} : Referansespenning for B logikk
- OE : *Output Enable*, aktivirer overføring av logikk
- GND : Jordtilkobling

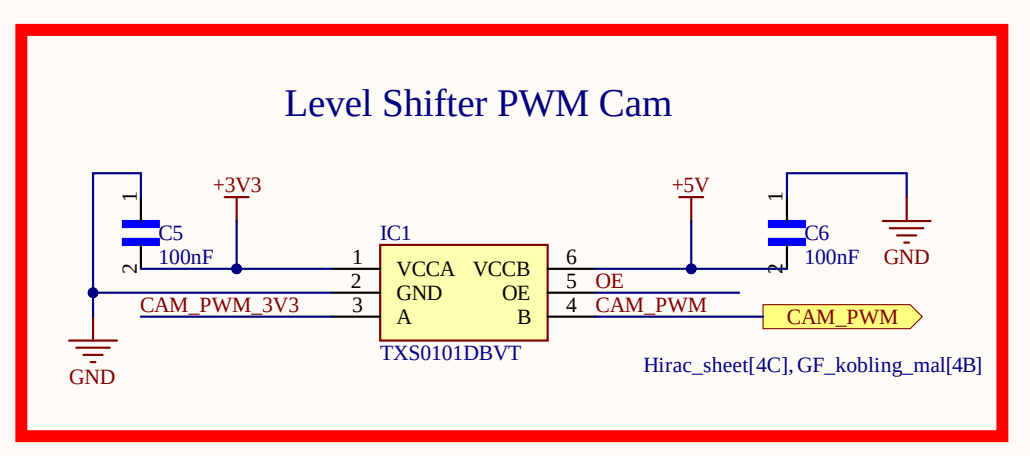
Grunnen til at denne kretsen var valgt er fordi den har rette spenningsområder for bruken vår. Den tar fra 1,65V til 3,6V på V_{CCA}, og fra 2,3V til 5V på V_{CCB}. Denne kan derfor kobles opp på kretskortet for å ta en 3,3V inn og gi 5V ut. På figur 6.26 er det vist et eksempel på shiftene-logikk der et signal går fra 1,8V på inngang A til 5V på utgang B.



Figur 6.26: *Shifting* logikk[163]

Ut fra figuren vil styresignalet(*markert i rødt*) fra inngang A, bli kopiert av utgangssignalet B som oppnår samme logiske nivå på en annen spenningsreferanse.

Inngang A kan derfor kobles opp mot en PWM-utgang på Jetson. PWM-signalet på 3,3V kan derfor sendes videre til servomotoren som 5V. Se figur 6.27 for å se hvordan dette er koblet opp.



Figur 6.27: Tegning for tilkobling av IC

I kretsen er to kondensatorer på $100nF$ er satt opp hver av de to forsyningene for å sikre stabilt spenningsnivå ved veksling. *OE*-pinnen kobles opp mot en utgang på jetson, slik at denne kan settes høyt eller lavt for å aktivere eller deaktivere signalet.

6.6.2.3.5 Temperatursensor

Temperatursensoren ble valgt av sensorgruppen, og er en overflatemontert IC fra `STMicroelectronics`, modell `STTS75`. Denne er en *Band-gap* temperatursensor, som bruker transistorens egenskaper og strømtetthet for å finne temperaturen. Den har et måleområde på -55 til 125 ℃, med en feilmargin på ± 0.5 ℃.

Denne skal drives gjennom I^2C og monteres på kommunikasjonskortet. For å minimere lengden på I^2C-banene, er temperatursensoren plassert så nærme pinnene til `SDL` og `SCL` som mulig. Siden Jetson er den mest kraftkrevende komponenten på kortet, vil det også være mest relevant å måle temperaturen nærmest denne.

6.6.2.4 Overflatebehandling

Likt som for bakplatene er det optimalt å bruke ENIG som overflatebehandling når det er gullbelagte PCI-kontakter. Siden det likevel ikke er store krav til verken flat overflate, eller krav til lengre driftstid - gikk vi for **HASL** som overflatebehandling. Prisforskjellen mellom de to er større enn verdien av å ha optimal overflatebehandling.

6.6.3 USB og CSI

Det ble også laget to ulike kort som skal hjelpe med montering av USB og CSI. Hensikten med de to var:

USB:	Siden det var begrenset med plass mellom Jetson utviklerkortet og kapslingen til elektronikkhuset, må USB-pluggen være kortere enn USB-pluggen på den medfølgende kabelen til kameraet. Dette oppnås ved å lage et egendefinert kretskort som kan loddes til en USB-kontakt
CSI:	CSI-kablene som medfølger stereokameraet er bare lange nok til at kortet kan stå fremst i rekken. Det er ønskelig å kunne plassere denne andre plasser. Medfølgende kabler er egendefinerte av leverandør til det spesifikke kameraet, og det var ikke mulig å finne tilsvarende kabler. Det ble derfor laget et kort som tillater en viderekobling av CSI-kablene

6.6.3.1 USB

For å spare plass er det mulig å bruke en overflatemontert USB-kontakt. Da tradisjonelle USB-kontakter er laget for å være solide og at det er mulig å få et godt grep, er overflatemonterte laget for å monteres rett på et kretskort. Se figur 6.28a og 6.28b for de to ulike typene.

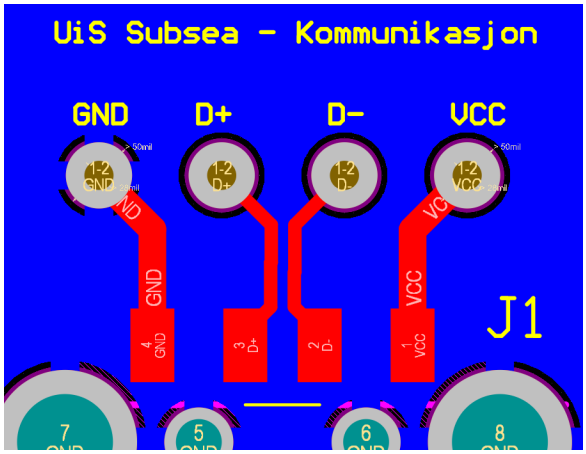
(a) Tradisjonell USB-kontakt med kabel [164]

(b) Overflatemontert USB-kontakt[165]

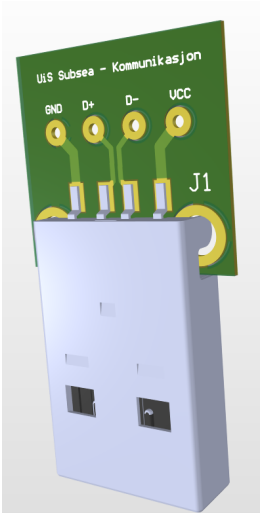
Ut fra figuren ser vi at den overflatemonterte tar mindre plass enn den tradisjonelle kontakten. Ved å lage et kretskort som denne kan monteres på, vil det kunne trekkes ledninger uten å bruke en slik kontakt. Kontakten vi valgte var fra *Molex*, og er vist på tidligere referert figur 6.28b.

USB-standarden har store krav til hvordan høyhastighets-USB skal føres, da USB er et differensialpar på samme måte som CAN-buss er. Dette er snakket mer om i seksjonen fra kretskort 6.4.2. Likevel er det kriterier for banelengde og forskjell i banelengde mellom paret som kan følges for å få et brukende kort, selv om ikke alle standarder blir opprettholdt [166]. Siden banene ikke skal rutes noe langt er det derfor mulig å følge kriteriene for å lage et kort som er to lag. Det er generelt rimeligere å lage et 2-lags kretskort enn det er for flere lag.

På figurer 6.29a og 6.29b er det mulig å se hvordan kretskortet er koblet opp og hvordan det ser ut fra *Altium*.

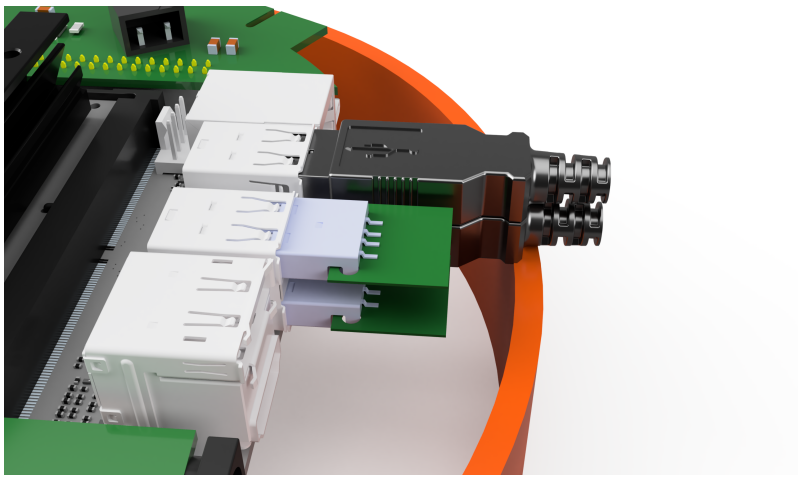


(a) Utlegg



(b) 3D modell

Som vist vil USB-kontakten overflatemonteres på kortet, og kortet vil deretter loddes ledninger på. Dette vil redusere behovet for plass betydlig. Se figur 6.30 der det vises med en illustrasjon.



Figur 6.30: Tradisjonell og egendesignet kort for USB-plugg

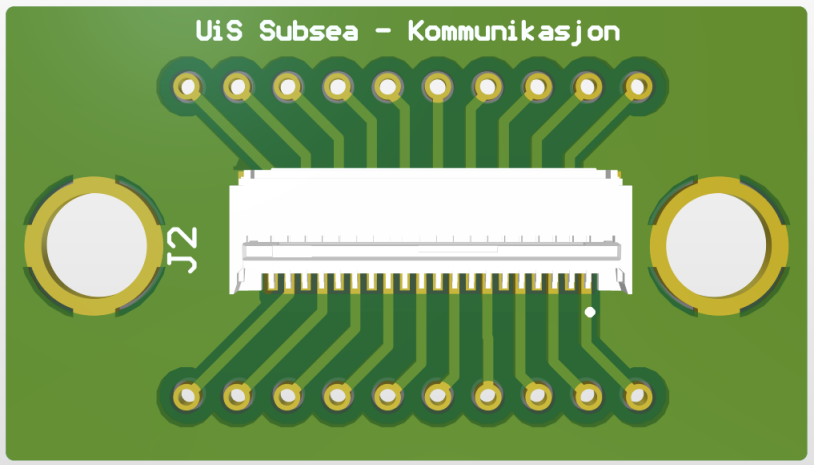
På bildet er det tegnet inn en kapsling i samme dimensjon for lett illustrasjon av hvordan en tradisjonell USB-plugg vil kollidere med ytterveggen.

6.6.3.2 CSI

CSI-kablene er av typen *FFC*[9], noe som er relativt lett og rimelig å få tak i. Kablene til kameraet går fra en *pitch*[10] til en annen. Kablene har også en forskjellig mengde med tilkoblinger i hver ende, da kun noen av tilkoblingene fra Jetson blir brukt. Siden de er så egentilpasset til kameraet, er det enklere å lage et kretskort for å forlenge disse kablene enn det er å lage egne kabler som er lengre.

Kablene som kommer fra kameraet er *0,5mm pitch* med 20 kontaktflater. For å viderekoble kan det brukes to stk plugger for denne kabelen. Kontakten som ble valgt er **Molex CONN FFC FPC 20POS 0.5MM**. Det monteres en på hver side av kortet. Kretskortet vises på figur 6.31.

[9] Flat Flexible Cable
[10] Avstand mellom kontaktflatene



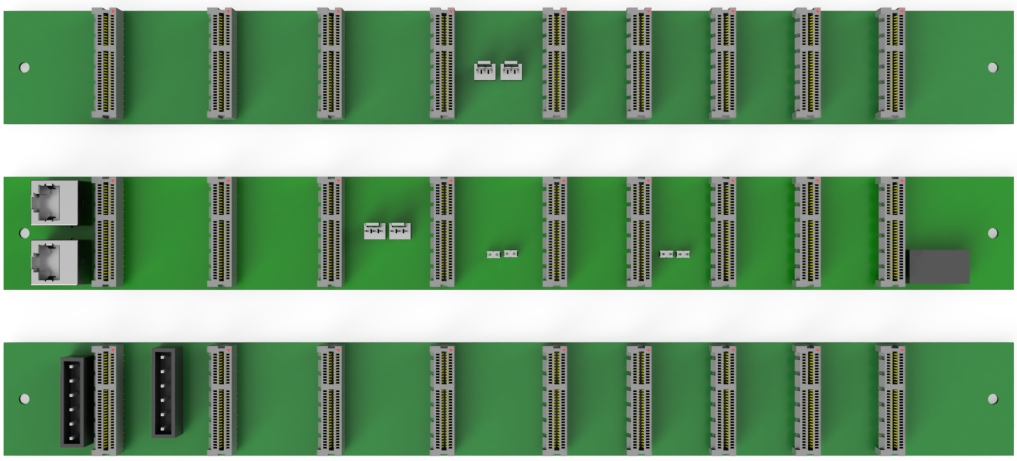
Figur 6.31: 3D modell av kretskort

Det må også bestilles et par med flatkabler som er lange nok til å rekke frem i ROV-en selv om kortet står lenger bak.

6.7 Resultat og oppsummering

	Bakplater
Design:	Bakplatene er lagd for å ligge horisontalt i ROV-en med 9 stk **PCI-kontakter** for feste av kretskort. Bakplatene forsyner kommunikasjon, kraft og signal mellom kretskort og videre ut av ROV-en. Bakplater er laget **365mm** høye og **41mm** brede.
Banebredde:	Banebredden er dimensjonert ut ifra hva PCI-kontaktene sin strømtrekk er oppgitt til. Det er tatt hensyn til spenningsfall og maks strømtrekk ved å velge hvor mange baner som trengs for hvert signal. Signalbaner trekkes med **24 mils** totalt, og kraftbaner trekkes med **55 mils**. Baner for 5V, 12V og jord ledes via kobberflater som dekker hele området der pinnene er koblet til.
Komponenter:	PCI-kontaktene som er valgt er **Samtec 64-way PCI**. Komponenter til bakplater ellers er dimensjonert for strømtrekket til de ulike signalene. Det er brukt **Molex KK254**, **RS PRO 4-pins terminal block**, **TE Connectivity RJ-45** og **RS PRO 6-pins terminal block** for de ulike tilkoblingene.
Overflate -behandling:	Det er valgt **HASL** som overflatebehandling, da prisen for ENIG var for høy i forhold til verdien.

Illustrasjon av resultat kan vises på figur 6.32.

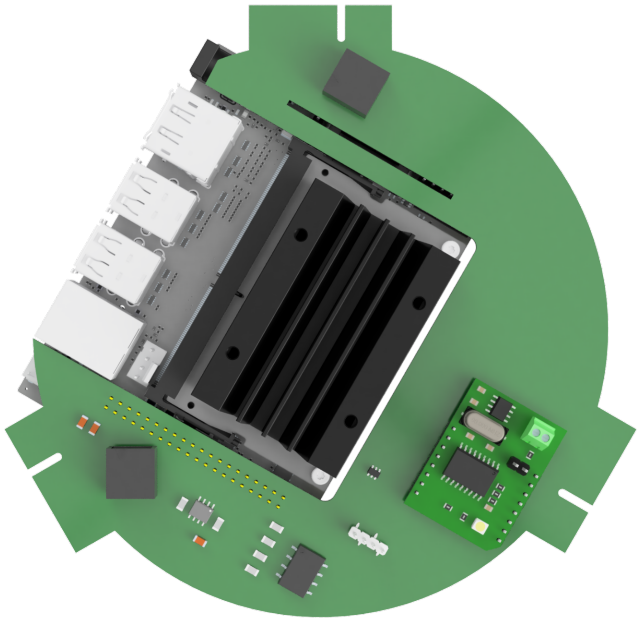


Figur 6.32: 3D-modell av kretskort

Kommunikasjonskort

Design:	Kommunikasjonskortet er laget for å holde modulkortet Nvidia Jetson Nano. Det kobles på bakplatene og blir brukt for å drive CAN-bussen for kommunikasjonen internt mellom kort. Kortet er lagd med en diameter på **140mm**, med en uthulning til Jetson og utstikkende gullfingrer.
Banebredde:	Banebredden til strømforsyning for Jetson er laget til å være **3mm**, for å ha god sikkerhetsmargin på banene. De andre banene er laget så store som mulig. Både for at kortet skal kunne lages lokalt, men også fordi det er ønskelig å ha minst mulig spenningsfall over banene.
Komponenter:	Det er montert på **DIP-switch**, **CAN SPI Click**, **STTS75**, **TXS0101DBVT** og **Jetson Nano**. For å realisere kretsene er det brukt overflatemonterte kondensatorer og resistanser for avlastning og strømreduksjon i oppkoblingen.
Overflate -behandling:	Det er valgt **HASL** som overflatelag, da prisen for ENIG var for høy i forhold til verdien.

Illustrasjon av resultat kan vises på figur 6.33.



Figur 6.33: 3D-modell av kretskort

Ved levering av kretskortene fant vi ut at den ene bakplaten måtte bestilles på nytt, da denne hadde et speilvendt fotavtrykk. Det ene fotavtrykket på kommunikasjonskortet var også feil, slik at signalet **INT** for CAN-modulen var på feil plass. Likevel var dette mulig å ordne med en lask, og det ble bestilt ny bakplate ilag med en annen bestilling av kretskort.

Kortene fungerte ellers som planlagt da funksjonene ble testet. Lodding gikk forholdsvis raskt, selv om det var mange kontakter å lodde. IC-en for *Level Shifter* fungerte ikke og ble fjernet etter litt testing og feiling. I ettertid viste det seg at servoen fungerte direkte med PWM-signalet fra Jetson.

Ved å lage kun 2-lags kort, og å bruke en rimelig overflatebehandling - ble kretskortene ganske rimelige. Se tabellen for oversikt:

Kretskort	**Pris**	**Toll**	**Total**
Bakplate PA	104,62 kr	26,18 kr	130,80 kr
Bakplate PB	104,62 kr	26,18 kr	130,80 kr
Bakplate PC	104,62 kr	26,18 kr	130,80 kr
Bakplate PB Ny	104,62 kr	26,18 kr	130,80 kr
Kommunikasjon	240,27 kr	60,01 kr	300,28 kr
USB	42,26 kr	10,56 kr	52,82 kr
CSI	42,26 kr	10,56 kr	52,82 kr
Frakt	-	-	258,09 kr
Total	-	-	**1318,01 kr**

Tabell 6.2: Priser for kretskort bestilt hos JLCPCB

Kortet for CSI hadde for liten *pitch* til å lodde for hånd. Det ble heller aldri gjort et forsøk for å lodde dette med loddepasta, da det var enklere å kjøpe et ferdig kretskort for å skøyte CSI-kablene. Dette kortet fant vi i etterkant av at vi lagde et selv. Kortet for USB-tilkobling derimot, fungerte bra og var driftssikkert. Diskusjon rundt hva som fungerte bra, og hva som fungerte dårlige angående kretskortdesign - vil bli diskutert videre i kapittel 8.

Kapittel 7

Programvare

Dette kapittelet tar for seg programvaren som er laget for å realisere prosjektet, og forklaringer rundt sentrale deler. All koden som er blitt laget er lagt ut på Github og kan finnes her: CAN-buss og Kommunikasjon 2023.

7.1 Problemstilling

Programvaren skal realisere fire ulike funksjoner:

- Motta data fra kontrollstasjonen via Ethernet og sende dette på CAN-bussen.
- Motta data fra CAN-bussen og sende dette til kontrollstasjonen via Ethernet.
- Ta inn bildedata fra kamera på USB og CSI. Denne skal sendes til kontrollstasjon over Ethernet.
- Tilting av stereokamera på front av ROV

Sending og mottak av data - Programvaren skal ta inn kjørekommandoer fra kontrollstasjonen. De skal sendes videre på CAN-bussen slik at kommandoene kan bli utført av kretskortene.

CAN-busskommunikasjon - Programvaren må kunne lese meldinger som blir sendt på CAN-bussen, som skal sendes til kontrollstasjonen.

Sende bildedata - Programvaren skal ta inn bildedata fra USB- og stereokamera, og komprimere denne før den sendes videre til kontrollstasjonen.

Tilting av stereokamera - Programvaren må kunne ta inn en vinkel fra kontrollstasjon og styre servomotoren til ønsket vinkel.

Behovs- og funksjonsspesifikasjonene til programvaren blir å realisere de funksjonene som er nevnt og diskutert i tidligere kapittel. Det er derfor ikke tatt med en egen liste av dette her.

7.2 Programvare for mikrokontrollere

Alle kortene ombord ROV-en skal kommunisere med hverandre over CAN-buss, og trenger dermed programvare som realiserer dette. I dette delkapittelet vil det gis en beskrivelse av de funksjonene som er laget, samt forklaring av sentrale kodeutdrag. Koden for CAN-buss kan finnes på UiS Subsea sin GitHub[1]. Koden til kommunikasjonskontrolleren Jetson Nano ligger under "Kommunikasjon-2023"[2] og foreslått kode for mikrokontrollere ligger under "CAN-Bus-STM32G431xx"[3].

7.2.1 C-kode til mikrokontrollere

Koden som er laget er innspirert av fjorårets gruppe [31] og en guide for oppsett av FDCAN på STM32-mikrokontrollere[167]. FDCAN er en videreutviklet versjon av CAN-protokollen, og er forklart i delkapittel 3.4.4. Mye av strukturen for oppsettet av CAN-buss er hentet fra koden som ble laget i fjor[31]. Men for å få denne koden til å fungere og kjøre, var det nødvendig å foreta noen modifikasjoner for å gjøre den kompatibelt med FDCAN-grensesnittet på mikrokontrollerene. De nødvendige modifikasjonene er basert på en guide for dette grensesnittet[167].

Mikrokontrollerene som skal kommunisere på CAN-bussen er Nucleo-kortene: Nucleo-32 STM32G431KB [63] og Nucleo-64 STM32G431RB [168]. Disse programmeres i programmeringsspråket C, og dette gjøres gjennom utviklingsverktøyet, STM32CubeIDE [169]. Det brukes mikrokontrollere på alle andre kort enn kommunikasjonskortet. Derfor er det laget et rammeverk for koden som kreves for å kommunisere over CAN-buss. Denne deles så ut til alle gruppene med mikrokontrollere slik at de enkelt kan koble seg opp til CAN-bussen.

7.2.1.1 Oppsett av FDCAN1 i STM32CubeIDE

For at mikrokontrollerene skal kunne kommunisere sammen med hverandre og med Jetson Nano på kommunikasjonskortet - må alle nodene på CAN-bussen bruke samme bitrate. For å kalibrere bitraten til mikrokontrollerene, må parametrene og klokkefrekvensen stilles inn som forklart i delkapittel om bittiming(3.4.3.4 og 3.4.3.5).

Alle nodene på CAN-bussen skal ha en bitrate på $\boxed{CAN_{bitrate} = 500kbps}$, som gir en nominell bittid på $\boxed{t_{NBT} = 2000ns}$. Hver bit skal leses av på omtrent 87% av bittet, med andre ord 87% av bittiden. Mikrokontrollerene STM32G431RB og STM32G431KB skal kjøre med en systemklokke på $\boxed{f_{sys} = 170MHz}$.

Som nevnt i delkapittel 3.4.3.5, må BRP være et heltall mellom 8 og 25, som også gir et heltall som antall tidskvantum. Velger her å bruke $BRP = 20$, og ved å bruke formel 3.10 fra delkapittel 3.4.3.5 kan følgende utledes:

$$N_{tidskvantum} = (Sync\_Seg + Seg1 + Seg2) = \frac{f_{sys}}{BRP \cdot CAN_{bitrate}} = \frac{170 \cdot 10^6 Hz}{20 \cdot 500 \cdot 10^3} = 17 \qquad (7.1)$$

[1] https://github.com/UiS-Subsea
[2] https://github.com/UiS-Subsea/Kommunikasjon-2023
[3] https://github.com/UiS-Subsea/CAN-Bus-STM32G431xx

Beregningen ovenfor gir 17 tidskvantum, hvor hvert tidskvantum er på $t_q = 117,65ns$. Det neste som må gjøres, er å beregne antall tidskvantum for $Seg1$. Dette gjøres også slik som i delkapittel 3.4.3.5 ved å ta utgangspunkt i ønsket avlesningspunkt. $Sync\_Seg$ er som vanlig lik 1.

$$\begin{aligned} Avlesning[\%] &= \frac{(Sync\_Seg + Seg1)}{N_{tidskvantum}} \cdot 100 = 87,5\% \\ Seg1 &= \frac{87,5}{100} N_{tidskvantum} - Sync\_Seg = \frac{87,5}{100} 17 - 1 = 13,875 \end{aligned} \tag{7.2}$$

–> Ikke OK

Siden de ulike segmentene i bittet må være et heltall, kan ikke beregnet verdi for $Seg1$ brukes. Ved å avrunde $Seg1$ opp til 14, kan avlesningspumktet beregnes og deretter sjekkes opp mot ønsket avlesningspunkt på $87,5\%$.

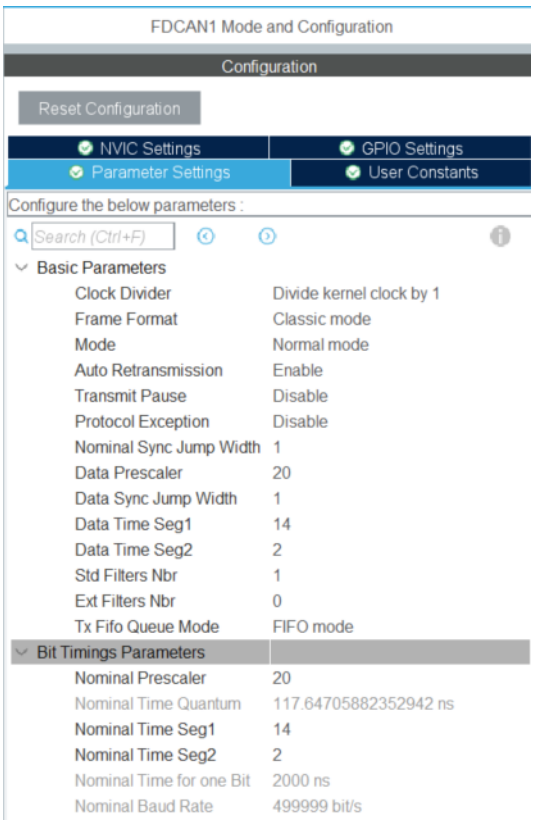
$$Avlesning[\%] = \frac{(Sync\_Seg + Seg1)}{N_{tidskvantum}} \cdot 100 = \frac{(1 + 14)}{17} \cdot 100 = 88,2\% \approx 87,5\% \tag{7.3}$$

–> OK

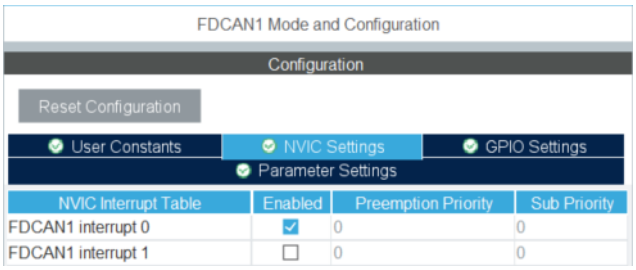
Avlesningspunktet blir på 88,2% ved 17 tidskvantum og $Seg1 = 14$. Dette er nært nok 87,5% og dermed akseptabelt. Ved at $Seg1 = 14$ og $Sync\_Seg = 1$, vil det resterende tidskvantumet plasseres i $Seg2$.

$$\begin{aligned} (1 + Seg1 + Seg2) &= 17 \\ Seg2 = 17 - 1 - Seg1 &= 17 - 1 - 14 = 2 \end{aligned} \tag{7.4}$$

Ved å bruke parametrene: $BRP = 20$, $Seg1 = 14$ og $Seg2 = 2$, vil mikrokontrollerene kjøre med en bitrate på 500kbps og et avlesningspunkt på 88,2% med en klokkefrekvens på 170MHz. Oppsettet av FDCAN-perifermodulen for mikrokontrollerene blir da som vist i figurene nedenfor.



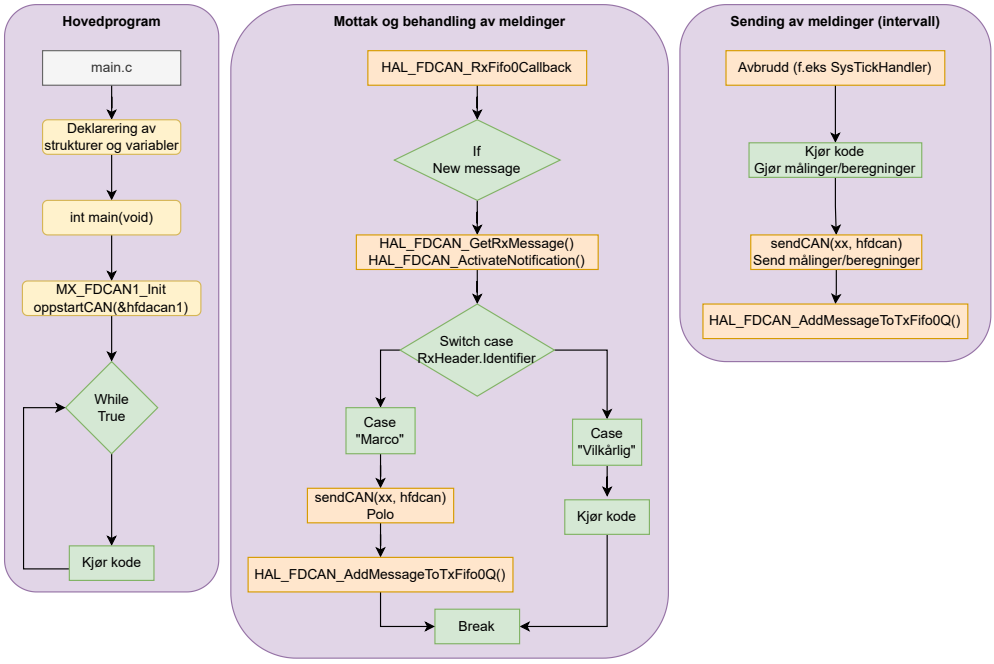
Figur 7.1: Parameter som er brukt for FDCAN1 på mikrokontrollere



Figur 7.2: Avbrudd aktivert for FDCAN1

7.2.1.2 Oppsett av CAN-bussen

Kodeutdragene som kommer nedover i dette delkapittelet er for å sette opp CAN-bussen. I figur 7.3 nedenfor, er den generelle flyten for CAN-buss på mikrokontrollerene illustrert med et flytskjema. Da det er de andre gruppene som bruker mikrokontrollere med utdelt kode for CAN-buss, er det opp til hver enkel gruppe for hvordan de løser flyten. Dette er bare en overordnet flyt for CAN-grensesnittet.



Figur 7.3: Generell flyt for CAN-buss på mikrokontrollere

De kommende kodeutdragene vil ta for seg oppsett av CAN-buss, mottak av meldinger på CAN-buss og sending av meldinger på CAN-buss. Dette er kode som er blitt utdelt til alle gruppene med mikrokontroller, og sørger for at de får en fungerende CAN-buss. Modifikasjoner kan forekomme fra deres side.

Når programmet på mikrokontrollerene starter, vil det først deklareres noen definisjoner for strukturer og variabler. Kodeutdraget 7.1 nedenfor viser et par definisjoner som blir kjørt ved initialisering av programmet.

Kode 7.1: Definering av variabler og strukturer for CAN-bussen

```
1 /* USER CODE BEGIN PV */
2 /* ----------------Startoppsett for CAN-Bus-----------------*/
3 FDCAN_TxHeaderTypeDef TxHeader;
4 FDCAN_RxHeaderTypeDef RxHeader;
5 uint8_t TxData[8] = {0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08}; // CAN-Bus Transmit ...
      Buffer
6 uint8_t RxData[8] = {0, 0, 0, 0, 0, 0, 0, 0}; // CAN-Bus Receive Buffer
7 /* USER CODE END PV */
```

I linje 5 og 6 blir ***TxData*** og ***RxData*** definert som en uint8 "*byte array*", hvor hver av de består av 8 byte. Det er disse variablene som inneholder datapakken som skal sendes eller mottas på CAN-bussen. I linje 3 og 4 blir strukturene ***TxHeader*** og ***RxHeader*** definert. Disse sendes eller mottas sammen med en datapakke, og inneholder all informasjon om CAN-bussmeldingen som ikke er en del av selve datapakken. Hva denne informasjonen er gås gjennom i kodeutdrag 7.3, men det kan for eksempel være

ID-en på meldingen og ID-typen.

Etter at strukturer og variabler er blitt deklarert, vil de ulike perifermodulene bli initialisert. Blant disse er den brukerlagte funksjonen `oppstartCAN()`, en av funksjonene som blir kjørt. Hva som blir initialisert er gjengitt i kodeutdrag 7.2 nedenfor.

Kode 7.2: Initialisering av perifermoduler og oppstartCAN

```
1   /* Initialize all configured peripherals */
2   MX_GPIO_Init();
3   MX_LPUART1_UART_Init();
4   MX_FDCAN1_Init();
5   /* USER CODE BEGIN 2 */
6   /* Starter CAN-buseen */
7   oppstartCAN(&hfdcan1);
8   /* USER CODE END 2 */
```

I funksjonen `oppstartCAN()`, er det to vesentlige ting som skjer. Det ene er oppsettet og konfigurasjonen av ***TxHeader*** som er gjengitt i kodeutdrag 7.3. Og det andre er oppsettet av filterkonfigurasjonene. Kodeutdrager for filterkonfigurasjonen er gjengitt i kodeutdrag 7.4

Kode 7.3: Konfigurering av TxHeader (del av oppstartCAN())

```
1  // Configure TX Header for FDCAN1
2  TxHeader.Identifier = 0x00;
3  TxHeader.IdType = FDCAN_STANDARD_ID;
4  TxHeader.TxFrameType = FDCAN_DATA_FRAME;
5  TxHeader.DataLength = FDCAN_DLC_BYTES_8;            // Antall byte som sendes
6  TxHeader.ErrorStateIndicator = FDCAN_ESI_ACTIVE;
7  TxHeader.BitRateSwitch = FDCAN_BRS_OFF;
8  TxHeader.FDFormat = FDCAN_CLASSIC_CAN;              // Bruker CAN og ikke FDCAN
9  TxHeader.TxEventFifoControl = FDCAN_NO_TX_EVENTS;
10 TxHeader.MessageMarker = 0;
11
12 HAL_FDCAN_ConfigFilter(canPort, &sFilterConfig);
13 HAL_FDCAN_Start(canPort);
14 HAL_FDCAN_ActivateNotification(&hfdcan1, FDCAN_IT_RX_FIFO0_NEW_MESSAGE, 0);
```

I kodeutdrag 7.3 ovenfor settes de ulike parametrene til strukturen ***TxHeader*** når funksjonen `oppstartCAN()` blir kalt på. Strukturen må konfigureres før en melding kan sendes på CAN-bussen. Disse parametrene blir lagret i ***RxHeader*** til mottaker av meldingen.

Identifier:	ID-en hvor meldingen skal sendes. Grunnen til at denne er satt til 0x00, altså ID 0 i dette tilfellet, er ID-en skal settes manuelt ved sending av meldinger gjennom funksjonen `sendCAN()` som gås gjennom i kodeutdrag 7.5.
IdType:	Sier noe om det er "standard-ID" eller "extended-ID" som er brukt. I dette tilfellet brukes det "standard-ID".
TxFrameType:	Markerer om det er en melding som sendes eller en forespørsel om melding som sendes. Ved å sette denne til `FDCAN_REMOTE_FRAME`, vil ***TxHeader*** markere enn forespørsel om data, mens ved `FDCAN_DATA_FRAME` som er brukt i dette tilfellet, vil ***TxHeader*** markere at det er en melding som blir sendt.
DataLength:	Sier hvor mange byte som sendes, og er satt til 8 byte i dette tilfellet.
ErrorState-Indicator:	Kan settes om man ønsker varsling ved feil i sending av meldinger. I dette tilfellet er den aktiv.
BitRateSwitch:	Avgjør om det skal kjøres ulik bitrate ved "arbitration" og sending av data. Dette er unikt for CAN-FD, men i dette tilfellet kjøres det CAN, og dermed er dette parametret av.
FDFormat:	Avgjør om det skal kjøres CAN-FD eller vanlig CAN. Som nevnt ovenfor kjøres det vanlig CAN.
TxEventFifo-Control:	Ikke i bruk
MessageMarker:	Ikke i bruk

Til slutt i funksjonen `oppstartCAN()`, konfigureres filteret etter filterinnstillingene og CAN-bussen startes. Avbrudd ved mottak av melding aktiveres i linje 12 til 14.

7.2.2 Filtrering av meldinger på CAN-buss

Når det settes strøm på mikrokontrollerene, initialiseres først CAN-bussen ved linjen `MX_FDCAN_Init();`, og deretter blir den lagde funksjonen `oppstartCAN(&hfdcan1);` kalt på. I `oppstartCAN` konfigureres flere ting, men her blir filterkonfigurasjonen fra kodeutdrag 7.4 gjennomgått.

Kode 7.4: Kodeutdrag for filterkonfigurasjon av mC (del av oppstartCAN())

```
1 // Configure filter for FDCAN1
2 FDCAN_FilterTypeDef sFilterConfig;
3 sFilterConfig.IdType = FDCAN_STANDARD_ID;
4 sFilterConfig.FilterIndex = 0;
5 sFilterConfig.FilterType = FDCAN_FILTER_MASK;
6 sFilterConfig.FilterConfig = FDCAN_FILTER_TO_RXFIFO0;
7 sFilterConfig.FilterID1 = 0x00;                    // FilterID1 fra listen ...
      nedenfor settes her. Definerer filterID
8 sFilterConfig.FilterID2 = 0x00;                    // FilterID2 fra listen ...
```

```
      nedenfor settes her. Definerer maskeID
9
10 /* ---------FilterID1 og FilterID2 for de ulike gruppene--------- */
11 // Reguleringskort: FilterID1 = 0x20, FilterID2 = 0xE0 --> Slipper gjennom ideer ...
      mellom 32-63 (0x20 - 0x3F)
12 // Sensorskort: FilterID1 = 0x40, FilterID2 = 0xE0 --> Slipper gjennom ideer ...
      mellom 64-95 (0x40 - 0x5F)
13 // Kraftkort: FilterID1 = 0x60, FilterID2 = 0xE0 --> Slipper gjennom ideer mellom ...
      96-127 (0x60 - 0x7F)
14 // Kommunikasjonskort: FilterID1 = 0x80, FilterID2 = 0xE0 --> Slipper gjennom ...
      ideer mellom 128-159 (0x80 - 0x9F)
15 // Sett inn rett ID-er for din gruppe
```

IdType:	Definerer om det er standard-ID på 11 bit eller utvidet-ID på 29 bit som er brukt på CAN-bussen. I dette tilfellet er det brukt standard-ID.
FilterIndex:	Brukes til å velge mellom ulike filter hvis det er flere filter konfigurert. I dette tilfellet er det kun behov for et filter, og den er dermed satt til 0.
FilterType:	Brukes for å velge hvilke type filter som skal brukes. Her kan det velges to filter som filtrerer ut ID-er mellom ID1 og ID2, et for standard-ID og et for utvidet-ID[66]. Det kan også velges et filter som spesifikt filtrerer ut ID1 og ID2. Den siste filtertypen er maskefilter filtrerer ID-er som slipper gjennom filter-ID (ID1) og maske-ID (ID2). Her er det valgt å bruke et maskefilter. En forklaring på hvordan dette fungerer kommer etter kodeutdraget.
FilterConfig:	Bestemmer hva som skjer med meldinger som slipper gjennom filteret. I dette tilfellet sendes meldingene til bufferen ***RXFifo0*** hvor meldingen lagres midlertidig før den sendes der den skal.
ID1 og ID2:	Brukes til å sette grensene til filtertypen som nevnt under ***FilterType***.

I linje 10 til 14 i kodeutdraget ovenfor, er det listet opp ulike ID-er som skal brukes i filtrene til de ulike gruppene. ID-ene som skal filtreres strekker seg fra 32 til 159. Der meldinger som skal til reguleringskortet har den høyeste prioriteten, og meldinger til kommunikasjonskortet har den laveste prioriteten.

For å illustrere hvordan maskefiltrering som er brukt her fungerer og settes opp, blir det videre gitt et eksempel på hvordan dette gjøres. Filteret skal i dette eksempelet slippe gjennom ID-er mellom 160 og 167. ID 160 er representert binært som $\boxed{1010\ 0000}$ og 167 er representert binært som $\boxed{1010\ 0111}$. Som filter-ID (ID1 i kodeutdrag), brukes den laveste adressen som skal slippes gjennom. I dette tilfellet er det 160, som er representert som 0xA0 på heksadesimal form. Noen av bittene i filter-ID vil bli sammenlignet med innkommende ID for å avgjøre om den skal aksepteres eller ikke. Hvilket av bittene som sammenligne blir bestemt av maske-ID.

For å finne maske-ID (ID2 i kodeutdrag), kan man sammenligne alle ID-ene i intervallet man vil slippe gjennom filteret. De gjenngående bitposisjonene i ID-ene som ikke endrer seg er markante. Disse settes da som 1 i maske-ID. Utledning 7.5 nedenfor av maske-ID illustrerer dette bedre.

$$
\begin{aligned}
ID\ 160: &\ 1010\ 0000 \\
ID\ 161: &\ 1010\ 0001 \\
ID\ 162: &\ 1010\ 0010 \\
ID\ 163: &\ 1010\ 0011 \\
ID\ 164: &\ 1010\ 0100 \\
ID\ 165: &\ 1010\ 0101 \\
ID\ 166: &\ 1010\ 0110 \\
ID\ 167: &\ \underline{1010\ 0111} \\
MaskeID = &\ \underline{\underline{1111\ 1000}}
\end{aligned} \tag{7.5}
$$

I utledning 7.5 ovenfor kommer det tydelig frem hvordan de fem første bittene fra venstre ikke endrer seg mellom ID 160 og 167. Som nevnt tidligere, settes de samme bittene i maske-ID like 1. Da blir maske-ID $\boxed{1111\ 1000}$, og kan representeres som 0xF8 heksadesimalt. En maske-ID som dette gjør at det kun er de fem første bittene i innkommende ID og filter-ID som sammenlignes.

Ved hjelp av filter-ID og maske-ID skal kun ID-er fra 160 til og med 167 slippe gjennom filteret. Filter-ID-en sammenlignes med innkommende ID, og maske-ID-en avgjør hvilke bit i filter-ID og innkommende ID som skal sammenlignes. Er bittene som skal sammenlignes (bruker OG-operator for sammenligning) like i både filter-ID og innkommende ID, vil meldingen slippe gjennom filteret. For bittene hvor maske-ID er 0, har det ingenting å si hva bittene i filter-ID og innkommende ID er.

I eksempelet nedenfor har vi en innkommende ID 168 eller $\boxed{1010\ 1000}$ binært. Med masken som den er satt opp, vil bit 1 til 3 bli ignorert(markert med "x" i resultat). Bit 5 til 8 i innkommende ID og filter-ID blir sammenlignet, og resultatet av disse blir godkjent. Men når bit 4 skal testes, blir resultatet lik 0, og dermed blir ID 168 avvist av filteret.

	Dec	Bin	Hex
Filter-ID	160	1010 0000	0xA0
Mask-ID	248	1111 1000	0xF8
Inn-ID	168	1010 1000	0xA8

x = don't care										
Inkommende ID		1	0	1	0	1	0	0	0	
Filter ID	&	1	0	1	0	0	0	0	0	
Før maske	=	1	1	1	1	0	1	1	1	
Maske ID		1	1	1	1	1	0	0	0	
Etter maske	=	1	1	1	1	0	x	x	x	Ikke godkjent

Figur 7.4: Eksempel på maskefiltrering av ID 168

.

Med 165 som innkommende ID, gitt som $\boxed{1010\ 0101}$ binært, blir alle bittene som skal sammenlignes lik 1. Denne ID-en slipper dermed gjennom filteret. Slik vil det være for alle ID-er mellom 160 og 167.

	Dec	Bin	Hex	x = don't care									
Filter-ID	160	1010 0000	0xA0										
Mask-ID	248	1111 1000	0xF8										
Inn-ID	165	1010 0101	0xA5	Inkommende ID		1	0	1	0	0	1	0	1	
				Filter ID	&	1	0	1	0	0	0	0	0	
				Før maske	=	1	1	1	1	1	0	1	0	
				Maske ID		1	1	1	1	1	0	0	0	
				Etter maske	=	1	1	1	1	1	x	x	x	Godkjent

Figur 7.5: Eksempel på maskefiltrering av ID 165

.

I eksempelet ovenfor er de tre laveste bittene i maske-ID satt til 0, og de tre laveste bittene i innkommende-ID og filter-ID vil dermed ikke bli sammenlignet ved OG-operasjonen. Det vil si at ID-en har et område på $2^3 = 8$ ID-er fra 160 til og med ID 167 før resultatet ikke blir godkjent. Hadde de fire laveste bittene vært satt til 0, hadde ID-en hatt et område på $2^4 = 16$ før resultatet ikke hadde blitt godkjent.

