Høgskulen på Vestlandet

# BACHELOROPPGAVE:

# BO21E-27-ABB: Akvakultur

<Andreas Gjervik Aaland>
<Andreas Håvardsholm Brekken>
<Jan Einar Ludvigsen Husa>
<Runar Grøneng>

2021

# Dokumentkontroll

| *Rapportens tittel:* BO21E-27-ABB: Akvakultur | *Dato/Versjon* 14.juni 2021/1.1 |
|---|---|
| | *Rapportnummer:* B021E-27 |
| *Forfatter(e):* Andreas Gjervik Aaland<br><br>Andreas Håvardsholm Brekken<br><br>Jan Einar Ludvigsen Husa<br><br>Runar Grøneng | *Studieretning:* KOM18<br>*Antall sider u/vedlegg* 55 |
| *Høgskolens veileder:* Guang Yang | *Gradering:* Åpen |
| *Eventuelle Merknader:* Vi tillater at oppgaven kan publiseres. | |

| *Oppdragsgiver*: ABB | *Oppdragsgivers referanse:* Simon-John Hamilton |
|---|---|
| *Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon):*<br><br>*Simon-john Hamilton*<br><br>Simon-john.hamilton@no.abb.com | |

| **Revisjon** | **Dato** | **Status** | **Utført av** |
|---|---|---|---|
| 0.1 | 11.05.21 | Førsteutkast | Andreas Gjervik Aaland<br>Andreas Håvardsholm Brekken<br>Jan Einar Husa<br>Runar Grøneng |
| 1.0 | 08.06.21 | Sisteutkast | Andreas Gjervik Aaland<br>Andreas Håvardsholm Brekken<br>Jan Einar Husa<br>Runar Grøneng |
| 1.1 | 14.06.21 | Korrigert sisteutkast | Andreas Gjervik Aaland<br>Andreas Håvardsholm Brekken<br>Jan Einar Husa<br>Runar Grøneng |

# Forord

Vi vil som gruppe takke våre veiledere gjennom dette prosjektet, både eksterne og interne som har bidratt oss med verktøyene og støtten for å komme gjennom denne bacheloroppgaven. Dette til tross for en usikker hverdag med Covid-19 situasjonen som har stadig vært i endring.

Takk til ABB som har gitt oss en oppgave hvor vi står igjen med enormt faglig utbytte og det er vår varmeste anbefaling at ABB fortsetter med å lage prosjekter for studenter som kommer etter oss.

Vi vil takke vår veileder Guang, som har vært en kunnskapsrik og god støttespiller for oss gjennom dette prosjektet.

Til slutt vil vi takke våre nærmeste som gjennom en krevende situasjon har holdt ut med oss selv når vi har vært fraværende og fullstendig fokusert på prosjektarbeidet.

# Sammendrag

Denne rapporten er skrevet for å presentere resultatet av den tildelte oppgaven fra ABB, "automatisering av oppdrettsnæring". Denne rapporten tar for seg og beskriver de ulike «hardware» og «software» verktøyene som er blitt brukt til å lage en prototype av en fiskemerd, en grafisk bruker-applikasjon, hvordan de kommuniserer med hverandre, og testing av dem som en sammensatt prototype.

Formålet med dette prosjektet var å komme på en ny og teknisk robust løsning for ABB for å øke automatisering av oppdrettsanlegg. En prototype er laget som en generisk modell av en representativ fiskemerd, med en tilhørende grafisk brukerapplikasjon som brukes for å styre denne prototypen. Prototypen og applikasjonen kommuniserer med hverandre ved å bruke en server/klient modell som overfører koordinater for å styre prototypen.

Vi har utviklet en rekke tester ment til å fremstille sammensatt hva prototypen er i stand til å gjøre og hvilket område den er funksjonell i. Disse testene gav oss verdifull informasjon om hvilke posisjoner prototypen kunne nå, og hvilke posisjoner den fysisk ikke var i stand til å nå. Denne informasjonen ble senere brukt til å definere yttergrenser som deretter ble programmert inn i brukerapplikasjonen.

Prosjektet i sin helhet er sett på som et bra fundament for å videreutvikle automatisering av oppdrettsnæringen, og har bredt applikasjonsområde innenfor akvakulturnæringen.

# 1 Innhold

# 1 Innledning

## 1.1 Organisering av rapporten

Rapporten starter med et sammendrag av oppgaven som gir en innføring i korte trekk om målet, prosessen og resultatet. Det gis også plass til å nevne litt om problemstillingen og de ulike løsningene som er foreslått.

Videre går rapporten gjennom de brukte teknologiene, og de tre største bidragene i oppgaven; motorstyring, brukergrensesnitt, og server/klient.

Til slutt gjøres det rede for testingen av de ulike komponentene i produktet vårt.

## 1.2 Oppdragsgiver

ABB AS er oppdragsgiver. ABB er et internasjonalt selskap og har en tilstedeværelse innenfor en hel mengde med industrier, som elektrifisering, robotisering, automatisering, motorer og omformere [1]. ABB i Bergen holder primært på med offshore vindkraft, akvakultur og landbasert infrastruktur.

Denne oppgaven er skrevet for avdelingen som har akvakultur som hovedfelt.

## 1.3 Problemstilling

I dag styres fôring og kamera (visuell tilbakemelding) av fiskeoppdrettet i stor grad manuelt. For å forandre kameraposisjon og for å følge med på fiskehelse er det operatører (røkter) som tar avgjørelser selv.
Dagens løsninger er i hovedsak trinser som styrer kamera i en akse på det horisontale plan, og en vaier som senker kamera ned i en fiskemerd [2]. I merden ligger det sensorer for salinitet og oksygeninnhold, fôr-slange og slange for fjerning av død fisk. Disse hindringen i merden forårsaker at det er posisjoner hvor kamera og riggen som styrer kamera ikke kan bevege seg fritt. I tillegg bruker kameraoperatøren mye tid på å styre kameraet til der det skal være.

## 1.4 Hovedidé til løsningsforslag

Ut ifra spesifikasjonene fra oppgavebeskrivelsen gitt av ABB, ble det tidlig tydelig at oppgaven mulig kunne bli svært omfattende. Gruppen var tidlig ute med store ideer og visjoner for hva som kunne gjøres, uten å kunne forutsi hvor mye tid de enkelte delene av oppgaven kunne ta. Sammen med ABB ble det bestemt at oppgaven skulle deles opp i flere deler, der hoveddelen av oppgaven skulle komme i fokus og mindre viktige deler skulle komme som ekstraoppgaver dersom det var tid.