Ved konstruksjon av et maskefilter, bør enn velge ID-er hvor det er 8, 16, 24, 32, 40 osv ID-er(eller bit) mellom øvre og nedre grense, som det for eksempel er gjort i dette eksempelet hvor det er 8 bit mellom 160 og 167. Dette er slik det ble gjort i fjor [31] og i år, ved at det er 8 filtergrupper hvor hver filtergruppe var på 32 ID-er.

7.2.2.1 Sending data på CAN-buss

For sending av meldinger på CAN-bussen er funksjonen `sendCAN()` laget. Denne funksjonen tar inn en 16-bits ID og setter ***identifier*** i ***TxHeader*** lik denne. Det blir også tatt inn en peker som peker på strukturen som inneholder konfigurasjonen av CAN-bussen. HAL-funksjonen `HAL_FDCAN_AddMessageToTxFifoQ` blir kalt opp etterpå og tar inn CAN-strukturen, ***TxHeader*** og ***TxData***. Det er denne funksjonen som faktisk behandler og sender meldingen på CAN-bussen.

Kode 7.5: Funksjon for sending på CAN-buss

```
1 void sendCAN(uint16_t id, FDCAN_HandleTypeDef *canPort) {   // For sending av ...
      melding pÃ¥ CAN-bussen
2     TxHeader.Identifier = id;
3     HAL_FDCAN_AddMessageToTxFifoQ(canPort, &TxHeader, TxData);
4 }
```

Datapakkene som skal sendes er som regel bygget opp av data fra flere ulike variabler. Dette kan for eksempel være data fra IMU-en,[4] hvor posisjonene i X-, Y- og Z-retning lagres i ulike variabler - men skal sendes i samme datapakke.

For å løse dette, er funksjonen `memcpy()` brukt. Denne funksjonen kopierer et gitt antall byte fra en variabel direkte til en annen variabel. Denne tar tre parameter, der den første parameteren er hvor dataen skal kopieres til - og den andre er adressen til byten som skal kopieres. Den siste parameteren spesifiserer hvor mange byte som skal kopieres.

Kode 7.6: Eksempel på bygging av datapakke før sending på CAN-buss

```
1  //    Bygger datapaken som skal sendes. Datapakken bestaar av 8 byte.
2  //    Koden nedenfor er bare et forslag pÃ¥ hvordan en datapakke kan settes sammen.
3
4  // Eksempel med 2x uint8, 1x uint16 og 1x uint32
5  memcpy(TxData,&forsteuint8, 1); // Kopierer 1 byte fra forsteuint8 til forste byte ...
       i TxData
6  memcpy(&TxData[2],&andreuint8, 1); // Kopierer 1 byte fra andreuint8 til andre ...
       byte i TxData
7  memcpy(&TxData[3],&forsteuint16, 2); // Kopierer 2 byte fra forsteuint16 til ...
       tredje og fjerde byte i TxData
8  memcpy(&TxData[5],&forsteuint32, 4); // Kopierer 4 byte fra forsteuint32 til ...
       femte, sjette, syvende og aattende byte i TxData
9
10 // Etter aa ha bygget pakken, kan meldingen sendes pÃ¥ bussen ved Ã¥ kalle opp ...
       sendCAN-funksjonen:
11       sendCAN(ID-paa-mottaker, &hfdcan1); //   ID-paa-mottaker settes til ID der ...
             meldingen skal gÃ¥
```

I kodeutdrag 7.6 ovenfor er det et eksempel hvor det bygges en datapakke med en byte fra to stykk uint8, to byte fra en uint16 og fire byte fra en uint32, til sammen 8 byte. De ulike bytene blir plassert i ***TxData***, og i linje 11 blir ***TxData*** sendt på CAN-bussen.

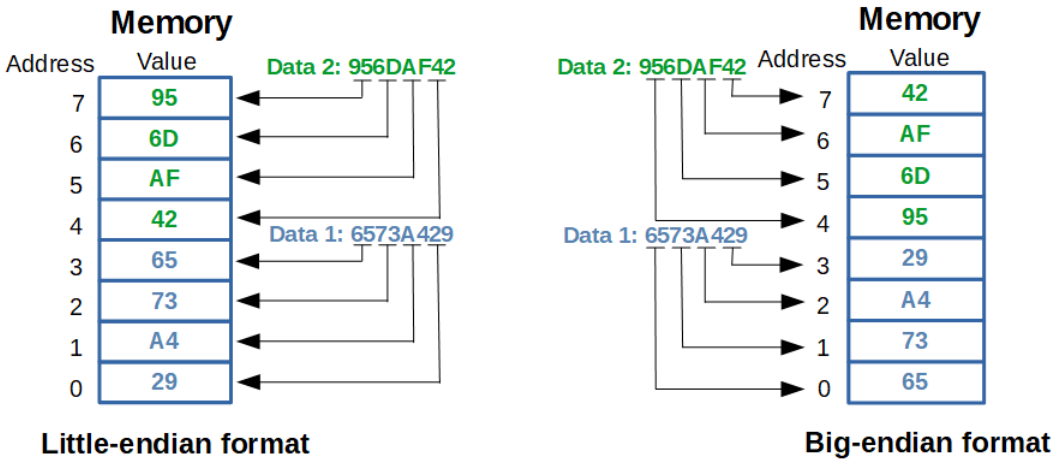
Det som kan være verdt å merke seg med funksjonen `memcpy()`, er hvordan den behandler datatyper som er mer enn en byte, altså uint/int16 og opp. Når en uint16 blir kopiert til ***TxData***, vil den mest signifikante byten (MsB) kopieres til den laveste adressen i ***TxData***, og den minst signifikante kopieres til den høyeste adressen.

[4] An inertial measurement unit (IMU) is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers.[170]

La oss si at den heksadesimale verdien for 10000 som er `0x2710` skal kopieres til de to første bytene i ***TxData***. Etter å ha kopiert dette til ***TxData***, vil den første byten i ***TxData*** være 0x10 og den andre vil være 0x27. Det samme vil gjelde for ***RxData*** hos mottakeren av denne meldingen. For at mottakeren skal klare lese 10000, må det gjøres en "byte-swap" slik at det leses 0x2710, og ikke 0x1027.

7.2.2.2 Minneformat og Little-endian

ARM-prosessoren på mikrokontrollerene baserer seg på minneformatet "*Little-endian*"(LE). Motparten til dette minneformatet er "*Big-endian*"(BE). Minneformatet sier noe om hvordan en sekvens med byte blir lagret i minnet. I et system som bruker BE-formatet, vil den mest signifikante byten (MsB) lagres i den laveste adressen i minneområdet. Mens den minst signifikante (LsB) blir lagret i den høyeste adressen[171]. For et LE-system vil det være motsatt. Dette er illustrert i figur 7.6 nedenfor.



Figur 7.6: Hvordan data blir lagret i de ulike minneformatene [172]

Bufferen ***TxData*** og ***RxData*** består av åtte stykk uint8, altså 8 byter. Når en uint16-variabel skal legges inn i ***TxData*** for sending, vil dataen til denne variabelen bli flyttet til minneområdet for ***TxData***. Da mikrokontrolleren er et LE-system, vil MsB legges i den høyeste adressen og LsB i den laveste adressen. Det er viktig å være klar over hvilke minneformat som brukes for å kunne sende og lese meldinger riktig mellom ulike enheter[61]. I dette prosjektet er både mikrokontrollerene og kommunikasjonskontrolleren Jetson Nano "*Little-endian*".

7.2.2.3 Mottak data på CAN-buss

I kodeutdrag 7.7 nedenfor er et forslag til hvordan de ulike gruppene kan behandle mottak av meldinger på CAN-bussen. Når en melding slipper gjennom filteret og blir lagt inn i bufferen ***RxFifo0***, blir det laget et avbrudd som kaller på funksjonen `HAL_FDCAN_RxFifo0Callback()` med strukturen til CAN-bussen og flagget ***RxFifo0ITs*** som sier om det er meldinger som venter på å bli mottat i bufferen.

Funksjonen `HAL_FDCAN_GetRxMessage()` leser meldingen fra bufferen ***RxFifo0*** og plasserer denne i ***RxData***. Headeren til meldingen blir også mottat og legges inn i ***RxHeader***. Deretter vil `HAL_FDCAN_ActivateNotification()` funksjonen sette flagget til avbruddet lavt igjen.

Kode 7.7: Mottak av meldinger på CAN-buss

```
1  // FDCAN1 Callback
2  void HAL_FDCAN_RxFifo0Callback(FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo0ITs){
3    if((RxFifo0ITs & FDCAN_IT_RX_FIFO0_NEW_MESSAGE) != RESET){
4      /* Retreive Rx messages from RX FIFO0 */
5      if (HAL_FDCAN_GetRxMessage(hfdcan, FDCAN_RX_FIFO0, &RxHeader, RxData) != HAL_OK){
6      /* Reception Error */
7      Error_Handler();
8      }
9      if (HAL_FDCAN_ActivateNotification(hfdcan, FDCAN_IT_RX_FIFO0_NEW_MESSAGE, 0) ...
           != HAL_OK){
10       /* Notification Error */
11       Error_Handler();
12     }
13     switch (RxHeader.Identifier) {      // Leser ID paa motatt melding. Casene maa ...
           lages ut fra ID-er enn mottar. (Sjekk "Interface-Overview" under ...
           "Overordnet prosjekt")
14         case 32:
15             //Kode
16             break;
17         case 33:
18             //Kode
19             break;
20         case 63: // IKKE FJERN DENNE                    // Endres til 63 for ...
               Regulering, til 95 for Sensor og til 127 for Kraft.
21             memcpy(&TxData, (uint8_t *) &"polo!\n", 6);
22             sendDataCAN(155);                           // Endres til 155 for ...
                   Regulering, til 156 for Sensor, til 157/158/159 for Kraft1/2/3.
23         }
24      }
25     }
26 }
```

Når en melding blir mottat på CAN-bussen, må enheten som mottok meldingen avgjøre hva den skal gjøre med meldingen. For å gjøre dette, kan det brukes en "switch" med "case"-betingelser. Et forslag på dette er også gitt i kodeutdrag 7.7. Her brukes ID-en som ligger i `RxHeader.Identifier` til å avgjøre hvilken kode som kjøres. Et eksempel på dette er "case 63" som blir satt når kommunikasjonskontrolleren sender "marco" på ID 63. Da svarer enheten som mottok denne meldingen med "polo!" på ID 155. Skal det kjøres mer krevende kode på en ID, vil det være hensiktsmessig å sette et flagg i "case"-en for så å kjøre denne koden utenfor avbruddet. Dette fordi krevende kode kan ta "lang" tid å kjøre, og mikrokontrolleren vil ikke kunne motta en ny melding før den har kommet seg ut av funksjonen `HAL_FDCAN_RxFifo0Callback()`.

7.2.3 Oppsett for CAN-buss på Jetson

For at kommunikasjonskontrolleren skal kunne kommunisere sammen med de andre mikrokontrollerene ombord ROV-en, må kommunikasjonskontrolleren kjøre med samme bitrate som mikrokontrollerene. På Jetson kan bitraten enkelt spesifiseres med en linje kode, og så blir de ulike parameterene satt. I figur 7.7 nedenfor er disse parametrene gitt ved en bitrate på 500kbps.

```
jetson@jetson-desktop:~$ sudo ip -d -s link show can0
4: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state UP mode DEFAULT group default qlen 10
    link/can  promiscuity 0
    can state ERROR-ACTIVE restart-ms 0
          bitrate 500000 sample-point 0.800
          tq 200 prop-seg 3 phase-seg1 4 phase-seg2 2 sjw 1
          mcp251x: tseg1 3..16 tseg2 2..8 sjw 1..4 brp 1..64 brp-inc 1
          clock 5000000
          re-started bus-errors arbit-lost error-warn error-pass bus-off
          0          0          0          1          1          0         numtxqueues 1 numrxqueues 1 gso_max_size 65536 gso_max_segs 65535
    RX: bytes  packets  errors  dropped overrun mcast
    803104     100388   1       0       1       0
    TX: bytes  packets  errors  dropped carrier collsns
    803120     100390   0       0       0       0
```

Figur 7.7: Parametrene for CAN-buss og bittiming på Jetson

Fra figuren ovenfor kommer det fram at Jetson bruker $BRP = 1$, $Prop\_Seg = 3$, $Phase\_Seg1 = 4$ og $Phase\_Seg2 = 2$, for å oppnå en bitrate på $\boxed{CAN_{bitrate} = 500kbps}$ og en nominell bittid på $\boxed{t_{NBT} = 2000ns}$. Avlesningspunktet er på 80%. Systemklokken er på $\boxed{f_{sys} = 5MHz}$, som er halvparten av krystalloscillatorens frekvens. Ved å bruke formlene fra delkapittel 3.4.3.4 og 3.4.3.5, kan dette verifiseres.

$$\begin{aligned} CAN_{bitrate} &= \frac{f_{sys}}{BRP \cdot (Sync\_Seg + Seg1 + Seg2)} \\ &= \frac{f_{sys}}{BRP \cdot (Sync\_Seg + Prop\_Seg + Phase\_Seg1 + Phase\_Seg2)} \\ &= \frac{5 \cdot 10^6}{1 \cdot (1+3+4+2)} = \frac{5 \cdot 10^6}{1 \cdot 10} = 500kbps \end{aligned} \tag{7.6}$$

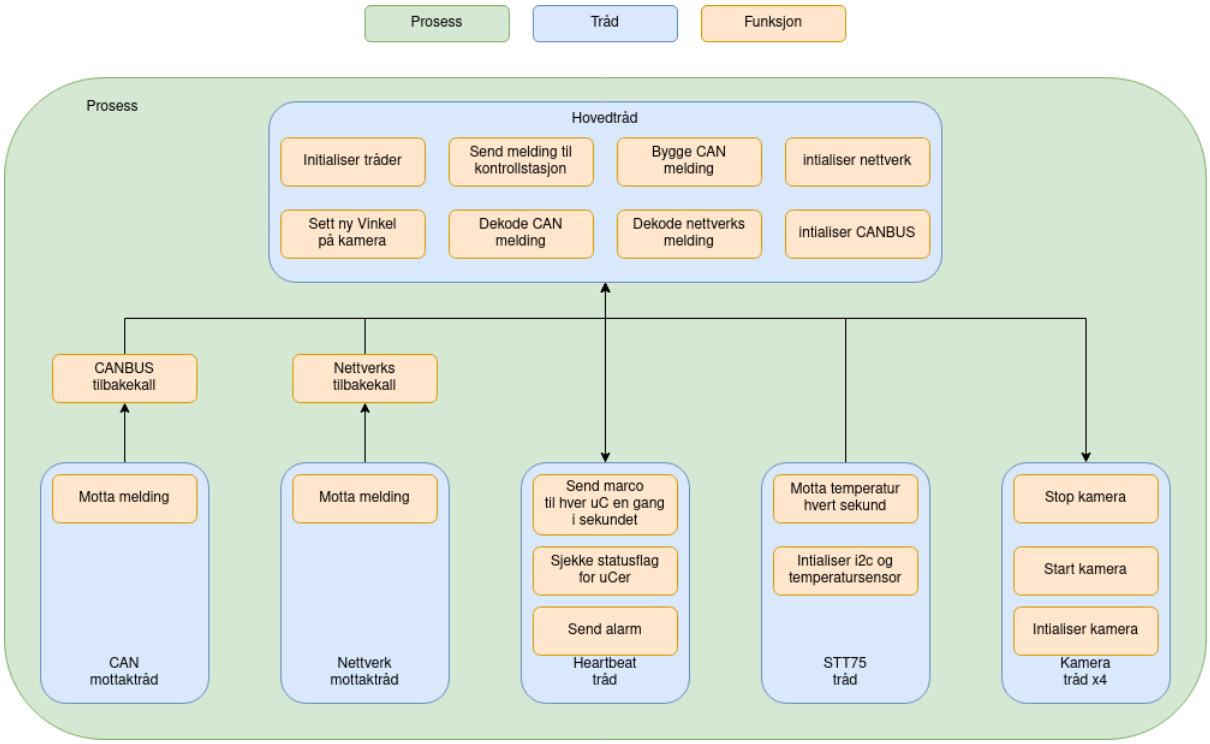
Som vist i beregningene ovenfor, gir parametrene i figur 7.7 en ønsket bitrate på 500kbps. Hvert tidskvantum er her på 200ns. Avlesningspunktet på 80% kan verifiseres på følgende vis:

$$Avlesning[\%] = \frac{(Sync\_Seg + Seg1)}{N_{tidskvantum}} \cdot 100 = \frac{(1+3+4)}{10} \cdot 100 = 80\% \tag{7.7}$$

Ved å bruke parametrene: $BRP = 1$, $Seg1 = 7$ og $Seg2 = 2$, vil Jetson også kjøre med en bitrate på 500kbps, og vil derfor være i stand å kommunisere med de andre mikrokontrollerene ombord i ROV-en.

7.3 Programvare for kommunikasjonskontroller

Kommunikasjonen mellom kontrollstasjon og ROV-en blir utført av kommunikasjonskontrolleren Nvidia Jetson Nano. Oppgavene til kommunikasjonkontrolleren er tredelt. Den ene oppgaven er å kommunisere med ROV-systemene over CAN-buss. Den andre er å kommunisere med kontrollstasjonen ved bruk av TCP over Ethernet. Den tredje oppgaven er å hente inn inn video og komprimere denne, for så å sende den opp til kontrollstasjonen. I tillegg til kommunikasjon, skal kommunikasjonskontrolleren også ha mulighet for å: Starte og stoppe GStreamer-rør for sending av video, avlese temperaturen på kretskortet ved bruk av en I2C temperatursensor, samt gi ut et PWM-signal for å justere vinkelen på stereokamera.



Figur 7.8: Flytskjema for funksjonalitet i programmet til kommunikasjonskontroller

.

I figur 7.8 ovenfor er en oversikt over tråder og funksjoner som programmet inneholder. For å realisere blokkene vist i figuren ovenfor, er det laget flere klasser og funksjoner i programmeringsspråket Python. Det er også laget en "Device-tree overlay" som vi bruker til å initialisere og sette opp GPIO-pinnene som trengs. For å klargjøre systemet ved oppstart er det også laget en "service" som kjører igang CAN-bussen.

7.3.1 Device-tree

Device-tree i Linux er en datastruktur for å beskrive hvilken maskinvare som er koblet til og hvordan den er tilkoblet. For å aktivere SPI, I2C og PWM-signal lages det en "dts"-[5] fil som vi kompilere til en "dtbo" [6] ved bruk av en "devicetree compiler". Dtbo-filen blir lagt inn i "devicetree" når maskinvaren startes. Deretter modifiseres det eksisterende treet slik at maskinvaren man ønsker å bruke, blir lagt til. Man setter opp "overlayet" ved å navngi og spesifisere hvilken maskinvare oppsettet er kompatibelt

[5] *Device-tree source file*
[6] Device-tree blob overlay

med. For overlayfilen vi har laget, ble det tatt utgangspunkt i "Linux for tegra [173]"[7] kernelen for Nvidia Jetpack 4.5.1 [174].

Kode 7.8: DTO filspesifikasjon

```
overlay-name = "UiS Subsea DTO CAN,PWM,I2C";
jetson-header-name = "Jetson 40pin Header";
compatible = ...
    "nvidia,p3449-0000-b00+p3448-0000-b00\0nvidia,p3449-0000-a02+p3448-0000-a02";
```

Man deler så opp de forskjellige funksjonene i fragmenter. Vi lager et fragment for klokken til CAN-bussen, et fragment til "SPI1" og et siste fragment for "pinmux" der vi velger hvilken funksjon hver pinne som vi bruker skal ha.

Kode 7.9: Fragment for CAN-klokke

```
fragment@0 {
        target-path = "/";
        __overlay__ {
            clocks {
                can_clock: can_clock {
                    compatible = "fixed-clock";
                    #clock-cells = <0>;
                    clock-frequency = <10000000>;
                    clock-accuracy = <100>;
    };};};};
```

I fragment 0 vist ovenfor settes det opp en fast klokke med frekvens 10MHz som skal ha en nøyaktighet på 100Hz. Denne frekvensen må være den samme som klokken på CAN-modulen SPI CAN Click, som er forklart i kapittel om maskinvare 4.4.4.2.

Kode 7.10: Fragment for SPI0

```
fragment@1 {
        target = <&spi0>;
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <0>;
            spi@0 {
                status = "okay";
                compatible = "microchip,mcp2515";
                reg = <0x0>;
                spi-max-frequency = <5000000>;
                nvidia,enable-hw-based-cs;
                nvidia,rx-clk-tap-delay = <0x7>;
                clocks = <&can_clock>;
                interrupt-parent = <&gpio>;
                interrupts = <TEGRA_GPIO(Z, 0) 0x1>;
                controller-data {
                    nvidia,cs-setup-clk-count = <0x1e>;
                    nvidia,cs-hold-clk-count = <0x1e>;
                    nvidia,rx-clk-tap-delay = <0x1f>;
                    nvidia,tx-clk-tap-delay = <0x0>;
                };};
            spi@1 {status = "disabled";};};};
```

I fragment 1 vist ovenfor settes perifermodulen for SPI0 opp. Her velges det hvilken modul den er kom-

[7] Tegra er ARM brikken Nvidia Jetson Nano er basert på

patibel med, som er CAN-kontrolleren MCP2515. SPI-kommunikasjons frekvensen settes, "can_clock" klokken fra fragment 0 knyttes opp, velger pinne for avbrudd, samt setter opp "Chip-select" og utgangssignalet på klokken for SPI-kommunikasjonen.

Kode 7.11: Fragment for pinmux

```
fragment@2 {
        target = <&pinmux>;
        __overlay__ {
            pinctrl-names = "default";
            pinctrl-0 = <&hdr40_pinmux>;

            hdr40_pinmux: header-40pin-pinmux {
                pin24 {
                    nvidia,pins = "spi1_cs0_pc3";
                    nvidia,function = "spi1";
                    nvidia,pull = <TEGRA_PIN_PULL_UP>;
                    nvidia,tristate = <TEGRA_PIN_DISABLE>;
                    nvidia,enable-input = <TEGRA_PIN_ENABLE>;
                    };
            };};};};};
```

I fragment 2 setter vi opp "pinmux"-en som fysisk kobler pinnene til periferimodulen den skal kobles mot. I kodeutdraget 7.11 er alle pinner utenom pinne 24 fjernet. Pinne 24 har vi satt opp til å være "Chip-select" og dras opp til 3.3V når den aktiveres. På Github [175] ligger den komplette "dts"-filen der alle pinnene er satt opp.

7.3.2 Python

I prosjektet ble høynivåprogrammeringspråket Python, valgt for å programmere kommunikasjonskontrolleren. Dette valget ble tatt med bakgrunn av tidligere erfaring med Python og fordeler med språket for hurtig prototyping. Mengden ekstrapakker som finnes i Python er også en fordel. Pythonpakker blir mer forklart i neste delkapittel 7.3.2.1. Språket har en enklere syntaks og er svært populært. Det egner seg godt til prototyping da det er et tolket språk som gjør at en slipper å kompilere koden hver gang en gjør endringer. Tolkingen av koden skjer med at man kjører programmet gjennom en tolker som konverter til maskinkode når programmet kjører [176]. Ulempen med dette er at gjennomkjøringstiden blir lengre. Python er i dag et av de mest populære programmeringsspråkene på grunn av disse funksjonene.

7.3.2.1 Pythonpakker

En Python-pakke er en samling av moduler, filer og kode som legger til funksjonalitet i applikasjonen man utvikler. Man legger pakkene til i prosjektet sitt ved å importere pakkene man ønsker å bruke. Pakkene som er blitt brukt for å utvikle koden til kommunikasjonskontrolleren går vi igjennom nedenfor:

- **Socket**-pakken gir et lavnivå grensesnitt for å programmere nettverkskommunikasjon. Vi bruker pakken til TCP-kommunikasjonen mellom kommunikasjonskontrolleren og kontrollstasjonen.
- **Threading**-pakken gir mulighet for tråd basert parallellisme og muliggjør at vi kan kjøre flere prosesser på likt. Tråder i Python gir ikke fullstendig parallellisme. Dette blir utdypet i delkapittel 7.3.2.2.

- **Sys**-pakken gir tilgang til noen systemvariabler og systemfunksjoner som for eksempel gir tilgang til nye filstier, sjekke enkoding på filer eller sjekke hvilken plattform koden kjører på.
- **Struct**-pakken brukes til å konvertere mellom Python-verdier og C-"structs". Vi bruker dette til å dekode og enkode pakkene som mottas eller sendes på CAN-bussen.
- **Time**-pakken gir tilgang til flere tidsfunksjoner. Noen av de funksjonene vi bruker er å ta nåværende tid og få programmet til å vente med "time.sleep()".
- **Json**-pakken er en enkoder- og dekodermodul for "JSON"-formatet. JSON er formatet som er valgt for å sende data mellom kommunikasjonskontroller og kontrollstasjonen.
- **Pytcon-CAN** er en pakke som gir støtte for CAN-buss i Python. Pakken har også mulighet for å sette opp filter, CAN-FD, eller detektere feil. Den gir også funksjonalitet for å sette opp lyttere og "Notifier" for mottak av meldinger i en egen tråd.

7.3.2.2 Tråder

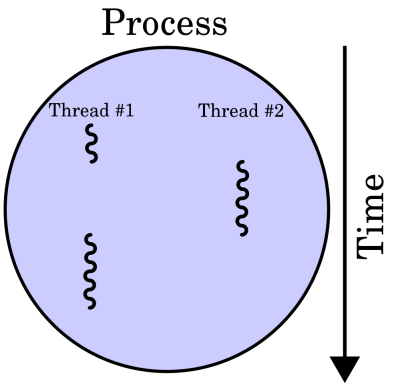
For å realisere ønsket funksjon av programvaren, tas det i bruk tråder. En tråd i programmering er en måte å dele oppgavene til koden i forskjellige deler slik at man kjører oppgavene samtidig.

Motivasjonen for bruk av tråder kommer fra at mottak av data over CAN og TCP, er blokkerende funksjoner. Dette betyr at programmet ikke går videre før den mottar en melding når disse funksjonene blir kalt. Et eksempel på dette er når det mottas en melding fra kontrollstasjonen over TCP. Da vil det ikke - uten tråder, være mulig å motta meldinger på CAN-bussen fra mikrokontrollerene samtidig. Tråder løser dette problemet, og programmet er derfor delt opp i flere tråder.

Tråd: En tråd innen programmering og maskinvare har forskjellige betydninger. Vi velger å skille dette i prosessortråd for maskinvare og programvaretråd for programmering.

En "CPU" eller prosessor har normalt sett i dag flere kjerner. Hver av disse kjernene kan ha flere tråder for å kjøre instruksjoner. Dette betyr at man kan kjøre instruksjoner parallelt. Dette er noe vi ikke har kontroll over i Python og dermed ikke kan utnytte.

En programvaretråd skiller som sagt ut oppgaver i applikasjonen slik at oppgavene kan kjørres parallelt. Python har begrensinger med at en prosess kun kan kjøres på en CPU-tråd. Ut fra dette bruker man da programvaretråder.



Figur 7.9: Illustrasjon på hvordan trådene til en prosess kjører. [177]

.

I figur 7.9 er det illustrert hvordan en programvareprosess der trådene ikke kjøres i parallell på maskinvarenivå, men maskinvaren bytter mellom programmvaretrådene så fort at det tilsynelatende kjøres parallelt.

I Python lager man en tråd ved å bruke "Threading"-biblioteket.

Kode 7.12: Oppsett og starting av tråd

```
1    enTraad = threading.Thread(name="minTraad", target=funksjon, daemon=True, ...
         args=(arg1, arg2, arg3))
2    enTraad.start()
```

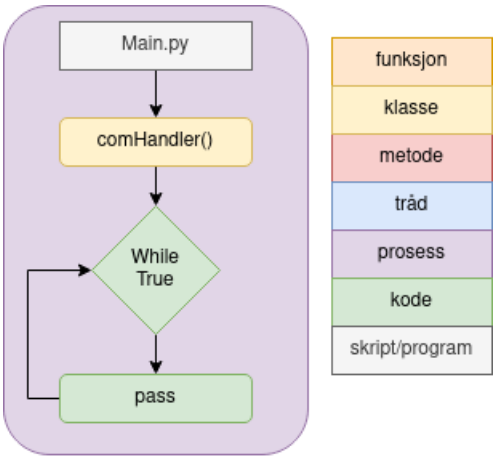
I kodeutdrag 7.12 lages først tråden "enTraad" med "threading"-pakken sin "thread"-modul. Tråden får et navn, og en funksjon som den skal kjøre. "daemon" betyr at variablene man sender inn skal bli delt mellom trådene. "args" spesifiserer hva man sender inn i funksjonen til tråden. Dette kan for eksempel være variabler, en funksjon, en metode, eller en klasse. Etter man har satt opp tråden, startes den med "enTraad.start()".

7.3.3 Pythonkode

I dette delkapittelet vil klassene og funksjonene som er laget for kommunikasjonskontrolleren gjennomgås, samt en forklaring på funksjonen de utfører. Koden i år følger en noe lik struktur som koden fra fjorårets bachelor[31]. Forskjellen er at det er lagt fokus på å gjøre den mer lesbar med at man bruker standardfunksjoner for pakker og at knytting av data som skal sendes skjer via hashtabeller.

Hovedprogrammet har en enkel oppgave, og det er å kjøre hovedklassen "comHandler()", som igjen har flere oppgaver.

Et enkelt flytskjema for denne prosessen er vist i figur 7.10 nedenfor.



Figur 7.10: Flytskjema for hovedprogram

.

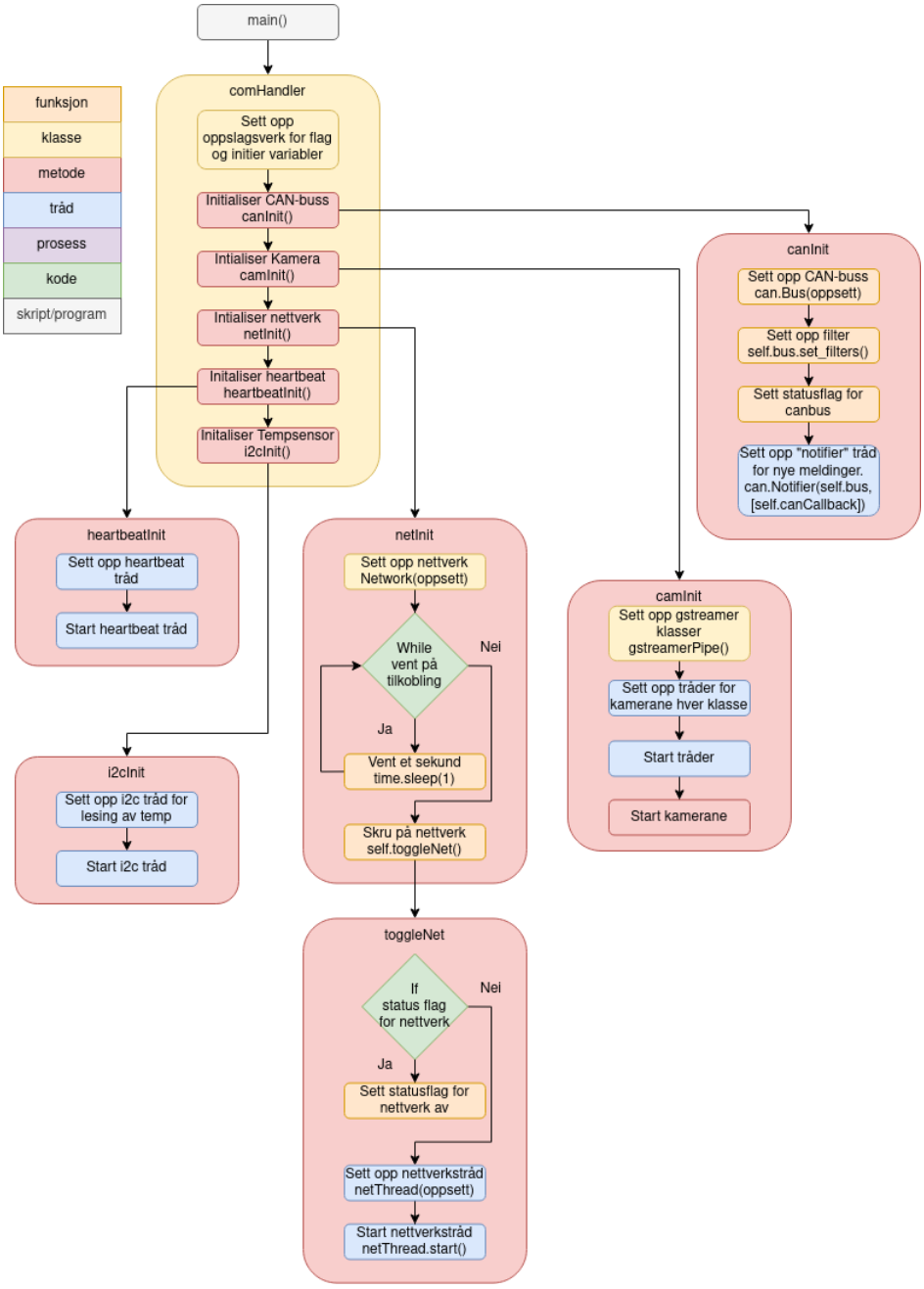
Hovedklassen "comHandler()" har flere oppgaver som er listet nedenfor:

- Initialisere klassene som er brukes
- Starte trådene som brukes

- Tilbakekall for mottatte TCP-meldinger
- Tilbakekall for mottatte meldinger på CAN-buss
- Sende TCP-meldinger
- Sende meldinger på CAN-buss
- Starte og stoppe GStreamer-rør for kamera.

7.3.3.1 Hovedprogram

comHandler()-klassen kan deles opp i to faser, initiering og kjøring. Et flytskjema som viser initialiseringsfasen er vist under i figur 7.11.



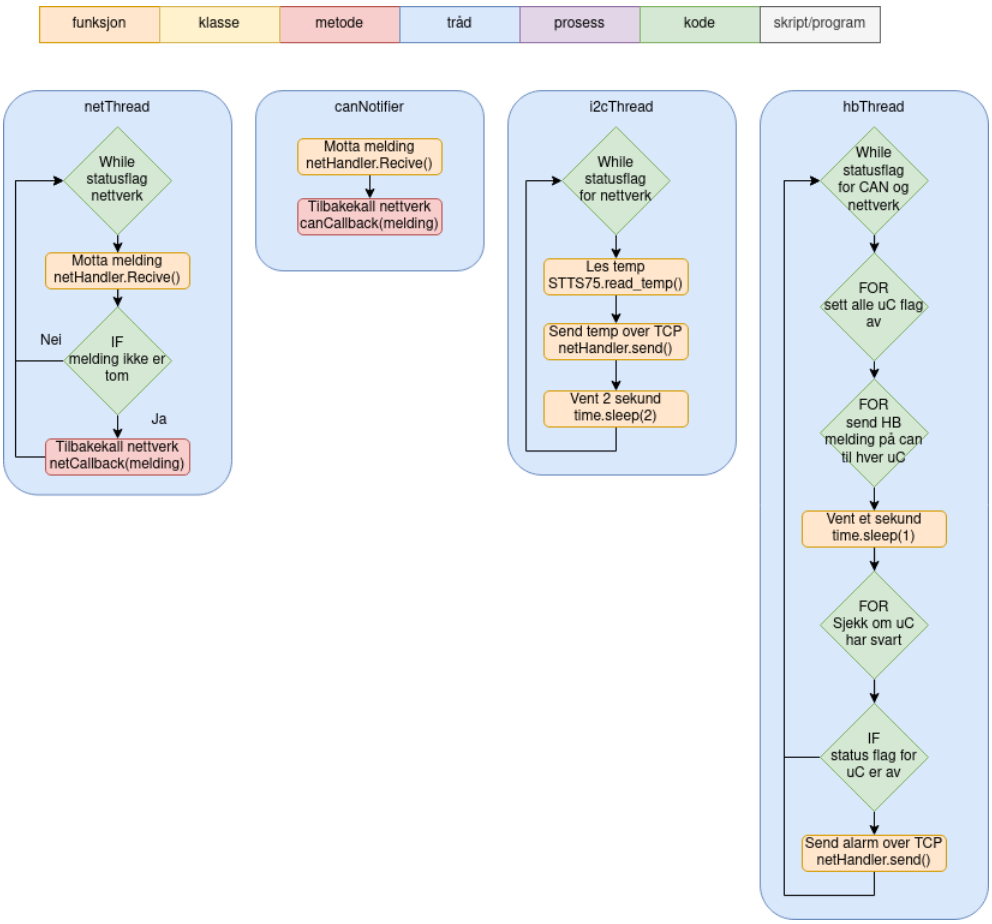
Figur 7.11: Initialiseringsprosess for **comHandler()**-klassen

.

Under initialiseringsfasen til klassen, setter den opp tabeller for statusflagg for trådene, mikrokontrollere og kamera. Klassen initierer så de forskjellige trådene og maskinvaren som programmet skal kjøre. Dette blir gjort av metodene forklart i listen under:

- **canInit()**-metoden setter først opp en instans for CAN-buss, "self.bus" med parameter for hva kanalen heter, hvilken type "socket" det er, om den skal motta egne meldinger og om det er CAN-FD eller vanlig CAN. Metoden setter så opp filteret, og setter deretter statusflagget for CAN på. Metoden setter så opp en notifier. Dette er en funksjon i "Python-CAN"-biblioteket som tar inn en buss og en funksjon eller metode som argument. Notifieren starter så en tråd som lytter på bussen og kaller den spesifiserte funksjonen eller metoden når det er mottat en ny melding på bussen.
- **camInit()**-metoden setter opp en "gstreamerPipe"-klasse for hvert kamera og starter så opp tråder for hvert av disse kameraene. Denne tråden tar inn den spesifikke "gstreamerPipe"-klassen som argument.
- **netInit()**-metoden setter opp "Network"-klassen med gitte parameter før den så venter på at en klient skal koble seg til. Når en klient har koblet seg til aktiverer den metoden **toggleNet()**.
- **toggleNet()**-metoden sjekker om systemflagget for nettverk er på. Hvis flagget er på, setter den flagget av. Er flagget av setter den opp og starter en tråd for nettverket. Tråden tar inn statusflagg for tråder, nettverksklassen og nettverks-tilbakekallet som argumenter.
- **heartBeat()**-metoden setter opp og starter en tråd for heartbeat-funksjonaliteten mellom kommunikasjonskontrolleren og mikrokontrollerene. Denne tråden tar inn statusflagg for mikrokontrollerene, statusflagg for tråder, metode for sending av CAN-meldinger og nettverksklassen som argumenter.
- **i2cInit()**-metoden initierer først STTS75-klassen som kommuniserer med en temperatursensor gitt av sensorgruppen. Metoden setter så opp og starter en tråd med STTS75 klassen, nettverksklassen og statusflaggene for tråder som argumenter.

Etter klassen er initiert, og hver av initaliseringsmetodene har kjørt - går vi inn i kjørefasen. I figur 7.12 er det vist et flytskjema for trådene.



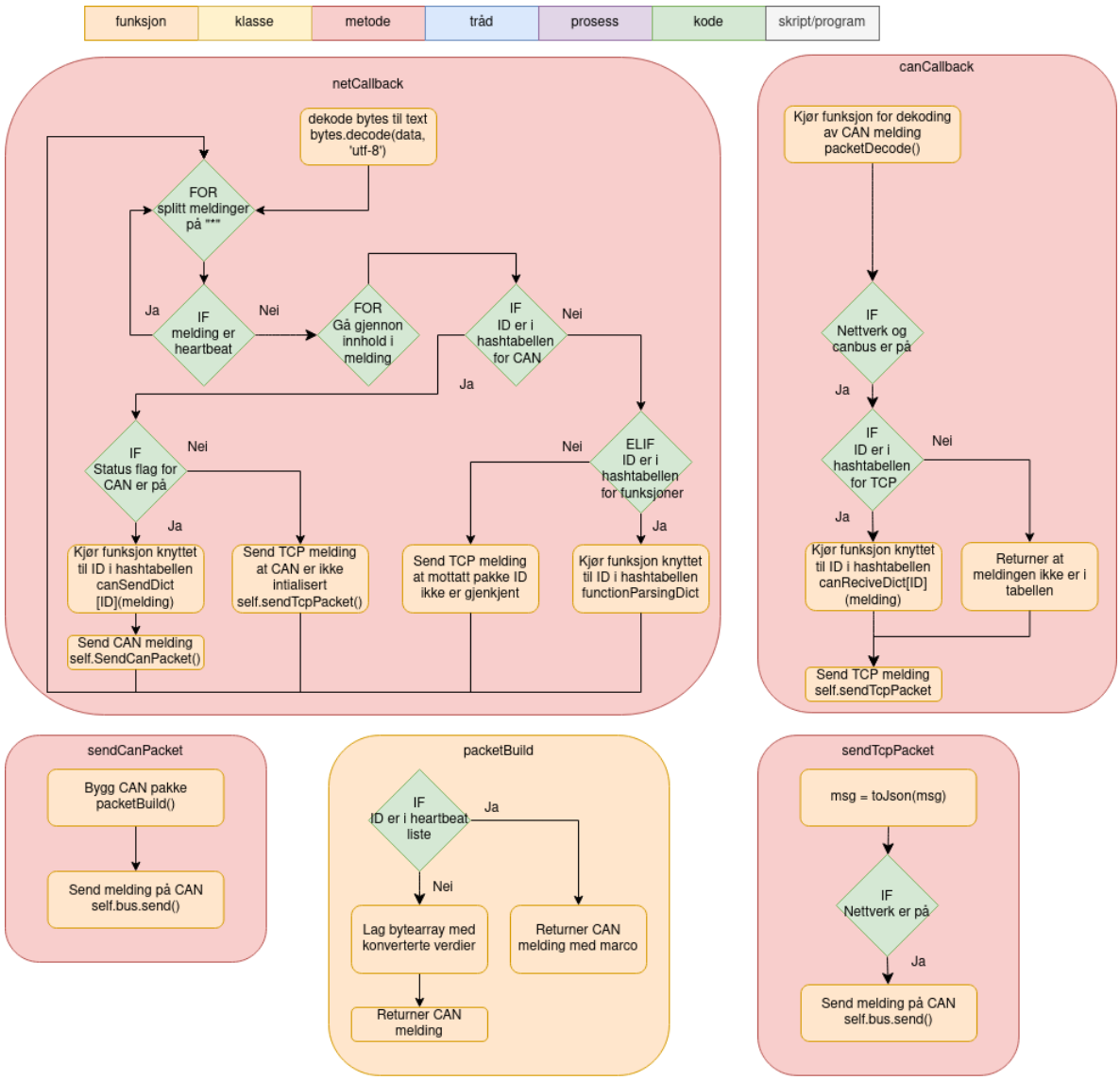
Figur 7.12: Kjøreprosessen for trådene

.

Trådene som kjøres blir forklart i listen under:

- **netThread()**-tråden kjører en "while"-løkke hvis nettverket er oppe. I løkken venter den på en melding og sjekker om det er innhold i meldingen. Hvis det er innhold i meldingen kaller den på tilbakekallsmetoden "netCallback()" i "comHandler()"-klassen.
- **canNotifier** ligger i "Python-CAN"-biblioteket og venter på en ny melding. Når den har mottatt en ny melding kaller den på tilbakekallsmetoden "canCallback()" i "comHandler()"-klassen
- **i2cThread()**-tråden kjører en "while"-løkke hvis nettverket er oppe. Løkken leser temperaturen fra STTS75-sensoren og sender temperaturen til kontrollstasjonen før den venter to sekunder.
- **hbThread()**-tråden kjøren en "while"-løkke hvis nettverket og CAN-bussen er oppe. Løkken setter først alle mikrokontrollerflaggene av. Den sender så ut en CAN-melding med "marco/n". Tråden venter så et sekund før den sjekker om statusflagget for mikrokontrollerene er på. Hvis et av flaggene ikke er på blir en alarm sendt til kontrollstasjonen.

Mottakstrådene aktiverer tilbakekall for behandling av motatt data. I figur 7.13 er det vist et flytskjema for de to tilbakekallsmetodene.



Figur 7.13: Tilbakekall for mottatte meldinger

.

7.3.3.1.1 Mottak av meldinger: Meldinger som blir mottatt, blir behandlet av tilbakekallsmetodene "netCallback" og "canCallback".

- **netCallback()**-metoden konverterer først innholdet i meldingen fra bytes til tekst i "utf-8"-formatet. Den splitter så opp meldingene til individuelle meldinger på den avtalte skille oppertøren, "*". Den går igjennom hver melding og konverterer til "JSON"-formatet. Det er så en "IF"-operatør som sjekker om meldingen er en heartbeat, som vil si den ikke skal viderebehandles. Hvis meldingen skal behandles sjekker den hvilken hashtabell ID-en hører inn i. I "netCallback()"-metoden er det to assosierte hashtabeller, "canSendDict" og "functionsParsingDict". Hvis ID-en er en "KEY" i tabellen for CAN, så sjekker den om CAN-bussflagget er på før den kjører funksjonen for å gjøre klar meldingen for "sendPacket"-metoden i "comHandler"-klassen. Hvis ID-en er en nøkkel i tabellen for funksjoner, så sjekker den hvilken aksjon den skal utføre med å sjekke om aksjonen er en nøkkel i en hashtabell som er innholdet til nøkkellen i funksjons hashtabellen. Den utfører så den gitte funksjonen med verdien gitt i meldingen. Nettverkstilbakekallet sender alarmer til kontrollstasjonen hvis ID-en ikke er en nøkkel i en av tabellene.
- **canCallback()**-metoden mottar en melding. Det blir først sjekket at statusflaggene til TCP-

kommunikasjonen og CAN-bussen er på. "Python-CAN"-bibliotektet blir så brukt til å sile ut Pakke-ID og dataen. Den kaller så på funksjonen assosiert med ID-en til pakken fra hashtabellen "canReciveDict" for å formatere bytearrayet i dataen til rett pakkestruktur før pakken sendes til kontrollstasjonen. Hvis ID-en ikke er knyttet til en funksjon, blir det sendt en feilmelding med at en ukjent pakke er mottatt.

7.3.3.1.2 Hashtabeller: For å gjøre det mer oversiktlig å legge til nye pakker som sendes, er det brukt hashtabeller ved å bruke Python-variabeltypen "Dictionary" som virker som et oppslagsverk. Hashtabellene er satt opp med at det gis en nøkkel som ikke kan endres på. Denne nøkkelen blir knyttet til en variabel som kan kalles på ved å spesifisere nøkkelen til tabellen.

Kode 7.13: Hashtabell

```
1  ROVCMD         = 40
2  MANICMD        = 41
3  SENSORFLAGS    = 66
4  SYS5VFLAGS     = 97
5  THR12VFLAGS    = 98
6  MANI12VFLAGS   = 99
7  canSendDict   = {
8         ROVCMD:        int8Parse,
9         MANICMD:       int8Parse,
10        SENSORFLAGS:   sensorflagsParse,
11        SYS5VFLAGS:    fuselightParse,
12        THR12VFLAGS:   fuselightParse,
13        MANI12VFLAGS:  fuselightParse
14      }
```

I dette kodeutradget 7.13 for hashtabellen som brukes for behandling av data mottatt over nettverket, lages det konstanter med ID-en som er knyttet pakken. I hashtabellen blir så hver nøkkel knyttet til funksjonen som skal brukes for å behandle pakken. [8]

7.3.3.1.3 Formateringsfunksjoner: For å kunne behandle de forskjellige pakkene som blir sendt, er det laget en rekke funksjoner. I mappen "functions" ligger det fem filer med funksjoner.

I "fFormating"-filen ligger det fire funksjoner for formatering av data. Disse er videreført fra i fjor[31] men skrevet noe om for å simplifisere dem. Funksjonene blir forklart i listen under:

- **getByte()**-funksjonen tar inn et argument som bestemmer formatet til dataen, for eksempel "'int16'" eller "'unit32'", og et argument som velger tallet som skal formateres. Den returnerer så en liste med tallet konvertert til bytes. Denne funksjonen blir ikke brukt i programmet men er likevel videreført hvis vi trenger funksjonaliteten.
- **getNum()**-funksjonen tar inn et argument som bestemmer formatet til dataen, for eksempel "'int16'" eller "'unit32'", og et argument med bytearrayet som skal formateres. Den returnerer så dette tallet.
- **getBit()**-funksjonen tar inn et tall, og et bit som skal sjekkes, og returner 0 eller 1 alt etter om bittet er lavt eller høyt.
- **setBit()**-funksjonen tar inn en liste med bits og returnerer det som en byte. Denne funksjonen blir ikke brukt i programmet men er likevel videreført hvis vi trenger funksjonaliteten.

[8] Parse eller Parsing innen programmering er å dele opp og stukturere data for videretolkning i et program

7.3.3.1.4 Funksjoner for netCallback: Når "netCallback"-metoden mottar en melding som skal sendes på CAN, kaller den først på funksjonen fra hashtabellen "canParsingDict". I kodeutradget under er en av disse funksjonene for å klargjøre en int16-melding.

Kode 7.14: int16Parse funksjon

```
1 def int16Parse(item):
2   msg = [item[0], ['int16', int(item[1][0])], ['int16', int(item[1][1])],
3                   ['int16', int(item[1][2])], ['int16', int(item[1][3])]]
4   return msg
```

Funksjonen lager en liste der ID-en kommer først. Den lager så en liste for hver "tag" som skal sendes. Listen for hver tag inneholder format og verdien.

I filen "fNetcallParsing.py" er det laget totalt 10 funksjoner for bruk i tabellen.

- **int8Parse** klargjør for en melding bestående av åtte int8-data.
- **uint8Parse** klargjør for en melding bestående av åtte uint8-data.
- **int16Parse** klargjør for en melding bestående av fire int16-data.
- **uint16Parse** klargjør for en melding bestående av fire uint16-data.
- **int32Parse** klargjør for en melding bestående av to int32-data.
- **uint32Parse** klargjør for en melding bestående av to uint32-data.
- **int64Parse** klargjør for en melding bestående av en int64 data.
- **uint64Parse** klargjør for en melding bestående av en uint64-data.
- **fuselightParse** klargjør melding for å skru av og på sikringer samt lys.
- **sensorflagsParse** klargjør melding for å sette styringsflagg for sensorkort.

7.3.3.1.5 Funksjoner for canCallback: Når "canCallback"-metoden mottar en melding som skal sendes videre på nettet, kaller den først på funksjonen fra hashtabellen "canReciveDict". Hver melding på CAN-bussen inneholder 8 byte med data. Det er derfor gjort et forsøk på å standardiserer pakkene til å inneholde 4 int/uint16 verdier eller 8 int/uint8 verdier i disse pakkene. Vi kan da lage funksjoner som kan brukes om igjen for å håndtere pakkene. Et eksempel på en slik funksjon er vist i kodeutradget under:

Kode 7.15: canint16Parse funksjon

```
1 def canint16Parse(canID, dataByte, ucFlags):
2   pack1 = getNum("int16", dataByte[0:2])
3   pack2 = getNum("int16", dataByte[2:4])
4   pack3 = getNum("int16", dataByte[4:6])
5   pack4 = getNum("int16", dataByte[6:8])
6   return {canID: (pack1, pack2, pack3, pack4)}
```

Denne funksjonen tar inn ID, et bytearray med data, og statusflagg for mikrokontrollerene. Funksjonen "getNum" blir kalt med formatet til dataen og aktuelle byte som skal behandles blir valgt ut. Funksjonen returnerer så en ferdig pakke som kan sendes til kontrollstasjonen.

Grunnen til "ucFlags" er med i alle funksjonene, er at vi bruker den i funksjonen "canHBParse" som vist i kodeutdraget 7.16 under:

Kode 7.16: canHBParse funksjon

```
  pack = dataByte[0:6].decode('utf-8')
  if pack == "polo!/n":
    if canID == 155:
        ucFlags['Reg'] = True
    elif canID == 156:
        ucFlags['Sensor'] = True
    elif canID == 157:
        ucFlags['12Vman'] = True
    elif canID == 158:
        ucFlags['12Vthr'] = True
    elif canID == 159:
        ucFlags['5V'] = True
  return {canID: (pack)}
```

Denne funksjonen sjekker om mottatt innhold i pakken er som forventet og setter på statusflagget for mikrokontrolleren hvis den har svart.

Det er i tillegg laget funksjoner for å videresende alarmer mottatt av sensorgruppen vist i kodeutdraget 7.17 under:

Kode 7.17: canSensorAlarmsParse funksjon

```
def canSensorAlarmsParse(canID, dataByte, ucFlags):
  imuErrors     = [False, False, False, False, False, False, False, False]
  tempErrors    = [False, False, False, False]
  pressureErrors= [False, False, False, False]
  lekageAlarms  = [False, False, False, False]
  for byteidx, byte in enumerate(dataByte):
    if byteidx == 0:
      for bit in range(8):
        if getBit(byte, bit):
          imuErrors[bit] = True
        else:
          imuErrors[bit] = False
    elif byteidx == 1:
      for bit in range(4):
        if getBit(byte, bit):
          tempErrors[bit] = True
        else:
          tempErrors[bit] = False
    elif byteidx == 2:
      for bit in range(4):
        if getBit(byte, bit):
          pressureErrors[bit] = True
        else:
          pressureErrors[bit] = False
    elif byteidx == 3:
      for bit in range(4):
        if getBit(byte, bit):
          lekageAlarms[bit] = True
        else:
          lekageAlarms[bit] = False
  return {canID: (imuErrors, tempErrors, pressureErrors, lekageAlarms)}
```

Denne funksjonen går gjennom hver byte, og tester hvert bit før den returnerer en liste med statusen på alarmene.

I filen "fCancallParse" er det lagd syv funksjoner for bruk i denne tabellen.

- **canint8Parse** klargjør pakke som sendes til kontrollstasjon fra pakke med åtte int8-verdier.
- **canuint8Parse** klargjør pakke som sendes til kontrollstasjon fra pakke med åtte uint8-verdier.
- **canint16Parse** klargjør pakke som sendes til kontrollstasjon fra pakke med fire int16-verdier.
- **canuint16Parse** klargjør pakke som sendes til kontrollstasjon fra pakke med fire uint16-verdier.
- **canSensorAlarmsParse** klargjør pakke som sendes til kontrollstasjon fra sensorkort med alarmer.
- **can12VParse** klargjør pakke som sendes til kontrollstasjon fra 12V-kort. Denne pakken inneholder alarmer og målinger.
- **canHBParse** sjekker om heartbeat fra mikrokontroller er mottatt.

7.3.3.1.6 Sending av meldinger: Når tilbakekallene har meldinger som skal sendes til ROV eller kontrollstasjon blir metodene "sendCanPacket" og "sendTcpPacket" brukt.

- **sendCanPacket**-metoden tar imot en melding og kjører funksjonen "packetBuild()" for konvertering til Python-CAN meldingsformatet, for så å sende meldingen ut på CAN-bussen.
- **sendTcpPacket**-metoden konverterer meldingen til Json-format, og så til bytes for å klargjøre til sending over TCP. Hvis nettverket er på, blir meldingen sendt.

7.3.3.1.7 Funksjoner for sending av meldinger: Når tilbakekallet for nettverks- eller canHB-tråden skal sende en melding, blir metoden "sendCanPacket()" brukt. Denne metoden kaller på funksjonen "packetBuild()" for å lage en CAN-melding som kan sendes.

Kode 7.18: packetBuild funksjon

```
1  can_types = {"int8": "<b", "uint8": "<B", "int16": "<h", "uint16": "<H", "int32": ...
       "<i", "uint32": "<I", "int64": "<q", "uint64": "<Q", "float": "<f"}
2  def packetBuild(tags):
3    if tags in [63, 95, 125, 126, 127]:
4      canID = tags
5      msg = bytearray('marco/n', 'utf-8')
6    else:
7      try:
8        canID, *idData = tags
9        idDataByte = []
10       for data in idData:
11         try:
12           idDataByte += struct.pack(can_types[data[0]], *data[1:])
13         except struct.error as e:
14           print(f"Error: {e}")
15           print(f"data={data}, format={data[0]}, value={data[1:]}")
16           print(f"idDataByte={idDataByte}")
17         msg = idDataByte
18     except ValueError as error:
19       return f"Error in building can message: {tags} "
20     print(msg)
21   return can.Message(arbitration_id=canID, data=msg, is_extended_id=False)
```

Funksjonen tar inn en liste med "tags", og bygger så meldingen. Hvis ID-en som skal sendes er innhold i listen for "heartbeat"-meldinger, sender den "marco/n". Er dette ikke tilfellet, går den gjennom en "for"-løkke som bygger bytearrayet for dataen i CAN-meldingen som returneres klar for sending.

Tilbakekallet for mottatte meldinger på CAN-buss eller en av de andre trådene som skal sende informasjon til kontrollstasjonen bruker metoden "sendTCPpacket" for å sende meldinger. Metoden kaller på funksjonen "toJson" vist i kodeutraget 7.19 nedenfor for å formatere meldingen før den sendes.

Kode 7.19: toJSON funksjon

```
1 def toJson(input):
2   packet_sep = json.dumps("*")
3   return bytes(packet_sep + json.dumps(input) + packet_sep, "utf-8")
```

Funksjonen tar inn en pakke, legger på "*"-skilleoperatoren, konverterer til JSON-formatet, og returnerer dette som bytes klar for sending over TCP. Dette er er en funksjon som er videreført fra fjorårets prosjekt[31] for å holde pakkeformatet over TCP likt.

7.3.3.2 Drivere

For å kontrollere nettverkstilkoblingen, lese temperatur, styre servomotor og starte kamera, er følgende klasser, forklart nedenfor, tatt i bruk:

- **Network()** - Er en klasse for å starte, holde, og avslutte TCP-kommunikasjon.
- **ServoPWM()** - Er en klasse for å initiere pinner, og oppdatere duty cycle på PWM()
- **STTS75()** - Er en klasse for å initiere temperatursensoren
- **gStreamerpipe()** - Er en klasse for initiering av kamera, samt starte og stoppe sending av videostrømmen.

7.3.3.2.1 Network() - Klassen ligger i networkHandler.py-filen og er videreført fra fjorårets prosjekt[31]. Vi har gått gjennom klassen for å forstå hvordan vi skal integrere den i "comHandler()"-klassen. Metodene til klassen:

- **init**-metoden til klassen setter opp variabler og starter "new_conn()"-metoden. Den starter også en tråd for "beat()"-metoden.
- **beat()**-metoden blir kjørt i en tråd og sender en "heartbeat" hvert 300ms.
- **new_conn()**-metoden setter først opp en TCP-socket og basert på variabelen "is_server". Denne variabelen bestemmer om en skal sette opp en server eller en klienttilkobling. Hvis det er en server, binder den seg til egen IP-adresse og port før den lytter etter nye tilkoblinger. Metoden starter så en tråd med metoden "wait_for_conn". Hvis det er en klient, kjører den en "while"-løkke som kobler til TCP-socketen.
- **wait_for_conn()**-metoden kjører en "while"-løkke som venter på en ny tilkobling. Når den får en ny tilkobling, bryter den løkken.

- **send()**-metoden sjekker om dataen som sendes er bytes, og sender dataen hvis dette er tilfellet. Kommer det en feilmelding når det prøves å sende en melding, prøver den å koble til socketen på ny.
- **recive()**-metoden mottar og returnerer data.

7.3.3.2.2 ServoPWM() -klassen ligger i filen camPWM.py. Klassen er utviklet med hensyn til servomotoren som er kjøpt inn og Jetson Nano. Dermed er klassen kun kompatibel med denne maskinvaren. Klassen har 4 metoder:

- **init**-metoden setter opp variabler og sjekker om pinnen som brukes til PWM er kompatibel for PWM. Metoden aktiverer så "initPWM()"-metoden.
- **initPWM()**-metoden setter opp GPIO-pinnen som en PWM-utgang, og setter frekvensen og pulsbredden som PWM-signalet skal kjøre med.
- **newAngle()**-metoden tar inn en vinkel mellom 0-180°. Den skalerer så denne vinkelen til rett pulsbredde, og oppdaterer utgangen.
- **cleanup()**-metoden stopper PWM-signalet og fjerner oppsettet av GPIO-pinnen. Dette setter pinnen lav til 0V.

7.3.3.2.3 STTS75() -klassen ligger i filen STTS75_driver.py. Klassen er laget som en driver for temperatursensoren som er utgitt av sensorgruppen til hvert kretskort som skal i elektronikkhuset. Driveren er utviklet sammen med sensorgruppen og har to metoder.

- **init**-metoden setter opp I^2C-bussen og den ene I^2C-slaveadressen vi leser og skriver til.
- **read_temp()**-metoden gjør først en skriveoperasjon som sier vi ønsker å lese fra register 0, før vi leser to bytes fra register 0 som inneholder temperaturen.

7.3.3.2.4 gStreamerPipe() -klassen ligger i filen camHandler.py. Klassen er laget for å kunne initiere, starte og stoppe gStreamer-rør fra Python. gStreamer er programvaren som er brukt for å ta inn, benhandle og sende videostrømmen fra kameraene på ROV-en. gStreamer-programvaren gås igjennom i delkapittelet 7.3.4.1. Klassen arver thread-klassen fra biblioteket threading og har metodene:

- **init**-metoden tar inn en "pipeId" og velger port og kilde basert på det. Er det en uspesifisert "pipeId" settes porten til 5000. Til slutt kjører den metoden "**createPipe()**"

Kode 7.20: Klasse for styring av rør og initiering av klassen

```
1  class gstreamerPipe(Thread):
2      def __init__(self, pipeId:str='test', multicastGroup:str = '224.1.1.1' , ...
           bitrate:str='8000000', port:str='5000') -> None:
3          Thread.__init__(self)
4          self.pipeId = pipeId
5          self.multicastGroup = multicastGroup
6          self.bitrate = bitrate
7          if self.pipeId == "stereo1":
8              self.port = '5000'
9              self.sourceID = '0'
10         elif self.pipeId == "stereo2":
```

```
11              self.port = '5001'
12              self.sourceID = '1'
13          elif self.pipeId == "bottom":
14              self.port = '5002'
15              self.sourceID = '2'
16          elif self.pipeId == "manipulator":
17              self.port = '5003'
18              self.sourceID = '3'
19          else:
20              self.port = port
21          self.pipe = self.createPipe()
```

- **createPipe()**-metoden klargjør røret med rett parameter og velger hvilke rør den skal kjøre før den til slutt starter røret. Hvis "pipeId" er ukjent tar den å starter en autogenerert videokilde med valgt portnummer.

Kode 7.21: Metode for å sette opp rør

```
1       def createPipe(self):
2           if self.pipeId == "stereo1" or self.pipeId == "stereo2":
3               gstStr = (f"nvarguscamerasrc sensor-id={self.sourceID} ! ...
                    video/x-raw(memory:NVMM), width=1920, heigth=1080, ...
                    framerate=30/1, format=NV12 ! nvv4l2h264enc ...
                    insert-sps-pps=true bitrate={self.bitrate} ! rtph264pay ! ...
                    udpsink host={self.multicastGroup} port={self.port} ...
                    auto-multicast=true")
4           elif self.pipeId == "bottom" or self.pipeId == "manipulator":
5               gstStr = (f"v4l2src device=/dev/video{self.sourceID} io-mode=2 ! ...
                    image/jpeg, format=MJPG, width=1280, heigth=720, ...
                    framerate=30/1 ! nvjpegdec ! video/x-raw ! nvvidconv ! ...
                    video/x-raw(memory:NVMM), format=NV12 ! nvv4l2h264enc ...
                    insert-sps-pps=true bitrate={self.bitrate} ! rtph264pay ! ...
                    udpsink host={self.multicastGroup} port={self.port} ...
                    auto-multicast=true")
6           else:
7               gstStr = (f"videotestsrc ! videoconvert ! x264enc ! rtph264pay ! ...
                    udpsink host={self.multicastGroup} port={self.port} ...
                    auto-multicast=true")
8           return Gst.parse_launch(gstStr)
```

- **runThread()** metoden starter en gstreamer-instanse og lager en tråd med denne. Den kjører så en evig løkke. Hvis løkken blir avsluttet, blir metoden "stopPipe()" aktivert.

Kode 7.22: Metode for å holde tråden i gang

```
1       def runThread(self):
2           Gst.init([])
3           self.mainLoop = GLib.MainLoop()
4           thread = Thread(target=self.mainLoop.run)
5           thread.start()
6           try:
7               while True:
8                   time.sleep(0.1)
9           except KeyboardInterrupt:
10              pass
11          finally:
12              self.stopPipe()
```

- **runPipe()**-metoden setter gstreamer i "playing" status som gjør at gstreamer-røret kjører.

Kode 7.23: Metode for å starte rør

```
1       def runPipe(self):
2           self.pipe.set_state(Gst.State.PLAYING)
3           print(f"Pipe {self.pipeId} set to state playing")
```

- **stopPipe()**-metoden setter gstreamer i "null"-status som stopper gstreamer-røret.

Kode 7.24: Medtode for å stoppe rør

```
1       def stopPipe(self):
2           self.pipe.set_state(Gst.State.NULL)
3           print(f"Pipe {self.pipeId} set to state null")
```

- **quitLoop()**-metoden stopper tråden som kjører klassen. Brukes når programmet stoppes.

Kode 7.25: Metode for å stoppe tråd

```
1       def quitLoop():
2           self.mainLoop.quit()
3           print(f"Pipe {self.pipeId} quit")
```

7.3.4 Programvare for bildebehandling og kamera

For å oppnå sending av videostrømmen til land med lav forsinkelse og ønskelig bildekvalitet, utnytter vi Nvidia Jetson Nano sin maskinvare. Det vil si dens dedikerte enkodere og dekodere. Denne maskinvaren utnyttes med hjelp av programvaren gStreamer for å hente videostrømmen, konvertere til rett format og sende videostrømmen.

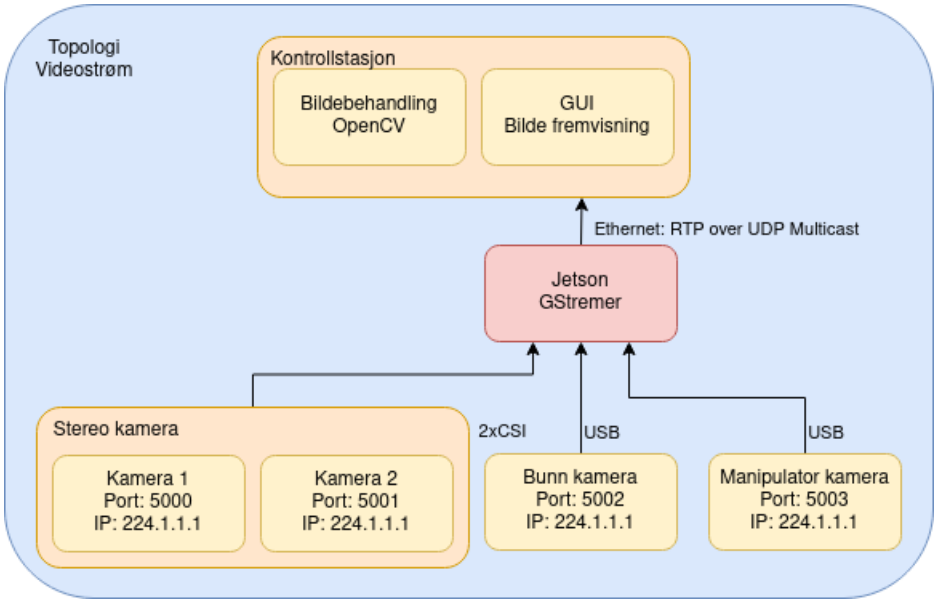
7.3.4.1 GStreamer

Teorien og veiledning for bruk av programvaren er hentet fra gStreamer nettsiden [178] og wikipedia [179].

GStreamer er et rørbasert multimedia rammeverk som knytter sammen forskjellige verktøy for mediaprosessering som brukes til å bygge komplekse mediaapplikasjoner. Typiske bruksområder er mediaavspillere, mediatranskodere og redigeringsapplikasjoner. GStreamer er gratis og har åpen kildekode med en *LGPL-2.1-or-later*-lisens. GStreamer ble lansert i 2001 av Erik Walthinsen og baserte seg på et forskningsprosjekt som ble utført hos *Oregon Graduate Institue*. Prosjektet blir støttet av næringslivet som bruker det i leveransene sine, samt videreutvikler og vedlikeholder rammeverket. GStreamer baserer seg på at man lager en rørledning hvor man legger inn elementer som er programvareutvidelser som behandler dataen. Eksempel på programvareutvidelser er enkodere, dekodere, filter, mediakilder og mediautganger. Hvert element har en eller flere "pads" eller puter som enten er en inngang, *sink* eller en destinasjon *source*. Kildeelementer som avlesning fra en fil eller et kamera har bare inngang, mens mediadestinasjoner har bare utgang.

7.3.4.2 GStreamer rørledning

For at vi skal oppnå ønsket resultat må vi ha fem forskjellige rørledninger. Topologien som er vist i figur 7.14 baserer seg på å bruke RTP-protokollen over UDP med *multicast* slik at flere applikasjoner i kontrollstasjonen kan hente samme videostrøm.



Figur 7.14: Topologi for videostrøm

.

For stereokameraet er følgende rørledning utviklet til bruk i GStreamer.

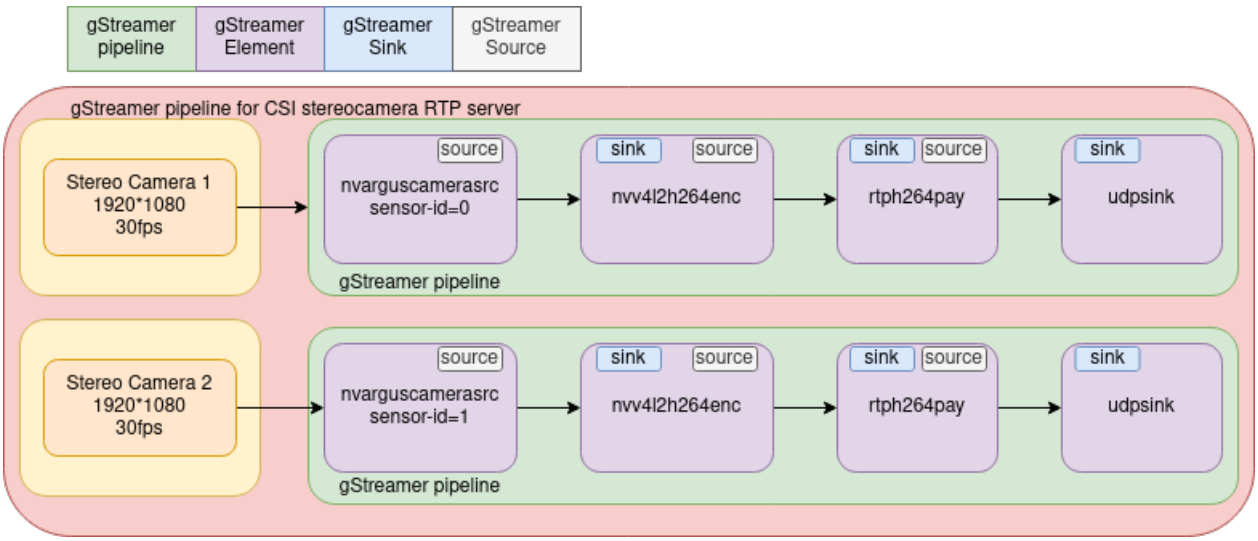
Kode 7.26: Åpning av rør for stereokamera strøm en

```
1 gst-launch-1.0 nvarguscamerasrc sensor-id=0 ! 'video/x-raw(memory:NVMM), ...
      width=1920, heigth=1080, framerate=30/1, format=NV12' ! nvv4l2h264enc ...
      insert-sps-pps=true bitrate=16000000 ! rtph264pay ! udpsink host=224.1.1.1 ...
      port=5000 auto-multicast=true
```

Kode 7.27: Åpning av rør for stereokamera strøm to

```
1 gst-launch-1.0 nvarguscamerasrc sensor-id=1 ! 'video/x-raw(memory:NVMM), ...
      width=1920, heigth=1080, framerate=30/1, format=NV12' ! nvv4l2h264enc ...
      insert-sps-pps=true bitrate=16000000 ! rtph264pay ! udpsink host=224.1.1.1 ...
      port=5001 auto-multicast=true
```

I figur 7.15 er det vist en forenklet rørledning som viser hvilke element som blir brukt.



Figur 7.15: Topologi for videostrøm

.

- **nvarguscamerasrc:** Er et element utgitt av Nvidia som bruker maskinvaren til Jetson Nano for å prosessere rådataen fra kamerasensoren. Dette elementet er spesifisert til å bruke NVMM-minnet som et buffer for videoen med oppløsning på 1080P og en bildeoppdateringsfrekvens på 30 bilder i sekundet med bilde formatet NV12.
- **nvv4l2h264enc:** Er et enkoderelement som også er utgitt av Nvidia. Dette elementet tar å omformere NV12 videoformatet til H.264. Elementet er spesifisert til å ha en bitrate på 16000000 bit/s samt at utgangsdataen inneholder SPS og PPS.
- **rtph264pay:** Er et standard element som lager RTP-meldinger av H.264-video.
- **udpsink:** Er et standard utgangelement som sender videostrømmen over RTP. Elementet er spesifisert med IP-adresse, hvilken port som skal motta og at det skal være multicast. Videostrømmen er tidstemplet slik at synkronisering er mulig hos mottakeren.

For USB-kamera er følgende rørledning utviklet til bruk i GStreamer.

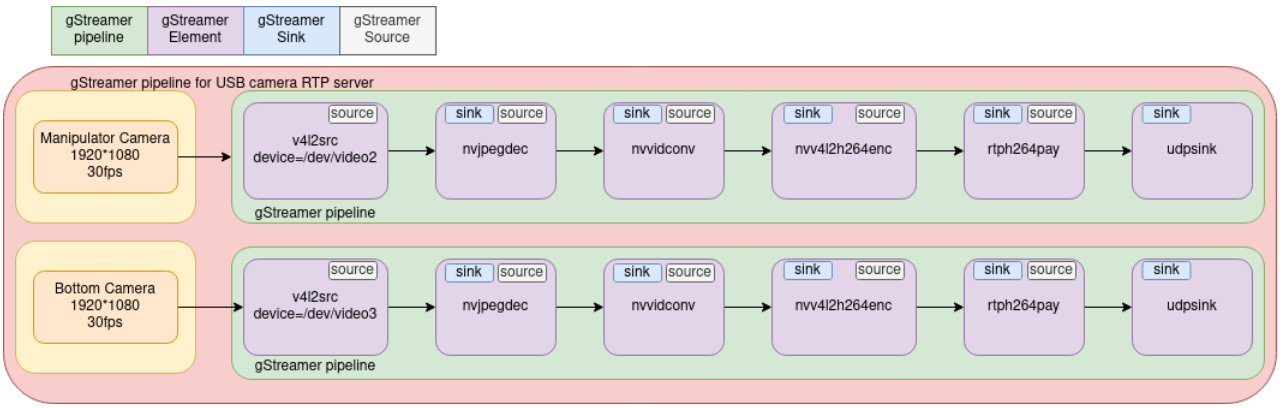
Kode 7.28: Åpning av rør for USB-kamera bunn

```
1 gst-launch-1.0 v4l2src device=/dev/video2 ! 'image/jpeg, format=MJPG, width=1920, ...
    heigth=1080, framerate=30/1' ! nvjpegdec ! 'video/x-raw' ! nvvidconv ! ...
    'video/x-raw(memory:NVMM), format=NV12' ! nvv4l2h264enc insert-sps-pps=true ...
    bitrate=16000000 ! rtph264pay ! udpsink host=224.1.1.1 port=5002 ...
    auto-multicast=true
```

Kode 7.29: Åpning av rør for USB-kamera manipulator

```
1 gst-launch-1.0 v4l2src device=/dev/video3 ! 'image/jpeg, format=MJPG, width=1920, ...
    heigth=960, framerate=30/1' ! nvjpegdec ! 'video/x-raw' ! nvvidconv ! ...
    'video/x-raw(memory:NVMM), format=NV12' ! nvv4l2h264enc insert-sps-pps=true ...
    bitrate=16000000 ! rtph264pay ! udpsink host=224.1.1.1 port=5003 ...
    auto-multicast=true
```

I figur 7.16 er det vist en forenklet rørledning som viser hvilke element som blir brukt.



Figur 7.16: Topologi for videostrøm

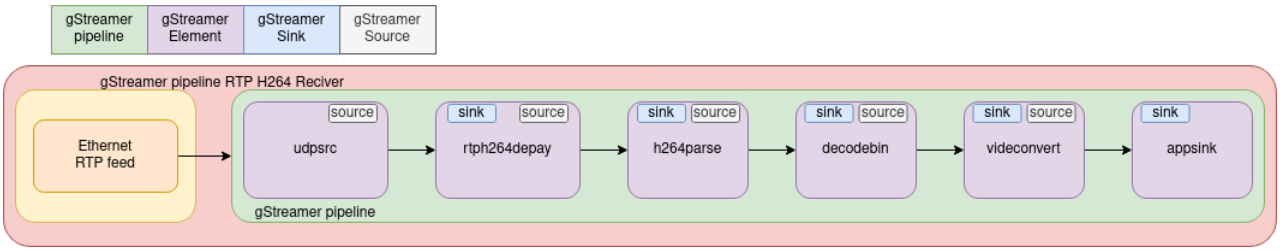
.

- **v4l2src:** Er et standard element som brukes til å hente data fra *Video for linux*-kilder som kamera. Elementet er konfigurert med bildeformat, oppløsning, og oppdateringsfrekvens.
- **nvjpegdec:** Er et dekoderelement som er utgitt av Nvidia. Dette elementet tar å omformere MJPEG formatet til NV12 formatet som blir brukt av resten av elementene lagt av Nvidia.
- **nvvidconv:** Dette elementet som også er lagt av Nvidia konverterer råvideo til Nvidia sin formatering av video og kan også brukes til å justere bildene.
- **nvv4l2h264enc:** Samme element og konfigurasjon som for CSI-rørledning.
- **rtph264pay:** Samme element og konfigurasjon som for CSI-rørledning.
- **udpsink:** Samme element med en annen konfigurert port en for CSI-rørledning.

For mottak av videostrømmen hos kontrollstasjonen har vi utviklet denne rørledningen til bruk i GStreamer.

Kode 7.30: Åpning av rør for mottak av RTP-strøm

```
1 gst-launch-1.0 -v udpsrc multicast-group=224.1.1.1 auto-multicast=true port=5000 ! ...
      "application/x-rtp, media=(string)video, clock-rate=(int)90000, ...
      encoding-name=(string)H264, payload=(int)96" ! rtph264depay ! h264parse ! ...
      decodebin ! videoconvert ! appsink sync=false
```



Figur 7.17: Topologi for videostrøm

.

- **udpsrc:** Er et standard element som brukes til å hente en videostrøm fra en UDP-kilde. Den er konfigurert med mulitcast, multicastgruppe(ip) og port.
- **rtph264depay:** Er et element som dekoder RTP-meldingene til en bytestrøm av H.264-video.

- **h264parse:** Er et element som henter ut H.264-videostrømmen fra bytestrømmen og konvertere til H.264 NAL[9]-enheter for hvert bilde. Elementet håndterer også feildeteksjon og feilhåndtering.
- **decodebin:** Er et element som tar et hvilket som helst format og konverterer til hvilket som helst format som passer i rørledningen.
- **videoconvert:** Er et element som konverterer videoen fra et fargespektrum til et annet. I følge GStreamer sin designguide, er den alltid anbefalt brukt når man går til generelle destinasjonselementer.
- **appsink:** Destinasjonselement som kan brukes i andre programvarer som OpenCV.

[9]NAL - Network Abstraction Layer

Kapittel 8

Diskusjon

Kapitteloversikt

8.1 Elektronikkhus

Det ble tidlig laget en modell av elektronikkhuset for å kunne utforme et design av innvendig elektronikk. For dette ble det brukt *Fusion 360*, og dette gjorde prosessen mye lettere. Det å ha en 3D-modell av innvendige kretskort gjorde at en enkelt kunne visualisere eventuelle problem som kunne oppstå. Modellen for det utvendige elektronikkhuset ble også laget for å vise ønskede størrelser til gruppen for ROV-design. Det ble lagt et fokus på å ha 3D-modeller av alle komponenter, noe som gjorde designprosessen mye lettere. Kretskort ble importert til stablen i Fusion for å sjekke for kollisjoner mellom komponenter. Dette fungerte meget bra, og avverget flere kollisjoner. Det ble også tenkt på at kort ligger omtrent 1cm fra kanten på elektronikkhuset. Ved eventuell lekkasje, kom ikke elektronikken i kontakt med vann før over en liter med vann lakk inn.

Konstruksjon av kamerahus og interne deler gikk igjennom mange ulike versjoner for å få ferdig produkt. Det var derfor bra at så mange design var testet med 3D-printing, da det gjorde at eventuelle endringer var enkelt å gjøre i CAD. Det anbefales at denne konstruksjonsmetoden blir tatt med videre. Konstruksjon av eksterne deler gikk også relativt bra, selv om noen deler ble laget for sent i prosjektet. Det kunne vært prioritert å få dette ferdig før, slik at designet hadde passet bedre på ROV-en.

Utregning av varmeavledningen på forhånd var veldig nyttig for konstruksjonen av elektronikkhus. Dette gjorde at kapslingen da hadde de nødvendige egenskaper for å holde temperaturen på elektronikken lav. Selv om det ønskelig skulle vært flere vifter i elektronikkhuset, er gruppen fornøyd med varmeavledningen i elektronikkhuset.

8.1.1 Utfordringer og forbedringer

I denne seksjonen vil det bli diskutert hvilke utfordringer vi traff ved utforming av elektronikkhuset. Det vil så bli belyst hva vi i retrospekt mener burde gjøres bedre neste gang.

Montering av kretskortstabel

Ved design av internt elektronikkhus ble det ikke tatt hensyn til montering av systemet. Montering er derfor mulig å gjøre feil i bortimot alle steg. Det er lite til ingen markører som sier hvordan ting skal orienteres for rett montering, noe som fører til at montering tar lengre tid. Montering brude derfor tidlig blitt prioritert, da systemet ofte monteres og demonteres.

Samarbeid for design av elektronikkhus

Gruppen for ROV-design fikk ansvar for å lage den eksterne kapslingen til elektronikkhuset. Dette var en oppgave som gruppen for kommunikasjon i fjor endte opp med å ta. Likevel fikk vi i ansvar å lage det interne i elektronikkhuset, og måtte utforme et internt design som skulle passe inn i deres design. Dette førte til at endringer i design av elektronikkhus har kostet gruppen like mye tid som det å lage hele elektronikkhuset hadde tatt gruppen. Selv om dette er en oppgave for ROV-design, burde denne i mye større grad blitt laget sammen med vår gruppe. Det burde også lages i samme tegning, slik at en hadde hatt en komplett tegning av ROV-en. Siden gruppe for ROV-design laget alt i *Inventor*, har tegningene problemer med kompatibilitet på tvers av hverandre.

Dette problemet har ført til problem med lekkasje, da det ble valgt en kuppel som ikke var laget til å brukes under vann. Dette skjedde da våres *ønsker* om størrelse av kuppel ble misforstått som *krav* av størrelse på kuppel. Andre misforståelser knyttet til lengde av elektronikkhus har gjort at volum på elektronikkhus er større enn det trenger å være, og fører til problem med oppdrift for ROV-en.

Justering av orientering av elektronikkhus

Under maskineringen av bakdel til elektronikkhus ble det gjort en feil av verkstedspersonell på UiS. Dette førte til at orienteringen på kretskortene ble feil. Konsekvensen av dette var at den påmonterte IMU-en som skulle gi vinkeldata hadde feil orientering på aksene. Det måtte derfor lages et nytt design av bakringen som muliggjorde at kretskortstablene kunne roteres til rett posisjon.

USB-kabel gjennom plugg

Å føre et USB-signal gjennom en plugg er ugunstig for signalintegriteten. Spesifikasjonene for den karakteristiske impedansen til USB er $50\,\Omega$, og disse spesifikasjonene er viktige for å redusere signalreflekssjoner og opprettholde signalintegriteten. Det er imidlertid vanskelig å opprettholde den karakteristiske impedansen gjennom en stor plugg. Når det er overganger til forskjellige tverrsnitt på ledningene, kan dette raskt bli et problem for signalintegriteten. Vi opplevde problemer med kabelen til kamerahuset når signalet gikk gjennom pluggen og videre til huset. Hvis avstanden mellom lederne ble for stor, mistet vi kommunikasjonen via USB. I kretskortdesign bestemmes den karakteristiske impedansen til et tvunnet par av avstanden mellom lederne, isolasjonens relative permittivitet og lederenes tverrsnitt, som vist i formel (6.9) i kretskortdesign. Det er derfor viktig å velge ledninger med riktige egenskaper

for å opprettholde den karakteristiske impedansen så godt som mulig gjennom kabelen. Løsningen for å få dette til å fungere var å tvinne alle ledningene godt sammen før montering, noe som virket stabilt. Dette er imidlertid en lite anbefalt løsning og har stort forbedringspotensial.

PLA inne i elektronikkhus

For prototyping ble det laget en 3D-printet del av PLA, som skulle være inne i elektronikkhuset. Dette ble gjort da printere med PLA, var lett tilgjengelig på UiS. Dette printet ble så brukt i elektronikkhuset da det var et fungerende design. Inne i elektronikkhuset blir det over 50℃, noe som sterkt påvirker PLA. Dette førte til at materialet bøyde seg kraftig. Siden det blir slike temperaturer inne i elektronikkhuset, burde alle slike print være laget av ABS, da dette materialet tåler mye høyere temperaturer.

8.2 Kommunikasjon

Det ble tidlig i prosjektet bestemt at vi ville gå for en kommunikasjonsløsning som innebar CAN-buss. Dette delvis på grunn av hvor bra CAN-buss har fungert i tidligere prosjekt. Men også på grunn av interessen og nysgjerrigheten i ved å forstå hvordan CAN-buss fungerte. For å realisere sending og mottak av data med kontrollstasjon, ble det brukt TCP over Ethernet. Videoen fra kameraene ble send ved hjelp av UDP og RTP over Ethernet.

Konseptet med å kommunisere på CAN-bussen ved hjelp av en ekstern CAN-modul koblet til en ettkortsdatamaskin ble testet tidlig i prosjektet. Dette ble gjort for å verifisere at vi hadde en kommunikasjonsløsning som ville fungere. Allerede i uke 8 hadde vi et konsept på plass for å teste CAN- og TCP-kommunikasjonen. Kommunikasjonen ble også testet underveis i prosjektet etterhvert som nye kretskort kom på plass. Det var betryggende for hele prosjektet å få en tidlig bekreftelse på at vi hadde et konsept for kommunikasjon som ville fungere. Dette fordi kommunikasjonen er et essensielt bindeledd mellom alle systemene. Alt i alt, fungerte kommunikasjonslinjene bra, og oppfylte kravene vi hadde satt i forhold til forsinkelse.

8.2.1 Utfordringer og forbedringer

I/O-liste

Det ble satt opp en I/O-liste for å få en oversikt over meldingene som skulle gå sendes i systemet. Dette for at alle prosjektgruppene skulle få kunnskap og en forståelse om hvordan kommunikasjonen var tiltenkt. Listen som er brukt i prosjektet kan finnes i vedlegg D.

I/O-listen fungerte som et bra verktøy for oss som gruppe. Noen av prosjektgruppene derimot, hadde problem med å skjønne behovet av en slik liste, og brukte dermed lite tid på å forstå og legge inn data i listen. Dette førte til problem lengre ut i prosjektet når systemene skulle knyttes sammen. Det var vanskelig for gruppene å knytte sammen systemene uten å være klar over hvilke data de ønsket å motta, og hvilke data andre ønsket at de skulle sende. Dette førte til at vi som gruppe måtte bruke mye tid på å sørge for at de ulike gruppene sendte og mottok den dataen de skulle ha, for så og fylle det inn i I/O-listen.

Det kan hende at I/O-listen ble for avansert for de ulike gruppene, og at de ikke forstod listen av den grunn. Men det var også tydelig at noen av gruppene ikke la inn en innsats for å forstå. I tillegg til

dette, var GUI-gruppen bestående av datastudenter en av gruppene som skulle jobbe tett med denne listen. Da disse ikke har hatt faget ELE210 (Datamaskinarkitektur), manglet de kanskje litt kunnskap om bitvise operasjoner.

Testing av kommunikasjonen

Etter at vi hadde fått alle komponentene og kretskortene på plass, ble det utført en del tester for kommunikasjonen. Dette for å finne ut hvilke begrensninger vi jobbet med. Disse testene er inkludert og gjengitt i vedlegg A.1. Fra testrapporten kan det ses at det oppnås bra resultat for de beregnede gjennomsnittstidene. Men det er ikke til å skyve under en stol at noen av makstidene er noe høye sammenlignet med gjennomsnittstiden.

Gjennomsnittstiden for en pakke tur-retur mellom sensorkort og kontrollstasjonen er på $2.13ms$, mens makstiden ligger på rundt 9ms. For en pakke tur-retur mellom sensorkort og kommunikasjonskort, ligger snittiden $1.8ms$ og makstiden på rundt 12ms. Hva som forårsaker de høye makstidene, og hvorfor makstidene er høyere mellom sensor- og kommunikasjonskort selv om det her er et mindre overføringsledd er uklart.

Det er blitt gjort en hel rekke testing med endringer i programvaren på Nvidia Jetson Nano, da det er kommunikasjonskontrolleren som er fellesnevneren i begge testene med noe høye makstider. Disse testene fikk ingen utslag for de høye makstidene. Uten å kunne konkludere med det, er det fortsatt sannsynlig at problemet ligger i programvare. Dette må undersøkes grundigere for å kunne optimalisere kommunikasjonen.