Hoveddelen av oppgaven går ut på å designe, konstruere og programmere en løsning for automatisk styring av kamera i fiskemerden. Dette kameraet skal kunne fjernstyres fra et kontrollrom eller lignende. Dette vil da si at gruppen sitt mål var å konstruere en ny mekanisk løsning for å flytte kamera i merden, samt programmere motorstyring, designe og programmere et brukergrensesnitt, og sette opp nettverksbasert kommunikasjon mellom disse.

Ekstraoppgavene gikk videre ut på å hente inn diverse informasjon fra sensorer i fiskemerden, som temperatur og strøm i vannet. Til slutt var det også et ønske om å lage en løsning for å kunne automatisk oppdage og gjenkjenne fôr og fisk.

# 2 Kravspesifikasjon

Gjennom samtaler med oppdragsgiver er det kommet frem til noen krav løsningen skal ha. Først og fremst at den skal være enkel og robust, ettersom miljøet installasjonen skal stå i ofte kan slite på konstruksjonen dersom det er mange utsatte komponenter. Selve merden som løsningen skal stå på veier bare noen få kilo i vannet, så løsningen må også være lett. Løsningen må også ta hensyn til eventuelle andre konstruksjoner som allerede er i merden, som fôringssystemer og lignende.

Noen punkter på hva som kreves:

- Konstruksjon/mekanisk løsning:
    - Enkel og robust
    - Lav totalvekt
    - Ikke komme i veien for eksisterende konstruksjoner
    - Skal kunne flytte kamera i alle retninger i et XYZ-plan
- Software
    - Kommunikasjon mellom kamera/kamerastyring og kontrollrom
    - Fjernstyring av kameramodul
    - Kamera skal kunne posisjonere seg automatisk i merden ut ifra koordinatene som blir gitt
    - Tilbakemeldinger slik at operatører vet hvor kameraet er til enhver tid
    - Gjenkjenne fôr og fisk
- Grensesnitt
    - En måte å vise informasjonen fra kameraet (GUI-løsning, bruker-Interface)

Ved hjelp av disse kravene ender vi opp med et definert mål for hva som skal utføres i løpet av hovedprosjektperioden, samt sette klare linjer for hvordan en prototype skal fungere.

# 3 Analyse av problemet

## 3.1 Innledning

Den komplette løsningen til problemet skal innbefatte styring av kamera, visuell representasjon av kameraposisjon i et grafisk grensesnitt, og kommunikasjon mellom en Raspberry Pi (RPi) og en PC med programvare.

### 3.1.1 Styring av kamera

Kameraet skal kunne fjernstyres ved å velge en posisjon i merden (ikke en joystick-løsning), og kunne bevege seg i fritt uten å komme i konflikt med annet utstyr. Den bør også kunne stilles i riktig posisjon for å kunne se død fisk. I tillegg bør den ha en løsning for å kalibreres. Det stilles ikke krav til valg av maskinvare eller programvare.

### 3.1.2 Grafisk Grensesnitt

Grensesnittet skal være en 3D-representasjon eller sammensatte 2D-bilder som viser hvor kameraet kan være i merden, og som gjør det mulig å «klikke» kameraet til en posisjon. Det skal også være mulig å velge kameraposisjon ved å taste inn koordinater. Programmet bør kunne legge begrensinger for kamerabevegelser, f.eks. ved å forby sektorer.

### 3.1.3 Kommunikasjon

Kamerastyringen som foregår via en RPi i merdskapet skal kunne kommunisere og sende data til dataprogrammet som sender data og instruksjoner andre veien. Informasjon skal veksles raskt nok til at det ikke er nevneverdig forsinkelse i systemet. Kommunikasjonen bør være robust og takle feilhendelser på en måte som ikke krever jevnlig tilsyn av operatøren.

### 3.1.4 Maskinlæring

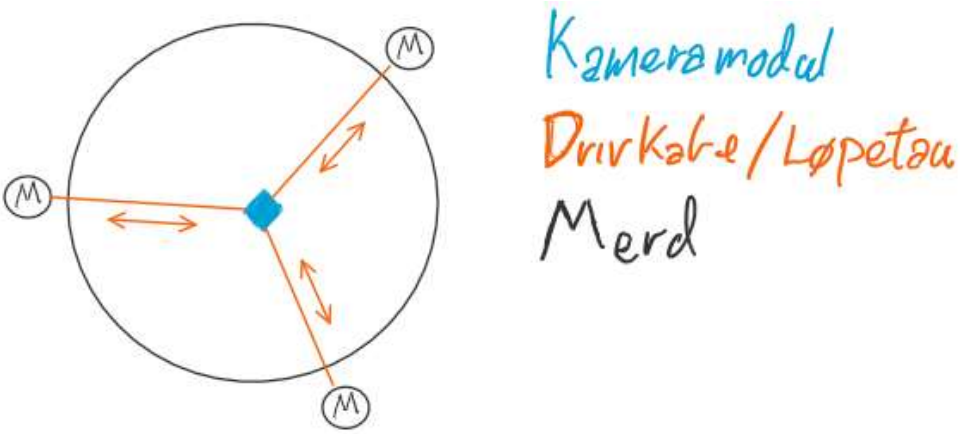
En funksjon som kan bruke kunstig intelligens til å kjøre datasett for å gjenkjenne pellets og fisk på bildene. Dette kan igjen brukes slik at kameraet kan «lete» etter pellets og fisk. For å benytte seg av dette må det naturligvis være en viss pålitelighet i programvaren.

## 3.2 Utforming av mulige løsninger

For å løse dette problemet er det krav til noe materiell og en del software. Motorer for å trekke og slippe kamera rundt i merden er nødvending, festeanordninger til riggen, kabling og diverse. Systemet er stort sett en automatisering av en eksisterende løsning, og det vil derfor være et mindre krav til brukeropplæring. Operatører vil måtte kunne bruke programvaren til å styre systemet, og helst ikke noe mer i daglig drift. Det må lages dokumentasjon i form av kildekode, arrangementstekning, koblingsskjema mm.

### 3.2.1 Løsningsalternativ 1

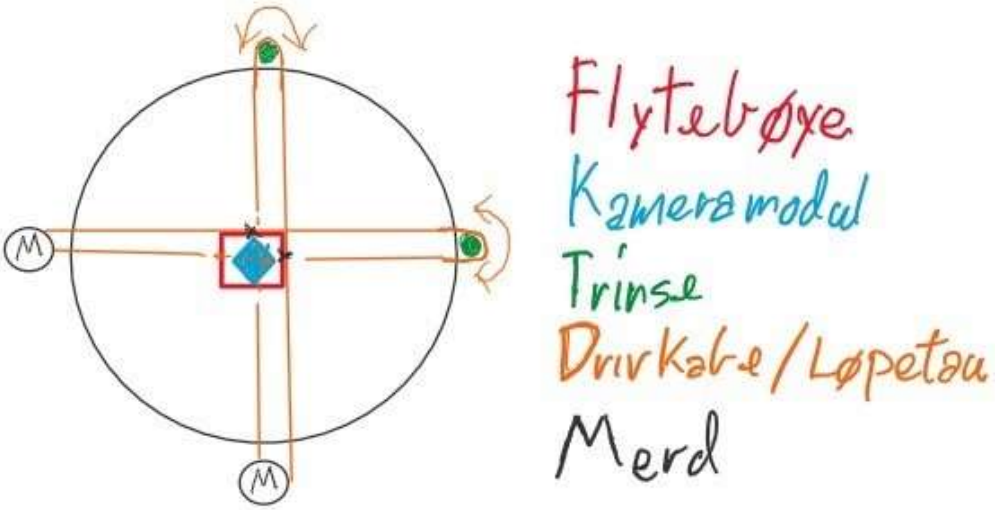
Det første konfigurasjonsalternativet er en såkalt “Delta/Trekant” (Eng: Delta/Wye) -løsning hvor servomotorene er plassert 120 grader fra hverandre rundt merden. Kameramodulen vil da henge i merden fra tre servomotorer. Kameramodulen vil kunne heves, senkes og flyttes rundt i merden ved ulike input fra servomotorene.




*Figur 1: Løsningsskisse med "Delta/Trekant"*

### 3.2.2 Løsningsalternativ 2

Det andre alternativet bruker en annen konfigurasjon for plassering av servomotoren, og bruker trinser og løpetau isteden. Løpetauet vil være festet i en flytebøye som kameramodulen er festet til. Dette er også hva dagens løsning er konfigurert som. Denne er beskrevet av ABB som en servomotor, som drar i en flytebøye ved bruk av et løpetau og en trinse. Disse er plassert offset fra senter av merden, på grunn av hindringer i merden.




*Figur 2: Løsningsskisse med trinser*

### 3.2.3 Vurderinger i forhold til verktøy og HW/SW komponenter

Valget falt på å bruke Raspberry Pi som hovedplattform i oppgaven. Vi har også valgt å programmere i Python da dette er et fleksibelt språk som vil kunne fungere på mange enheter. Om vi tar i bruk objekt-detektering, så er de mest brukte verktøyene for dette kompatibelt med Python. Når det gjelder motorer til styring av kamera var det opprinnelig planlagt å bruke ABBs egne servomotorer da det var et ønske om dette. På grunn av tilgjengelighet valgte vi å bruke enkle DC motorer i stedet.
For å deaktivere motorene når de ikke er i bruk vil MOSFETs kunne være et godt alternativ da de også vil kunne skaleres i forhold til størrelsen på motorene.

## 3.3 Konklusjon av analysen

I henhold til kravspesifikasjonen fra oppdragsgiveren har vi som gruppe drøftet og kommet fram til en rekke løsningsalternativer. Gjennom veiledning med oppdragsgiver har vi valgt ut to løsninger som vi og oppdragsgiver oppfatter som reelle og gjennomførbare.

Det disse to alternativene har til felles for hverandre er løsningene på HW/SW. Det vil si komponentene som Raspberry Pi som hovedplattform, GUI for bruker-Interface, Server-Klient modell for å kommunisere mellom de ulike komponentene.

Det som skiller dem fra hverandre er den faktiske plassering av servomotorene rundt merden. Gruppen har gjennom samtaler med ABB valgt å gå videre med “Delta/trekant”-løsningen. Det legges vekt på at dagens løsning, som går ut på å flytte kameraet rundt i merden ved bruk av trinser og flytebøyer, har en rekke begrensinger som ABB ønsker utbedret. Det omfatter blant annet hindringer som for eksempel foringsmerden som er plassert i senter, dette gir bevegelsesbegrensinger i forhold til hvor flytebøyen kan plasseres.

“Delta/trekant”-løsningen vil ha kameramodulen hengene under vannoverflaten. Ettersom mesteparten av annet utstyr ligger i vannoverflaten, vil kameramodulen ha større bevegelsesfrihet, ved bruk av “Delta/trekant”-løsningen.

# 4 Realisering av valgt løsning

## 4.1 Innledning

Denne oppgaven var til å begynne med ganske bredt rettet. I starten var det en viss usikkerhet om hva den innbefattet. Etter noen innledende møter med ABB kom vi frem til en kravspesifikasjon og et mål med oppgaven som vi kunne gå ut ifra.

Vi hadde tidlig utarbeidet en idé på hva vi ønsket å oppnå med oppgaven. Det viktigste var å få på plass en fjernstyring av kameraet der operatør kan velge fritt posisjonen til kameraet i merden, i motsetning til manuell styring av kameraet som var dagens løsning. I tillegg ønsket vi en ny måte for kameraet å bevege seg, mekanisk sett.

Det vi endte opp med var kamerastyring fra en «operatørkonsoll», i form av en PC, som styrer tre motorer i en delta-konfigurasjon. Det vil si at de er separert 120 grader seg imellom, og at høyden og sideveis plassering er bestemt av lengden på tauene til alle tre motorene.