Feiltolking av meldinger

Når mikrokontrolleren sendte en datapakke, hendte det at kommunikasjonskontrolleren feiltolket byter i denne datapakken ved mottak. I datapakkene som ble mistolket, var det konsekvent byte en og syv som var feil. Det ble koblet opp skop til CAN-bussen for å analysere meldingene på bussen. Meldinge på bussen var korrekt sammenlignet med meldingen som ble sendt. Dette tydet på at feilen lå mellom CAN-modulen og kommunikasjonskontrolleren.

Det første som ble gjort, var å endre på konfigurasjonen av CAN-bussparametrene for bit-timing. Dette ble gjort på både kommunikasjonskontrolleren og mikrokontrolleren som sendte meldingen. Meldingene fortsatte å bli feiltolket, selv etter disse endringene. Dette tydet på at problemet muligens lå på selve SPI-linjen mellom enhetene og oppsettet av denne.

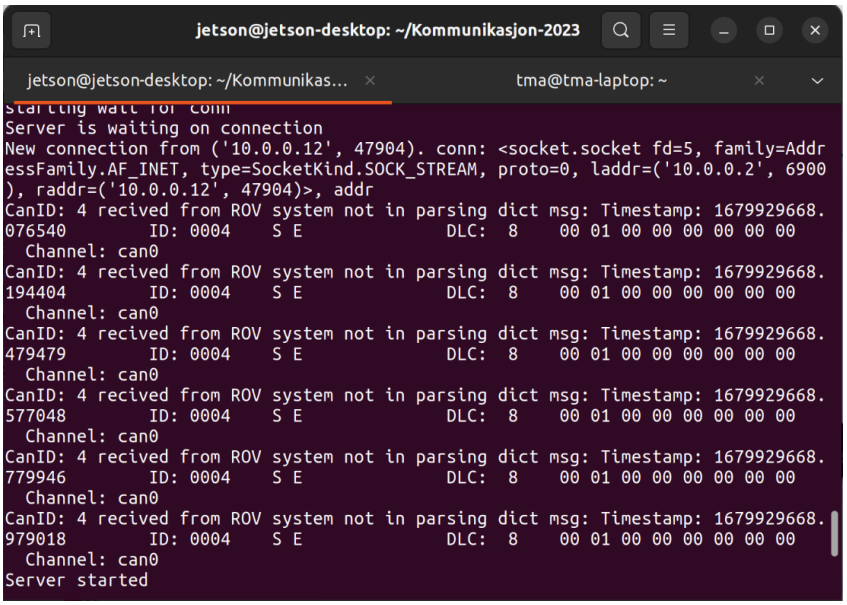
Det ble satt opp et skop på klokkesignalet og MISO-linjen for SPI-kommunikasjonen, for å sammenligne klokkesignalet med skrivingen på MISO-linjen. Målingene av klokkesignalet viste at signalet oscillerte mye etter nivåendring. Selv om det ikke ble tid til å teste dette bedre, er det mulig dette var en del av problemet. På grunn av dårlig tid, ble det gjort en halvering av kommunikasjonshastigheten, altså frekvensen på klokkesignalet, noe som førte til at datapakkene ble tolket korrekt. Da SPI-linjen opprinnelig opererte med en frekvens på 10MHz, mens CAN-bussen ikke kjørte på mer en 500kbps, var det ingen problem å halvere frekvensen til 5MHz.

Under analysering av SPI-linjen på skop, ble det med ren tilfeldighet merket at meldingene ble lest riktig av kommunikasjonskontrolleren når måleproben var tilkoblet MISO-linjen. Når måleproben ble koblet fra, ble meldingene feiltolket igjen. Meldingene ble også tolket rett når vi loddet en lask direkte mellom CAN-modulen og kommunikasjonskontrolleren på MISO-linjen. Denne løsningen ble ikke brukt, da det var tilstrekkelig å halvere bussfrekvensen til 5MHz.

CAN-modulen SPI CAN Click (4.10a) skal i følge databladet, være kompatibel med en master som kjører en bussfrekvens på opptil 10MHz på SPI-linjen. Nvidia Jetson Nano skal kunne kjøre en klokkefrekvens på opptil 65MHz[84]. Etter at SPI-bussfrekvensen ble satt ned til 5MHz fungerte bussen uten å feiltolke meldinger men også med endret elektrisk karakteristikk på banen. Disse tingene tatt til betraktning, tyder på at det er utlegget på de fysiske SPI-banene som er problemet. Hvordan SPI-banene burde ha vært, er forklart og gjennomgått i diskusjonen som omhandler kretskortdesign 8.6.

Sporadisk ukjent melding

På et punkt under testing av kommunikasjonen, begynte det å dukke opp en ukjent melding i konsollen som viser alle de ukjente meldingene kommunikasjonskontrolleren mottar. I figur 8.1 nedenfor, er det vist et skjermbilde av konsollen med fem av disse feilmeldingene.



Figur 8.1: Konsoll for meldinger på CAN-bussen

Den ukjente meldingen som dukket opp i konsollen var alltid den samme, og så ut som den i figuren ovenfor. Meldingen ble alltid tolket som ID 4 med de samme åtte bytene. Det ble satt opp et skop for å analysere meldingene på CAN-bussen, og det viste seg at denne meldingene aldri ble sendt over CAN-bussen. Problemet måtte da ligge mellom CAN-modulen og kommunikasjonskontrolleren.

Dette problemet vedvarte selv etter vi hadde satt ned frekvensen på SPI-bussen, men hyppigheten av antall feilmeldinger ble redusert. Det er dermed uvisst om korrekt dimensjonerte SPI-baner ville løst dette problemet. Det som er sikkert, er at det er en "Rx Error"-melding, som er indikert av "E" i figuren ovenfor. CAN-modulen har et register for feilmeldinger, men det er ikke funnet noe i dokumentasjonen [88], som forklarer en feilmelding som denne.

Problemet med den ukjente meldingen så ikke ut til å påvirke systemet negativt, da alle meldingene som skulle sendes og mottas kom frem feilfritt. Da problemet dukket opp mot slutten av prosjektet når alle systemene koblet på bussen, ble det ikke prioritert å finne ut hva som forårsaket denne feilmeldingen. Dette fordi sluttfasen av prosjektet bestod av mye testing av ROV-en i sin helhet, som gjorde det vanskelig å feilsøke på dette problemet.

8.3 Maskinvare

Maskinvaren som ble valgt etter å ha analysert behovene, har møtt nærmest alle av kravene som ble satt. For videosending var det et stort fokus på å finne kraftig nok maskinvare som kunne gjøre oppgavene energieffektivt, og ikke lage for mye varme i prosessen. Nvidia Jetson Nano har fungert som en passende plattform for dette. Å utnytte maskinvareakselererte enkoding og dekoding av videoen gav oss nok tilgjengelig prosessorkraft til å utføre kommunikasjonsoppgavene i tillegg. CAN-modulen SPI CAN Click gav Jetson Nano CAN-buss kompatibilitet, med blandede resultat.

CAN-driver kretsen med SN65HVD230DR fungerte utmerket, men har ikke CAN-FD-kompatibilitet. Dette satte begrensninger på 1Mbps for overføringshastigheten. Servomotoren SG90 fungerer til tenkt bruk, men viste seg å ha en litt lav oppløsning hvis det skal gjøres mer presis styring.

8.3.1 Utfordringer og forbedringer

I denne seksjonen vil det bli diskutert hvilke utfordringer vi traff ved valg av maskinvare. Det vil så bli belyst hva vi i retrospekt mener burde gjøres bedre neste gang.

MCP2515

En av utfordringene med årets design av CAN-bussen, var CAN-kontrolleren MCP2515 på CAN-modulen SPI CAN Click. Det å kombinere CAN-modulen med Nvidia Jetson Nano, var noe som tok mye tid. Her måtte det brukes mye tid i testing og oppsett av programvaren for å kunne kommunisere med CAN-kontrolleren MCP2515.

Det å gå for en ekstern CAN-kontroller mener vi var et bra valg i oppstartsfasen av prosjektet. Men underveis i prosjektet, lanserte Nvidia kortet Jetson Orin Nano 8GB[180], som er et utvikler-kort med CAN-kontroller. Et kort som dette kunne spart oss for mye tid på å implementere CAN-busskommunikasjon for kommunikasjonskortet.

CAN-FD

De andre gruppene brukte mikrokontrollere av typen Nucleo32-STM32G431KB og Nucle64-STM32G431RB. Disse mikrokontrollerene har perifermodul for CAN-FD. I dette prosjektet, ble det bestemt av oss at det skulle brukes CAN, og ikke CAN-FD. Dette satte begrensninger for overføringshastigheten som kunne kjøres på CAN-bussen. Sett i ettertid, hadde realiseringen av CAN-FD-kommunikasjon vært spennende.

Programvaren som ble brukt på Nvidia Jetson Nano for å kommunisere på CAN-bussen, hadde også kompatibilitet for både CAN og CAN-FD. Hadde vi gått for en annen maskinvare for CAN-kontrollere og CAN-drivere i prosjektet, kunne vi kjørt CAN-FD. I kapittelet som omhandlet maskinvaren (4), ble det sett på både en CAN-kontroller og en CAN-driver som kunne brukes med CAN-FD. Dette var CAN-kontrolleren MCP2517FD[90], og CAN-driveren TJA1044GT/3Z[94]. Hadde vi droppet bruken av en CAN-modul, og heller implementert en krets for CAN-buss direkte på kretskortet, kunne vi ha brukt MCP2517FD og TJA1044GT/3Z.

8.4 Kamera og video

Det var bestemt at det skulle monteres tre kamera på ROV-en. Ett stereokamera i front slik at bildebehandlingsgruppen kunne drive med 3D-modellering. Ett USB-kamera i bunn av ROV for oversikt av omgivelser under ROV-en. Og ett USB-kamera som skulle gi en oversikt over manipulatoren. Det ble bestilt et stereokamera av typen IMX219-83, og to USB-kamera av typen OV2710. Basert på funksjonsspesifikasjonen ble det beregnet hvilke kameralinser som ville oppfylle kravene til ønsket synsfelt.

USB-kameraene OV2710 fungerte til sin hensikt. De gav et brukbart synsfelt, samtidig som de leverte bilder av god kvalitet og oppløsning. Det var oppgitt at disse kameraene ville fungere bra i mørke omgivelser på grunn av sin CCD-kamerasensor, noe de også gjorde. Responsen fra ROV-operatørene var positive.

8.4.1 Utfordringer og forbedringer

I denne seksjonen vil det bli diskutert hvilke utfordringer vi traff ved kamera og video. Det vil så bli belyst hva vi i retrospekt mener burde gjøres bedre neste gang.

IMX219-83

Etter at testing av ROV i vann hadde begynt, ble det oppdaget at synsfeltet på IMX219-83 var for smalt, og at kameraet fungerte dårlig i "mørke" omgivelser.

Responsen fra ROV-operatørene under vanntest, var at synsfeltet på stereokameraet var for smalt, på tross av beregningene av synsfelt på forhånd. Synsvinklene til stereokameraet var oppgitt til 83/73/50 grader diagonalt, horisontalt og vertikalt. Det kom også frem at det var for lite lys i bassenget for at kameraet skulle fungere bra.

For å løse dette, burde det ha blitt bestilt en ny linse med kortere fokallengde og større blenderåpning. Dette ville gitt et større synsfelt, og også økt mengden lys som slapp gjennom linsen.

Da IMX219-83 hadde fungert dårlig under første vanntest til manøvrering, ble det bestemt at det ene USB-kameraet skulle benyttes innvendig i kuppelen slik at testen kunne gjennomføres. Dette viste seg å gi gode resultat. Vi valgte derfor til slutt å montere USB-kamera av typen OV2710 på permanent basis og bestille inn et nytt for manipulatoren og bare bruke IMX219-83 til bildebehandling.

8.5 Programvare

Det ble utviklet programvare til kommunikasjonskontrolleren og funksjoner for CAN-buss til mikrokontrollerene. Programvaren for kommunikasjonskontrolleren er todelt, der den ene delen omhandler kommunikasjon og den andre video.

Videosending ved bruk av gStreamer fungerte bra, og det var en stor effektbesparelse å bruke maskinvareakselerert enkoding og dekoding av video sammenlignet med en tradisjonell prosessor. Dette frigjor også prosessorkraft som kunne brukes til å kjøre andre applikasjoner. Forsinkelsen var akseptabel under testene, men varierte avhengig av hvordan mottakeren spilte av videoen. Ved bruk av gStreamer på Windows slik testene ble gjort, var forsinkelsen omtrent 100ms. Ved bruk av samme laptop med

AMD-prosessor på Ubuntu 22.04, var forsinkelsen omtrent 160ms. Det ble prøvd en laptop med Intel-prosessor og samme Ubuntu-versjon, og her var resultatet omtrent 100ms forsinkelse. Det er usikkert hva som er den underliggende flaskehalsen her, og det bør utføres mer testing for å forbedre forsinkelsen ved bruk av AMD-prosessorer og Ubuntu. Ytelsen på kommunikasjonskontrolleren var god, men den totale opplevelsen av resultatet varierte mye avhengig av hvordan videoen ble spilt av.

Kommunikasjonen er realisert ved flere Pythonprogram som i prinsippet har to oppgaver. Motta data fra kontrollstasjon, så kjøre en funksjon eller videresende pakken over CAN og motta data fra ROV-en og videresende denne til kontrollstasjon. Fokuset i koden var at det skulle være enklere å legge til nye "ID-er" som skal sendes eller mottas. Dette fungerte godt, programmet legger til noe forsinkelse på meldingene men av uproblematisk størrelse. Det er også lagd noen program som kan kjøre funksjoner for å starte og stoppe videostrømmer, styre servomotoren for tilting av kamera og avlesning av temperatur på kretskortet. Disse funksjonene fungerte godt, men koden for styring av videostrømmene kunne nok ryddes litt opp i.

Programmering av tråder og nettverk har gruppen lite erfaring med fra tidligere, og gruppen har hatt et stort læringsutbytte fra dette. "gStreamer" var også en ny erfaring og har krevd mangfoldige timer med lesing av dokumentasjon.

Funksjonene lagd for mikrokontrollerene er programert i C og setter et rammeverk for kommunikasjon mellom systemene til gruppene.

8.5.1 Utfordringer og forbedringer

I denne seksjonen vil det bli diskutert hvilke utfordringer vi traff ved utforming av programvare. Det vil så bli belyst hva vi i retrospekt mener burde gjøres bedre neste gang.

MCP2515

Vi opplevde tidlig problemer med drivere for MCP2515 CAN-kontrolleren. Driveren som ligger i kernelen til Jetpack 4.6.1 var ikke særlig stabil. Etter mye feilsøking gikk vi ned en versjon til Jetpack 4.5.1 der Nvidia kunne verifisere at driveren var kompatibel og testet.

Databehandling

Behandling av meldinger burde kanskje bare foregått i kontrollstasjonen. Dette kan forenkle programvaren for kommunikasjonskontrolleren og gjør at meldingene blir behandlet færre ganger. Ulempen med dette er at Kontrollstasjons programvaren tar lengre tid å ferdigstille og andre grupper får da kanskje ikke mulighet til å teste kommunikasjonen sin før kontrollstasjon programmet er klart. Dette innbærer også et ekstra ledd i testingen som kan være ineffektivt.

Programmeringspråk

Det bør tas en vurdering om en skal endre til et kompilert programmeringspråk for å bedre ytelsen til programmet. Vist en ønsker å gjøre mer bildebehandling og kunstig intelligens oppgaver på ROV-en, burde man bruke C++. For å utnytte potensialet til Jetson må man bruke "CUDA" prosessoren og

minnet noe man ikke har tilgang til i Python. Ulempen med dette er at C++ er mer komplisert å programmere i, det finnes flere fallgruver og prototyping av koden tar lengre tid.

Egen fil for CAN-buss kode

Istedenfor å sende kodeutragene av funksjonene som skulle brukes for CAN-buss kommunikasjonen på mikrokontrollerene burde vi lagd en ".h" og ".c" fil som kunne inkluderes i koden til gruppene. Mange grupper endte opp med å implementere koden og funksjonene i "main.c" som ofte fører til et uoversiktlig program.

Retningslinjer for timere

For CAN-buss implementasjonen ble det sendt ut et forslag til timer oppsett og parameter konfigurasjon for CAN-bussen som passet dette. Det ble lagd et "Interface overview" dokument hvor all dataen som sendes ble kartlagt. I dette dokumentet ble sendings intervallet spesifisert. Hvordan gruppene valgte å implementere dette var ulikt og skapte noe problem. Det burde lages et bedre rammeverk for timere og hvordan gruppene skal implementere dette for kommunikasjonen.

8.6 Kretskort

Ingen på gruppen hadde tidligere erfaring med kretskortdesign. Det ble derfor en stor prosess å sette seg inn i dette for å utforme et ferdig kretskort. Gruppen startet derfor tidlig å sette seg inn i programmet *Altium Designer* for å kunne lage kretskort. Mal av utforming av kretskort skulle også deles med de andre gruppene, da dette skulle monteres i et design med tre bakplater. Gruppen gikk derfor for å hente konsultasjon fra foreleser i emnet *Datamaskinkonstruksjon (ELE340*, Kristian Thorsen. Han gav oss tilgang til ressurser ved å legge oss til i emnet sitt om datamaskinkonstruksjon på Canvas.

Prosessen vi har hatt med å lage kretskort bærer stort preg av at vi tilegnet oss kunnskapen til kretskortdesign samtidig som vi lagde kretskort. Likevel er gruppen relativt fornøyd med resultatet, og spesielt fornøyd med å få ferdig kretskort tidlig i prosjektet. Gruppen er fornøyd med det generelle designet bak det modulære designet, og anvendelsen av dette. Løsningen med bakplater er direkte hentet fra suksessen fra fjoråret, og denne ideen er derav bare videreutviklet. Videreutviklingen på dette har hatt bra resultat, og det antas at dette blir brukt videre til neste år. Det å ha felles komponenter med krets for CAN-buss og temperatursensor ferdig rutet når malen ble gitt ut, var også noe som standardiserte prosessen av kretskortdesign for alle gruppene, og det anbefales at dette videreføres.

8.6.1 Utfordringer og forbedringer

I denne seksjonen vil det bli diskutert hvilke utfordringer vi traff ved utforming av kretskortene. Det vil så bli belyst hva vi i retrospekt mener burde gjøres bedre neste gang.

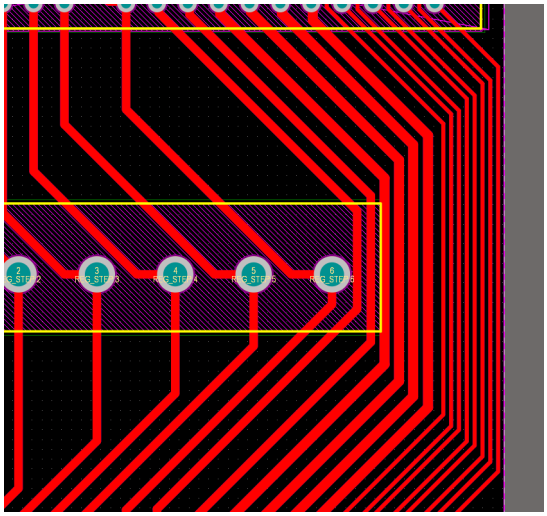
Individuelle baner på bakplate

Bakplatene sine baner burde vært trukket individuelt. Siden **24 mils**(12 på begge sider) banebredde i ytre lag fortsatt gir under 35℃ i temperaturøke ved det maksimale strømtrekket per pinnepar(2,4A), var det ikke nødvendig med bredere baner. Dette forårsaket tvert imot bare problemer, og ekstra kostnad. Det hadde nemlig vært mulig å bestille kun en variant av kretskortet dersom hvert kort hadde individuelle baner, da minstebestilling uansett var fem stk kort. På denne måten hadde vi kun hatt behov for en bestilling istedenfor tre.

Interferens mellom baner på bakplate

En annen feil vi gjorde er at baner på bakplaten ikke har kobber mellom banene for å redusere interferens mellom banene. Denne kobberen kan også oppføre seg som en antenne. Noe som fører til at den kan plukke opp uønsket støy. Denne skjerminga av signalbaner kalles Faradaybur". Det ble tenkt på hvilke signal som burde legges på siden av hverandre når bakplatene ble lagd, men det ble ikke tenkt på at en burde redusere interferens mellom banene.

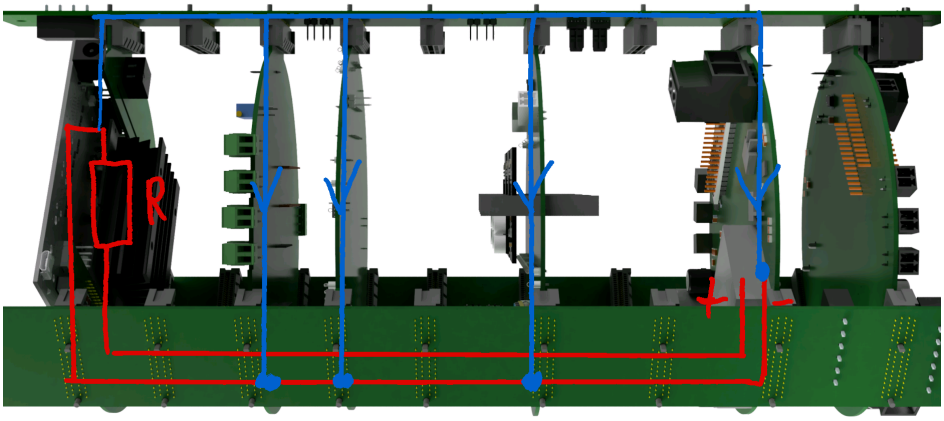
Enkelte plasser er det for liten avstand mellom banene, som spesielt åpner for mer interferens enn det trenger å være. Selv om dette unntaksvis ble gjort for å få plass til plugger, er dette likevel uheldig. På figur 8.2 vises det hvordan noen baner ble for nærme hverandre, noe som åpnet for interferens og krysstale mellom banene.



Figur 8.2: Avstand mellom baner er på det minste *10 Mils*

Jord på bakplater

I to av bakplatene ble jord koblet i et felles jordlag. Hensikten bak dette var å få en bedre jording og mindre resistans for returstrømmer. I motsetning, så åpner dette for jordsløyfer og returstrømmer mellom de to bakplatene. På figur 8.3 er det illustrert returstrømmer.



Figur 8.3: Returstrømmer vist i blått og tiltenkte baner vist i rødt

Returstrømmer kan som vist, gå igjennom jordlaget til andre sine kretskort enn lasten og bakplaten sitt jordlag. På figuren er det vist returstrømmer fra kommunikasjonskortet til kraftforsyningen. De store strømmene fra thrusterene er koblet gjennom ledning ut fra kraftkortet, og går ikke på bakplaten. Det er da kun mindre returstrømmer som vil gå igjennom de andre kretskortene, selv om dette heller ikke er optimalt.

Det samme gjelder også for strømbanene til 5V-forsyningen. Da denne også er koblet på to bakplater istedenfor en. På samme vis som jord, er det mange som har et 5V-lag som et koblet i begge bakplatene. For å forhindre dette burde all kraftforsyning gå igjennom kun en bakplate, og ikke flere.

For mange ulike plugger på bakplate

Det ble brukt for mange forskjellige plugger til bakplaten. Det var lite nødvendig å ha så mange ulike plugger, og det burde vært et standard fotavtrykk for en plugg som tåler det samme strømtrekket som pinnene til PCI-kontaktene(*Maks 2,4A*). Denne burde vært tilgjengelig nederst på bakplaten, der alle signal kunne blitt rutet til hver sin via i fotavtrykket. Eksempelvis å bruke to stk `RS PRO, 30 Way, 2 Row, Vertical PCB Header`.

Fotavtrykk for flere PCI-kontakter

Selv om en plassering av PCI-kontakter ble gjort tidlig, burde disse ha fotavtrykk flere plasser langs bakplaten. Dersom det da i etterkant var ønskelig å ha en annen plassering på kretskortene, kunne det loddes på en PCI-kontakt på et ledig fotavtrykk. Dette hadde åpent for et mer modulært design, og gjort det lettere å ha en mer egentilpasset avstand mellom ulike kretskort. Vi så tidlig at noen kort kom til å kreve mer plass enn andre kretskort, men at det ville bli vanskelig å vite hvilke før kretskortene var designet.

Monteringsvanskeligheter av kretskortstabel

Bakplatene burde vært laget slik at det ikke var mulig å montere kretskort på feil plass. Dersom den ene bakplaten hadde hatt PCI-kontakten speilvendt fra de andre, ville denne bare passet den ene veien. Det vill da vært mye enklere å montere kretskortene inn i stablen rett vei.

SPI-buss ruting

Ruting av SPI-bussen ble ikke gjort med hensyn på hvordan et høyhastighetsignal burde rutes på et kretskort. Fra en kilde om ruting av SPI-buss [181], står det fire tips for ruting av SPI-signal. Der står det at signalene burde rutes så korte som mulig, holdes til samme lengde, og ha konsekvent impedans. For vårt design er ingen av disse faktorene tatt hensyn til, noe som kan ha ført til feil ved SPI-kommunikasjonen. Dette har blitt mer diskutert ved problemer med kommunikasjon 8.2.1.

For å redusere ringing på SPI-bussen, kunne vi brukt seriemotstander og kontrollert impedans. Ut fra formelen for dempningsfaktor, ville høyere resistans ført til mindre oversving.

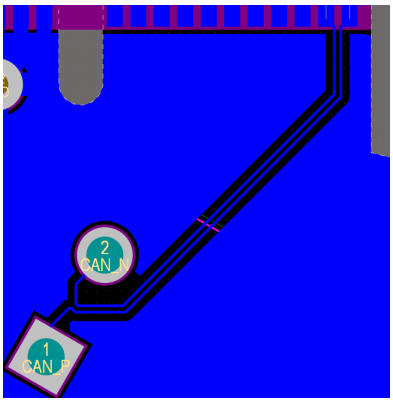
$$\zeta = \frac{R}{2\sqrt{L/C}} \tag{8.1}$$

Dempningsfaktoren er viktig for å få rett stigningstid t_r, og stabiliseringstid t_s på bussen. Tiden for et klokkesignal til SPI-bussen ved 10 MHz vil være 100 ns. Stabilisereingstiden til bussen må derfor være innenfor denne tiden for å kunne lese bittet på stigende flanke.

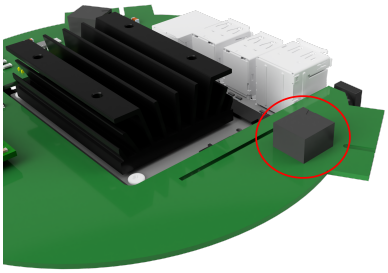
På en kilde fra Altium [182], brukes det 22 Ω eller 33 Ω resistanser for dette.

Plugg for CAN-buss

Det å bruke en plugg for å føre CAN-buss gjennom et kretskort er generelt dårlig praksis. Se figur 8.4a og 8.4b, her vises det hvordan rutingen går fra en via til gullfingrer. Dette gir en uforutsigbar karakteristisk impedans igjennom både via og plugg, selv om rutingen på kretskortet er med rett karakteristisk impedans. En vil også få kapasitansendring igjennom via-en og pluggen, som kan føre til signalrefleksjoner.



(a) Utlegg for ruting av CAN-buss



(b) 3D modell som viser plugg

Selv om dette kan kontrolleres med ledninger i tvunnede par med et viss tverrsnitt og avstand, vil det være uvisst når den skal termineres gjennom en via på kretskortet. Det burde istedenfor finnes et modulkort som tillater direkte tilkobling til kretskortet gjennom pinner for å redusere overganger. Alternativt burde hele kretsen lages på kretskortet for å eliminere problemet helt. Dette var noe vi tidlig vurderte, men gikk for modulkortet da gruppen hadde lite kjennskap til kretskortdesign. Designet ville også inneholt en krystalloscillator, noe som ikke er spesielt enkelt og krever et nøye design. Utlegget for modulkortet var derimot mulig å få tak i fra databladet, og kunne blitt kopiert til eget kretskort.

Logisk nivå PWM-signal og Level Shifter

Denne IC-en skulle sikre at det var tilstrekkelig spenning på PWM-signalet. Denne skulle omgjøre PWM-signalet fra 3,3V til 5V som beskrevet i seksjon 6.6.2.3.4. Selv om det ut fra databladet ser ut som denne skulle fungere bra, ble det under testing oppdaget at denne ikke gikk under laveste spenningsnivå for V_{CCA}. Dette resulterte i at laveste spenningsnivå lå omtrent på 1,8V, noe som er over V_{IL} fra PWM-inngangen på servoen. Utgangssignalet ble heller ikke like høy som verdien av V_{CCB} som lovet av databladet. Etter et par runder med testing, ble det konkludert med at dette var uinteressant å få til å fungere da PWM-signalet fungerte uten. Andre oppgaver ble prioritert og IC-en ble fjernet. Det ble laget en liten lask for å koble PWM-signalet fra Jetson direkte på banen der utgangen til IC-en skulle vært.

Grunnen til at dette ble gjort var fordi PWM-servoen sin V_{IH} var ukjent da dette ikke stod i noe datablad. Det ble derimot kun spesifisert at forsyningsspenningen til servoen var 4,8V til 6V. Det var derfor fare for at denne ikke hadde fungert med et signal på 3,3V. Det ble derfor forsøkt å øke denne spenningen fra 3,3V til 5V. Ved testing ble det likevel oppdaget at V_{IH} var lavere enn V_{OH} fra PWM-utgangen. Servoen kunne derfor kjøres direkte uten å øke spenningen.

Senere ble det gjort en dokumentert test på dette for å finne hvilke støymarginer PWM-signalet har. Denne testen er lagt med som vedlegg A2. I denne konkluderes det med at V_{IL} er 1,2V og V_{IH} er 2,3V. Dette burde vært testet mye tidligere, da det kunne vært unngått å bruke tid på en Level Shifter. Dette ble også nevnt av laboratorieansvarlig Jon Fidjeland tidlig i prosessen, som oppfordret oss til å se på støymarginer før en ser på løsninger for å øke spenningsnivået.

Seriemotstander til DIP-switch

Seriemotstanden på DIP-switchen var mye større enn nødvendig. Med en utgangsspenning på 3,3V og en motstand på $10\,\mathrm{k\Omega}$ blir strømmen som kan gå igjennom $\boxed{\frac{3{,}3V}{10\,\mathrm{k\Omega}} = 0.33\,\mathrm{mA}}$. Jetson skal ikke drive nok utganger til at det ville vært et problem å tilrettelegge for høyere strømtrekk. Jetson driver GPIO-pinnene sine gjennom en *TXB108 Level Shifter*. Det står i databladet at "the output drivers are very weak", uten at det står mer om hvor mye de kan trekke. Dersom en antar omtrent det samme som en Raspberry Pi, vil maksimal strømtrekk være 16 mA. En $300\,\Omega$ resistans ville da gitt et strømtrekk på $\boxed{11\,\mathrm{mA}}$, som antageligvis ville vært nok.

Det som lurte oss i databladet var at det var oppgitt en drivekapasitet på $20\,\mu\mathrm{A}$ [183]. Dette gjorde oss bekymret for strømtrekket til Jetson. Det antas likevel etter konsultasjon med Jon Fidjeland at dette var strømforbruk av å drive logikken til en port. Likevel ble dette en usikkerhet rundt dette som gjorde at resistansen ble koblet i serie for å sikre at spenningsnivået ikke droppet ved for stort strømtrekk.

Optimalt burde utgangene likevel gått igjennom et buffer, slik at kraftforsyningen på 3,3V fra Jetson kunne brukes til å drive utgangen. Dette hadde eliminert bekymringen rundt drivekapasiteten til portene.

Vifter for kjøling

Viftene som ble bestilt var for dyre og hadde for lav effekt for jobben. Istedenfor å tilrettelegge for at en vifte skulle monteres mellom PCI-kontaktene, burde viftene heller monteres på hvert kort individuelt. Selv om viftene aldri var tiltenkt å erstatte behovet for en vifte på kretskort med høy varmeutvikling, var virkningsgraden såpass neglisjerbar at det bare ble en unødvendig kostnad.

8.7 Prosjektet internt

Selv om alle på gruppen har vært involvert i avgjørelsene relatert til de forskjellige arbeidsoppgavene vi har hatt i prosjektet, har hovedansvaret for de ulike oppgavene i prosjektet vært tredelt:

- **Joar:** fikk hovedansvaret for design av kretskortene, og utlegg for disse. I tillegg til dette, tok han på seg hovedansvaret for 3D-printing av deler til den interne delen av elektronikkhuset.
- **Thomas:** fikk hovedansvaret for det meste som angikk kommunikasjonskontrolleren. Dette innebar mottak av meldinger fra CAN-bussen, sending av meldinger til kontrollstasjon og sending av videostrøm opp til kontrollstasjon.
- **Nils Helge:** fikk hovedansvaret for CAN-bussen, og for hvordan denne skulle settes opp og programmeres på mikrokontrollerene til de andre gruppene. I tillegg tok han på seg ansvaret for kamera.

Gruppen tok tidlig fatt på oppgaven, og fikk derav tidlig på plass konsepter for å løse oppgaven. Det ble lagt inn timer tidlig for å undersøke hvordan en skulle få disse konseptene til å fungere. Dette gjorde at det ble lite endringer i den originale planen, noe som førte til at gruppen fikk tid til å fullføre prosjektet. Gruppen samles daglig i tidlig fase av prosjektet, og planer ble lagt i plenum for å sikre at alle var på samme linje. Gjennom hele prosjektet har det vært en rød tråd at alle på gruppen har hatt et samlet sluttmål som gruppen arbeidet mot. Det har vært enighet rundt prioriteringer og mål for prosjektet, som har ført til en effektiv gjennomføring. Medlemmene har hatt stor individuell autonomitet, og søkt konsultasjon hos hverandre og eksternt ved behov.

Det å dele oppgaven i flere deler, har vært essensielt for å få nok tid til å fullføre oppgaven. Det har gjort at hver av medlemmene kan spesialisere seg i sin egen del av oppgaven, uten at alle medlemmene bruker tid på dette. Det frigjør nok tid til å ta et nødvendig dypdykk i stoffet for å gjøre oppgaven skikkelig. Dette førte til et mye bedre sluttresultat, og gruppen er fornøyd med denne avgjørelsen.

Tidsbruk har jevnt over vært relativt likt, selv om det har vært perioder der det ene medlemmet har vært bortreist. Dette førte til merarbeid for medlemmet for å ta igjen timer. Et annet medlem av hadde også prosjektlederansvar, noe som førte til mye ekstraarbeid knyttet til det overordnede prosjektet. Det å ha lengre opphold i arbeid fører også til litt dårligere kontinuitet i prosjektet. Dette er ikke optimalt å ha slike opphold for gruppen, men gruppen som en helhet har løst dette bra. Samlet har alle vært enige om at det skal gjøres en stor innsats i prosjektet, og det føler vi at alle medlemene har vist.

I vedlegg E, er det gjengitt et Gantt-skjema og månedsrapporter. Fra månedsrapportene kan det ses hvordan framgangen i prosjektet var ved månedsskiftet i januar og februar. Allerede i månedsskiftet fra januar, hadde vi komt frem til et generelt konsept rundt hvordan oppgaven skulle løses. Dette viser at timene som ble lagt ned i planleggingsfasen, gjorde at vi kom frem til et konsept som ikke trang store endringer underveis.

Gantt-skjemaet viser hvordan progresjonen for prosjektet internt har vært. Her planla vi med en timebruk på 1874 timer totalt på alle gruppemedlemmene gjennom hele prosjektet. Ved slutt av prosjektet er det dokumentert 2209 timer brukt. To store poster som overskred tidestimatet var rapportsskriving og produksjon/prototyping.

8.7.1 Bistand og testing

Selv om mesteparten av timene har blitt lagt ned i vårt eget prosjekt, har det også gått en hel del timer til å hjelpe andre prosjektgrupper på UiS Subsea. Underveis i prosjektet, har flere av gruppene bedt oss om assistanse vedrørende ulike problemer de har støtt på. Som kommunikasjonsgruppen blir vi fort involvert i andres problemer når det testes. Alle tre medlemmene har tidligere utdanning med fagbrev innen elektro eller automatisering. Joar har også erfaring med plastarbeid, maskinering og sveising. Gruppen endte derfor opp med å bistå med diverse problemstillinger når de dukket opp. Noen utvalgte problemstillinger som ble brukt tid på:

- **Thrustere:** Når thrustere skulle testes ble det bistått med feilsøking av kode for å verifisere at meldinger kom frem, samt finne årsaken til at pådraget var ustabilt. Det ble funnet et problem med hvordan mottak av data var implementert i mikrokontrolleren, samt register som overskrev hverandre.
- **Manipulator:** Når manipulator skulle testes ferdigmontert, var det problematikk med oppkobling, og alle signalene måtte kartlegges på ny. Det var også et problem med at motordriveren for rotasjonen av griperleddet trengte fem sekunder med null pådrag før den var initialisert og kunne kjøres.
- **Dreiing av flyter:** Når flyteren skulle produseres, hadde ROV-designgruppen en stor arbeidsmengde, og Joar ble derfor forespurt om han kunne plastsveise og dreie flyterhuset.
- **Kretskort:** Da andre grupper brukte lengre tid på å designe kretskort, ble det satt av en del tid for konsultasjon til andre grupper sitt utlegg for å få de ferdig.

8.8 Prosjektet overordnet

8.8.1 Prosjektledelse

Årets prosjekt består av totalt ni bacheloroppgaver som omhandler ROV-en Yme, og flyteren Balder. Totalt er det 24 studenter som har deltatt i årets prosjekt. Hver av oppgavene tar for seg et spesifikt delsystem, hvor alle disse inngår i et totalprodukt. For å ha en overordnet plan over hvordan totalproduktet skulle ende opp til slutt, ble det satt sammen en prosjektgruppe. Prosjektgruppen bestod av:

- **Prosjektleder:** Joar Rodrigues de Miranda
- **Prosjektleder konkurranser-koordinator:** Thomas Matre
- **Teknisk ansvarlig elektro:** Jesper A. Flatheim
- **Teknsik ansvarlig data:** Filip Sølvberg Herrera
- **Teknisk ansvarlig maskin:** Aleksander Schei

Prosjektgruppen ble veiledet av Otto Nessa Ljosdal, Tomas Royal Choat og Vebjørn Riiser, som alle deltok i fjorårets prosjekt for UiS Subsea. Det første prosjektgruppen gjorde, var å kartlegge hva årets prosjekt skulle inneholde. På grunn av det store antallet personer som hadde meldt interesse for å skrive oppgave for UiS Subsea, ble det bestemt at det skulle bygges en ny ROV. Interessen for å fortsette

og konkurrere i MATE ROV Competition var også der. Dette gav klarere rammer for utformingen av ROV-en, og noen tekniske begrensninger vi måtte forholde oss til. De forskjellige gruppene ble presentert med følgende oppgaver:

- **Elektro:** Kraftforsyning
- **Elektro:** Sensorsystem
- **Elektro:** Regulering og styring
- **Elektro:** Kommunikasjon og kamera
- **Elektro:** Flyter
- **Data:** Brukergrensesnitt til kontrollstasjon
- **Data:** Bildebehandling
- **Maskin:** Flyter og ROV-design
- **Maskin:** Manipulatordesign

Fordelingen av oppgavene ble gjort før prosjektet startet i januar, slik at gruppene kunne begynne planleggingen av oppgaven tidligst mulig. Dette gav oss et godt utgangspunkt for å komme i mål innen tidsfristen.

8.8.1.1 Planleggingsfasen

I starten av et prosjekt er det viktig å kartlegge behovende rundt prosjektet. Dette for å få et inntrykk av omfanget implementeringen av oppgavene innebærer. Det ble derfor laget en behovsmatrise for å kartlegge og analysere funksjonaliteten som var ønsket, eller krevdes for å delta i konkurransen. Hensikten med behovsmatrisen var at de ulike gruppene skulle få et klart bilde av hva som måtte leveres, og dermed komme raskt i gang med den individuelle planleggingen rundt deres oppgave. Det aktuelle innholdet i matrisen som angikk kommunikasjonsgruppen, er vist i figur 8.5 nedenfor.

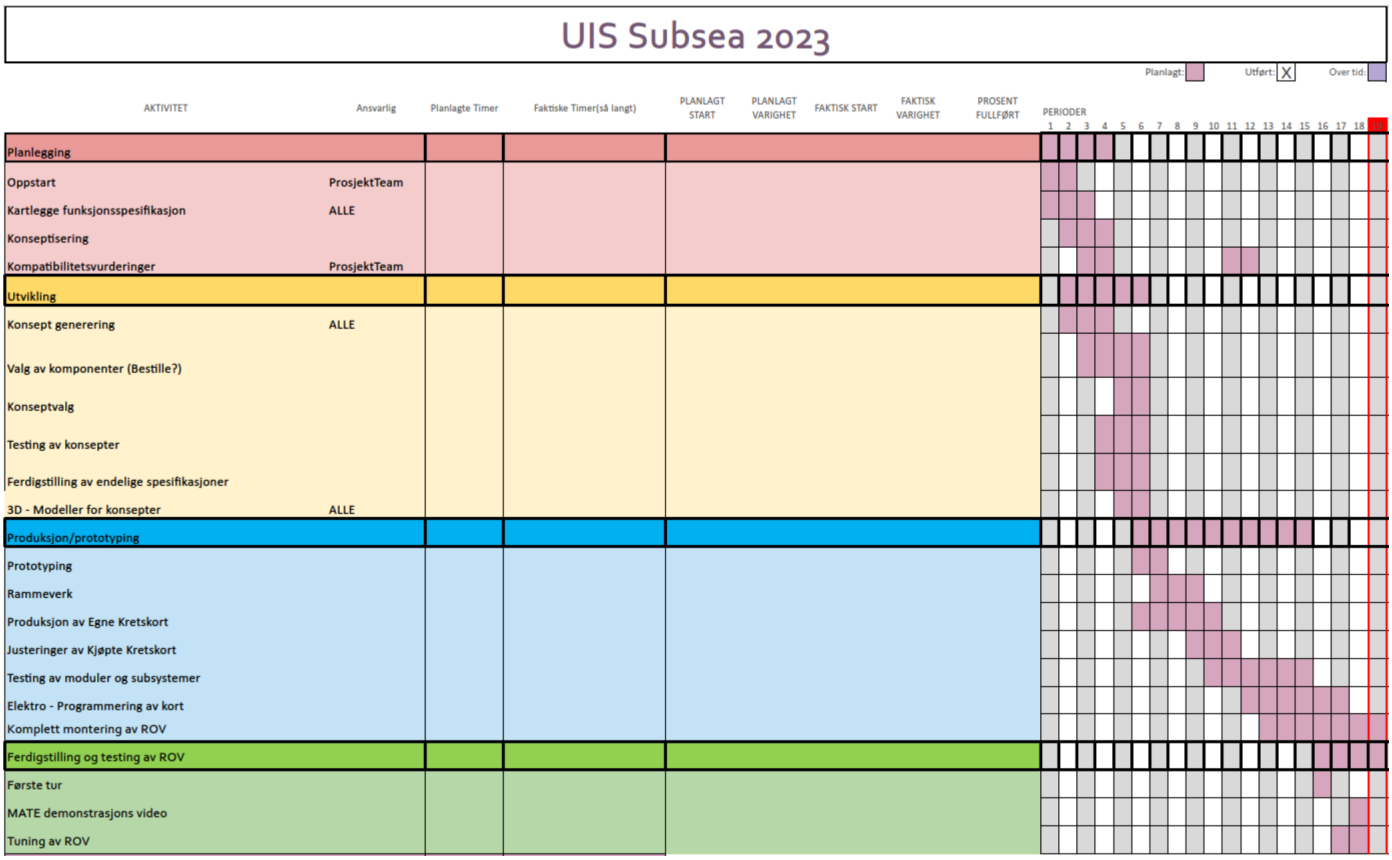
Kartlegging av Behov & Nødvendigheter:

N	Req. type	Hvem - Gruppenavn	Req. Beskrivelse	Vanskelighetsgr	Avklaringer	Priorite	Ansvarlig pers	Begrunnelse	Verifisering
15	Hardvare - Mjukvare - Nettverk	E - Kommunikasjon ROV	Kamera	Middels	Trenger stereo kamera for dybdesyn. Lav forsinkelse, flere kammera for å se flere vinkler. God synsvinkel	Høy		Dybde synn, bildestrøm for orientering	Sjekke med stoppeklokke, sjekke at synsvinkler er og
16		E - Kommunikasjon ROV	Sonar	Middels/Høy	N/A	Lav		For autonom navigering	
17	Hardvare	E - Kommunikasjon ROV	Grensesnittkort	Middels	Koblingskretskort for all elektronikk i ROV-en, f.eks. vhja. PCI.	Høy		Rask og effektiv montering, kommunikasjon og modulær utvidelse.	
19	Hardvare - Mjukvare - Nettverk	E - Kommunikasjon ROV	Ekstern Kommunikasjonskrets	Middels	Kommuikasjons mot topside system, valg av protokoll	Høy		For å kunne sende data mellom kommunikasjons kontroller, og topside	Verifisere at data stemmer og kan sendes begge veier.
20	Hardvare - Mjukvare - Nettverk	E - Kommunikasjon ROV	Kommunikasjonskrets	Middels	Design av kommunikasjons krets og valg av protokoll	Høy		For å kommunisere mellom mikrokontrollere samt også mellom kommunikasjonskontroller og mikrokontroller	Verifisere at data utveksling virker mellom mikrokontrollere og kommunikasjonskontroller

Figur 8.5: Behovsmatrise for kommunikasjon

Behovsmatrisen opplevdes som et nyttig verktøy i planleggingsfasen av prosjektet, og satt klare krav til hva hver gruppe måtte få på plass. For at et prosjekt med så mange involverte parter skal fungere,

må det være klare krav og tidsfrister som man skal forholde seg til. En overordnet tidsplan basert på erfaringer de satt igjen med fra fjorårets prosjekt ble da utarbeidet. Datoen vi jobbet mot som siste frist var 15. mai, som var fristen for innlevering av bacheloroppgaven og for å levere inn kvalifikasjonsvideo til MATE-konkurransen. Dette gav oss fire måneder på å ferdigstille prosjektet før denne fristen.