## 4.2 Brukte Teknologier

### 4.2.1 Python


*Figur 3: Python logo [3].*

Vi har valgt å bruke Python som programmeringsspråk på tvers av alle våre plattformer. Python ble skapt av Guido van Rossum (1989). Python ble skapt med hensikt på å skape et språk hvor syntaksen skulle være mer lesbar som klartekst og en alt i en pakke. Python er objektorientert høynivå programmeringsspråk og har en rekke av aspektene som vi ser på som kan være en fordel for dette prosjektet. En annen stor fordel ved bruk av Python er at koden ikke må kompileres før den kjøres. Dette gjør det mye mer effektivt å teste koden fortløpende under utviklingen. [4] [5]

### 4.2.2 PyCharm

*Figur 4 - PyCharm logo [6]*

Vi har valgt å bruke utviklermiljøet (IDE) PyCharm for å programmere. PyCharm er utviklet av JetBrains. PyCharm er en kryssplattform IDE som gir oss muligheten til å programmere på ulike plattformer som Windows, MacOS og Linux uten at koden må endres vesentlig for å fungere på tvers av disse plattformenene. Dette er av svært gunstig betydning for vårt prosjekt siden ikke alle i gruppen på prosjektet bruker de samme plattformene som arbeidsverktøy. [7]

### 4.2.3 Raspberry Pi

*Figur 5 - Raspberry Pi logo [8]*

Raspberry Pi (RPi) er en alt på et kretskort (Single-Board) datamaskin. RPi er produsert av Rasberry Pi Foundation. RPi ble utviklet med tanke på å være et lavterskels produkt for å lære grunnleggende informatikk samtidig som RPi er et allsidig produkt som kan brukes som en plattform for å utvikle en rekke ulike produkter. [9]

RPI vil for oss fungere som en plattform for motorstyring av merd prototypen og som Server/Klient for å koble seg opp til GUI og bruker.

*Figur 6 - Bilde av Raspberry pi [33]*

### 4.2.4 Github


*Figur 7 - GitHub logo [10]*

Det ble valgt å ta i bruk GitHub til versjonskontroll i dette prosjektet.

Dette gir oss mulighet til å lagre koden i skyen og på denne måten gjøre det lettere å samarbeide om koden. GitHub beholder også tidligere versjoner, slik at man enkelt kan rulle tilbake til en tidligere versjon om dette blir nødvendig.

PyCharm har også god integrasjon med GitHub innebygget i programvaren.

### 4.2.5 Fusion 360


*Figur 8 - Fusion 360 logo [11]*

For å designe deler til prototypen ble Autodesk Fusion 360 tatt i bruk. Programmet har mange muligheter og er intuitivt å bruke. I tillegg tilbyr de gratis utgave av sin programvare i full versjon til studenter.

### 4.2.6 Microsoft Teams

For å samarbeide effektivt har vi brukt Microsoft Teams. Teams gir oss mulighet til å samle filer og redigere dokument sammen i sanntid. I tillegg bruker vi dette programmet til å gjennomføre møter. [12]

### 4.2.7 Zoom

Vi har også brukt Zoom for å holde møter, og delta på forelesningene i faget. [13]

### 4.2.8 PrusaSlicer

*Figur 9 - PRUSA logo [14]*

For å kunne 3d-printe en 3d modell må filen først gjøres om til instruksjoner som 3d-printeren kan forstå. For å gjøre dette må man bruke en «slicer».

Den lager en fil med nøyaktige instruksjoner om hvordan printeren skal bevege seg, hvordan temperaturen skal være til enhver tid og hvor mye delen som printes ut skal kjøles ned underveis.

Dette programmet har allerede mange ferdig testede profiler for mange forskjellige 3d printere, noe som gjør at man raskt kommer i gang med å printe uten for mye testing og justering av innstillinger.

PrusaSlicer er åpen kildekode, og er videreutviklet fra Slic3r.

### 4.2.9 3D-Printer

*Figur 10- Wanhao logo [15]*


*Figur 11 - Bilde av 3D printer*

3D-printeren som er tatt i bruk er en modifisert Wanhao Duplicator 9 400.

Grunnet problemer med stabilitet har hovedkort og «extruder» blitt byttet ut.

Hovedkortet mottar informasjon fra sensorene til printeren og sender informasjon til motorkontrollerne om hvordan motorene skal beveges. Printeren kjører programvaren Marlin som er en av de mest brukte nå til dags. Denne programvaren er også åpen kildekode.

Extruderens oppgave er å presse en nøyaktig mengde filament(materiale) ned i skrivehodet til printeren.

## 4.3 Merd Prototypen

### 4.3.1 Design og utforming

I henhold til valgt løsning av oppgaven så var det nødvendig å designe og konstruere en fysisk prototype som er representativ for en fiskemerd i et oppdrettsanlegg.

Selve modellen av fiskemerden bygget vi av gjengestenger og patentbånd som det er god tilgjengelighet på i butikker som for eksempel Biltema.

*Figur 12 - Bilde av fiskemerd modell*

### 4.3.2 Komponenter

#### *4.3.2.1 Motorer*

Prototypen bruker 3 N20 likestrøm (DC) børstemotorer med gir-reduksjon til å flytte på kameramodulen i merden. Det var opprinnelig tenkt å benytte en utgave som gir 120 omdreininger/min, men på grunn av mangel på tilgjengelighet ble det besluttet å gå for en utgave med 160 omdreininger/min.

*Figur 13 - Bilde av DC motor*

#### *4.3.2.2 Sensorer*

Vi bruker en sensor som måler rotasjon på et løpehjul for å kunne finne ut hvor langt tauet beveger seg ut og inn.

I første omgang var det planlagt å bruke en «Hall Effect» sensor da denne gir god nøyaktighet og ikke skaper ekstra friksjon på systemet da den ikke trenger å berøre løpehjulet som tauet beveger seg over.

Denne fungerer ved at man sender en konstant strøm gjennom et ledende materiale og måler hvordan spenningen over dette endrer seg når det blir påvirket av et magnetisk felt. [16]

Dette innebærer at man i tillegg til sensoren måtte hatt en magnet på løpehjulet.