Figur 8.6: Overordnet tidsplan

Målsatt dato for første vanntest var 11. april, noe som i ettertid viste seg å være et optimistisk mål. Dette for å ha en margin for å kunne fikse uforutsette feil og problemer, samt for å ha mer tid til skriving av bachelor i sluttfasen av prosjektet. Dette gav gruppene tre måneder til å planlegge, designe og produsere sine systemer.

Alle gruppene fikk i oppgave å utlede en aktivitetsplan. Denne skulle gjenspeile den overordnede tidsplanen, og hvordan oppgaven til gruppen skulle utføres. For å opprettholde kontroll på prosjektets fremgang og samkjøre gruppene, ble det holdt ukentlige møter der prosjektgruppen og en representant fra hver gruppe deltok. Formålet med disse møtene var å samkjøre gruppene og diskutere problemstillinger. Løsninger, og tidligere prosjekt for UiS Subsea ble analysert for å se hva som hadde fungert bra og dårlig tidligere år.

I et tverrfaglig prosjekt med mange involverte parter, slik som dette prosjektet, er det oppgaver som faller utenfor de definerte oppgavene. Prosjektgruppen må holde en oversikt over disse oppgavene, og deretter delegere ansvar til grupper med ledig kapasitet.

8.8.1.2 Utviklingsfasen

I neste fase av prosjektet, der gruppene hovedsaklig jobbet med design og produksjon av systemene sine, endret prosjektgruppens oppgave seg litt fra et overordnet fokus til mer individuelle problemstillinger

som dukket opp. De ukentlige møtene fortsatte som før, der agendaen endret seg noe til å fokusere mer på fremdrift, komplikasjoner og forespørsler. Typisk agenda for møtene var:

- Felles generell info (Hinder, personer som ikke er tilgjengelig fremover, milepæler)
- Hva har gruppen gjort siden forrige møte?
- Hva er planen til gruppen frem til neste møte?
- Hva trenger gruppen fra andre grupper eller leverandører?
- Status på komponenter med leveringstid

En ny utfordring som oppstod, var å effektivt dele informasjon om hva de forskjellige gruppene behøvde fra hverandre. Det ble derfor oppfordret til at gruppene skulle sette opp møter med hverandre for å få klarert og kartlegge hvordan systemene deres skulle fungere sammen. Også her var det nye ekstraordinære prosjektoppgaver som kom frem i lyset, og prosjektgruppen måte dermed få kontroll på disse og delegere ansvaret.

8.8.1.3 Sluttfasen

Sluttfasen av et prosjekt som dette gjenspeiler ofte den totale innsatsen lagt inn i de tidligere fasene av prosjektet. Det som ofte skjer i slike prosjekt, er at ting tar mer tid en først estimert, da folk har en tendens til å være tidsoptimistiske. I første delen av denne fasen, ble det fokusert på testing og ferdigstilling av individuelle systemene. Etter hvert som systemene ble ferdigstilt og koblet sammen, kom den faktiske progresjonen til de ulike gruppene frem. Alle gruppene hadde feil med sine systemer, men det ble tydelig at alvorlighetsgraden til manglene, gjenspeilte tidligere innsats i prosjektet. Det var også flere tilleggsoppgaver som måtte fordeles, noe som førte til uenigheter om hva de forskjellige gruppenes ansvar var. Dette førte til at enkelte grupper fikk et merarbeid som ikke var forventet. Konsekvensen av dette ble unødvendig lange kvelder og arbeid i helgene. Etter et par uker med testing, ble problemer utbedret og nye løsninger ble designet, produsert og testet. I sluttfasen var det også en del arbeidsoppgaver angående MATE-konkurranse som måtte fullføres. Dette gav en ekstra belastning på en allerede stor arbeidsmengde.

8.8.1.4 Erfaringer

Totalt sett har prosjektet vært vellykket. Det er lagt inn en stor mengde arbeid, og det er produsert mye på et kort tid. Ingen i prosjektet hadde spesielle forkunnskaper eller erfaringer med ROV-er og undervanns-teknologi, noe som gir prosjektet et utfordrende utgangspunkt. Det har oppstått mange utfordringer gjennom prosjektet, og flere hadde vært mulig å unngå. Det var flere uenigheter mellom grupper om hva hver gruppes ansvar var. Det endte ofte opp med at disse oppgavene ble utsatt eller at ingen av gruppene ønsket, eller hadde mulighet til å ta ansvar for oppgavene.

Begynnelsen av prosjektet og starten av utviklingsfasen var preget av at flere prosjektdeltakere hadde eksamener, kontinuitets-eksamener, eller prosjektdeltakere som ikke var tilgjengelige den første måneden av prosjektet på grunn av utveksling. Dette er helt legitime grunner for å ikke være fullstendig tilgjengelig, men viste seg å være problematisk med den store arbeidsmengden et slikt prosjekt krever. Flere av prosjektdeltakerene reiste også på studieturer de første månedene som bidro til denne utfordringen. Dette fører til en "ripple-effekt" av forsinkelser.

Alle prosjektdeltakerene er studenter. Krav til innsats og resultater, vil derfor komme fra gjensidig respekt for at den individuelle gruppens progresjon kan ramme deres medstudenters progresjon. Likevel skjer det forsinkelser og mangler som det er vanskelig for prosjektgruppen å sette inn ekstraordinære tiltak for å gjøre noe med.

En av de store fordelene til en studentorganisasjon som UiS Subsea, er å ha tilgang til kontorer på UiS-campus. Dette ble flittig brukt av mange grupper, og er en viktig kanal for deling av informasjon. Organisasjonen har også tilgang til lab-arealer med måleutstyr samt maskineringsverksted. Flesteparten av gruppene valgte å oppholde seg på disse områdene, og det ble dannet en naturlig informasjonsflyt mellom gruppene som oppholdte seg der. Grupper som isolerte seg fra disse fellesområdene ble mindre involverte i prosjektet og falt utenfor den viktige informasjonskanalen. Et problem er at UiS Subsea sitt kontor ikke har nok plass til å akkommodere alle gruppene. En annen utfordring var at grupper arbeidet på forskjellige tidspunkt. Noen grupper jobbet stort sett i normal arbeidstid, mens andre kom gjerne klokken 14:00 og jobbet utover kvelden. Dette gjør det vanskelig å koordinere samarbeid og gjør arbeids- og informasjonsflyten mindre effektiv, og forsterker "ripple-effekten" av forsinkelsene.

I en etablert studentorganisasjon som UiS Subsea har et slikt prosjekt som dette en fordel med at det eksisterer et etablert rammeverk for prosjektet, og tidligere medlemmer har vært behjelpelig med å dele kunnskapen de sitter på. Dette gjør at et så stort prosjekt kan realiseres, noe som ikke hadde vært mulig hvis en skulle begynt helt på nytt.

Prosjektlederansvaret har vært krevende, og tatt en stor del av tiden som var satt av til bacheloren. Prosjektgruppen har kanskje vært for stor i år og ansvarsområder har ikke vært tydelig nok definert. To prosjektlederroller har fungert delvis, med at den ene tok mer ansvar for organisatoriske problemstillinger samt mekanisk ansvar, og den andre tok ansvar for konkurranser og sammenknyttning av system. Problemet med to prosjektledere er at det bidrar til mer utydelige ansvarsområder. Møtene opplevdes som noe ineffektive, og det foreslås en struktur der teknisk leder tar møte med grupper relatert til ansvarsområdet, og at det heller er et prosjektledelsemøte. Det bør legges mer vekt på at tidsfrister skal opprettholdes og at det kreves mer grunngivning for forsinkelser.

Et prosjekt av denne størrelsen er krevende, og innebærer en god del ekstraarbeid utover å skrive en bacheloroppgave. Resultatet av prosjektet er sterkt knyttet til den ekstra innsatsen som legges inn av gruppene. For at prosjektet skal realiseres er det helt avgjørende at alle bidrar. Oppgaver som ikke tilhører en spesifikk gruppe, faller under prosjektgruppens ansvar og må delegeres. Oppgavene som blir delegerte blir ofte nedprioriterte av mottakende gruppe da arbeidsmengden med egen bacheloroppgave og system er stor. Gruppene har også måtte velge mellom å bruke tid på prosjektet, og å bruke tid på sin egen bacheloroppgave. Dette har blitt prioritert forskjellig på de ulike gruppene, og det har vært krevende å få til en god kombinasjon av de to.

Totalt sett har alle gruppene gjort en solid innsats med å ha levert et flott sluttprodukt. Det har vært god moral i prosjektet og gruppene har samarbeidet godt. Det har oppstått minimalt med konflikter mellom gruppene, og de konfliktene som har oppstått har kommet fra en stor arbeidsmengde. Beviset på et vellykket prosjekt kan sees i videoen under, hvor det er lagd en demonstrasjonsvideo for MATE-konkurransen, for å kvalifisere oss til denne.

Link til video for MATE kvalifisering: https://www.youtube.com/watch?v=yg9I5MSf8y8

8.8.2 Økonomi og budsjett

For at en studentorganisasjon som UiS Subsea skal være i stand til å gjennomføre et prosjekt som dette, er den økonomiske siden av prosjektet helt essensiell. Gruppen ansvarlig for kraftforsyningen,

bestående av Tomas Håkonsen og Håvard Thomlevold, ble delegert ansvaret som sponsoransvarlige i år.

Sponsoransvarlige i år har gjort en formidabel jobb, og sørget for at de økonomiske midlene til å gjennomføre prosjektet har kommet inn. Den største sponsorposten kommer fra Universitetet i Stavanger, men bedrifter fra næringslivet har også kommet med betydningsfull støtte. I tillegg til økonomisk støtte, har også UiS og bedrifter bidratt med blant annet komponenter og halvfabrikata. I fjor ble det for første gang på mange år arrangert en "Subsea-Dag". Denne dagen ble også arrangert i år med god suksess. I tillegg til den økonomiske siden av dette, ble dette også en god mulighet for studenter å komme i kontakt med bedrifter innen subsea-næringen.

Et budsjett som angir hva inntektene skal gå til, er helt avgjørende for god økonomisk styring. I tabell 8.1 nedenfor er årets budsjett for de ulike gruppene som bygger ROV-en og flyteren.

Fordelingsgrunnlag	**Budsjett [kr]**
Regulering (ELE)	3 000,00
Sensor (ELE)	6 000,00
Kommunikasjon (ELE)	10 000,00
Kraftforsyning (ELE)	10 000,00
Float (ELE)	8 000,00
ROV-design (MASK)	12 000,00
Manipulatordesign (MASK)	8 000,00
GUI (DATA)	2 500,00
Bildebehandling (DATA)	2 500,00
Felles	45 000,00
Total	**107 000,00**

Tabell 8.1: Årets budsjett for de ulike gruppene

Budsjettet ovenfor viser hva det var forventet at de ulike gruppene skulle bruke på sin del av prosjektet. Den desidert største posten her var fellesutgifter. Dette var til ting som f.eks. thrustere, tjoren og ny PC til kontrollstasjon. Av budsjettet, ble det satt av 10 000,00,- til vår del av prosjektet. Hvordan vi har brukt disse midlene er gjengitt i tabell 8.2 nedenfor.

Utstyr	Antall	Enhetspris [kr]	Totalt [kr]
64GB SD-kort	1	299,90	299,90
Nvidia Jetson Nano 4GB	1	1 863,30	1 863,30
OV2710 USB-kamera	2	544,73	1 089,46
IMX219-83 Stereokamera	2	476,83	953,66
Servomotor SG90	1	59,68	59,68
Kretskort (Hovedkort)	5	48,054	240,27
Kretskort (Bakplate PA)	5	20,924	104,62
Kretskort (Bakplate PB)	5	20,924	104,62
Kretskort (Bakplate PC)	5	20,924	104,62
Kretskort (CSI)	5	8,452	42,26
Kretskort (USB)	5	8,452	42,26
Samtec PCIe 64 Way	40	36,46	1 458,4
Frakt			258,09
Diverse plugger			1 122,53
Toll diverse			1 225,84
Sum			-8 970,31
Tildelt budsjett			10 000,00
Overskudd/Underskudd			**1 029,69**

Tabell 8.2: Årets regnskap for Kommunikasjonsgruppen

Tabellen ovenfor gjenspeiler utgiftene vi har hatt relatert til vår oppgave i år. Innebakt i posten "Diverse plugger", inngår også "standardplugger". Dette er plugger som ble bestilt for at alle gruppene kunne benytte seg av dem. Så noe av denne posten skulle nok i realiteten blitt dekket av fellesposten i budsjettet. CAN-modulen SPI CAN Click som ble brukt i prosjektet er ikke oppført i budsjettet, da UiS Subsea hadde et par av disse modulene liggende på verkstedet.

Fra tabellen, kommer det frem at vi endte opp med å bestille to IMX219-83 stereokamera. Stereokameraet kom med to CSI-kabler av typen FFC. Under montasje av det første stereokameraet, ble den ene av disse kablene ødelagt. Disse kablene var vanskelige å oppdrive og det virket som om de var spesiallagd for IMX219-83. Vi så dermed ingen annen løsning enn å bestille et nytt kamera for å få tak i en ny kabel.

Alle utgiftene til oppgaven vår summerte seg til slutt opp til 8 970,31,-. Med et budsjett på 10 000,- satt vi til slutt igjen med et overskudd på 1 029,69,- etter at ROV-en var ferdigutviklet.

Kapittel 9

Videre arbeid

Selv om vi har en fungerende ROV innen tidsfristen, er det fortsatt små forbedringer som kan gjøres for å optimalisere den. Arbeidet som gjenstår, og som vil bli jobbet med videre, vil bli tatt opp i dette kapittelet. Dette i håp om at sluttproduktet kan forbedres, og dermed prestere bedre i de konkurransene det skal deltas i.

Granske mer på SPI-buss: For å finne ut hvorfor feilmeldinger kommer fra SPI-bussen, ønsker vi å koble opp bussen på et skop. Vi vil så samenligne klokkesignalet med signalet fra MISO for å se om bittet treffer ved stigende flanke på klokkesignalet. Dersom vi ser uregelmessig timing på MISO i forhold til klokkesignalet, vil vi se nærmere på hva som kan gjøres med SPI-linjene. Siden vi har reserve av kommunikasjonskort, vil vi koble opp denne kretsen på reservekortet med resistanser loddet på SPI-linjene. Dersom dette viser seg å fungere, vil vi øke hastigheten på bussen tilbake til 10 MHz.

Printe nye deler i ABS: Delene som ble laget av PLA for bruk inne i elektronikkhus, vil erstattes med ABS for å bedre kunne tåle temperaturene inne i elektronikkhuset.

Kamera: Valgt USB-kamera var godt egnet for bruk under vann, og det ble bestemt at det skulle bestilles enda et USB-kamera av typen OV2710. Dette betyr at det må designes og produseres et nytt design for permanent løsning for både stereokamera og OV2710 i kuppelen. Også dette designet vil bli 3D-printet i ABS. Det blir også bestilt nye linser til stereokamera for å kunne videreføre eventuell erfaring med hvordan resultatet blir.

Justering av vinkel for elektronikkhus: For å få rett vinkel hver gang, og for å slippe å justere denne inn med data fra IMU, ønsker vi å lage en bedre festemetode for kretskortstabelen. Ved å måle mer nøyaktig på bakdelen, kan vi finne et design for ringene som kun har et hull for montering der orienteringen av elektronikkhuset er rett.

Granske høye makstider for kommunikasjon: For å optimalisere kommunikasjonen i systemet, er vi nødt til å finne ut hva årsaken til de høye makstidene i testrapport A.1, og utbedre det. Som nevnt i diskusjonen, mistenkes det at problemet ligger i programvaren. Det vil derfor utvikles ny programvare for å lokalisere problemet. Dette vil være programvare hvor det eneste formålet, er sending og mottak av meldinger. Denne skal være så minimalistisk som mulig. Hvis problemet forsvinner, vil det bli lagt til en og en funksjon for å lokalisere problemet.

Vifter på kort for motorkontrollere: Siden dette ikke ble gjort ved design av kretskort for montorkontrollere, ønsker vi å sette inn en større vifte for å kjøle ESC-ene. Vi vil da bruke større vifter som kan monteres mellom kretskortene i stabelen i vertikal retning. Dette muliggjør for mer plass til større vifter med høyere effekt.

Kapittel 10

Konklusjon

Gruppen har over en periode på fire måneder designet og utviklet et system som realiserer kommunikasjon mellom de ulike systemene internt i ROV-en, men også mellom ROV-en og kontrollstasjonen. Oppgaven har også inkludert innhenting av videostrømmer som skal sendes opp til land. Prosessen underveis ved å lage dette systemet og komme i mål, opplevdes som suksessfullt, og prosessen har vært givende og læringsrikt.

Elektronikkhus Utformingen ved å ha vertikale kretskort i en stabel viste seg å fungere bra. Designet med tre bakplater viste seg som en solid og effektiv koblingsmetode mellom kretskortene. Å bruke PCI-kontakter viste seg å være en god løsning, både med tanke på mekanisk styrke, og for tilkoblingsmuligheter. Disse løsningene er i stor grad tatt fra fjorårets oppgave. Å ha mange tilgjengelige baner for ruting av signal i bakplantene reduserte mengden kabler i elektronikkhuset, noe som gjorde det ryddig. Ved å ha kretskortene i en stabel, ble også testing av kretskort enklere. En problemstilling vi hadde med kretskortstabelen var at denne var mulig å montere feil sammen, da det ikke var noen mekanisk forigling som forhindret dette. Ved å bruke CAD i *Fusion*, ble prosessen bak designet av kretskortstabelen visualisert. Dette gjorde at montering av deler kunne gjøres på forhånd i tegningen for å forhindre kollisjoner. Ved å ha designet som CAD, ble det også enkelt å prototype tilpassede deler som kunne 3D-printes. Det ble også importert 3D-modeller for blant annet kamera og servo, noe som for eksempel gjorde design av kameratilt mye lettere. Designet av maskinerte kamerahus med O-ringer, viste å være tett ved første vanntest. Dette er vi fornøyd med.

Kommunikasjon: Valgte kommunikasjonsstandarder og protokoller er godt egnet for formålet. Videreføringen av fjorårets løsning med CAN-buss for kommunikasjonen internt i ROV-en, viste seg å være en robust og fleksibel løsning. Bruk av TCP-protokollen over Ethernet er valgt for kommunikasjonen mellom kontrollstasjonen og ROV-en. Denne løsningen gir en stabil kommunikasjon mellom ROV og land, med tilfredstillende lav forsinkelse for kontrolldata og prosessdata. UDP-protokollen er valgt for sending av video fra ROV-en til land, med RTP-protokollen for å håndtere videopakkene. Dette muliggjør bruk av "multicast". Resultatet fra testrapport A.1 og testrapport A.3 viser at vi har oppnådd tilfredstillende lav forsinkelse for både kommunikasjon og videosending. Forsinkelsen mellom mikrokontrollerene er $\approx 232\mu s$. Gjennomsnittlig forsinkelse for en TCP-melding er $0.535ms$. Den gjennomsnittlige forsinkelsen mellom mikrokontrollerene og kontrollstasjonen tur-retur er $2.13ms$. Resultatet av dette er at kommunikasjonen har en gjennomsnittlig forsinkelse tur-retur per melding som er $\approx 1ms$ lavere en fjorårets prosjekt[31]. Testerapport A.3, viser at forsinkelsen på videosendingen ligger mellom $63ms$ og $129ms$, som var bedre enn målsatt funksjonsspesifikasjon. Det ene problemet vi opplevde under testingen av kommunikasjonen, var SPI-linjen mellom CAN-modulen og kommunikasjonskontrolleren. Her ble klokkefrekvensen midlertidig satt ned for å bli kvitt problemet. Her burde

det ha vært mer fokus på kretskortdesign av SPI-banene i planleggingsfasen for å unngå problem.

Maskinvare Bruk av tilgjengelig maskinvare muliggjorde verdifull testing av konsept. Vi fikk tidlig låne modulkortene Nvidia Jetson Nano og SPI CAN Click, samt et stereokamera for videostrøm. Det var dermed mulig å teste disse komponentene ilag og bekrefte konseptvalg. For å kunne bruke USB- og CSI-kamera, var det viktig å bevise at Jetson Nano var kraftig nok til å prosessere bildedataen fra disse. Å ha maskinvare på plass gjorde at vi kunne utvikle programvare tidlig, noe som førte til at kommunikasjonskretsen fort ble operasjonell etter levering av kretskort.

Kretskort Selv om prosessen rundt design av kretskort var utfordrende for gruppen, er vi relativt fornøyd med et fungerende resultat. Spesielt er vi fornøyd med å være tidlig ute med testing av kretskort og oppsett. Utlegg på kretskort burde tidligere vært sendt til inspeksjon for å utelukke flere feil. Sjekk av fotavtrykk burde vært grundigere, da pinnen for INT på SPI CAN Click var rutet feil plass. Gruppen burde også hatt mer konsultasjon generelt rundt kretskortdesign. Likevel har læringsprosessen vært givende. Bakplatene sitt design ble meget bra, og viste seg å være et solid og driftsikkert. Det å senke Jetson Nano til å monteres midt på kretskortet, viste seg å spare mye plass i elektronikkhuset. Det gjorde også at Jetson enkelt kan demonteres, da sammenkoblingen skjer via en header. Selv om valg av komponenter til kretskort var litt dyrere enn nødvendig, har dette hatt liten betydning for budsjettet. Det å lage tolagskort var et økonomisk rett valg, og hadde liten effekt på designet av kretskortet. Dårlig ruting av linjene til SPI-bussen viste seg å være et problem for signalintegriteten mellom SPI CAN Click og Jetson Nano. Dette kommer av for lite kjennskap til både PCB-design, men også kjennskap til SPI som høyhastighetssignal. Komponentene på modulkortet SPI CAN Click burde vært rutet direkte på kretskortet for å unngå problem med signalintegritet, spesielt for å rute CAN-bussen med rett karakteristisk impedans. Skjemategninger til denne lå tilgjengelig på databladet til modulkortet, noe som hadde gjort kretsen enkel å designe på kommunikasjonskortet.

Kamera Det ble valgt to USB-kamera og ett CSI-kamera for ROV-en, der CSI-kamera var et stereokamera av typen IMX219-83. Stereokamera ble valgt for å drive med avstandsmåling til bildebehandling, og krav til oppløsning ble satt til 1920x1080p. USB-kameraene var av typen OV2710 og ble brukt for oversikt under ROV, og mot manipulator. Det var derfor ikke satt samme krav til oppløsning på USB-kameraene, da denne ble satt til 1280x720p. Synsvinkelen til stereokameraet viste seg å være for smal til kjøring av ROV. Selv om dette ble regnet på, undervurderte vi hvor bred synsvinkel som var hensiktsmessig ved kjøring. I tillegg til dette, viste det seg at stereokameraet var dårlig egent til undervannsbruk, da kameraet var avhengig av mye lys for å få et klart bilde. For å løse dette burde det vært valgt en linse med bredere synsvinkel, og større blenderåpning. USB-kamerane derimot, fungerte som ønsket og gav bilder med tilstrekkelig kvalitet.

Programvare Programmet som er utviklet i Python er stabilt og fungerer som tiltenkt. Som merket i testrapport A.1 er makstiden på meldingene noe høy, selv om gjennomsnittstiden er tilfredstillende lav. Hva dette skyldes er usikkert men dette kan komme fra at koden er skrevet i tolket språk, og vil bli undersøkt ytterligere. Målet med koden for kommunikasjon var at den skulle være modulær, og enkel å utvide. Dette mener vi er oppnådd etter det ble tatt i bruk hashtabeller for å legge til ny data som skal sendes eller mottas. Det er lagd funksjoner for lesing av temperatur, endre vinkelen på servokamera, og styre gStreamer-rørene. Disse har lagt til ekstra funksjonalitet som har kommet godt med under bruk av systemet. I demonstrasjonsvideoen kan man se at tilting av kamera blir flittig brukt for å justere synspunktet.

gStreamer har fungert meget bra for prosjektet, men har gitt oss en bratt læringskurve for å oppnå ønsket resultat. Forsinkelsen på videostrømmen ble målt til omtrent $96ms$. Maskinvareakselererte en-

koding og dekoding gjør denne prosessen effektivt uten å påvirke kommunikasjonen negativt. Etter å ha vært på bransjemesser og snakket med leverandører, er alt under $120ms$ sett på som målet for et hvert kamerasystem over Ethernet. Vi ser derfor på dette som et vellykket resultat.

Selv om prosjektet har blitt betegnet som "Speedrun i ROV-bygging", har prosjektet i sin helhet vært vellykket. ROV-en klarte de nødvendige oppgavene for å kvalifisere seg til MATE konkurransen i USA. Det å ha en ferdig ROV før innlevert bachelor var et stort mål for organisasjonen, og krevde mye av medlemmene. Det gjenstår enda arbeid for å optimalisere ROV-en fram mot konkurransen, slik at denne får best mulig vinnersjanse. Kvalifiseringsvideo med ferdig ROV er tilgjengelig via denne linken: https://www.youtube.com/watch?v=yg9I5MSf8y8

Bibliografi

[1] R. Jehangir, "What is an underwater rov?" https://bluerobotics.com/learn/what-is-an-rov/, 2022 July 18th | Retrieved: 2023 February 9th.

[2] N. T. Centre, "Remotely operated vehicle (ROV) services." [Online]. Available: https://www.standard.no/pagefiles/978/u-102r1.pdf

[3] BlueRobotics. BlueROV2. ROV Klasse ! [Online]. Available: https://bluerobotics.com/store/rov/bluerov2/

[4] R. T. S. LTD. Rovtech solutions - remotely operated vehicles (ROVs) product gallery. ROV Klasse II. [Online]. Available: https://rovtechsolutions.com/products/underwater-rovs/

[5] OCEANEERING. ROV systems | oceaneering. ROV Klasse III. [Online]. Available: https://www.oceaneering.com/rov-services/rov-systems/

[6] SCANMUDRING. Equipment - scanmudring %. Crawler. [Online]. Available: https://scanmudring.no/equipment/

[7] A. A. I. V. A. Center. BPAUV - AUVAC. ROV Klasse VI. [Online]. Available: https://auvac.org/34-2/

[8] K. S. TECHNOLOGI. Survey ROV. ROV Klasse VII. [Online]. Available: https://kystdesign.no/rovs/surveyor/

[9] W. H. O. Institution. Henry melson stommel medal - woods hole oceanographic institution. [Online]. Available: https://www.whoi.edu/who-we-are/about-us/people/awards-recognition/henry-melson-stommel-medal/

[10] G.-B. G. O. B. Array. Floats | GO-BGC. [Online]. Available: https://www.go-bgc.org/floats

[11] Ecomagazine, "Robotic floats provide new look at ocean health and global carbon cycle," https://www.ecomagazine.com/news/research/robotic-floats-provide-new-look-at-ocean-health-and-global-carbon-cycle, 2023.

[12] MATE. MATE ROV competition, MATE-II. [Online]. Available: https://web.archive.org/web/20221130104515/https://www.materovcompetition.org/content/about-us/

[13] ——. About MATE-II, MATE-II. [Online]. Available: https://web.archive.org/web/20221203081732/https://mateii.org/about-mate-ii/

[14] ——, "MATE konkuranse manual, MATE-II." [Online]. Available: https://files.materovcompetition.org/2023/2023_EXPLORER_Manual_FINAL_1_17_2023_withcover.pdf

[15] Havforskningsrådet. Havforskningstiåret. [Online]. Available: https://www.forskningsradet.no/om-forskningsradet/portefoljer/hav/havforskningstiaret/

[16] "trykk – fysikk." [Online]. Available: http://snl.no/trykk_-_fysikk

[17] O. Lohne, "flytegrense." [Online]. Available: http://snl.no/flytegrense

[18] "Yield (engineering)." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Yield_(engineering)&oldid=1124425305

[19] G. M. Haarberg, "korrosjon." [Online]. Available: http://snl.no/korrosjon

[20] B. Pedersen, "rust." [Online]. Available: http://snl.no/rust

[21] R. Johnsen, "CORROSION CHALLENGES WITH THE USE OF ALUMINIUM IN OFFSHORE AND MARINE ENVIRONMENTS."

[22] "Heat transfer coefficient, wikipedia," https://en.wikipedia.org/wiki/Heat_transfer_coefficient, (Sett: 23.01.2023).

[23] "Thermal conductivity, wikipedia," https://en.wikipedia.org/wiki/Thermal_conductivity, (Sett: 23.01.2023).

[24] J. Fourier, *The Analytical Theory of Heat*, 1822, (Sett: 23.01.2023).

[25] "Newton's law of cooling." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Newton%27s_law_of_cooling&oldid=1146584085

[26] "termisk konduktivitet." [Online]. Available: http://snl.no/termisk_konduktivitet

[27] Nondestructive Evaluation Physics : Materials. [Online]. Available: https://www.nde-ed.org/Physics/Materials/Physical_Chemical/ThermalConductivity.xhtml

[28] "toleranse – teknikk." [Online]. Available: https://snl.no/toleranse_-_teknikk

[29] "termoforming." [Online]. Available: https://snl.no/termoforming

[30] Ultimate 3D Printing Material Properties Table. [Online]. Available: https://www.simplify3d.com/resources/materials-guide/properties-table/

[31] T. M. o. C. N. Mats Røste, "Uis subsea: Bildebehandling og kommunikasjon," Universitet i Stavanger, Tech. Rep., 2022, (Sett: 23.01.2023).

[32] CPSdrone – Making Underwater Drones. [Online]. Available: https://www.cpsdrone.com/

[33] A. Almar-Næss, "bruddanvisning – mekanikk." [Online]. Available: https://snl.no/bruddanvisning_-_mekanikk

[34] "halvfabrikat." [Online]. Available: http://snl.no/halvfabrikat

[35] "dreiing." [Online]. Available: http://snl.no/dreiing

[36] O. Skjeggedal, "fres." [Online]. Available: http://snl.no/fres

[37] "Junction temperature." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Junction_temperature&oldid=1110084091

[38] Overall Heat Transfer Coefficient Table Charts and Equation. [Online]. Available: https://www.engineersedge.com/thermodynamics/overall_heat_transfer-table.htm

[39] Original prusa i3 mk3s+. Polyalkemi. [Online]. Available: https://polyalkemi.no/produkt/original-prusa-i3-mk3s/

[40] Formbot Voron V0.1 120x120x120mm. Technology Outlet. [Online]. Available: https://technologyoutlet.co.uk/products/formbot-voron-v0-120x120x120mm

[41] . M. The dangers of ABS Filament Fumes. Purex International Ltd. [Online]. Available: https://www.purex.co.uk/blog/the-dangers-of-abs-filament-fumes/

[42] "buesveising." [Online]. Available: https://snl.no/buesveising

[43] Fusion 2 | inter-supply as. [Online]. Available: https://inter-supply.no/produkt/fusion-2/

[44] E. S. Joakim Skaar, "Sveisekvalitet ved ekstrudersveising av PEHD."

[45] "i^2c, wikipedia," https://en.wikipedia.org/wiki/I2C , (Sett: 03.02.2023).

[46] "i^2c," https://no.wikipedia.org/wiki/I2C , (Sett: 03.02.2023).

[47] "Pull-up resistor, wikipedia," https://en.wikipedia.org/wiki/Pullup_resistor , (Sett: 03.02.2023).

[48] "What is open drain : Configuration & its applications," https://www.watelectronics.com/open-drain/ , (Sett: 20.02.2023).

[49] "Um10204," https://www.pololu.com/file/0J435/UM10204.pdf , (Sett: 23.03.2023).

[50] "Difference between i2c and spi (i2c vs spi)," https://medium.com/@rjrajbir24/difference-between-i2c-and-spi-i2c-vs-spi-c6a68d7242c4 , (Sett: 21.02.2023).

[51] "i^2c primer: What is i^2c? (part 1)," https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html , (Sett: 21.02.2023).

[52] G. Brown, "Discovering the stm32 microcontroller," Tech. Rep., 2012, (Sett: 26.01.2023).

[53] "Serial peripheral interface, wikipedia," https://en.wikipedia.org/wiki/Serial_Peripheral_Interface , (Sett: 24.01.2023).

[54] "I2c vs spi vs uart – introduction and comparison of their similarities and differences," https://www.totalphase.com/blog/2021/12/i2c-vs-spi-vs-uart-introduction-and-comparison-similarities-differences/ , (Sett: 31.01.2023).

[55] "Can bus, wikipedia," https://en.wikipedia.org/wiki/CAN_bus, (Sett: 23.01.2023).

[56] "Practical tips: Can-bus," https://www.kmpdrivetrain.com/paddleshift/practical-tips-can-bus/, (Sett: 23.01.2023).

[57] "Sn65hvd23x 3.3-v can bus transceivers datasheet," https://www.ti.com/lit/ds/symlink/sn65hvd230.pdf , (Sett: 15.02.2023).

[58] "Can bit monitoring and stuffing," https://embedclogic.com/can-protocol/can-bit-monitoring-and-bit-stuffing/ , (Sett: 27.01.2023).

[59] "Controller area network (can) programming tutorial 10: bit wise bus arbitration animation," https://www.youtube.com/watch?v=XdtdBUHEdOI , (Sett: 23.03.2023).

[60] "Cyclic redundancy check, wikipedia," https://en.wikipedia.org/wiki/Cyclic_redundancy_check#CRC-15-CAN, (Sett: 24.01.2023).

[61] M. Tengesdal, "Frå transistor til datamaskin." [Online]. Available: https://ebooks.uis.no/index.php/USPS/catalog/book/196

[62] A. B. Florian Hartwich, "The configuration of the can bit timing," http://www.oertel-halle.de/files/cia99paper.pdf, Robert Bosch GmbH, Tech. Rep., (Sett: 17.02.2023).

[63] "Nucleo-g431kb," https://www.digikey.no/no/products/detail/stmicroelectronics/NUCLEO-G431KB/10231583?sN4IgTCBcDaIHIFUDCAZAogeQLQHEAsAzAIwDSAQiALoC%2BQA , (Sett: 07.02.2023).

[64] "Calculate can bit timing parameters," https://www.cnblogs.com/shangdawei/p/4716784.html , (Sett: 24.03.2023).

[65] "Can-fd, wikipedia," https://en.wikipedia.org/wiki/CAN_FD, (Sett: 24.01.2023).

[66] "Stm32 fdcan in loopback mode," https://controllerstech.com/stm32-fdcan-in-loopback-mode/ , (Sett: 21.02.2023).

[67] "Can fd challenges fieldbuses and industrial ethernet in special purpose machinery," https://www.automation.com/en-us/articles/2014-1/can-fd-challenges-fieldbuses-and-industrial-ethern , (Sett: 21.02.2023).

[68] "Osi model, wikipedia," https://en.wikipedia.org/wiki/OSI_model , (Sett: 10.02.2023).

[69] "Osi model, cisco," https://learningnetwork.cisco.com/s/article/osi-model-reference-chart , (Sett: 10.02.2023).

[70] "Metropolitan area network, wikipedia," https://en.wikipedia.org/wiki/Metropolitan_area_network , (Sett: 31.01.2023).

[71] "What are the ethernet standards," https://www.elandcables.com/the-cable-lab/faqs/faq-what-are-the-ethernet-standards , (Sett: 02.02.2023).

[72] "Fiber-optic cable, wikipedia," https://en.wikipedia.org/wiki/Fiber-optic_cable , (Sett: 02.02.2023).

[73] "Transmission control protocol, wikipedia," https://en.wikipedia.org/wiki/Transmission_Control_Protocol , (Sett: 01.02.2023).

[74] "Internet protocol suite, wikipedia," https://en.wikipedia.org/wiki/Internet_protocol_suite , (Sett: 10.02.2023).

[75] "User datagram protocol, wikipedia," https://en.wikipedia.org/wiki/User_Datagram_Protocol , (Sett: 01.02.2023).

[76] "Real time transport protocol, wikipedia," https://en.wikipedia.org/wiki/Real-time_Transport_Protocol , (Sett: 20.02.2023).

[77] "Usb, wikipedia," https://no.wikipedia.org/wiki/USB , (Sett: 24.02.2023).

[78] "Usb: Port types and speeds compared," https://tripplite.eaton.com/products/usb-connectivity-types-standards , (Sett: 24.02.2023).

[79] "full-duplex," https://www.techtarget.com/searchnetworking/definition/full-duplex , (Sett: 14.03.2023).

[80] "Datamaskinarkitektur (ele210)," https://www.uis.no/nb/course/ELE210_1 , (Sett: 27.02.2023).

[81] "Stm32g431kb," https://www.st.com/en/microcontrollers-microprocessors/stm32g431kb.html , (Sett: 24.02.2023).

[82] "Raspberry pi 4," https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ , (Sett: 06.02.2023).

[83] "Nvidia jetson, wikipedia," https://en.wikipedia.org/wiki/Nvidia_Jetson , (Sett: 05.05.2023).

[84] "Jetson nano system-on-module data sheet, nvidia," Jetson Nano Developer Kit , (Sett: 09.02.2023).

[85] "Okdo rock 4 model c+ 4gb single board computer rockchip rk3399-t arm cortex-a72," https://www.okdo.com/us/p/okdo-rock-4-model-c-4gb-single-board-computer-rockchip-rk3399-t-arm-cortex-a72/ , (Sett: 07.02.2023).

[86] "Single-board computer." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Single-board_computer&oldid=1128135516

[87] "Arduino can bus tutorial | interfacing mcp2515 can module with arduino," https://how2electronics.com/interfacing-mcp2515-can-bus-module-with-arduino/ , (Sett: 07.02.2023).

[88] "Mcp2515 stand-alone can controller with spi interface data sheet," https://ww1.microchip.com/downloads/en/DeviceDoc/MCP2515-Stand-Alone-CAN-Controller-with-SPI-20001801J.pdf , (Sett: 09.02.2023).

[89] "Mcp2515," https://www.digikey.no/en/products/detail/microchip-technology/MCP2515-I-SO/593001 , (Sett: 09.02.2023).

[90] "Mcp2517fd," https://www.digikey.com/en/products/detail/microchip-technology/MCP2517FD-H-SL/7691348 , (Sett: 09.02.2023).

[91] "Mcp2510," https://www.digikey.no/no/products/detail/microchip-technology/MCP2510-I-ST/319403 , (Sett: 09.02.2023).

[92] "Hi-3111psif," https://www.digikey.no/en/products/detail/holt-integrated-circuits-inc/HI-3111PSIF/3585724 , (Sett: 09.02.2023).

[93] "Mcp2551," https://www.digikey.no/no/products/detail/microchip-technology/MCP2551-E-SN/509446 , (Sett: 15.02.2023).

[94] "Tja1044gt/3z," https://www.digikey.no/no/products/detail/nxp-usa-inc/TJA1044GT-3Z/7648357 , (Sett: 15.02.2023).

[95] "Sn65hvd230dr," https://www.digikey.no/no/products/detail/texas-instruments/SN65HVD230DR/404367 , (Sett: 15.02.2023).

[96] "Tja1050 high speed can transceiver," https://www.nxp.com/docs/en/data-sheet/TJA1050.pdf , (Sett: 15.02.2023).

[97] "Can spi click 3.3v," https://www.mikroe.com/can-spi-33v-click , (Sett: 15.02.2023).

[98] "Mcp2515 can bus modul," https://www.digitalimpuls.no/diverse/149913/mcp2515-can-bus-modul-can-v20b-5v-spi , (Sett: 15.02.2023).

[99] "Can module," https://joy-it.net/en/products/SBC-CAN01 , (Sett: 16.02.2023).

[100] "Adding can to the raspberry pi," https://www.beyondlogic.org/adding-can-controller-area-network-to-the-raspberry-pi/ , (Sett: 16.02.2023).

[101] "Can modul," https://joy-it.net/files/files/Produkte/SBC-CAN01/SBC-CAN01-Datasheet.pdf , (Sett: 16.02.2023).

[102] Mg90s metall gear micro servo. Modell Hobby. [Online]. Available: https://modellflybutikken.no/produkt/servoer/mg90s-metall-gear-micro-servo/

[103] Mg996r Metal Gear Servo Motor (360 Degree Rotation). indiamart.com. [Online]. Available: https://www.indiamart.com/proddetail/mg996r-metal-gear-servo-motor-360-degree-rotation-25335766097.html

[104] Hitec HS-65MG Mighty MG Feather servo - RC Modellflysport. [Online]. Available: https://www.rc-modellflysport.no/products/hitec-hs-65mg-mighty-mg-feather-servo

[105] TowerPro SG 90 Micro Servo Motor : Amazon.in: Industrial & Scientific. [Online]. Available: https://www.amazon.in/Robodo-Electronics-Tower-Micro-Servo/dp/B00MTFFAE0

[106] "Karbonite gears." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Karbonite_gears&oldid=1090063681

[107] I. K. G. og Marius Brataas, "Kommunikasjon, videostrøm og regulering av fjernstyrt undervannsfarkost," Universitet i Stavanger, Tech. Rep., 2018, (Sett: 06.03.2023).

[108] High Speed Connection of Cameras Using USB. Oxford Instruments. [Online]. Available: https://andor.oxinst.com/learning/view/article/universal-serial-bus

[109] R. Sætre, "Teori – oppsummering (HW & DR)." [Online]. Available: https://www.ntnu.no/wiki/download/attachments/78972107/Uke47-Teori%20oppsummering_1av2_detaljertHWogDR.pdf?version=1&modificationDate=1479902081000&api=v2

[110] "Ip camera," https://en.wikipedia.org/wiki/IP_camera , (Sett: 01.03.2023).

[111] "H.264," https://no.wikipedia.org/wiki/H.264 , (Sett: 01.03.2023).

[112] "What internet speed do i need for remote viewing security cameras?" https://www.getscw.com/support/faq/internet-speed-camera-viewing , (Sett: 01.03.2023).

[113] "Stereo camera," https://en.wikipedia.org/wiki/Stereo_camera , (Sett: 01.03.2023).

[114] "Computer stereo vision," https://en.wikipedia.org/wiki/Computer_stereo_vision , (Sett: 02.03.2023).

[115] "What is fov — the importance of field of view in photography," https://www.studiobinder.com/blog/what-is-fov-definition/ , (Sett: 06.03.2023).

[116] "In defense of the zoom lens," https://petapixel.com/2017/05/15/defence-zoom-lens/ , (Sett: 06.03.2023).

[117] "Laowa 4mm f/2.8 mft fisheye," https://laowa.com.au/product/4mm/ , (Sett: 06.03.2023).

[118] "Fov calculation: Understanding field of view in photography," https://shotkit.com/field-of-view/ , (Sett: 03.03.2023).

[119] "Field of view and angular field of view," https://www.princetoninstruments.com/learn/camera-fundamentals/field-of-view-and-angular-field-of-view , (Sett: 06.03.2023).

[120] "Data compressionm wiki," https://en.wikipedia.org/wiki/Data_compression , (Sett: 04.04.2023).

[121] "Introduction to photography," https://alison.com/course/1326/resource/file/resource_200-1513762488755404360.pdf , (Sett: 05.04.2023).

[122] "H.264 profiles," https://www.rgb.com/h264-profiles , (Sett: 05.04.2023).

[123] "Advanced video coding, wiki," https://en.wikipedia.org/wiki/Advanced_Video_Coding , (Sett: 10.04.2023).

[124] "An overview of h.264 advanced video coding," https://www.vcodex.com/an-overview-of-h264-advanced-video-coding/ , (Sett: 10.04.2023).

[125] "Inter frame, wiki," https://en.wikipedia.org/wiki/Inter_frame , (Sett: 11.04.2023).

[126] "Motion compensation, wiki," https://en.wikipedia.org/wiki/Motion_compensation , (Sett: 11.04.2023).

[127] "Mobileasl," https://mobileasl.cs.washington.edu/ , (Sett: 11.04.2023).

[128] "Intra-frame coding," https://en.wikipedia.org/wiki/Intra-frame_coding , (Sett: 11.04.2023).

[129] "Video basics," https://support.biamp.com/General/Video/Video_Basics , (Sett: 11.04.2023).

[130] "Baseline h.264 encoder ip," https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=vecto
, (Sett: 11.04.2023).