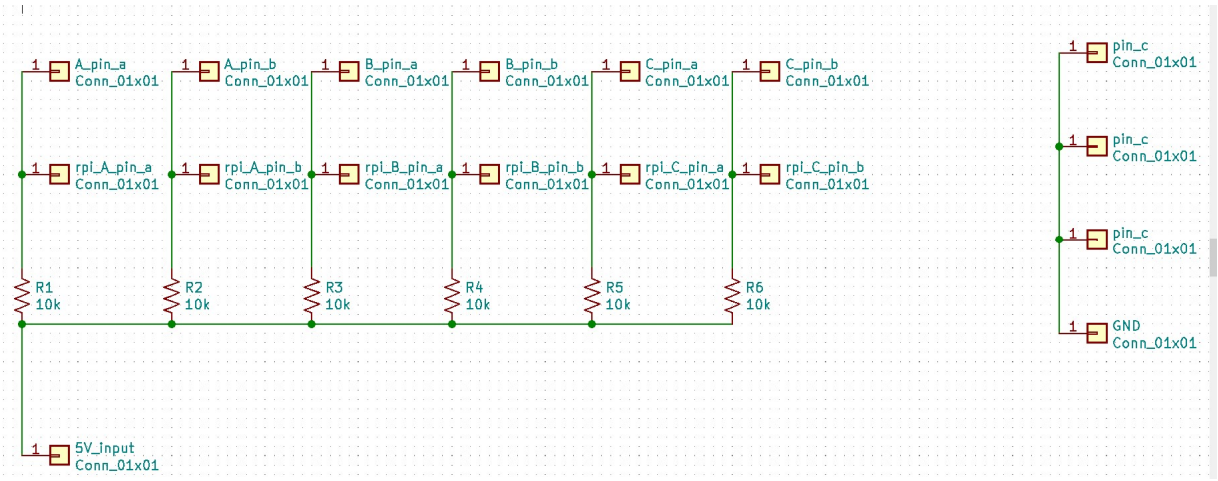
På grunn av høy kostnad for en slik sensor ble det valgt å bruke en «Rotary Encoder». Denne sender ut en puls etter å ha rotert en viss avstand. Den har 2 utganger og det vil variere hvilke av utgangene pulsen vil bli komme på først avhengig av hvilken retning den roteres. I denne oppgaven ble det benyttet en sensor av typen Alps STEC16B03 som sender ut 24 pulser på 360°.

Det er en mekanisk sensor som sender ut pulser når pinnene på innsiden kommer i kontakt med ledende spor som er festet på den roterende delen.

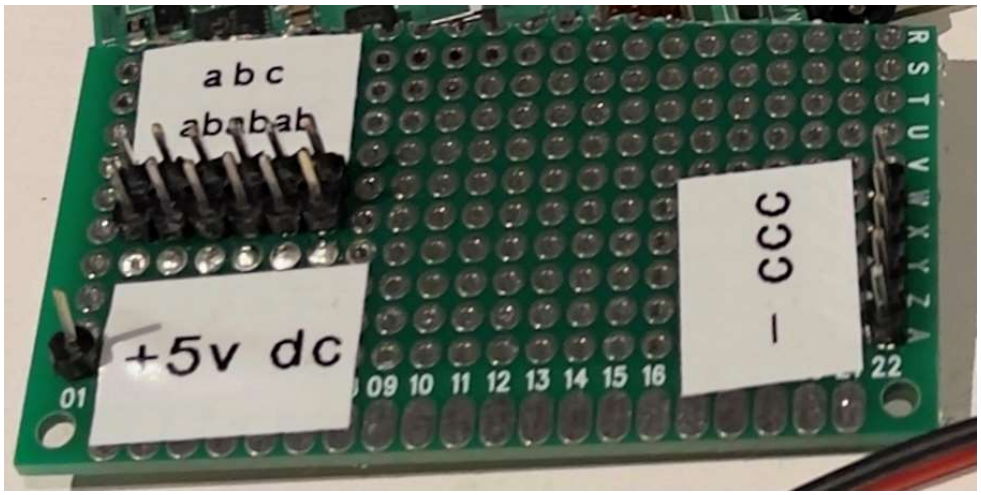
Dette skaper mer friksjon og unøyaktighet siden vi ikke får noen input på hvor den befinner seg mellom pulsene, men vil fungere godt nok til denne prototypen.

***Figur 14 - Bilde av "Rotary Encoder"***

Det ble også laget et eget kretskort får å gi et høynivå input til RPi-en når sensoren ikke er aktivert, i tillegg til å fordele jord til sensorene. Sensoren jorder inputen slik at man mottar et lavt signal når det blir sendt en puls.




*Figur 15 - Diagram av pullup kretskort*




*Figur 16 - Bilde av pullup kretskort*

#### *4.3.2.3 H-Bro*

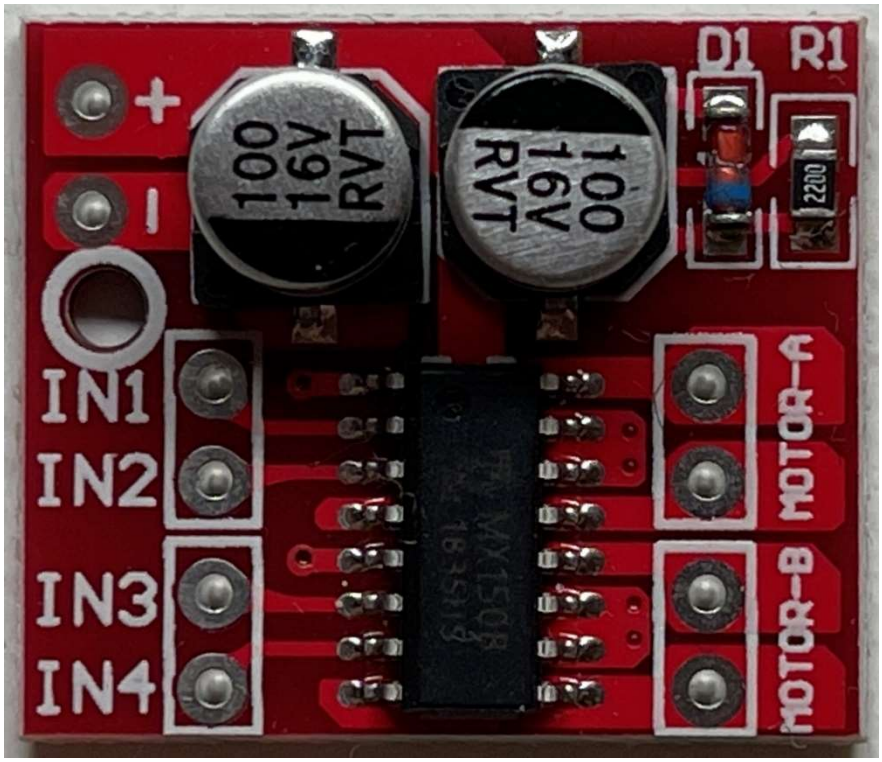
En H-bro gjør at man kan styre strømmens retning, som igjen gjør at man kan styre motoren begge veier. For å kontrollere hastighet brukes også PWM (Pulse Width Modulation). For å styre retning ved hjelp av H-broen, benyttes to inngangssignaler som kan motta høye og lave signaler.

Det er tatt i bruk en motorkontroller som er bygget rundt en MX1508 H – bro.

Denne støtter spenninger mellom 2 – 9,6V. Den støtter en kontinuerlig utgangsstrøm på 0,8A og en maks utgangsstrøm på 1,5A. Den har i tillegg en beskyttelseskrets som kutter strømmen om temperaturen blir for høy (150 grader celsius).

I vårt tilfelle ble det valgt å kjøre den på 9V for at motoren skal kunne yte mest mulig effekt.

Denne motorkontrolleren ble i hovedsak valgt på grunn av tilgjengelighet da vi allerede hadde disse tilgjengelig.



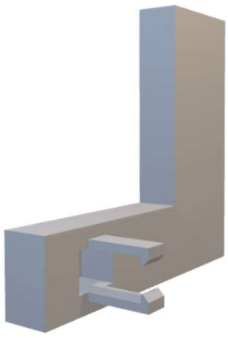

***Figur 17 - Bilde av motorkontroller***

#### *4.3.2.4 Braketter*

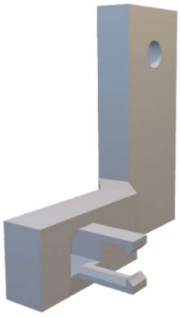
VI har designet og 3d-printet tre braketter som holder motorer og sensorer, og som igjen fester disse til merd-prototypen.

Den endelige utgaven har en guide for å sikre at tauet holder seg på sensoren, men denne er fjernet fra den fysiske modellen da den skapte for mye friksjon mot tauet.

Versjon 1 ble laget med tanke på motorfeste. Versjon 2 hadde feste for sensor, men det manglet et hull for å få plass til en metallbit som stikker ut fra sensoren. I versjon 2 ble også motorfeste midtstilt slik at sensorhjulet kommer inn på midten av spolen. I versjon 3 ble det laget et ekstra hull for at sensoren skal passe. I tillegg ble det laget en guide for at tauet ikke skal hoppe av sensorhjulet. I versjon 4 ble guiden flyttet lenger ut fra sensorhjulet.

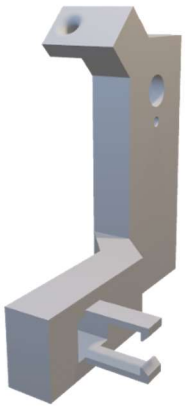


*Figur 18 - Brakett versjon 1*



*Figur 19 - Brakett versjon 2*

*Figur 20 - Brakett versjon 3*

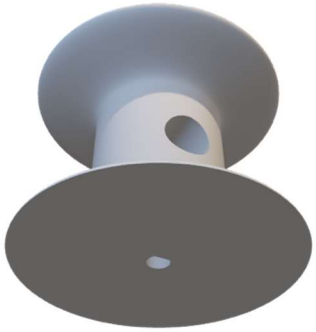


*Figur 21 - Brakett versjon 4*

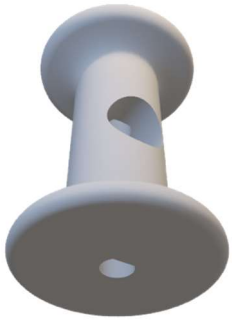
#### *4.3.2.5 Motor tau spole*

Vi har 3d-printet og designet tauspoler som samler opp og mater ut tau for å flytte på kameramodulen.

Første utkast har fungert som den skal, men for å senke hastigheten på tauet og for å få mer moment ble det designet en utgave med mindre diameter.



***Figur 22 - Spole versjon 1***

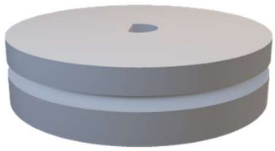


***Figur 23 - Spole versjon 2***

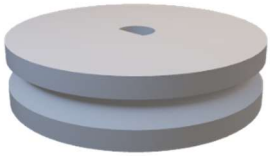
#### *4.3.2.6 Sensor Løperhjul (pulley)*

Vi har designet og 3d-printet løpehjul. Disse løpehjulene er designet med et rillet spor for å forhindre at tauet glir som vil føre til at motorene flytter kameramodulen til feil posisjon. Rotasjons-tellerne er montert med løpehjul som tauet roterer på etter hvert som kameramodulen flyttes rundt.

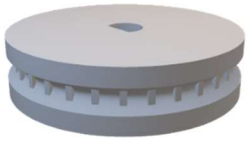
Versjon 1 hadde for smalt spor for at tauet kunne passe, dette ble derfor utvidet i versjon 2. I versjon 3 ble det lagt til riller for å få mer friksjon. Disse rillene ble endret til å gå helt ut i kanten på versjon 4. I denne versjonen ble også kantene gjort høyere for at tauet ikke skal dette ut så lett.



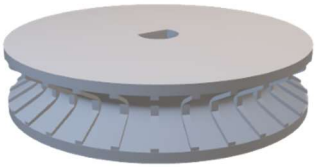
*Figur 24 - Løpehjul versjon 1*



*Figur 25 - Løpehjul versjon 2*



*Figur 26 - Løpehjul versjon 3*



*Figur 27 - Løpehjul versjon 4*

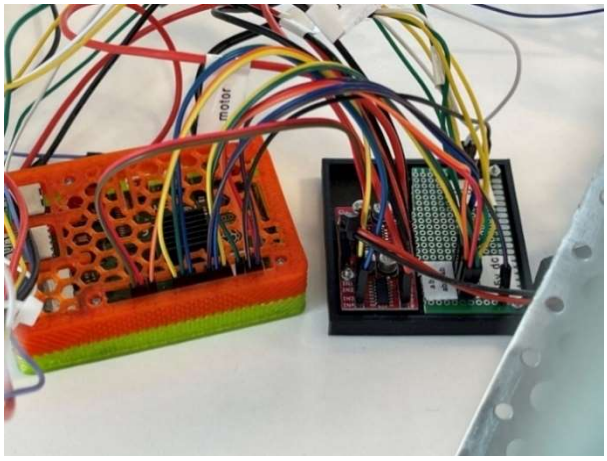
### 4.3.3 Ferdig løsning

Under er det bilder av den sammensatte løsningen for prototypen.