[131] "Vcodex: Introduction to video coding," https://www.youtube.com/watch?v=gxefuXizO04 , (Sett: 12.04.2023).

[132] "Human vision and color," https://biomachina.org/courses/imageproc/121.pdf , (Sett: 12.04.2023).

[133] "Imx219-83 stereo camera," https://www.waveshare.com/wiki/IMX219-83_Stereo_Camera#Specification , (Sett: 03.03.2023).

[134] "Elp, synchronized dual lens stereo usb camera 1.3mp hd 960p," http://www.webcamerausb.com , (Sett: 18.04.2023).

[135] "Mipi csi-2® mipi camera serial interface 2," https://www.mipi.org/specifications/csi-2 , (Sett: 20.04.2023).

[136] "Mipi cameras vs usb cameras: a detailed comparison," https://www.edge-ai-vision.com/2022/01/mipi-cameras-vs-usb-cameras-a-detailed-comparison/ , (Sett: 20.04.2023).

[137] "Imx219, datablad," https://www.opensourceinstruments.com/Electronics/Data/IMX219PQ.pdf , (Sett: 22.04.2023).

[138] "Imx219 camera module," https://www.waveshare.com/imx219-d160.htm , (Sett: 24.04.2023).

[139] "Ov2710 2mp usb camera (a)," https://www.waveshare.com/wiki/OV2710_2MP_USB_Camera_(A) , (Sett: 10.03.2023).

[140] "What does f 2.8 mean in photography," https://cameraprism.com/what-does-f-2-8-mean-in-photography/ , (Sett: 10.03.2023).

[141] "Key differences between ccd and cmos imaging sensors," https://www.flir.eu/support-center/iis/machine-vision/knowledge-base/key-differences-between-ccd-and-cmos-imaging-sensors/ , (Sett: 10.03.2023).

[142] IPC, "IPC2221." [Online]. Available: http://www-eng.lbl.gov/~shuman/NEXT/CURRENT_DESIGN/TP/MATERIALS/IPC-2221A(L).pdf

[143] The Why and How of Differential Signaling - Technical Articles. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/the-why-and-how-of-differential-signaling/

[144] "Differential signalling." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Differential_signalling&oldid=1140569670

[145] "Impedance matching." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Impedance_matching&oldid=1144269112

[146] "Characteristic impedance." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Characteristic_impedance&oldid=1149320425

[147] "Reflections of signals on conducting lines." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Reflections_of_signals_on_conducting_lines&oldid=1125740033

[148] S. Campbell. How to Design a PCB Layout. Circuit Basics. [Online]. Available: https://www.circuitbasics.com/make-custom-pcb/

[149] "Ground loop (electricity)." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Ground_loop_(electricity)&oldid=1148548709

[150] T. Instruments, "PCB Design Guidelines For Reduced EMI." [Online]. Available: https://www.ti.com/lit/an/szza009/szza009.pdf?ts=1676356107576&ref_url=https%253A%252F%252Fwww.google.com%252F

[151] Preventing Ground Loops in Your PCB Design. Altium. [Online]. Available: https://resources.altium.com/p/preventing-ground-loops-your-pcb-design

[152] Through Hole Technology | American Products Inc. [Online]. Available: https://www.americanproductsinc.com/contract-manufacturing/through-hole-technology/

[153] Pick, Place Podcast | Electronics Manufacturing Podcast (When to use through hole vs surface mount components). Pick, Place, Podcast. [Online]. Available: https://www.pickplacepodcast.com

[154] "Reflow soldering." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Reflow_soldering&oldid=1142074041

[155] ENIG Rohs VS OSP VS HASL Comparison Table: Which Works Best? | Titoma. [Online]. Available: https://titoma.com/blog/pcb-surface-finish-enig-hasl-osp-isn-iag

[156] H. Endres. Gold or Tin versus Gold and Tin? Experience Molex. [Online]. Available: https://experience.molex.com/gold-or-tin-versus-gold-and-tin/

[157] "Pulse-width modulation." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Pulse-width_modulation&oldid=1144637466

[158] "Peripheral Component Interconnect." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Peripheral_Component_Interconnect&oldid=1134940417

[159] Pcie-164-02-f-d-th. Digi-Key Electronics. [Online]. Available: https://www.digikey.no/no/products/detail/samtec-inc/PCIE-164-02-F-D-TH/6561619

[160] "Buck converter." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Buck_converter&oldid=1137755397

[161] R. Johnsen and H. Øverby, "fiberoptikk." [Online]. Available: http://snl.no/fiberoptikk

[162] Twisted Pair Impedance Calculator: What is Twisted Pair Impedance and How is it Calculated? | Do Supply Tech Support. [Online]. Available: https://www.dosupply.com/tech/2021/12/20/twisted-pair-impedance-calculator-what-is-twisted-pair-impedance-and-how-is-it-calculated/

[163] T. Instruments, "Txs0101." [Online]. Available: https://www.ti.com/lit/ds/symlink/txs0101.pdf?HQS=dis-dk-null-digikeymode-dsf-pf-null-wwe&ts=1679903124889&ref_url=https%253A%252F%252Fwww.ti.com%252Fgeneral%252Fdocs%252Fsuppproductinfo.tsp%253FdistId%253D10%2526gotoUrl%253Dhttps%253A%252F%252Fwww.ti.com%252Flit%252Fgpn%252Ftxs0101

[164] 0480372200. Digi-Key Electronics. [Online]. Available: https://www.digikey.no/en/products/detail/molex/0480372200/2421361

[165] Usb-a til usb-b 2.0 kabel svart (0.5 meter). [Online]. Available: https://www.maxgaming.no/no/usb-kable/usb-a-til-usb-b-20-kabel-05-meter

[166] Routing Requirements for a USB 2.0 Impedance Interface on a 2-Layer PCB. Altium. [Online]. Available: https://resources.altium.com/p/routing-requirements-usb-20-2-layer-pcb

[167] "Stm32 fdcan in loopback mode," https://controllerstech.com/stm32-fdcan-in-loopback-mode/ , (Sett: 12.02.2023).

[168] "Nucleo-g431rb," https://www.digikey.no/no/products/detail/stmicroelectronics/NUCLEO-G431RB/10231584?s=N4IgTCBcDaIHIFUDCAZAogeQLQHEAsAzAIwBKAQiALoC%2BQA , (Sett: 23.02.2023).

[169] "Integrated development environment for stm32," https://www.st.com/en/development-tools/stm32cubeide.html , (Sett: 23.02.2023).

[170] "Inertial measurement unit," https://en.wikipedia.org/wiki/Inertial_measurement_unit , (Sett: 27.02.2023).

[171] "Endianness, wiki," https://en.wikipedia.org/wiki/Endianness , (Sett: 09.03.2023).

[172] "Little-endian vs big-endian in embedded systems," https://open4tech.com/little-endian-vs-big-endian-in-embedded-systems/ , (Sett: 09.03.2023).

[173] "Tegra, wikipedia," https://en.wikipedia.org/wiki/Tegra , (Sett: 05.05.2023).

[174] "Linux for tegra, nvidia," https://developer.nvidia.com/embedded/jetpack-sdk-451-archive , (Sett: 05.05.2023).

[175] "Uis subsea 2023 kommunikasjon repo, github," https://github.com/UiS-Subsea/Kommunikasjon-2023 , (Sett: 05.05.2023).

[176] "Python, wikipedia," https://en.wikipedia.org/wiki/Python_(programming_language) , (Sett: 05.05.2023).

[177] "Threading, wikipedia," https://en.wikipedia.org/wiki/Thread_(computing) , (Sett: 10.03.2023).

[178] "Gstreamer, gstreamer," https://gstreamer.freedesktop.org/ , (Sett: 30.02.2023).

[179] "Gstreamer, wikipedia," https://en.wikipedia.org/wiki/GStreamer , (Sett: 30.02.2023).

[180] "Nvidia jetson orin nano developer kit," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/ , (Sett: 11.05.2023).

[181] Tips for Optimal High Speed SPI Layout Routing. [Online]. Available: https://resources.pcb.cadence.com/blog/2019-tips-for-optimal-high-speed-spi-layout-routing

[182] Is There an SPI Trace Impedance Requirement? Altium. [Online]. Available: https://resources.altium.com/p/there-spi-trace-impedance-requirement

[183] "SP-09732-001_v1.1." [Online]. Available: https://pbrobinson.fedorapeople.org/SP-09732-001_v1.1.pdf

Tabeller

Figurer

Listings

Vedlegg A

Tester

- A.1: Testing av kommunikasjon
- A.2: Testing av PWM
- A.3: Testing av video
- A.4: Testing av temperatursensor opp mot Jetson Nano

23A.1 - Intern

Prøverapport

Testing av kommunikasjon

UiS Subsea, testing av kommunikasjon og forsinkelse

Forfatter(e)
Nils Helge H. Sandsbråten, Joar Dale Landa, Thomas Davidsen Matre

30. mars 2023

Postadresse:
www.uis.no

EMNEORD:
CAN, Ethernet, TCP

Prøverapport

Testing av kommunikasjon

UiS Subsea, testing av kommunikasjon og forsinkelse

RAPPORTNUMMER	VERSJON	DATO
23A.1	1.1	30. mars 2023

FORFATTER(E)
Nils Helge H. Sandsbråten, Joar Dale Landa, Thomas Davidsen Matre

OPPDRAGSGIVER(E)	OPPDRAGSGIVERS REFERANSE
Morten Tengesdal	Morten Tengesdal

PROSJEKT	GRADERING	ANTALL SIDER OG VEDLEGG
Kommunikasjon ROV	Intern	14

PRØVEOBJEKT	PRØVEOBJEKT MOTTATT
Kommunikasjon	Mars 10, 2023

PRØVEPROGRAM	PRØVESTED	PRØVEDATO
N/A	UiS - KE E-462	Mars 30, 2023

SAMMENDRAG
Testrapporten innholder testing av kommunikasjonen internt ombord ROV og kommunikasjonen mellom ROV og kontrollstasjon

Prøveresultatene gjelder kun de objekter som er prøvd.

Historikk

VERSJON	DATO	VERSJONSBESKRIVELSE
1.0	21-03-2023	Opprettet testdokument.
2.0	30-3-2023	Gjennomføring av tester og skriving av testrapport.
3.0	27-4-2023	Revisjon av testrapport hvor forsinkelse på TCP ble tatt med.

Innhold

1 Hva skal testes?

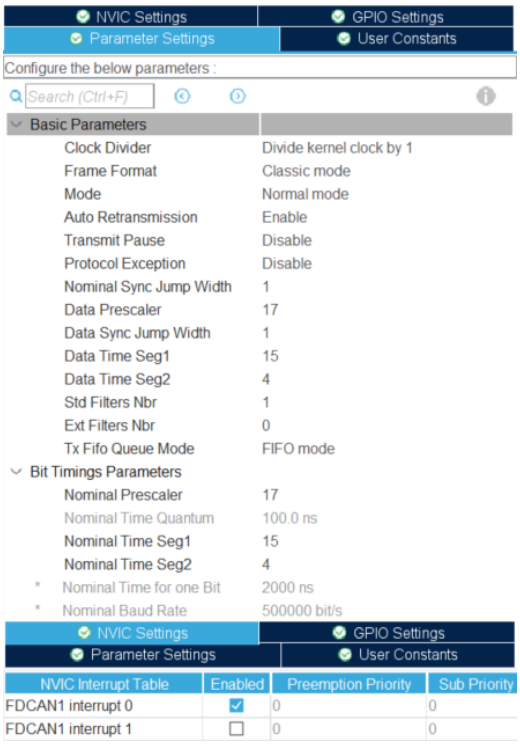
Det skal kjøres fire forskjellige tester, hvor hovedfokuset er å måle forsinkelsen på datapakker rundt forbi i systemet. Den første testen skal måle forsinkelsen på en datapakke med styresignal som sendes fra kontrollstasjon ned til en mikrokontroller og tilbake igjen. Andre testen skal måle forsinkelsen på meldinger internt ombord ROV mellom mikrokontroller og Nvidia Jetson Nano. Meldingen som sendes går gjennom flere ledd før den når mottakeren, og det samme på veg tilbake. Dette er vist i blokkskjemaet i figur 3. Av denne grunn, er det også interessent og vite hvor stor forsinkelsen er over Ethernetlinjen hvor TCP-protokollen brukes. Dette blir den tredje testen. Den fjerde testen er hvor mange meldinger mikrokontrollerene ombord ROV-en klarer sende til hverandre i et gitt tidsintervall.

I den første testen vil også integriteten til medlingen vil bli verifisert ved å sammenligne mottatt melding opp mot sendt melding. I tillegg til dette vil det kobles opp et scope til CAN-bussen for å sjekke hvor lang selve meldingen er, og for å verifisere intergriteten til meldingen ytterligere.

2 Utstyr og innstillinger

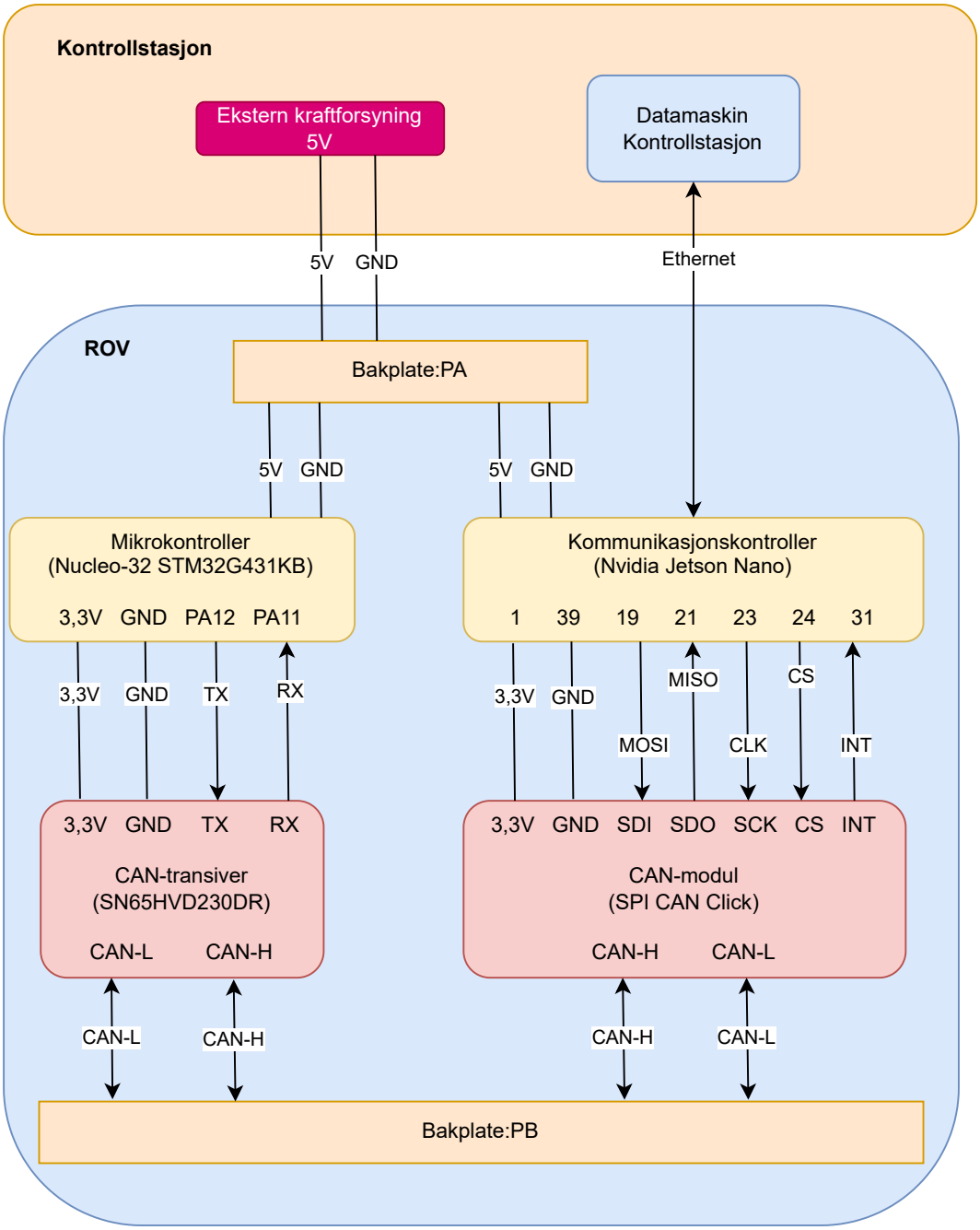
- Nvidia Jetson Nano 4GB på kommunikasjonskort som kjører skriptet “latencyServer.py”, “latencyCan.py” eller “latencyTCPserver.py”
- Datamaskin ved kontrollstasjon som kjører skriptet “latencyClient.py” eller “latencyTCPclient.py”
- Mikrokontroller på sensorkort (Nucleo-32 STM32G431KB, klokkefrekvens: 170MHz) hvor “testSTM32G431KB” kjøres i “main.c”
- Mikrokontroller på kraftkort (Nucleo-32 STM32G431KB, klokkefrekvens: 170MHz) hvor “testSTM32G431KB_mottaker” kjøres i “main.c”
- Mikrokontroller på kraftkort (Nucleo-32 STM32G431KB, klokkefrekvens: 170MHz) hvor “testSTM32G431KB_sender” kjøres i “main.c”
- CAN-modul av typen SPI CAN Click 3,3V koblet til Nvidia Jetson Nano 4GB
- CAN-transiver koblet til mikrokontroller på sensorkort (SN65HVD230DR)
- Bakplatene PA og PB
- Ekstern kraftforsyning (5V)
- Diversekabler til kraft
- Ethernetkabel med en lengde på 2 meter (Cat6)
- Keysight InfiniiVision MSOX3012A Scope

Innstillinger FDCAN1 på mikrokontroller:

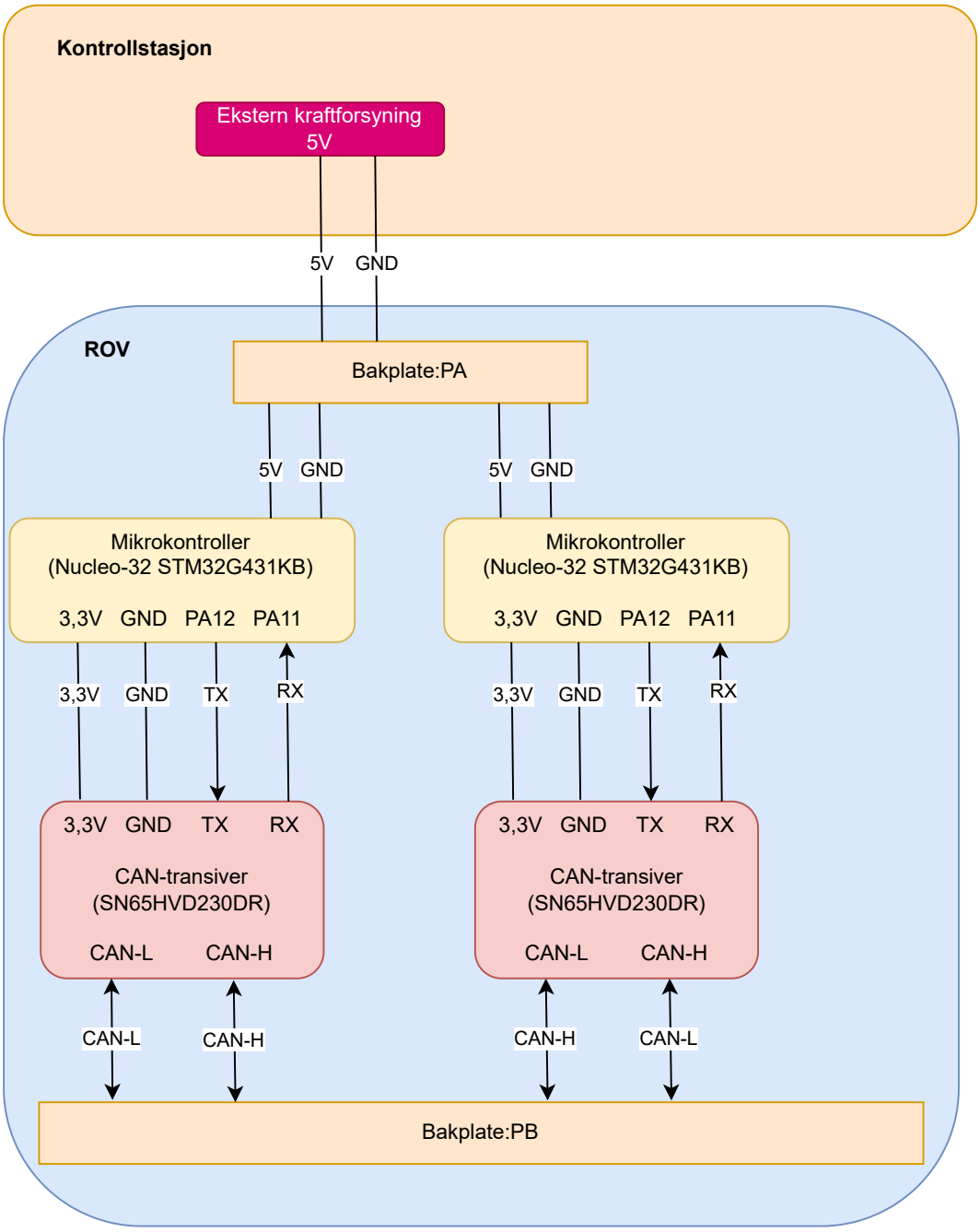


Figur 1: Parameter for FDCAN1 på mikrokontrollere

3 Oppkobling / Koblingsskjema



Figur 2: Oversikt over oppkobling for tester som gjennomføres mellom mikrokontroller og kontrollstasjon/Jetson Nano



Figur 3: Oversikt over oppkobling for tester som gjennomføres mellom to mikrokontrollere

4 Programverktøy

- STM32CubeIDE
- Python 3.10
- Python 3.6

5 Programfiler

Programfilene kan finnes på Github: https://github.com/UiS-Subsea/Kommunikasjon-2023/tree/main/tests

- latencyClient.py
- latencyServer.py
- testSTM32G431KB legges inn i main.c
- testSTM32G431KB_mottaker legges inn i main.c
- testSTM32G431KB_sender legges inn i main.c
- latencyCan.py
- latencyTCPclient.py
- latencyTCPserver.py

6 Fremgangsmåte

Skriptet latencyClient.py sender en intialiseringsmelding før den går inn i en løkke som sender et bestemt antall meldinger, som blir satt av variabelen "NoOfPacks". Før hver melding blir sendt blir tiden lagret i variabelen "start". Vi venter så på å motta en melding igjen. Vi sjekker så om pakke-ID er den vi forventer. Hvis pakke-ID er korrekt tar vi å regner ut tiden for å sende en melding til mikrokontroller og motta svar, ved å gjøre "tid = nåtid - start".

Skriptet latencyServer.py mottar meldinger fra klienten og videresender meldingen på CAN-buss. Den mottar også meldinger fra mikrokontrolleren fra CAN-buss og sender disse videre til klienten.

Skriptet latencyCan.py virker likt som klienten men sender data direkte ut på CAN-buss, og er kun internt ombord ROV-en. Dette for å måle forsinkelsen internt mellom mikrokontroller og Nvidia Jetson Nanno.

Skriptet latencyTCPClient.py går inn i en løkke som sender et bestemt antall meldinger, som blir satt av variabelen "NoOfPacks". Før hver melding som blir sendt, blir tiden lagret i variabelen "start". Vi venter så på å motta den samme meldingen igjen. Vi sjekker så om meldingens innhold er det vi samme som vi sendte ned. Hvis pakke-ID er korrekt, tar vi å regner ut tiden for å sende og motta svar. Dette gjøres ved å ta "tid = nåtid - start".

Skriptet latencyServer.py mottar meldinger fra klienten og sender medlingen tilbake.

Framgangsmåte for å finne forsinkelsen tur-retur fra kontrollstasjon til mikrokontroller.

- Programmer STM-mikrokontroller ved å stille inn FDCAN1 med parametrene i figur 1, og kopier test-STM32G431KB inn i main.c
- Koble opp etter koblingskjema vist i figur 3
- Start Python-skriptet for å simulere kommunikasjonskontrolleren: latenecyServer.py
- Start Python-skriptet for å simulere kontrollstasjon: latencyClient.py
- Les av snittid, makstid og minstetid.

Framgangsmåte for å finne forsinkelsen tur-retur fra kommunikasjonskontroller til mikrokontroller.

- Programmer STM-mikrokontroller ved å stille inn FDCAN1 med parametrene i figur 1, og kopier test-STM32G431KB inn i main.c
- Koble opp etter koblingskjema vist i figur 3
- Start Python-skriptet for å simulere kommunikasjonskontrolleren: latenecyCan.py
- Les av snittid, makstid og minstetid.

Framgangsmåte for å finne forsinkelsen tur-retur mellom kommunikasjonskontroller og kontrollstasjon, over TCP.

- Koble opp etter koblingskjema vist i figur 3
- Start Python-skriptet for å simulere kommunikasjonskontrolleren: latenecyTCPserver.py
- Start Python-skriptet for å simulere kontrollstasjon: latencyTCPclient.py
- Les av snittid, makstid og minstetid.

Fremgangsmåte for å finne hvor mange meldinger som kan sendes mellom to mikrokontrollere i et gitt tidsintervall.

- Programmer en av STM-mikrokontrollerene ved å stille inn FDCAN1 med parametrene i figur 1, og kopier testSTM32G431KB_mottaker inn i main.c
- Programmer den andre STM-mikrokontrollerene ved å stille inn FDCAN1 med parametrene i figur 1, og kopier testSTM32G431KB_sender inn i main.c
- Debug STM-mikrokontrolleren som kjører testSTM32G431KB_mottaker, og sett et breakpoint på linje 391
- Tidsintervallet kan endres på linje 390, og er i utgangspunktet satt til 10000ms eller 10 sekund
- Kjør skriptet og vent til det stopper ved breakpoint
- Les av variabelen "antallfor å se hvor mange meldinger som ble sendt i intervallet

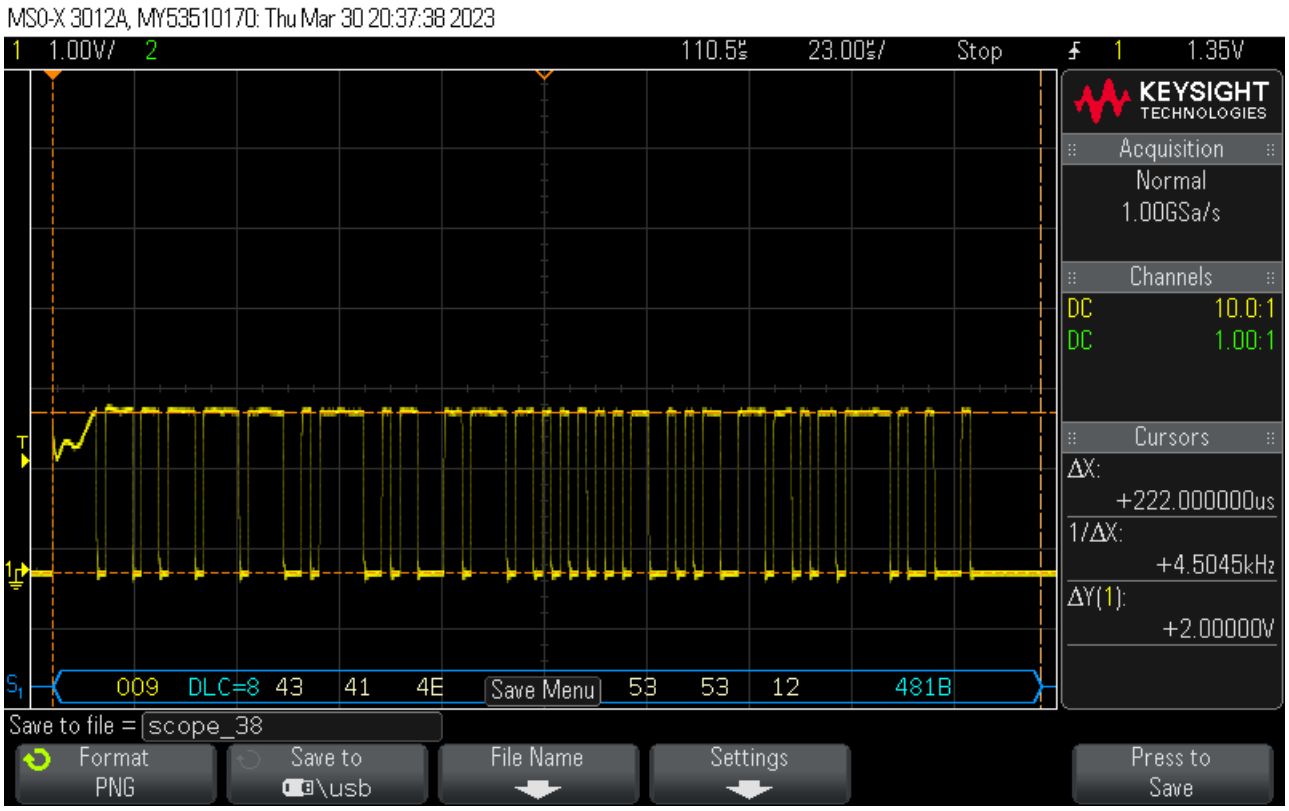
7 Resultat

Resultat fra sending av meldinger mellom kontrollstasjon og sensorkort:

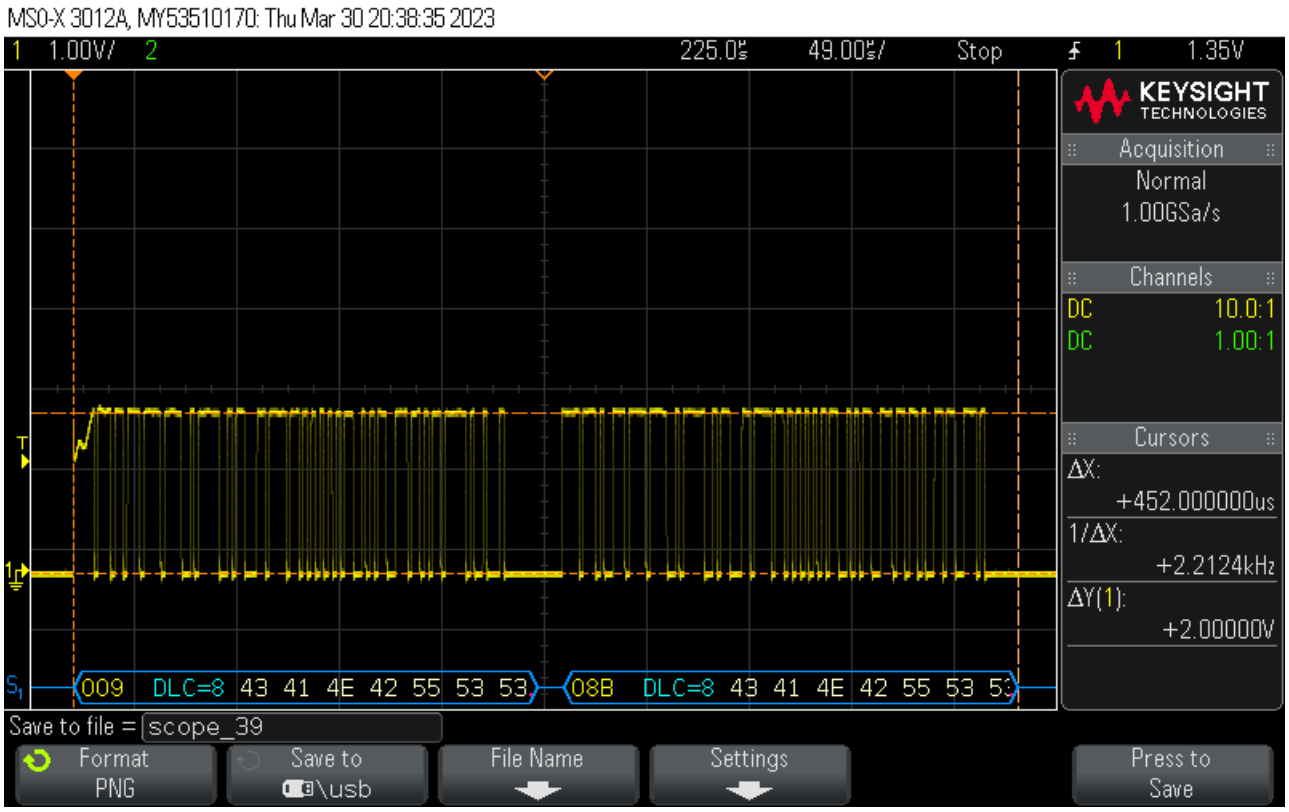
Test nr.	Antall meldinger	Snittid Tur-Retur [ms]	Makstid Tur-Retur [ms]	Minstetid Tur-Retur [ms]
1	500	2,26	8,27	1,38
2	500	2,22	8,63	1,20
3	2000	2,10	8,01	1,27
4	2000	2,13	8,57	1,26
5	5000	1,94	10,76	1,28
6	5000	2,11	9,47	1,23

Tabell 1: Tabell med ulike tider for en melding tur-retur internt ombord ROV-en mellom mikrokontroller og kontrollstasjon

Meldingen som sendes på CAN-bussen er på 8 byte, og er som følger:
$0x43, 0x41, 0x4E, 0x42, 0x55, 0x53, 0x53, 0x12$.



Figur 4: Tiden for en melding på CAN-bussen. Målt differensialspenning mellom CAN-H og CAN-L



Figur 5: Tiden for to meldinger tur-retur på CAN-bussen. Målt differensialspenning mellom CAN-H og CAN-L

Resultat fra sending av meldinger mellom kommunikasjonskort og sensorkort:

Test nr.	Antall meldinger	Snittid Tur-Retur [ms]	Makstid Tur-Retur [ms]	Minstetid Tur-Retur [ms]
1	500	1,25	12,79	0,92
2	500	1,31	7,76	0,95
3	2000	1,16	14,86	0,82
4	2000	1,19	13,14	0,84
5	5000	1,08	10,78	0,82
6	5000	1,09	11,36	0,82

Tabell 2: Tabell med ulike tider for en melding tur-retur internt ombord ROV-en mellom mikrokontroller og kommunikasjonskontroller

Resultat fra sending av meldinger mellom kommunikasjonskort og kontrollstastjon over TCP:

Test nr.	Antall meldinger	Snittid Tur-Retur [ms]	Makstid Tur-Retur [ms]	Minstetid Tur-Retur [ms]
1	500	0,67	2,35	0,32
2	500	0,45	3,94	0,25
3	2000	0,54	3,37	0,20
4	2000	0,56	3,55	0,18
5	5000	0,42	3,69	0,17
6	5000	0,57	3,75	0,18

Tabell 3: Tabell med ulike tider for en melding tur-retur over TCP

Resultat fra sending av meldinger mellom to mikrokontrollere:

Test nr.	Tid [s]	Antall meldinger	Snittid en melding [μs]
1	1	4313	231,87
2	1	4314	231,80
3	3	12935	231,93
4	3	12938	231,88
5	10	43133	231,84
6	10	43125	231,88

Tabell 4: Tabell med ulike tider for en melding tur-retur internt ombord ROV-en mellom mikrokontroller og kommunikasjonskontroller

8 Analyse og konklusjon

Ved å analysere utklipp 4 og 5 fra scope, kan integriteten av meldingen på CAN-bussen verifiserest. Scopet viser også differansespenningen på $\Delta Y = +2,00V$ mellom CAN-H og CAN-L som er forventet differensialspenning. Det første scopebildet, figur 4, viser også tiden for en melding. Denne tiden er på $\Delta X = +222,00\mu s$, som igjen er som forventet, da en melding består av 111 bit, og hvert bit har en bittid på 2000ns, som gir $2000ns \cdot 111 = 222,00\mu s$.

I tabell 1, er resultatet for sending av meldinger mellom mikrokontroller ombord i ROV-en opp til kontrollstasjon. Ved å ta et nytt gjennomsnitt av alle snittiden tur-retur, får vi en snittid på 2,13ms. Med andre ord, er tiden for en pakke fra kontrollstasjon til mikrokontroller og en tilbake, altså to pakker, omtrent 2,13ms. Hvor mange pakker som kan sendes på CAN-bussen og opp til kontrollstasjon kan da regnes grovt som:

$$Antall\ pakker = \frac{1000ms}{\frac{2,13ms}{2\ pakker}} \approx 939 pakker/s \tag{1}$$

I tabell 2, er resultatet for sending av pakker mellom mikrokontroller og kommunikasjonskontrolleren, Nvidia Jetson nano ombord i ROV-en. Som for det første resultatet, beregnes det også her et nytt gjennomsnitt av alle pakkene tur-retur. Snittiden på en pakke tur-retur mellom mikrokontroller og kommunikasjonskontroller blir da på 1,8ms. Hvor mange pakker som kan sendes på CAN-bussen mellom mikrokontroller og kommunikasjonskontroller kan da regnes grovt som:

$$Antall\ pakker = \frac{1000ms}{\frac{1,18ms}{2\ pakker}} \approx 1695 pakker/s \tag{2}$$

Fra resultatet i de to første testene, kommer det tydelig frem at sending over TCP-linjen opp til kontrollstasjon bidrar til mer forsinkelse. Denne forsinkelsen er blitt målt, og resultatet er gjengitt i 3. Over de seks testene, blir snittiden på forsinkelsen TCP-linjen bidrar med på omtrent 0,535ms.

Resultatene for den siste testen hvor det skulle testes hvor mange pakker som kunne sendes mellom to mikrokontrollere i et gitt tidsintervall er vist i tabell 4. Her kan det ses at snittiden på en pakke er omtrent konstant i de ulike testene. Med andre ord er det et lineært forhold i mellom tiden og antall pakker. Snittiden på en pakke mellom mikrokontrollerene er på omtrent $231,87\mu s$. Antall pakker per sekund basert på denne snittiden kan da beregnes som:

$$Antall\ pakker = \frac{1000ms}{231,87\mu s} \approx 4313 pakker/s \tag{3}$$

I de to første testene hvor Nvidia Jetson Nano var involvert i kommunikasjonen, opplevde vi på de første meldingene som ble sendt etter initialisering av testen, at forsinkelsen var høy. Eksempler på størrelsen av forsinkelsen kan ses i makstidene tur-retur i begge tabellene. Som sagt var dette de første meldingene, men vi ser fra minstetiden og snitttiden at snittiden gradvis går mot minstetiden ved økt antall meldinger. Hva dette kommer av er uklart. Men noen av mistankene på hva som kan forårsake dette, er programvaren som er blitt laget for Nvidia Jetson Nano og det faktum at det kjøres i Python som ikke er annerkjent som et effektivt programmeringsspråk. En annen årsak kan være CAN-modulen SPI CAN Click, og hvordan den er blitt konfigureret.

23A.1 - Intern

Prøverapport

Testing av Servo PWM

UiS Subsea, test av servo PWM signal og støymarginer

Forfatter(e)
Nils Helge H. Sandsbråten, Joar Dale Landa, Thomas Davidsen Matre

23. april 2023

Postadresse:
www.uis.no

EMNEORD:
PWM, Signal, Støymargin

Prøverapport

Testing av Servo PWM

UiS Subsea, test av servo PWM signal og støymarginer

RAPPORTNUMMER	VERSJON	DATO
23A.1	1.1	23. april 2023

FORFATTER(E)
Nils Helge H. Sandsbråten, Joar Dale Landa, Thomas Davidsen Matre

OPPDRAGSGIVER(E)	OPPDRAGSGIVERS REFERANSE
Morten Tengesdal	Morten Tengesdal

PROSJEKT	GRADERING	ANTALL SIDER OG VEDLEGG
Kommunikasjon ROV	Intern	7

PRØVEOBJEKT	PRØVEOBJEKT MOTTATT
Servo PWM	April 23, 2023

PRØVEPROGRAM	PRØVESTED	PRØVEDATO
N/A	UiS - KE E-462	April 23, 2023

SAMMENDRAG
Testrapporten tar for seg PWM-signalet til en SG90 servo, og tester hvilke spenningsnivåer denne har for lav og høy.

Prøveresultatene gjelder kun de objekter som er prøvd.

Historikk

VERSJON	DATO	VERSJONSBESKRIVELSE
1.0	23-4-2023	Opprettet testdokument.
2.0	23-4-2023	Utførte test og skrev testrapport.

Innhold

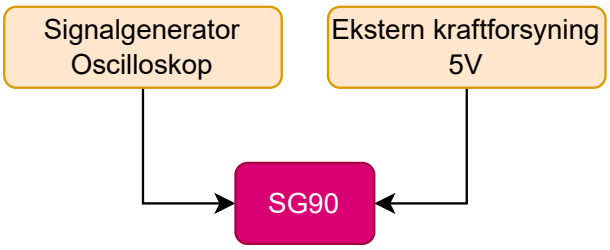
1 Hva skal testes?

Styring av en SG90 servo ved PWM signal med et gitt område for duty cycle. Dette skal sjekkes opp mot hvilke posisjoner som servo-en er i ved oppgitt signal. Det skal også testes hvilke verdier for V_{IH} og V_{IL} som servo-en har.

2 Utstyr og innstillinger

- Servo SG90
- Oscilloskop Keysight InfiniiVision MSOX3012A Scope
- Ekstern kraftforsyning (5V)
- Diverse ledninger til oppkobling
- 3D-printet stativ for servo til kamera
- Kamera IMX-219-83

3 Oppkobling



Figur 1: Oversikt over oppkobling for tester som gjennomføres

4 Fremgangsmåte

Servo-en kobles opp med tre ledninger. En til 5V, en til jord og en til PWM-signal. Jord og 5V kobles til strømforsyningen, signalet kobles til Oscilloskopet.

For å teste vinkler, blir det satt opp med en periodetid på 20ms. Området som skal testes er mellom 0°og 90°, dette tilsvarer 1ms til 2ms. Det skal testes tre punkt: 0°, 45°, og 90°.

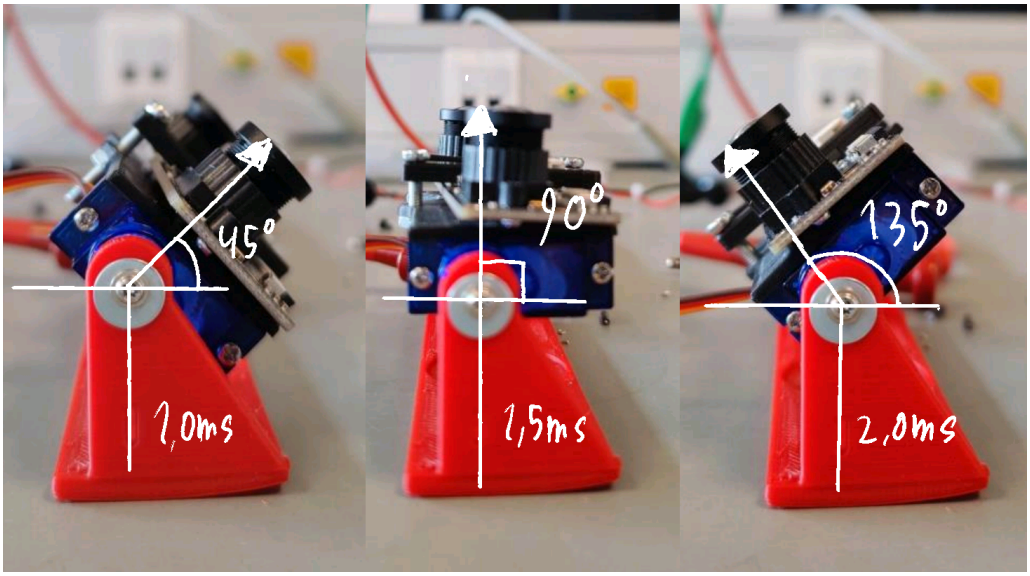
Etterpå skal det testes hvilke støymarginer dette signalet har. Det vil si å skru opp bunnpunktet gradvis for å finne hvor signalet ikke er stabilt lenger. Så vil toppunktet bli skrudd ned og se ved hvilken spenningsverdi dette ikke lenger er stabilt.

Framgangsmåte for å teste PWM.