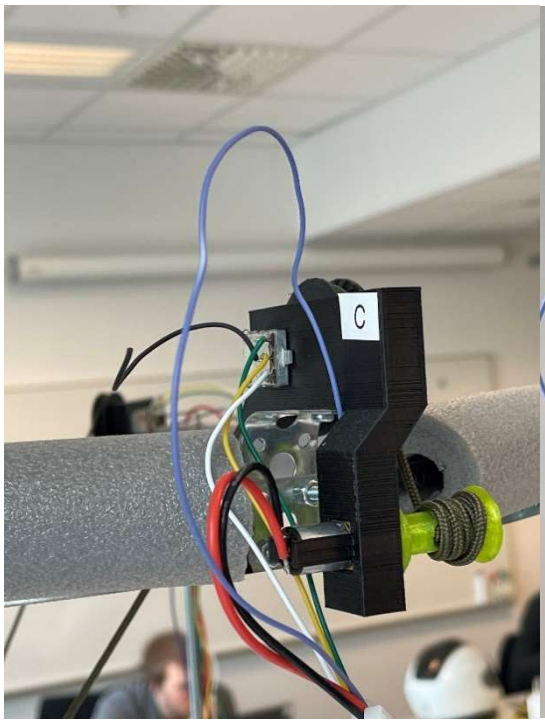
Koblingsskjema for RPi-en er dokumentert i «variables.py» filen i kildekoden.

I denne filen kan man også endre en rekke variabler for størrelse av merden og sensorhjul og IP/port som skal brukes for tilkobling til server.

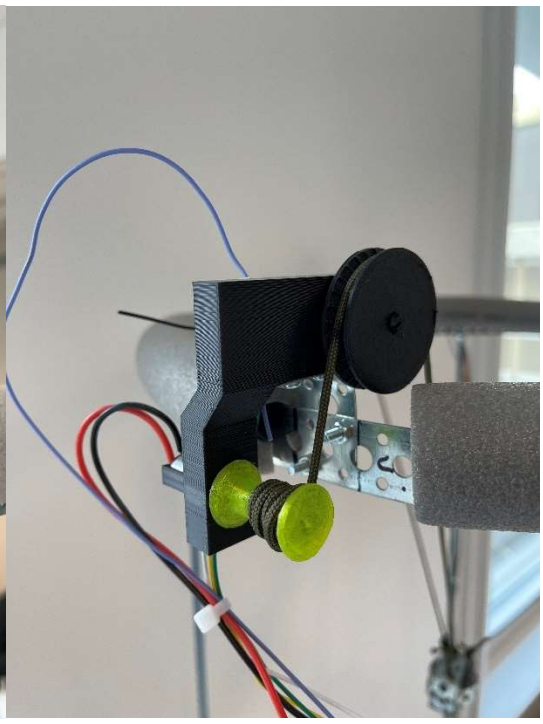
I tillegg kan man endre hastigheter på motorene i begge retninger. Dette fordi kamera modulen også får hjelp av tyngdekraften når den senkes ned.



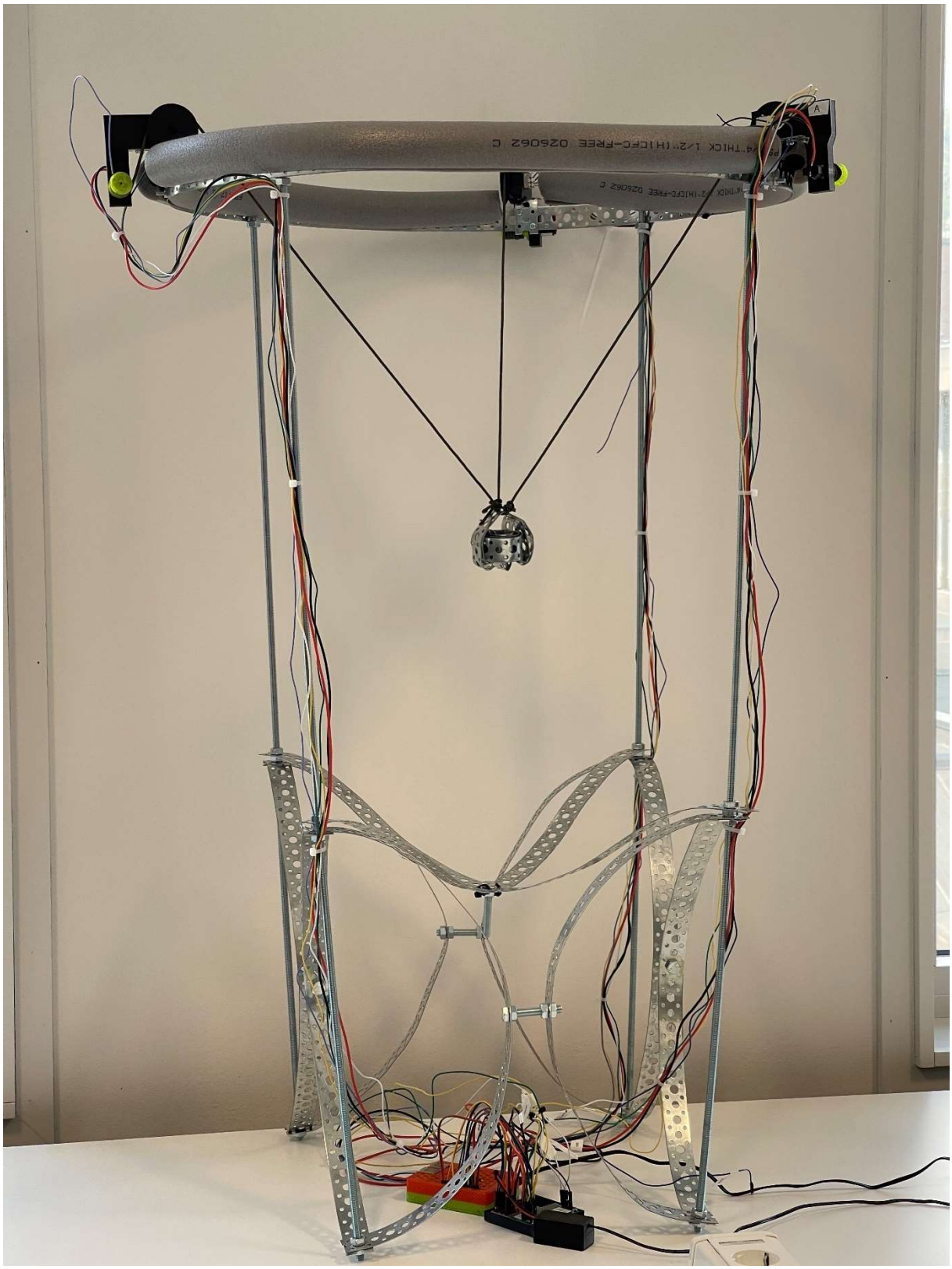
*Figur 28 - Oppkoblet Raspberry pi, motorkontroller og pullup kort*