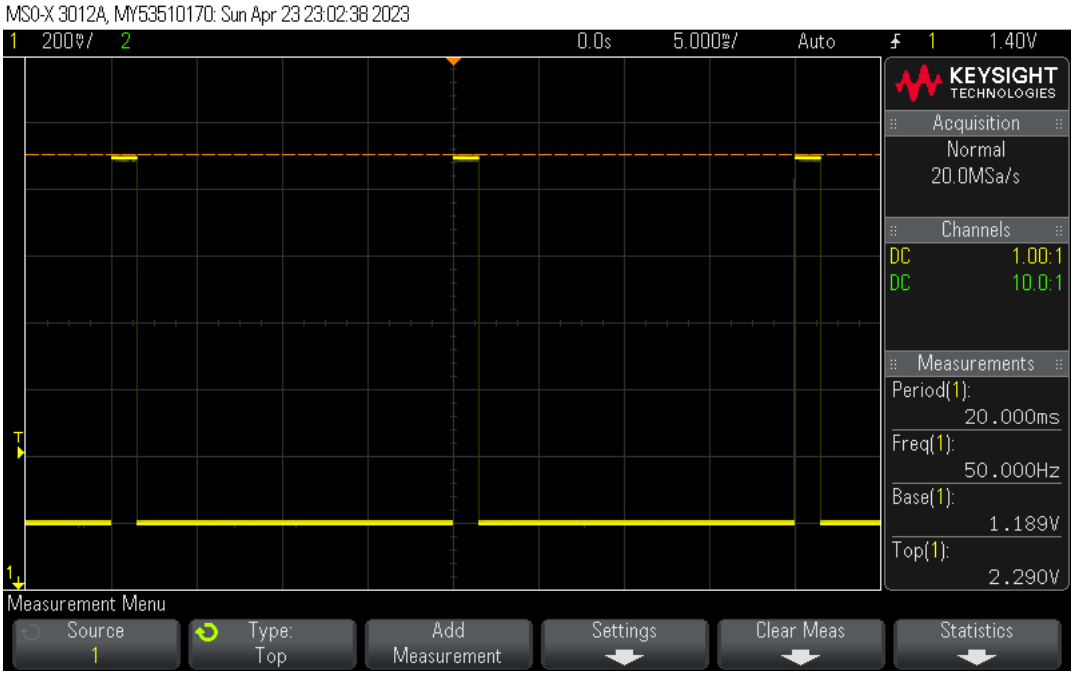
- Koble opp etter koblingskjema 1
- Still inn ocsilloskopet

- Sjekk posisjon ved 1ms, 1,5ms og 2ms
- Sjekk V_{IL} ved å øke bunngrense
- Sjekk V_{IH} ved å redusere toppgrense

5 Resultat



Figur 2: Posisjoner observert ved 1,0ms, 1,5ms og 2,0ms



Figur 3: Bilde tatt av skopmåling ved klart signal med V_{IH} og V_{IL}.

6 Analyse og konklusjon

Fra testen ser vi at posisjoner ved tidspunktene resulterer i:

- 1,0ms : 45°
- 1,5ms : 90°
- 2,0ms : 135°

Dette resultatet er som forventet.

Med spenningsgrenser fant vi

- $V_{IL} \approx 1{,}2\text{V}$
- $V_{IH} \approx 2{,}3\text{V}$

23A.3 - Intern

Prøverapport

Testing av video

UiS Subsea, testing av videostrøm og forsinkelse

Forfatter(e)
Nils Helge H. Sandsbråten, Joar Dale Landa, Thomas Davidsen Matre

20. april 2023

Postadresse:
www.uis.no

EMNEORD:
UDP, TCP, RTP, Ethernet

Prøverapport

Testing av video

UiS Subsea, testing av videostrøm og forsinkelse

RAPPORTNUMMER	VERSJON	DATO
23A.3	1.1	20. april 2023

FORFATTER(E)
Nils Helge H. Sandsbråten, Joar Dale Landa, Thomas Davidsen Matre

OPPDRAGSGIVER(E)	OPPDRAGSGIVERS REFERANSE
Morten Tengesdal	Morten Tengesdal

PROSJEKT	GRADERING	ANTALL SIDER OG VEDLEGG
Kommunikasjon ROV	Intern	10

PRØVEOBJEKT	PRØVEOBJEKT MOTTATT
Videostrøm	Mars 10, 2023

PRØVEPROGRAM	PRØVESTED	PRØVEDATO
N/A	UiS - KE E-462	April 20, 2023

SAMMENDRAG
Testrapporten innholder testing av videostrømmen fra kamera mellom ROV og kontrollstasjon

Prøveresultatene gjelder kun de objekter som er prøvd.

Historikk

VERSJON	DATO	VERSJONSBESKRIVELSE
1.0	21-03-2023	Opprettet testdokument.
2.0	20-4-2023	Gjennomføring av tester og skriving av testrapport.

Innhold

1 Hva skal testes?

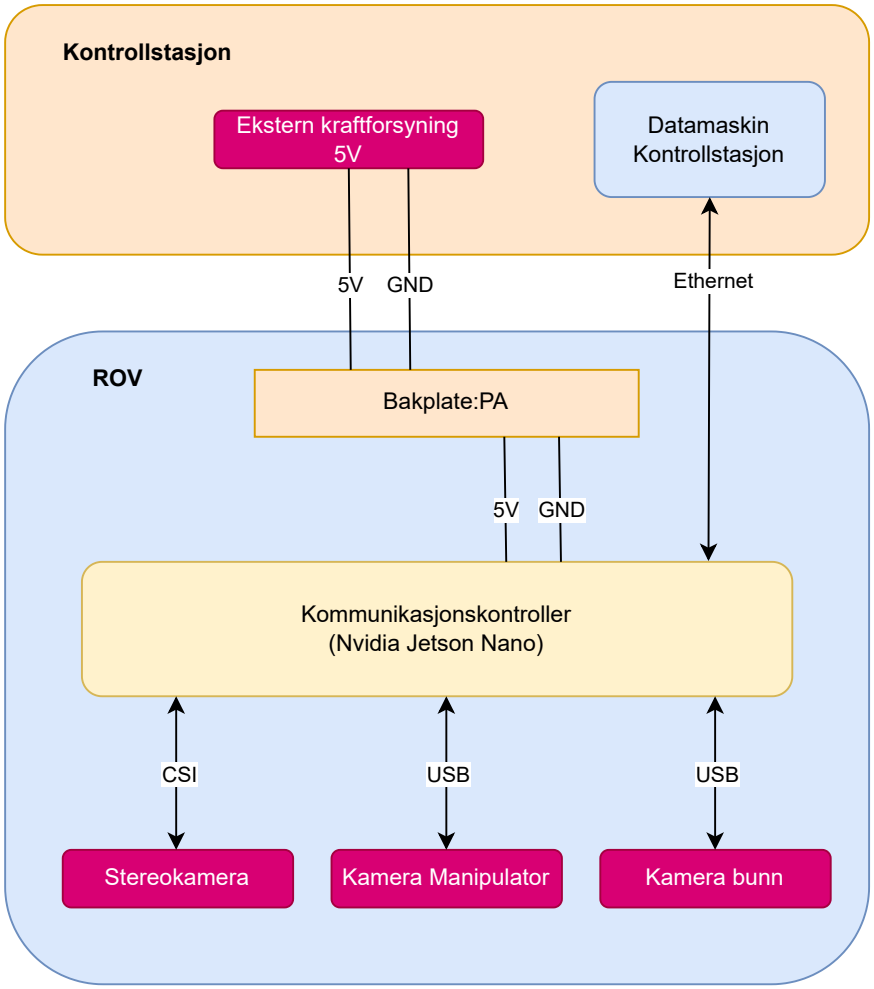
Det skal først testes forskjellige metoder for komprimering samt forskjellige måter å sende videostrømmen på. Fra disse metodene skal det sammenlignes forskjellen i CPU-last og forsinkelse. Fra valgt metode for sending og komprimering skal det så verifiseres at videostrømmene blir mottatt, og måle forsinkelse på denne når alle kamereane kjører.

2 Utstyr og innstillinger

- Nvidia Jetson Nano 4GB på kommunikasjonskort som kjører skriptet “main.py” eller “TCPclient.py”
- Datamaskin ved kontrollstasjon som kjører skriptet “network_handler” og “camHandler” eller “TCPServer.py”
- Netsiden https://www.online-stopwatch.com/
- Ekstern kraftforsyning (5V)
- Stereokamera IMX219-83
- 2 stk USB-kamera OV2710
- Ethernetkabel med en lengde på 2 meter (Cat6)

Innstillinger på jetson: Set ip på jetson til 10.0.0.2 og legg til 224.0.0.0 rute for å ta i bruk “multicast”. All video sending blir gjort på 720P oppløsning.

3 Oppkobling / Koblingsskjema



Figur 1: Oversikt over oppkobling for tester som gjennomføres

4 Programverktøy

- STM32CubeIDE
- Python 3.10
- Python 3.6

5 Programfiler

Programfilene kan finnes på Github: https://github.com/UiS-Subsea/Kommunikasjon-2023/tree/main/tests

- gStreamer
- tegrastats

- vTCPsender.py
- cTCPreciver.py
- vUDPsender.py
- cUDPreciver.py
- main.py
- camHandler.py

6 Fremgangsmåte

For å teste sending av bildestrøm over UDP blir OpenCV biblioteket brukt for avlesning av kamera samt for komprimering til JPEG. Python skriptet “vUDPsender.py” leser av videostrømmen, komprimerer så sender ut strømmen på port 5000. Skriptet “vUDPreciver.py” kobler seg til porten henter ut daten og dekoder den for avspilling. Vi leser av CPU-last med tegrastats kommandoen.

- Start tegrastats i terminal for sjekking av CPU-last.
- Start sending av videostrøm fra jetson ved å starte python skriptet “vUDPsender.py”.
- Start mottak av videostrøm ved å starte python skriptet “vUDPreciver.py”
- Film PC-skjerm med nettsiden https://www.online-stopwatch.com/ på.
- Ta skjermbilde for å sjekke forsinkelse.
- Sjekk CPU-last i terminal som kjører tegrastats.

For å teste sending av bildestrøm over TCP blir OpenCV biblioteket brukt for avlesning av kamera samt for komprimering til JPEG. Python skriptet “vTCPsender.py” leser av videostrømmen, komprimerer så sender ut strømmen på port 5000. Skriptet “vTCPreciver.py” kobler seg til porten, henter ut daten, og dekoder den for avspilling. Vi leser av CPU-last med tegrastats kommandoen.

- Start tegrastats i terminal for sjekking av CPU-last.
- Start sending av videostrøm fra jetson ved å starte python skriptet “vUDPsender.py”.
- Start mottak av videostrøm ved å starte python skriptet “vUDPreciver.py”
- Film PC skjerm med nettsiden https://www.online-stopwatch.com/ på.
- Ta skjermbilde for å sjekke forsinkelse.
- Sjekk CPU-last i terminal som kjører tegrastats.

For å teste sending av bildestrøm over UDP med RTP-pakker blir programvaren gStreamer brukt. Dette gir oss også tilgang til å bruke maskinvareenkoding og dekoding av videoen.

- Start tegrastats i terminal for sjekking av CPU-last.

- Start sending av videostrøm fra jetson ved å kjøre kommandoen under i terminalen:

Kode 1: Åpning av rør for USB kamera manipulator

```
gst-launch-1.0 v4l2src device=/dev/video2 ! 'image/jpeg, format=MJPG, ...
    width=1280, heigth=720, framerate=30/1' ! nvjpegdec ! 'video/x-raw' ! ...
    nvvidconv ! 'video/x-raw(memory:NVMM), format=NV12' ! nvv4l2h264enc ...
    insert-sps-pps=true bitrate=8000000 ! rtph264pay ! udpsink ...
    host=224.1.1.1 port=5000 auto-multicast=true
```

- Start mottak av videostrøm ved å kjøre kommandoen under i terminalen:

Kode 2: Åpning av rør for mottak av RTP strøm

```
gst-launch-1.0 -v udpsrc multicast-group=224.1.1.1 auto-multicast=true ...
    port=5000 ! "application/x-rtp, media=(string)video, ...
    clock-rate=(int)90000, encoding-name=(string)H264, payload=(int)96" ! ...
    rtph264depay ! h264parse ! decodebin ! videoconvert ! appsink sync=false
```

- Film PC skjerm med nettsiden https://www.online-stopwatch.com/ på.
- Ta skjermbilde for å sjekke forsinkelse.
- Sjekk CPU-last i terminal som kjører tegrastats.

For å teste alle kamerane samtidig

- Start tegrastats i terminal for sjekking av CPU-last.
- Start sending av videostrøm fra jetson ved å starte python skriptet “main.py”.
- Start mottak av videostrøm ved å starte python skriptet “camHandler.py”
- Film PC-skjerm med nettsiden https://www.online-stopwatch.com/ på.
- Ta skjermbilde for å sjekke forsinkelse.
- Sjekk CPU-last i terminal som kjører tegrastats.

7 Resultat

Resultat fra sending av videostrøm UDP; Manipulatorkamera :

Test nr.	Forsinkelse [ms]	CPU last [%]
1	185	100
2	180	100
3	182	99
4	186	99
5	181	100
Gjennomsnitt	182.8	

Resultat fra sending av videostrøm TCP; Manipulatorkamera:

Test nr.	Forsinkelse [ms]	CPU last [%]
1	385	99
2	322	100
3	320	99
4	256	99
5	289	100
Gjennomsnitt	334.4	

Resultat fra sending av videostrøm RTP over UDP; Manipulatorkamera :

Test nr.	Forsinkelse [ms]	CPU last [%]
1	97	38
2	95	32
3	96	37
4	95	43
5	96	29
Gjennomsnitt	108.4	

Resultat fra sending av videostrøm RTP over UDP (alle kamera samtidig); Stereokamera 1 :

Test nr.	Forsinkelse [ms]
1	96
2	97
3	96
4	96
5	96
Gjennomsnitt	96.2

Resultat fra sending av videostrøm RTP over UDP (alle kamera samtidig); Stereokamera 2 :

Test nr.	Forsinkelse [ms]
1	99
2	95
3	64
4	97
5	96
Gjennomsnitt	90.2

Resultat fra sending av videostrøm RTP over UDP (alle kamera samtidig); Manipulatorkamera :

Test nr.	Forsinkelse [ms]
1	63
2	95
3	128
4	64
5	64
Gjennomsnitt	82.2

Resultat fra sending av videostrøm RTP over UDP (alle kamera samtidig); Bunnkamera :

Test nr.	Forsinkelse [ms]
1	127
2	95
3	96
4	95
5	129
Gjennomsnitt	108.4

8 Analyse og konklusjon

Fra resultatene ser vi at TCP strømmen er betydelig senere og egner seg ikke for vår applikasjon. UDP sending er bedre men vi har en flaskehals med at OpenCV som komprimerer bildene ved bruk av prosessoren, bruker all tilgjengelig prosesseringskraft. Når vi bruker gStreamer-programvaren oppnår vi mindre forsinkelse ved sending over UDP med RTP pakker selv med mer “overhead” i meldingene. Dette viser at flaskehalsene her er prosessorkraft og TCP.

23A.4 - Intern

Prøverapport

Testing av temperatursensor og PWM

UiS Subsea, testing av temperatursensor og PWM opp mot Jetson Nano

Forfatter(e)
Nils Helge H. Sandsbråten, Joar Dale Landa, Thomas Davidsen Matre

04. april 2023

Postadresse:
www.uis.no

EMNEORD:
Temperatursensor, I2C, PWM

Prøverapport

Testing av temperatursensor og PWM

UiS Subsea, testing av temperatursensor og PWM opp mot Jetson Nano

RAPPORTNUMMER	VERSJON	DATO
23A.4	1.1	04. april 2023

FORFATTER(E)
Nils Helge H. Sandsbråten, Joar Dale Landa, Thomas Davidsen Matre

OPPDRAGSGIVER(E)	OPPDRAGSGIVERS REFERANSE
Morten Tengesdal	Morten Tengesdal

PROSJEKT	GRADERING	ANTALL SIDER OG VEDLEGG
Kommunikasjon ROV	Intern	8

PRØVEOBJEKT	PRØVEOBJEKT MOTTATT
Temperatursensor og PWM	Mars 10, 2023

PRØVEPROGRAM	PRØVESTED	PRØVEDATO
N/A	UiS - KE E-462	April 04, 2023

SAMMENDRAG
Testrapporten inneholder testing av temperatursensor og PWM fra og til Jetson Nano

Prøveresultatene gjelder kun de objekter som er prøvd.

Historikk

VERSJON	DATO	VERSJONSBESKRIVELSE
1.0	04-4-2023	Opprettet testdokument.
2.0	04-4-2023	Utførte test og skrev testrapport.

Innhold

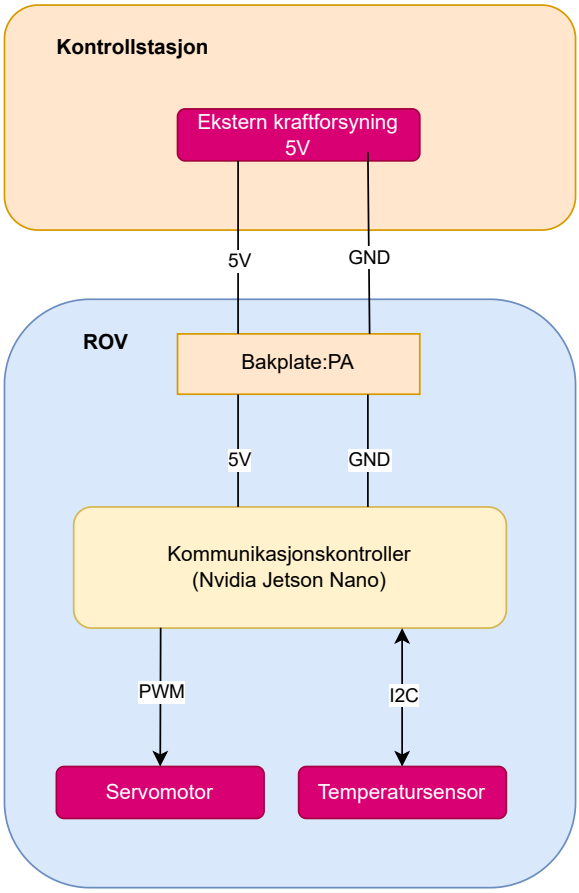
1 Hva skal testes?

PWM-signalet ut fra Nvidia Jetson Nano skal testes og verifiseres. Dette er PWM-signalet som skal gå til tilten for stereokameraet. Det skal også testes og verifiseres at Nvidia Jetson Nano får inn korrekt temperatur fra temperatursensor over I^2C. Koblingsskjemaet for testen er vist i figur 1 nedenfor.

2 Utstyr og innstillinger

- Nvidia Jetson Nano 4GB på kommunikasjonskort som kjører skriptet "camPWM.py" eller "STTS75_driver.py"
- STTS75 temperaturføler loddet på kretskort
- Fluke 62 MAX+ IR Thermometer
- Ekstern kraftforsyning (5V)
- Diverse kabler til kraft
- Keysight InfiniiVision MSOX3012A Scope

3 Oppkobling



Figur 1: Oversikt over oppkobling for tester som gjennomføres

4 Progamverktøy

- Python 3.6

5 Programfiler

Programfilene kan finnes på Github: https://github.com/UiS-Subsea/Kommunikasjon-2023/tree/main/drivers

- camPWM.py
- STTS75_driver.py

6 Fremgangsmåte

Skriptet camPWM.py inneholder en klasse med en metode som intialiserer GPIO-pinnen for PWM-signalet. Her kan man konfigurere om man vil bruke pin 32 eller 33. Man setter også opp frekvensen til PWM-signalet og velger initiell duty cycle. En annen metode brukes til å sette en ny duty cycle på signalet. Her er det satt opp til at metoden tar inn en vinkel mellom 0 til 180grader og skalerer det om til duty cylce for servomotoren vi bruker.

Framgangsmåte for å teste PWM.

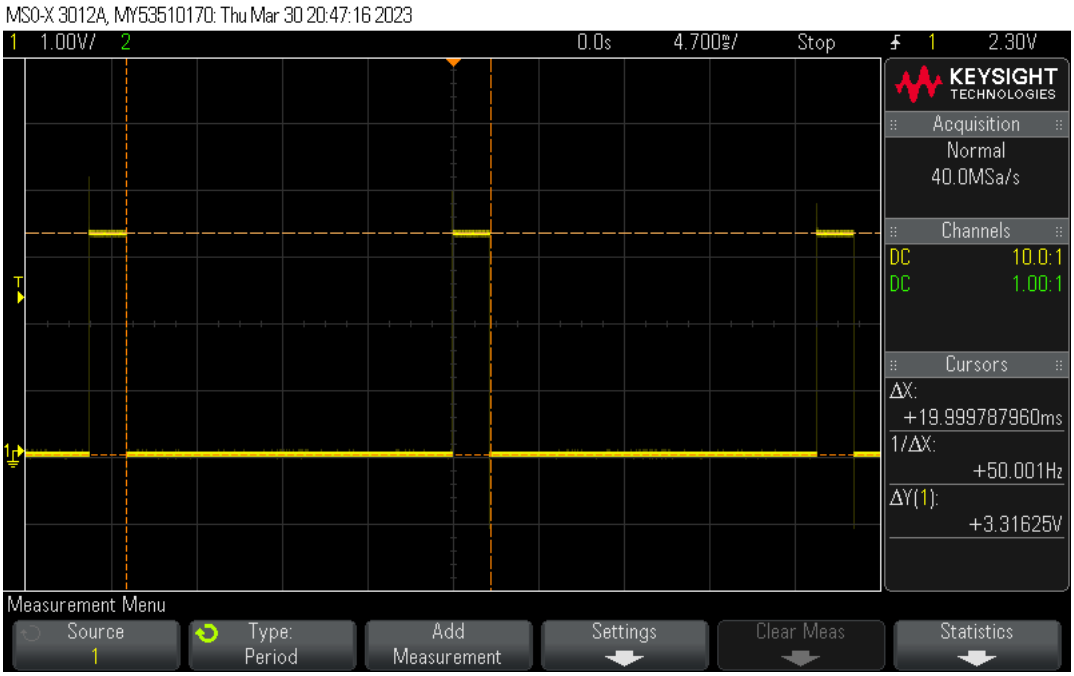
- Koble opp etter koblingskjema 1
- Start python skriptet for å styre PWM signalet. Velg pinne 32: pwmCAM.py
- Mål frekvens og duty cycle.

Skriptet STTS75_driver.py setter opp I2C kommunikasjon for STTS75 temperaturmåleren. Det tar så å henter ut signal hvert sekund.

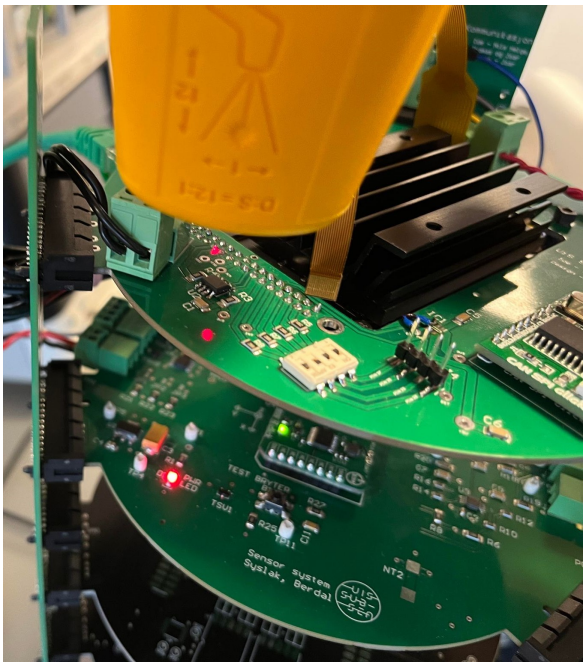
Framgangsmåte for å teste temperatursensor.

- Koble opp etter koblingskjema 1
- Start python skriptet for å måle temperaturen på kortet: STTS75_driver.py
- Mål temperatur med termometer og sammenlign med avlest data i konsoll.

7 Resultat



Figur 2: Scopebildet av PWM-signalet generert av Nvidia Jetson Nano



(a) Måling av temperatur rundt temperatursensor

(b) Avlesning av temperatur med thermometer

```
jetson@jetson-desktop:~/Kommunikasjon-2023/drivers$ python3 STTS75_driver.py
1
25.0
```

Figur 4: Temperatur avlest i konsoll på Nvidia Jetson Nano

8 Analyse og konklusjon

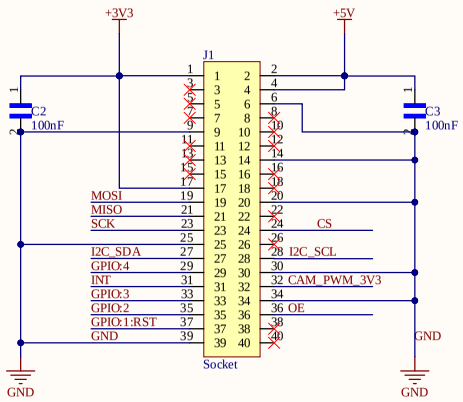
I testen er Nvidia Jetson Nano stilt inn til å gi ut et PWM-signal på 50Hz, altså periodetid på 20ms. Fra scopebildet i figur 2, kan det bekreftes at PWM-signalet er som ønsket.

For å verifisere temperaturen avlest av Nvidia Jetson Nano, er temperaturen rundt temperatursensoren først målt med et thermometer til å være $25,1°C$. Den avleste temperaturen i konsollen til Nvidia Jetson Nano viser en temperatur på $25,0°C$. Avlesning av temperatur over I^2C er dermed verifisert.

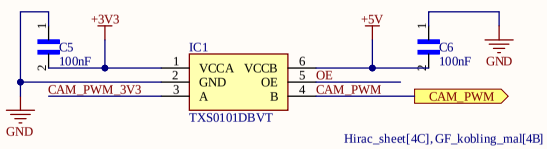
Vedlegg B

Skjemategninger

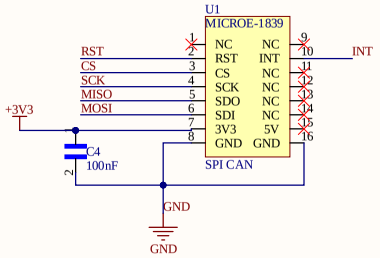
40-pin Header Jetson



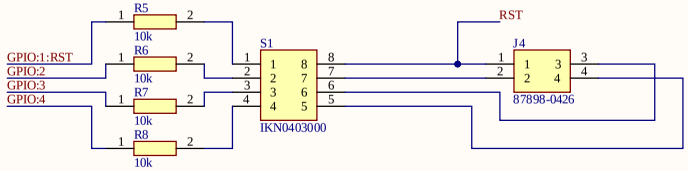
Level Swtich PWM Cam



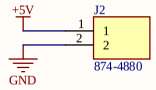
SPI til CAN-bus



DIP-switch

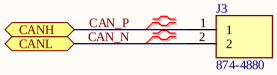


Plugg 5V Jetson

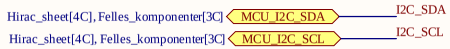


Plugg CAN-bus

Hirac_sheet[4C], GF_kobling_mal[4B], GF_kobling_mal[5B]
Hirac_sheet[4C], GF_kobling_mal[4B], GF_kobling_mal[5B]



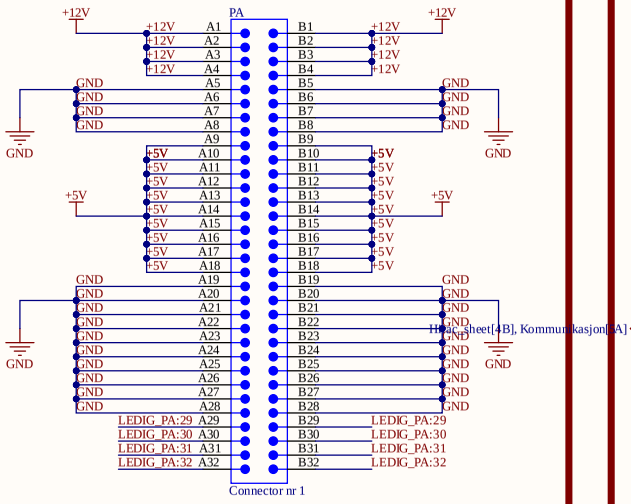
Innganger temperatursensor



Title		
Size A3	Number	Revision
Date: 5.10.2023		Sheet of
File: C:\Users\..\Kommunikasjon.SchDoc		Drawn By:

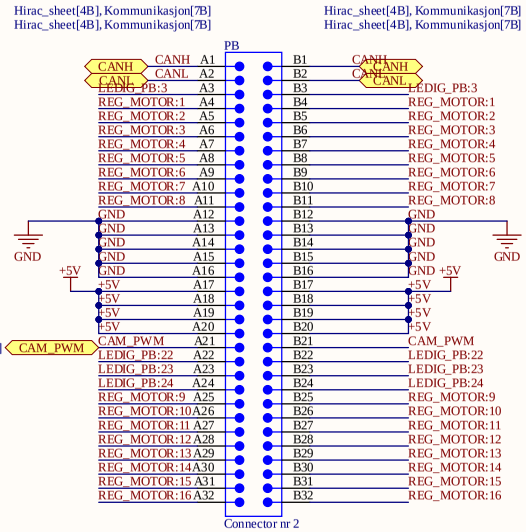
Plugg PA

Power bakplate med 12V og 5V



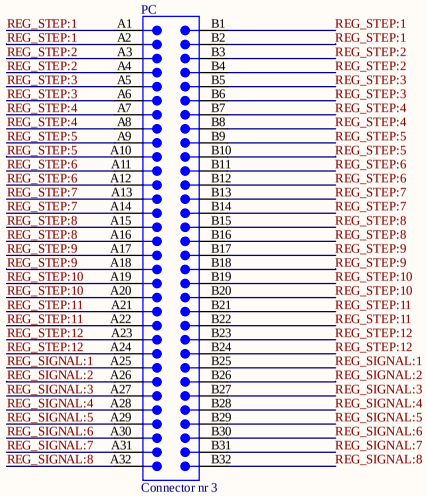
Plugg PB

CAN-BUS og PWM til propellermotorer
og kamera



Plugg PC

Manipulator



Informasjon:

Det er satt opp tre styk bakplater som har ulikt formål. Hovedårsaken til dette er for å få et enkelt med sterkt mekanisk festepunkt for alle kortene. Det er likevel behov eller ønske fra flere å bruke denne bakplaten til kommunikasjon, plugger eller andre ting. Slik det er satt opp, er det veldig mye ledig. Det er derfor fritt fram for å ønske seg en bane til et formål. Dette kan gjøres ved å finne seg en aktuell bane og sende dette til en av oss i kommunikasjonsgruppen.

Litt praktisk som kanskje ikke er åpenbart:

A1 = B1

Alle signal som er på A1 blir automatisk koblet til B1 også, og det er derfor det bare viser A1-32 som tilkoplingspunkt, da det er likt på andre siden.

Alle har alle koblingspunktene felles, slik at denne er lik for alle kort. Dersom du ønsker å bruke en bane mellom et kort, vil denne komme for alle. Det er derfor viktig at du gir beskjed, slik at signalet ikke blir brukt av noen andre.

I malen her ligger PCB'en med rette mål på 4-lags 1,6mm utlegg. Dette er noe som kreves for PCI kontakter, og er noe som JLCPCB klarer å produsere. Det er antatt at du bytter lagene slik at de passer ditt kort best. Dersom du har en grunn for mer eller mindre enn 4 lag(eller en annen størrelse), gi beskjed til oss i kommunikasjonsgruppen.

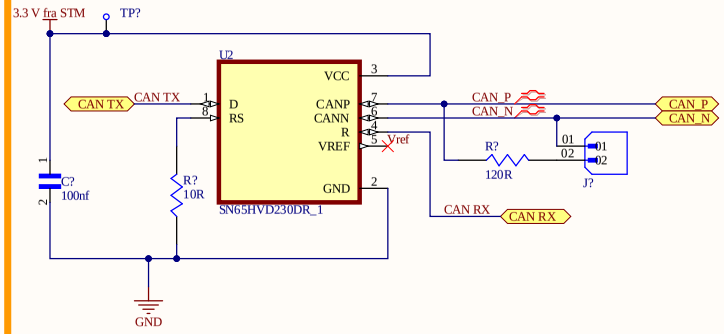
Dersom du må plassere gullfingrene(GF) på nytt, så skal de plasseres med enden av "Solder" laget 11 mils utenfor rammen av PCB utlegget. Er du i tvil, send oss et bilde eller ta kontakt.

Husk også å ikke jobbe i denne malen, men lag din egen kopi av den.

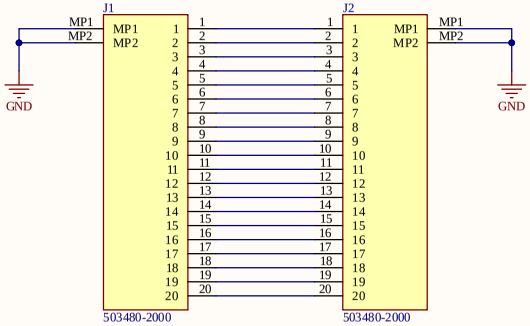
Title		
Size A3	Number	Revision
Date: 5.10.2023		Sheet of
File: C:\Users\..\GF_kobling_mal.SchDoc		Drawn By:

Mindre kort og felles tegninger

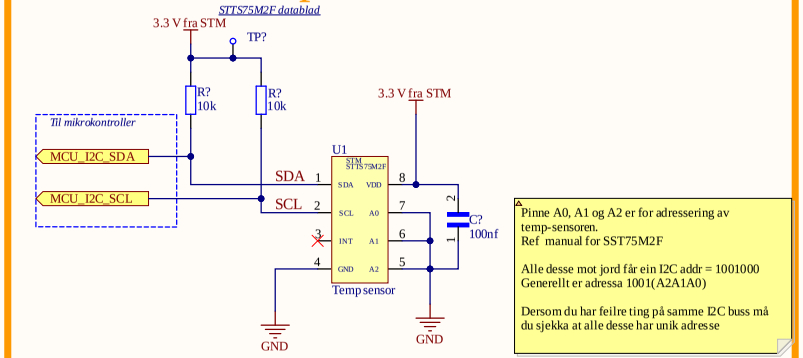
CAN-driver



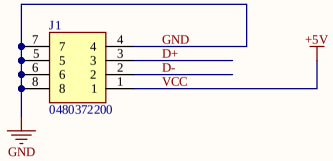
CSI-Overgang



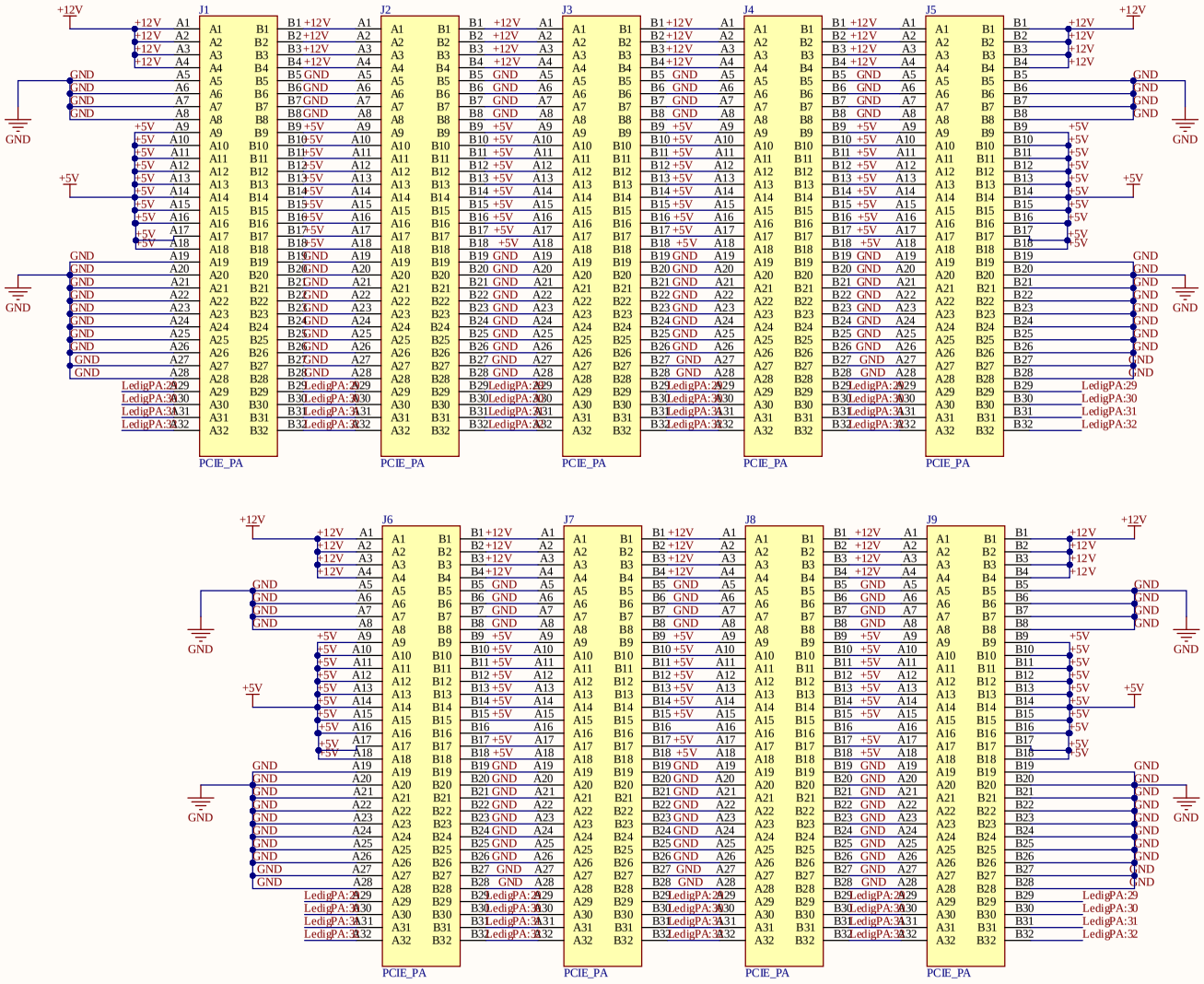
Temperatursensor



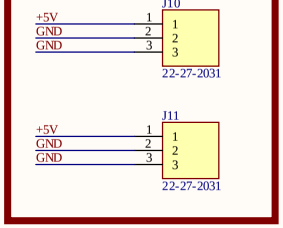
USB



Title		
Size A3	Number	Revision
Date: 5.10.2023	Sheet of	
File: C:\Users\..\Felles_skjema_med_småkort.SchDoc	Drawn By:	

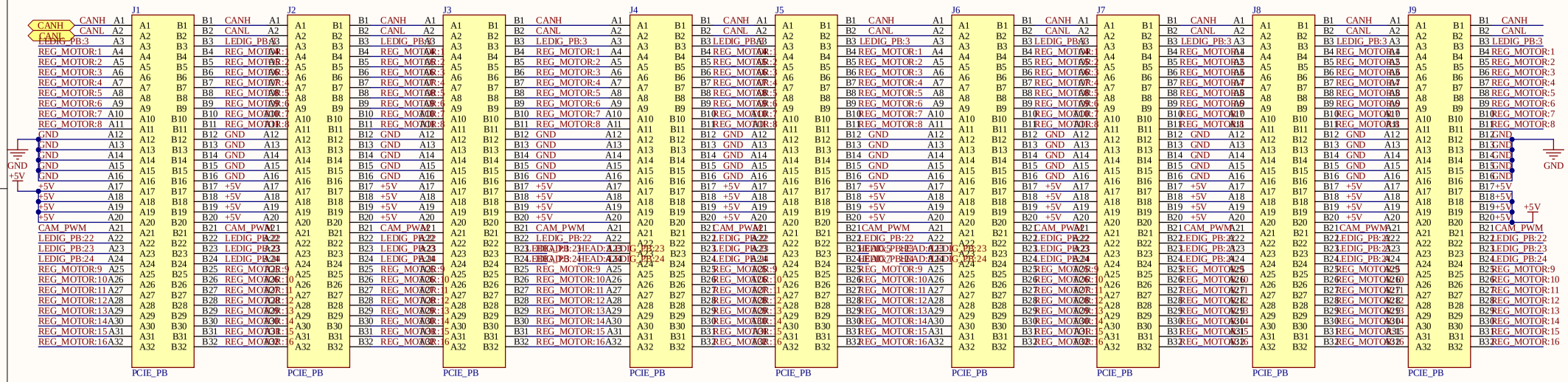


Plugger til vifter

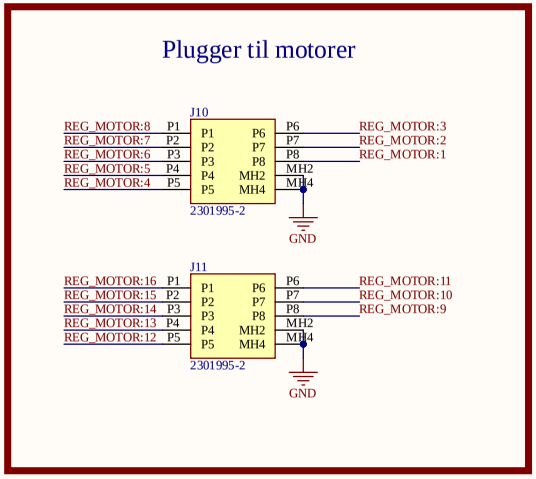


Title		
Size A3	Number	Revision
Date: 5.10.2023		Sheet of
File: C:\Users\..\Bakplate_PA.SchDoc		Drawn By:

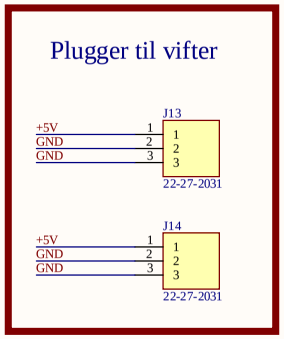
Kobling bakplate PB



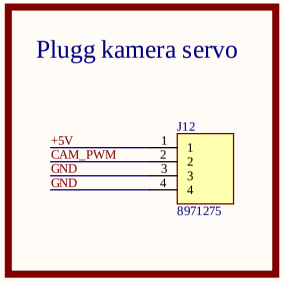
Plugger til motorer



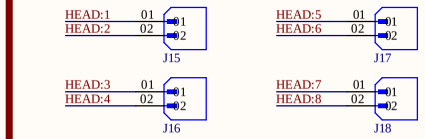
Plugger til vifter



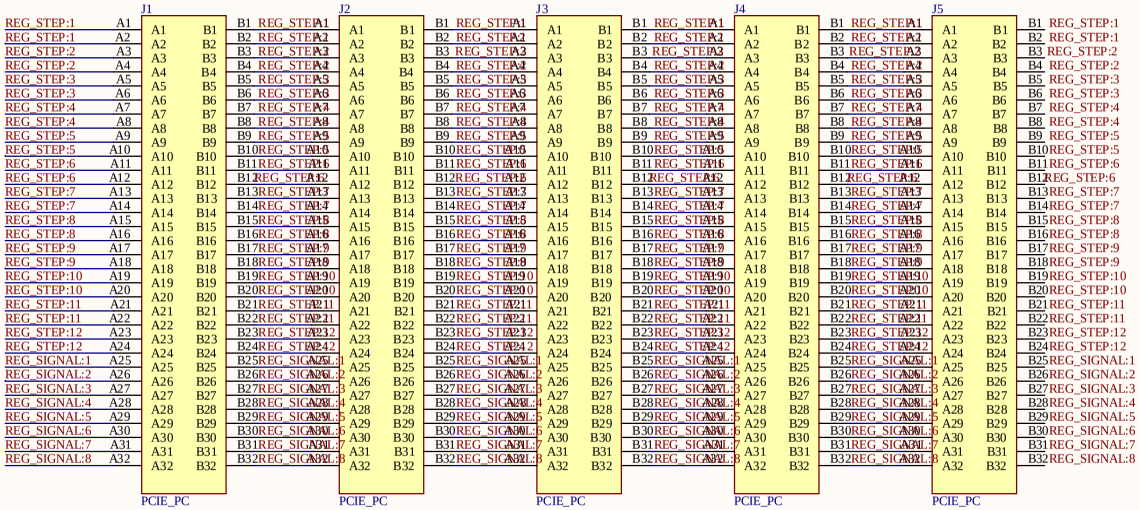
Plugg kamera servo

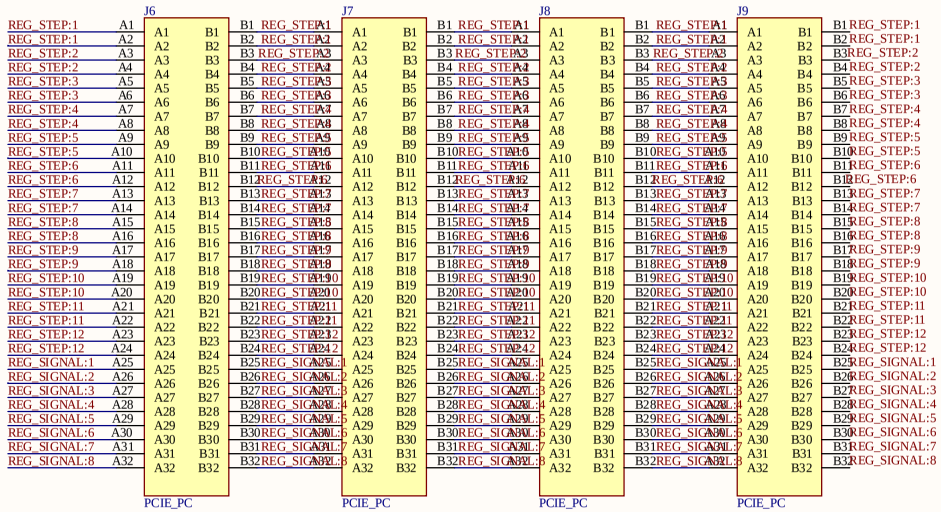


Header til viderekobling

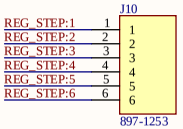


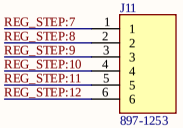
Title		
Size: A3	Number	Revision
Date: 5.10.2023		Sheet of
File: C:\Users\..\Bakplate_PB.SchDoc		Drawn By:





Plugger til steppermotorer

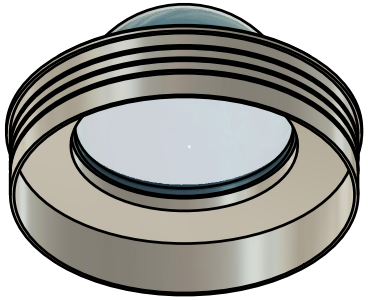


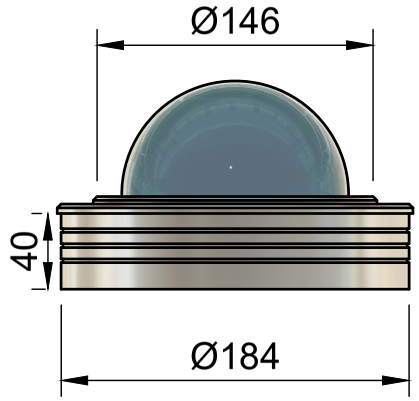


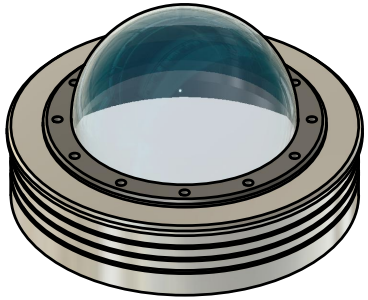
Title			
Size A3	Number		Revision
Date: 5.10.2023		Sheet of	
File: C:\Users\..\Bakplate_PC.SchDoc		Drawn By:	

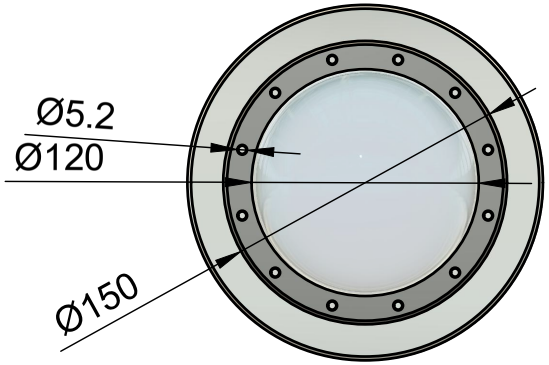
Vedlegg C

Mekaniske vedlegg

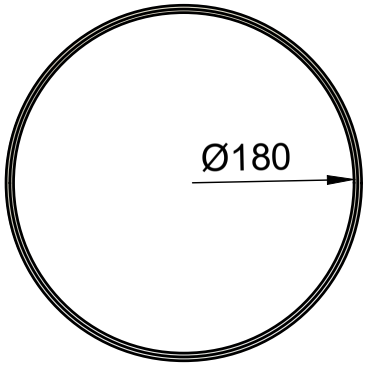


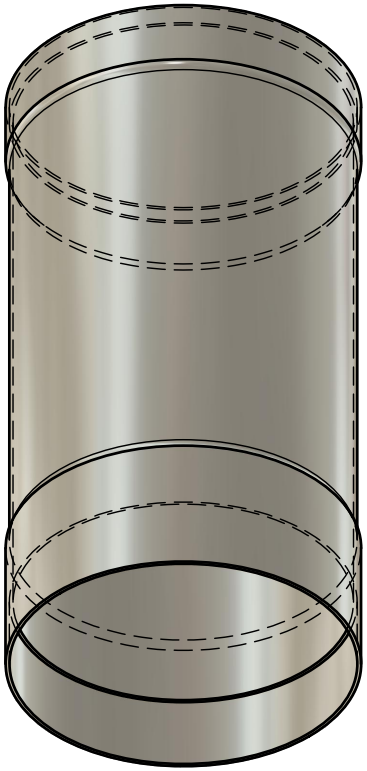


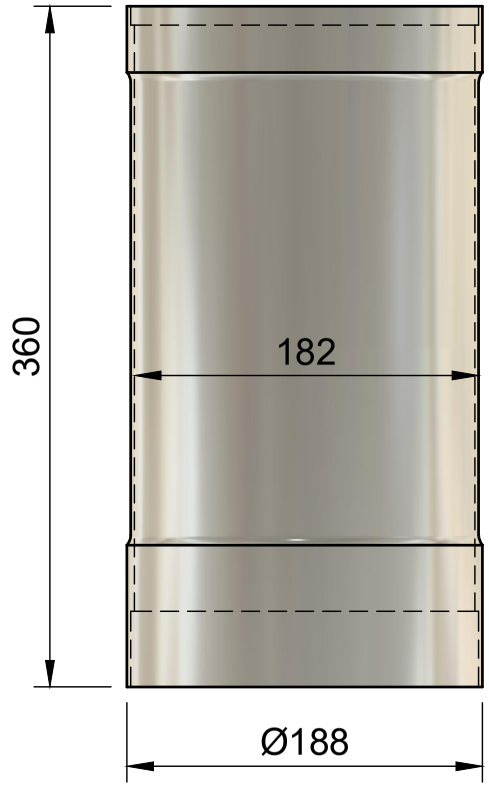




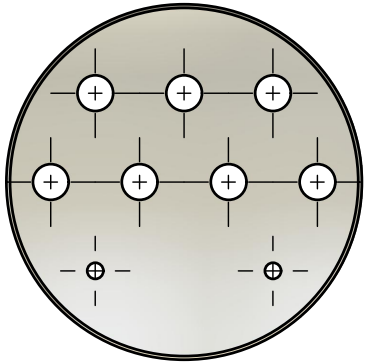
Dept.	Technical reference	Created by Joar Landa 10.05.2023	Approved by
		Document type	Document status
		Title Elektronikkhus Front med kuppel	DWG No.
			Rev. / Date of issue / Sheet 1/1

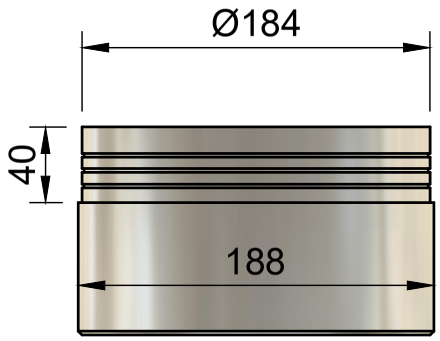


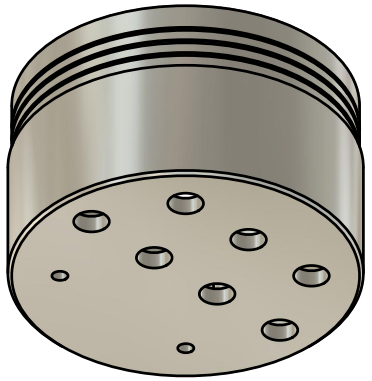


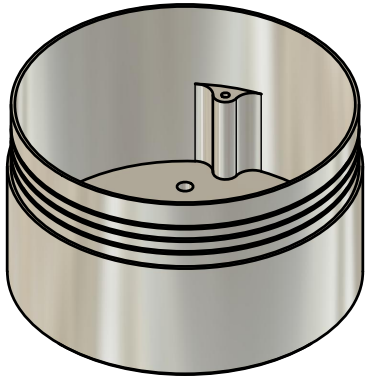


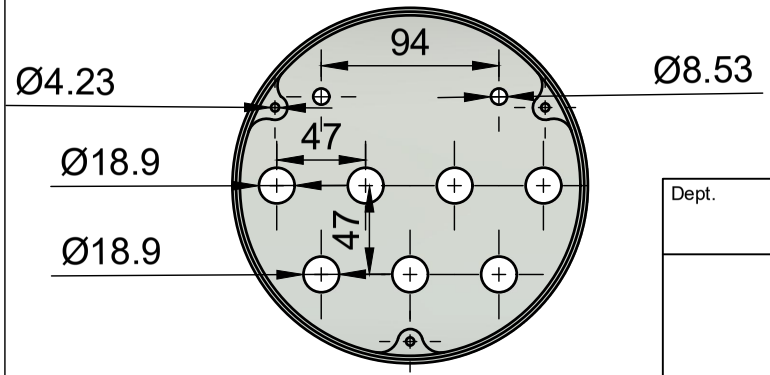
Dept.	Technical reference	Created by Joar Landa 10.05.2023	Approved by
		Document type	Document status
		Title Elektronikkhus Rørkropp	DWG No.
			Rev. / Date of issue / Sheet 1/1



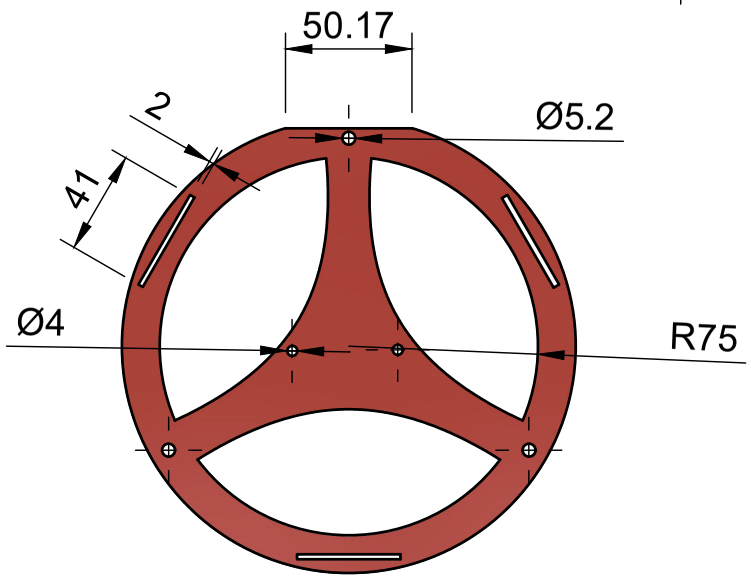


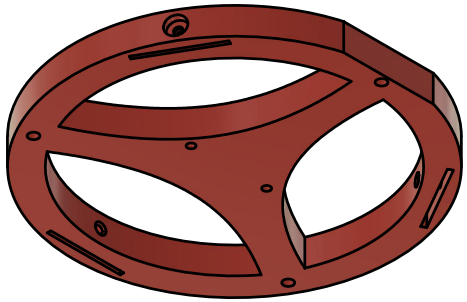


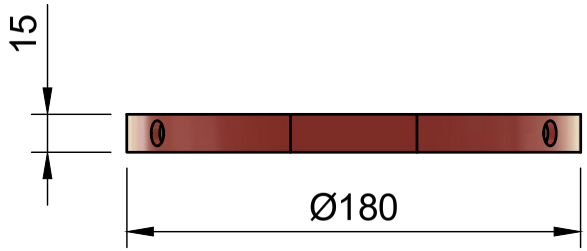




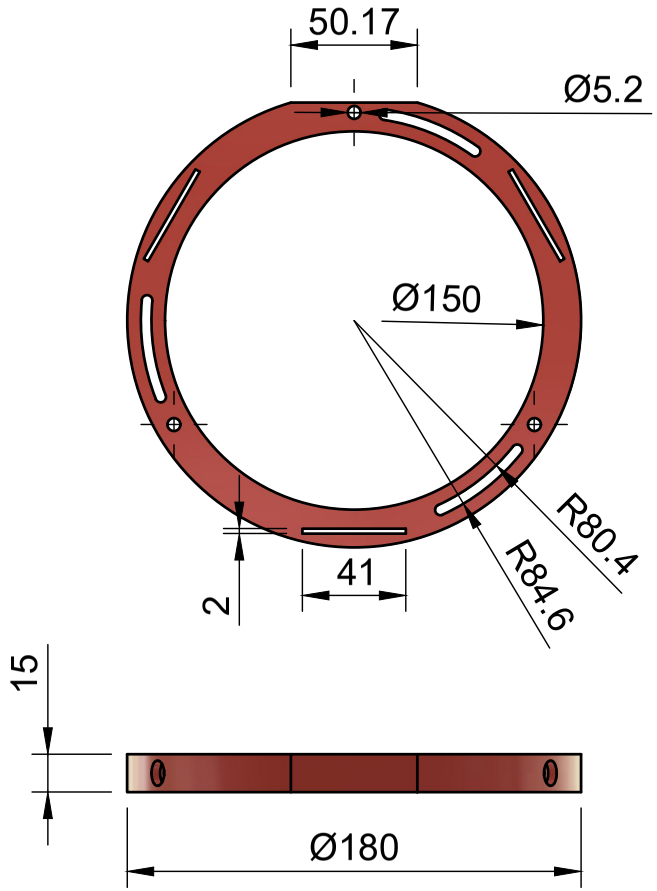
Dept.	Technical reference	Created by Joar Landa 10.05.2023	Approved by
		Document type	Document status
		Title Elektronikkhus Bunn	DWG No.
			Rev. / Date of issue / Sheet 1/1

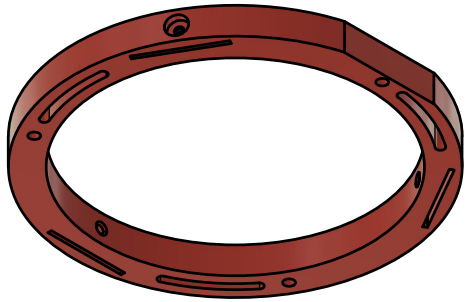




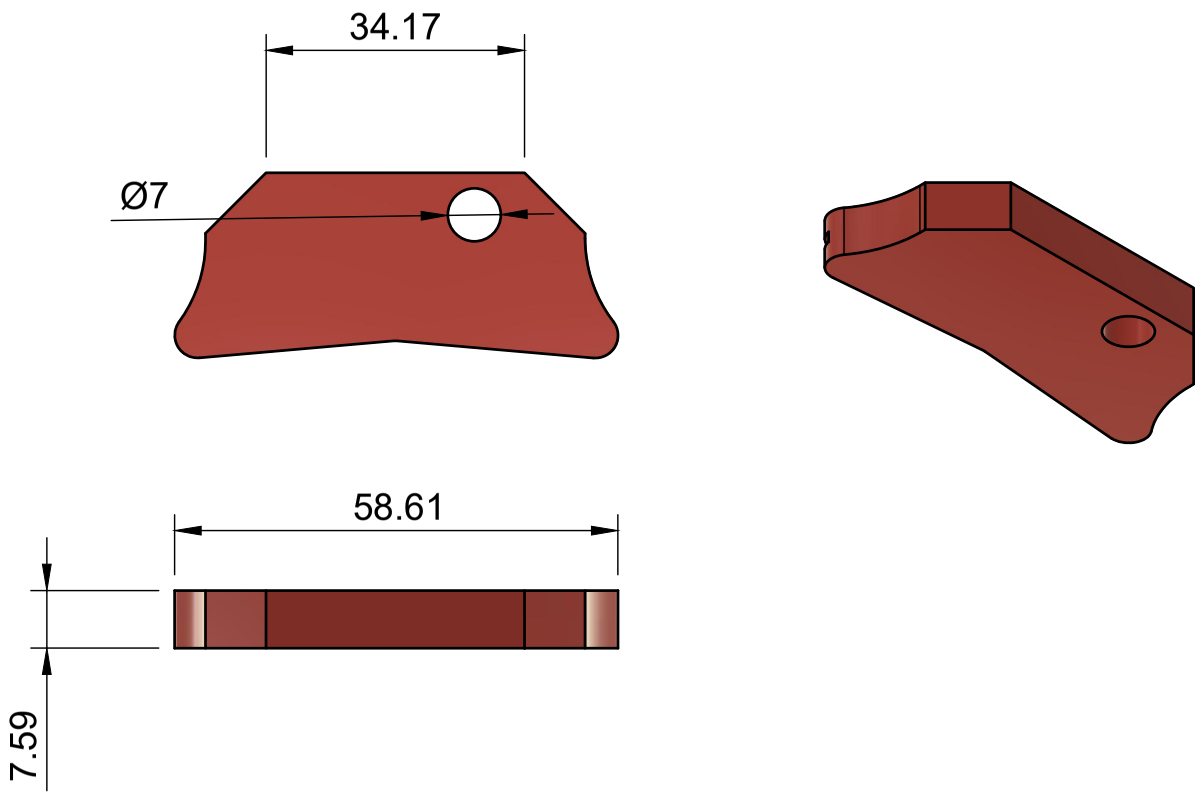


Dept.	Technical reference	Created by Joar Landa 10.05.2023	Approved by
		Document type	Document status
		Title Elektronikkhus Ring framme	DWG No.
			Rev. / Date of issue / Sheet 1/1

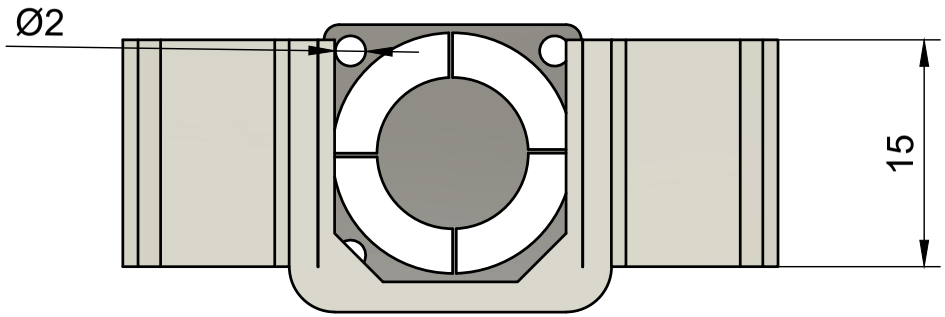


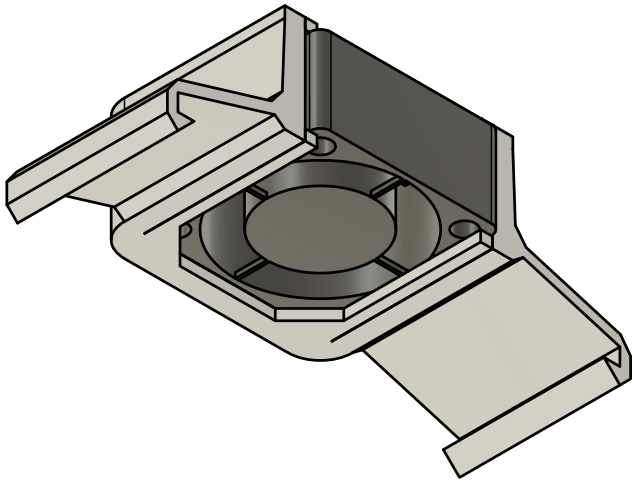


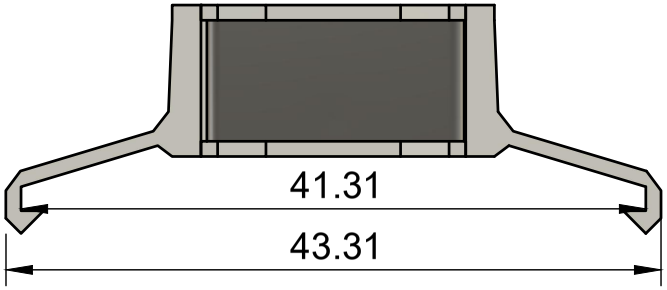
Dept.	Technical reference	Created by Joar Landa 10.05.2023	Approved by
		Document type	Document status
		Title Elektronikkhus Ring bak	DWG No.
			Rev. / Date of issue / Sheet 1/1



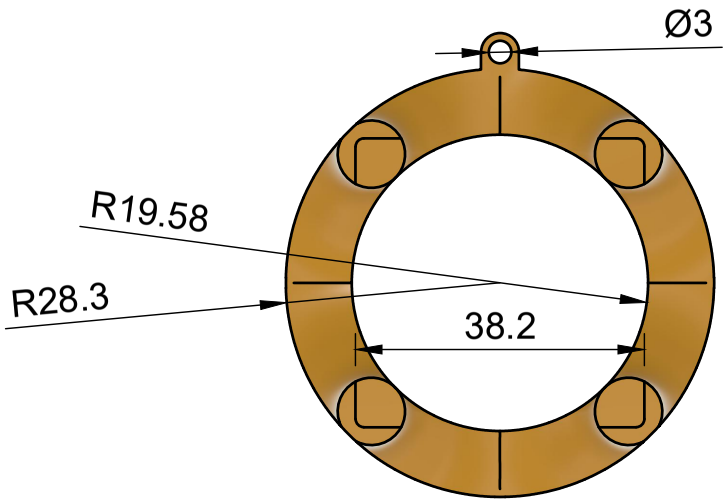
Dept.	Technical reference	Created by	Approved by
		Joar Landa 10.05.2023	
		Document type	Document status
		Title: Holder for lekkasjesensor	DWG No.
			Rev. / Date of issue / Sheet: 1/1

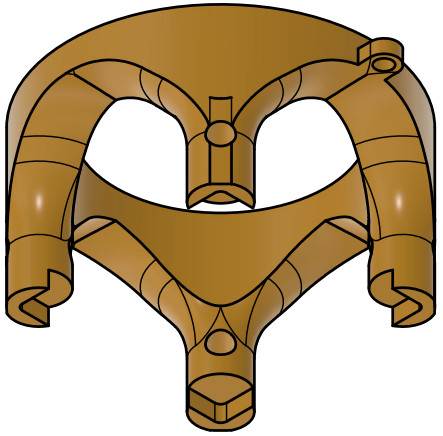


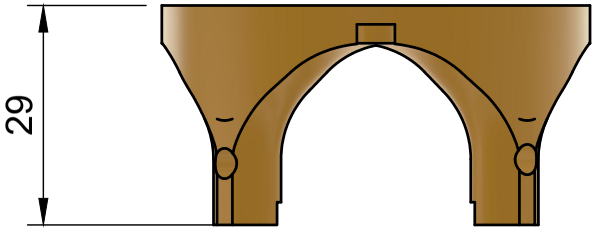




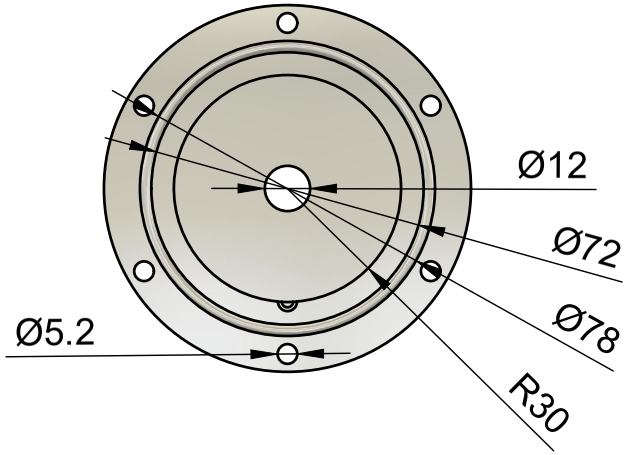
Dept.	Technical reference	Created by Joar Landa 10.05.2023	Approved by
		Document type	Document status
		Title Vifteholder	DWG No.
			Rev. / Date of issue / Sheet 1/1

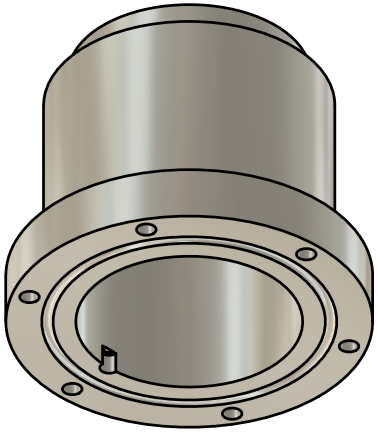


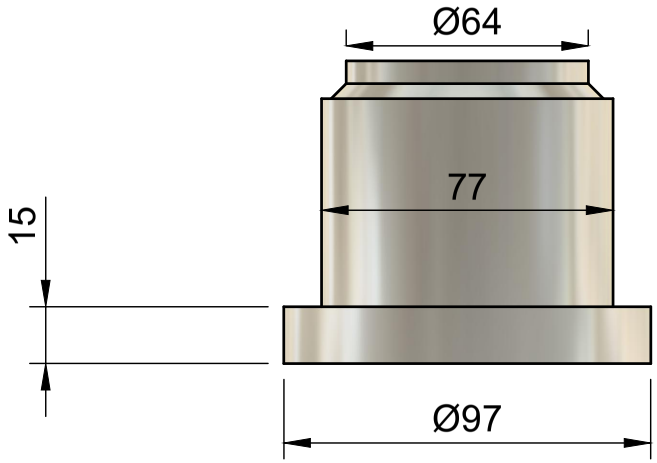




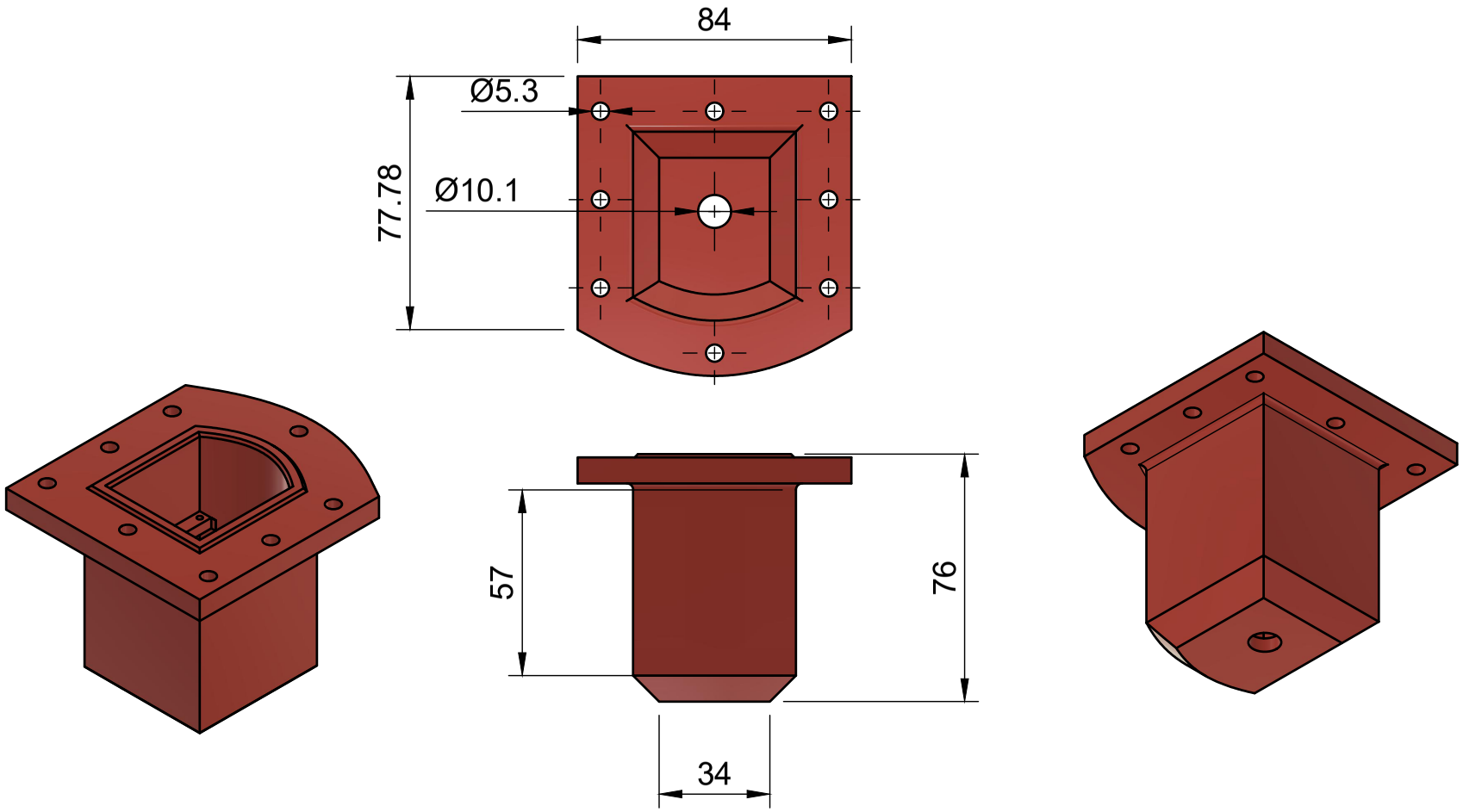
Dept.	Technical reference	Created by Joar Landa 10.05.2023	Approved by
		Document type	Document status
		Title Kamerahus_alu_innside	DWG No.
			Rev. / Date of issue / Sheet 1/1







Dept.	Technical reference	Created by Joar Landa 10.05.2023	Approved by
		Document type	Document status
		Title: Kameraus_alu	DWG No.
			Rev. / Date of issue / Sheet: 1/1



Dept.	Technical reference	Created by Joar Landa 11.05.2023	Approved by
		Document type	Document status
		Title Lite Kamerahus USB kamera	DWG No.
			Rev. / Date of issue / Sheet 1/1

Vedlegg D

I/O-lister

Canbus interface: Regulering 50ms cycle
Receiver -> Regulering

canID	Byte	Bit	Type	Description/Message	Bit Description	Scale	Warnign/Alarm setpoint	Sender	Note
32				Control Word				IDX_Kom	Init
	B0		bytearray						
		b0	bit	0: All regulering deaktivert					1 = kan aktiveres, 0 = av
		b1		1: Aktiver rull reg					
		b2		1: Aktiver stamp reg					
		b3		1: Aktiver dybde reg					
	B1								
	B2								
	B3								
	B4								
	B5								
	B6								
	B7								
33				Styredata ROV				IDX_Kom	Styredata ROV
	B0		int8	X_axis					ROV framover/bakover
	B1		int8	Y_axis					ROV venstre/høyre
	B2		int8	Z_axis					ROV opp/ned
	B3		int8	Rotation (gir)					ROV snu venstre/høyre
	B4								
	B5								
	B6								
	B7								
34				Styredata Manipulator				IDX_Kom	Styredata Manipulator
	B0		int8	Manipulator IN/OUT					
	B1		int8	Rotation Left/Right					
	B2		int8	Tilt Up/Down					
	B3		int8	Claw grip/release					
	B4								
	B5								
	B6								
	B7								
35				Sensor Data				IDX_Sensor	Regulerings data
	B0		int16	Rull Vinkel		Deg/100			Rull Vinkel
	B2		int16	Stamp Vinkel		Deg/100			Stamp Vinkel
	B4		int16	Dybde		Deg/100			
	B6			spare					
	B7			spare					
36									
	B0		int16	strømmåling 12V høyre		A/1000		IDX_Kraft	Måles i mA, 1000-60000
	B2								
	B3								
	B4								
	B5								
	B6								
	B7								
37									

Canbus interface: Sensor 50ms cycle
Receiver -> Sensor

canID	Byte	Bit	Type	Description	Bit Description	Scale	Warnign/Alarm setpoint	Sender	Note	Comment
64										
65										
66				Control Word				IDX_Kom		
	B0		Bitarray	Initiaiserings flag					Initiaiserings flag	
		b0		Nullpunk Dybde					Set nullpunkt dybde	
		b1		Nullpunk vinkler					Set nullpunkt vinkler	
		b2		Kalibrer IMU					Kalibrer IMU	
		b3		Vanntype					0 = feskvann, 1 = saltvann	For dybdeutregning
		b4								
		b5								
		b6								
		b7								
	B1									
	B2									
	B3									
	B4									
	B5									
	B6									
	B7									
	B7									
95			str	("marco\n")				IDX_Kom	heartbeat	

Canbus interface: Kraftforsyning 50ms cycle
Receiver -> Kraftforsyning

canID	Byte	Bit	Type	Description	Bit Description	Scale	Warnign/Alarm setpoint	Sender	Note
96									
97				Control Word 5V				IDX_Kom	
	B0		bitarray	Function Control					
	B1								
	B2								
	B3								
	B4								
	B5								
	B6								
	B7								
98				Control Word 12V Thruster supply				IDX_Kom	Thruster = Høyre
	B0		bitarray	Function Control					
	B1		uint8	Front light dimming SP		%/1			0-100%
	B2								
	B3								
	B4								
	B5								
	B6								
	B7								
99				Control Word 12V Manipulator supply				IDX_Kom	Manipulator = Venstre
	B0		bitarray	Function Control					
	B1		uint8	Bottom light dimming SP		%/1			0-100%
	B2								
	B3								
	B4								
	B5								
	B6								
	B7								
124									
125			str	("marco\n")				IDX_Kom	heartbeat 12V MAN
126			str	("marco\n")				IDX_Kom	heartbeat 12V THR
127			str	("marco\n")				IDX_Kom	heartbeat 5V

Canbus interface: Kommunikasjon 100ms cycle
Receiver -> Kommunikasjon

canID	Byte	Bit	Type	Description	Bit Description	Scale	Warnign/Alarm setpoint	Sender	Note
128									
129				Thrustpaadrag				IDX_Reg	Thrustpådrag
	B0		INT8	HHF					
	B1		INT8	HHB					
	B2		INT8	HVB					
	B3		INT8	HVF					
	B4		INT8	VHF					
	B5		INT8	VHB					
	B6		INT8	VVB					
	B7		INT8	VVF					
130				Temperatur				IDX_Reg	
	B0		int16	Temp Reg kort		c/100			
	B2		int16	Temp Driver kort		c/100			
	B4		int16	Settpunkt		mm/1			
	B6			REG_AKTIVER					
	B7			REG_AKTIVERDYBDE					
131									
132									
133									
134									
135				Akselerasjonsdata				IDX_Sensor	Kun for logging
	B0		int16	Accel raw x		ms^2/Accel scale factor		IDX_Sensor	
	B2		int16	Accel raw y		ms^2/Accel scale factor		IDX_Sensor	
	B4		int16	Accel raw z		ms^2/Accel scale factor		IDX_Sensor	
	B6		int16	Accel scale factor		*1000?		IDX_Sensor	
	B7			Accel scale factor MSB				IDX_Sensor	
136				Gyrodata				IDX_Sensor	Kun for logging
	B0		int16	Gyro raw x		deg/ gyro scale factor		IDX_Sensor	
	B2		int16	Gyro raw y		deg / gyro scale factor		IDX_Sensor	
	B4		int16	Gyro raw z		deg / gyro scale factor		IDX_Sensor	
	B6		int16	Gyro scale factor		*1000		IDX_Sensor	
	B7			Gyro scale factor MSB				IDX_Sensor	
137				Magnetometer data				IDX_Sensor	Kun for logging
138				Vinkler				IDX_Sensor	
	B0		int16	Rull		deg/100		IDX_Sensor	
	B2		int16	Stamp		deg/100		IDX_Sensor	
	B4		int16	Gir		deg/100		IDX_Sensor	Usikker om denne blir brukt
	B6		int16	Spare				IDX_Sensor	
	B7			Spare MSB				IDX_Sensor	
139				Temp / dybde				IDX_Sensor	
	B0		int16	Dybde		mm/1		IDX_Sensor	
	B1			Dybde MSB				IDX_Sensor	
	B2		int16	vann temp		c/100		IDX_Sensor	

Canbus interface: Kommunikasjon
Receiver -> Kommunikasjon

ID	Function	Value	Type	Message	Description	Scale	Warnign/Alarm setpoint	Sender	Note
Error				Error:	feilmelding			IDX_Kom	exceptions i program
Alarm				Alarm:	Alarm			IDX_Kom	alarmer feks. uc som ikke kommuniserer
200				[200, [Function, Value]]	Control Word Kamera			IDX_GUI	
	tilt					Deg/1			0-90* final scaling TBD.
		value		[200, [tilt, 45]]	Kamera tilt	Deg/1			
	start								
		stereo1		[200, [start, stereo1]]	Start stereo1				
		stereo2		[200, [start, stereo2]]	Start stereo2				
		bottom		[200, [start, bottom]]	Start bottom				
		manipulator		[200, [start, stereo1]]	Start manipulator				
	stop								
		stereo1		[200, [stop, stereo1]]	Stop stereo1				
		stereo2		[200, [stop, stereo2]]	Stop stereo2				
		bottom		[200, [stop, bottom]]	Stop bottom				
		manipulator		[200, [stop, manipulator]]	Stop manipulator				

Vedlegg E

Gantt og månedsrapporter

UIS Subsea 2023 - ELEKTRO Kommunikasjon

Planlagt: Utført: X Over tid:

AKTIVITET	Ansvarlig	Planlagte Timer	Faktiske Timer(så langt)	PLANLAGT START	PLANLAGT VARIGHET	FAKTISK START	FAKTISK VARIGHET	PROSENT FULLFØRT	PERIODER 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Planlegging av startfasen		**424**	**450**					**100 %**	X
Oppstart	**ALLE**	**20**	**20**	**1**	**2**	**1**	**2**	**100 %**	X
Overordnet design		10	**15**	**1**	**2**	**1**	**2**	**100 %**	X
Strukturering av arbeidsplan		10	**5**	**1**	**2**	**1**	**2**	**100 %**	X
Kartlegge behovs - og funksjonsspesifikasjon	**ALLE**	**14**	**20**	**1**	**3**	**1**	**5**	**100 %**	X
Behovsspesifikasjon		3	**5**	**1**	**2**	**1**	**2**	**100 %**	X
Funksjonsspeifikasjon		9	**10**	**1**	**3**	**1**	**3**	**100 %**	X
Long Lead Items		2	**5**	**1**	**3**	**1**	**5**	**100 %**	X
Konseptisering	**ALLE**	**60**	**70**	**2**	**3**	**2**	**6**	**100 %**	X
Utkast av design		20	**30**	**2**	**3**	**2**	**6**	**100 %**	X
Valg av maskinvare		20	**30**	**2**	**3**	**2**	**2**	**100 %**	X
Valg av kamera		20	**10**	**2**	**3**	**2**	**4**	**100 %**	X
Kompatibilitetsvurderinger	**ALLE**	**30**	**30**	**3**	**4**	**1**	**5**	**100 %**	X
Møte med andre grupper for undersystem		15	**15**	**1**	**4**	**1**	**7**	**100 %**	X
Møte med bildebehandlingsgruppe		15	**15**	**1**	**4**	**1**	**7**	**100 %**	X
Research	**ALLE**	300	**310**	**1**	**11**	**1**	**13**	**100 %**	X
Utvikling: Kortdesign og Modellering		**485**	**451**					**100 %**	
Konsept generering	**ALLE**	**220**	**140**	**2**	**3**	**4**		**100 %**	X
Kretskort design		180	**90**	**2**	**3**	**4**		**100 %**	X
Mekanisk design med kretskort		40	**50**	**2**	**3**	**4**		**100 %**	X
Valg av komponenter (Bestille?)		**20**	**25**	**3**	**4**	**3**		**100 %**	X
Konseptvalg		**15**	**16**	**5**	**2**	**5**		**100 %**	X
Styring av kamera		10	**6**	**5**	**2**	**5**		**100 %**	X
Håndtering av kretskort i ROV		5	**10**	**5**	**2**	**5**		**100 %**	X
Testing av konsepter - Simulert		**150**	**140**	**4**	**3**	**3**		**100 %**	X
Ferdigstilling av endelige spesifikasjoner		**40**	**70**	**4**	**3**	**6**		**100 %**	X
3D - Modeller for konsepter	**ALLE**	**40**	**60**	**5**	**2**	**4**		**100 %**	X
Produksjon/prototyping		**460**	**668**					**96 %**	
Prototyping		**75**	**110**	**6**	**8**	**5**		**100 %**	X
Rammeverk		**15**	**8**	**7**	**10**	**8**		**100 %**	X
Produksjon av Egne Kretskort		**75**	**40**	**6**	**11**			**100 %**	X
Justeringer av Kjøpte Kretskort		**50**	**25**	**9**	**12**			**90 %**	
Testing av moduler og subsystemer		**100**	**210**	**10**	**16**			**90 %**	
Elektro - Programmering av kort		**100**	**150**	**11**	**18**			**100 %**	X
Komplett montering av ROV		**45**	**125**	**12**	**20**			**95 %**	
Ferdigstilling og testing av ROV		**55**	**40**						
Første tur		**20**	**4**						X
MATE demonstrasjons video		**20**	**16**						X
Tuning av ROV		**15**	**20**						
Rapport		**450**	**600**						
Rapportskriving		**450**	**600**						

	Planlagt	Brukt
Totalt timeantall:	1 874	2 209

Rapport fra UiS Subsea Kommunikasjon gruppe - Januar

Alle konsepvalg er gjort:

- Single Board Computer for behandling av bilde opp
  - Under 100ms forsinkelse på kamera
  - Kan kjøre 4 usb kamera
  - Trenger ikke switch
- Sendes med ethernet opp, bilde med UDP
- Har CAN-bus intern kommunikasjon
  - Fått til å fungere på Pi
  - Gjenstår jetson
- Har bestemt innvendig størrelse på elektronikkhus
  - Di = 180mm, L = 435mm
- Har ferdig design innvendig for kretskort
  - Mal for utlegg for PCB er ferdig og levert til andre
  - Tre ulike bakplater
    - CAN/Signal
    - Power
    - Plugg og målepunkt

Det som er pressende nå er å tegne bakplate og kretskort i altium, få CAN-bussen til å fungere og lage klare rammer for hvordan andre kan implimentere dette. Som for eksempel å lage utlegget til CAN reciever kretsen, og lage kode for bruk av denne på stm32 ide'en.

Vi skriver det vi har begynt på, men er som alle andre meget usikker på hvor mye som er bra å skrive. Prøver å gå igjennom hverandre sitt materiale, og tenker å bruke bibliotektet ofte for gjennomgang.

Timer brukt er mellom 130-150 på alle, noe som er veldig likt og viser godt samarbeid. Vi møter rutinemessig på skolen og har god kommunikasjon rundt dette. Ser fort at det overskrides den antatte tiden som skal brukes på bacheloren.

> Ønsker å avtale et møte dersom du har tid en gang i framtiden. Joar drar på studietur neste uke, slik at dersom du har tid på fredag får vi gjort det før det. Ellers så er et ønskelig med et møte med Nils Helge og Thomas.

Rapport fra UiS Subsea Kommunikasjon gruppe - Februar

Hva som er gjort i perioden:

1. CAN-bus er testet og fått til å fungere mellom flere mikrokontrollere og Jetson Nano. Oppdateringshastiget kjørt er 500 kHz som er vel innenfor frekvensen som ønskes.
2. Adresseområder er gitt ut til de ulike gruppene slik at de nå kan begynne å koble funksjoner opp mot adressene. Det er også laget en oversikt der gruppene kan sette inn funksjoner som ønskes slik at det er mulig for andre å finne ut hvor informasjonen kan leses.
3. Protokoll mellom kontrollstasjon og ROV er testet. USB kamera er koblet opp ilag med stereokamera der begge fungerer. Gstreamer er valgt som programmvare for å sende bildestrømmen opp. Det er brukt enkoder på Jetson for å minnimere datastrømmen som blir sendt.
4. Jetson 4GB er kommet, denne holdes det på med å sette inn programm på tilsvarende det som har vært på den tidligere.
5. Programmeringskrets er droppet da Jetson ikke har noe støtte for direkte programmering av mikrokontrollere. Det vil bli forsøkt å bruke USB tilkobling fra Jetson og programmering fra kontrollstasjon.
6. Festemetode for innvendig kretskort har gått fra idefase til designfase.
7. PCB design for kretskort er ferdig og det mangler kvalitetssjekk av denne. Det er gjort rede for alle funksjoner og testet i elektronikkhus for å sjekke passform.
8. Bakplater er designet ferdig med de baner som skal tideles. Det er tildelt mange baner til reguleringsgruppen, for å redusere mengden kabler som blir sendt gjennom elektronikkhuset.
9. Det er gjort en hel del skriving på foskjellige kapittel, men hovedsaklig er det kommunikasjonskapittelet som har fått mest fokus. Mye teori er forklart, og vi er nå i gang med å skrive om utføring.
10. Mal for innholdsfortegnelse er relativt standardisert. Det gjenstår et par små endringer som ikke har blitt prioritert enda.

Timebruk er fordelt slik:

Thomas	**Nils Helge**	**Joar**
275	277,5	267

Generelt er det fortsatt god stemmning innad i gruppen og vi er alle veldig engasjert i prosjektet. Vi ser at timeantall fort vil gå forbi 600 før prosjeket er ferdig. Samarbeid med andre grupper er relativt bra, og prosjektet ser an til å gå i havn.

Forhåpentligvis går det fort når alt skal monteres og testest, da mye av programmeringen er gjort på forhånd. Det gjenstår myye lodding. Montering av modulen(*uten kapsling*) er forhåpentligvis i gang med en gang PCB-kort kommer, slik at test av elektrisk system kan skje lenge før elektronikkhus er ferdig.

Vi tror selv at vi har nok materiale til bacheloren, og har ikke tatt på oss noe særlig nye oppgaver. Satser her på å bli ferdig med våre oppgaver og dokumentasjon av dette først. Antar så at prosjektet blir med tidkrevende når det nærmer seg fysiske tester i vann og innsending av demo video.

> Møte er å foretrekke å ha i uke 11. I mellomtiden er kommunikasjon på mail helt supert