*Figur 29 - Ferdig montert brakett fra venstre*



*Figur 30 - Ferdig montert brakett fra høyre*



***Figur 31 - Ferdig montert prototype***

## 4.4 Funksjon av Merd prototypen

### 4.4.1 Motorstyring/Invers-kinematikk

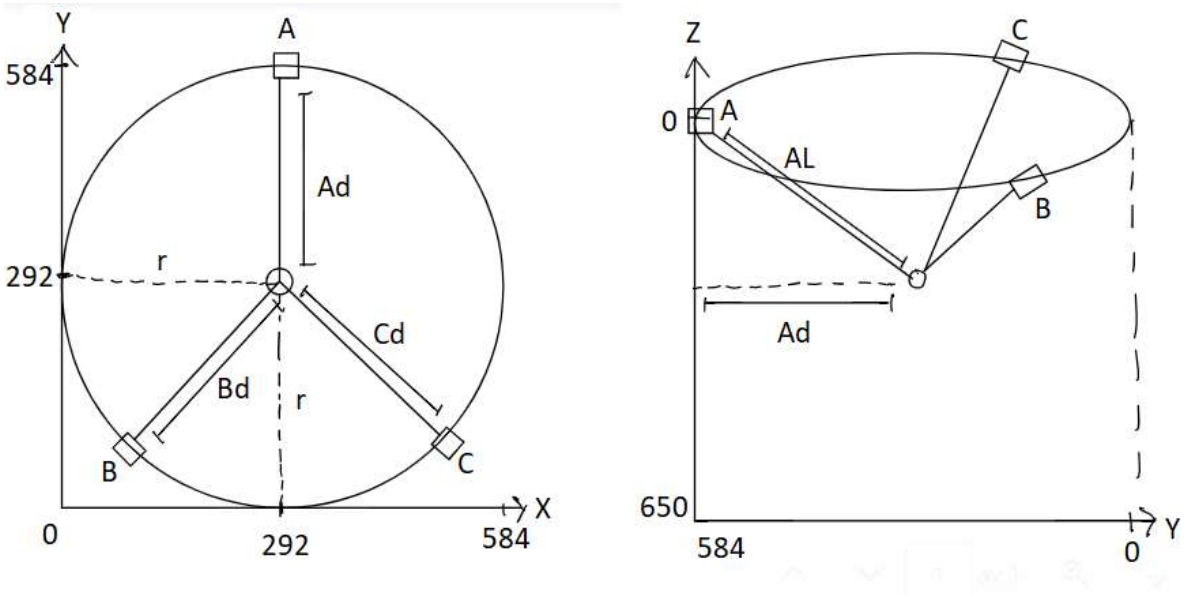
Motorkontrollene er koblet til RPi-en med 2 ledninger. For å styre motoren ene retningen sender man høy på den ene og lav på den andre. Om man reverserer hvilken pinne som er høy/lav vil motoren skifte retning. Lav/lav vil gjøre at motoren står stille. Det benyttes også PWM til å styre hastigheten på motorene.

For å måle hvor langt tauet blir trukket ut og inn er det satt et løpehjul på sensoren som tauet beveger seg over. Løpehjulet har en omkrets på 125mm, og vi vet derfor at den har beveget seg 5,21 mm når det blir mottatt en puls fra sensoren.

Med denne løsningen kan man ikke ta i bruk x, y og z koordinater direkte. Disse må i steder omregnes til lengder av tau på de 3 forskjellige spolene. Da må man ta i bruk «Inverse Kinematics».

Her var det mulig å omgjøre koordinatene ved å bruke Pytagoras setning 2 ganger.

Først i x, y planet, hvor resultatet fra dette settes inn sammen med z koordinaten.




*Figur 32 - Illustrasjon av invers kinematikk*

$A = (r, 2r)$

$B = (r - \frac{\sqrt{3}}{2}r, \frac{r}{2})$

$C = (r + \frac{\sqrt{3}}{2}r, \frac{r}{2})$

$Ad = \sqrt{(x - A_x)^2 + (y - A_Y)^2}$

$AL = \sqrt{Ad^2 + z^2}$

$Bd = \sqrt{(x - B_x)^2 + (y - B_Y)^2}$

$BL = \sqrt{Bd^2 + z^2}$

$Cd = \sqrt{(x - C_x)^2 + (y - C_Y)^2}$

$CL = \sqrt{Cd^2 + z^2}$

Ved å sette inn koordinatene kameraet skal bevege seg til og koordinatene til spolene vil man få ut lengden som tauet må ha for å bevege seg til den gitte posisjonen.

Koden som styrer motorene, er delt opp i flere deler. Den første delen gjør om (x, y, z) koordinater til taulengder som de tre spolene må ha for at kameraet skal være i riktig posisjon.

Denne taulengden blir deretter sendt videre til koden som henter data fra sensorene og styrer motorene ut ifra dette. Denne koden kjøres i hver sin tråd for å sikre at den mottar all dataene fra de forskjellige sensorene samtidig. For å registrere pulsene fra sensorene brukes også «event handlere».

For å slippe å gå mye inn på lavnivå delen av inn og utganger er biblioteket «gpiozero» tatt i bruk. [17]

## 4.5 GUI Prototypen

### 4.5.1 Valg av GUI-løsning

Da det kom til valg av GUI-løsning hadde vi flere muligheter når det gjaldt hvilke språk eller utviklingsverktøy vi kunne tenke oss å jobbe med. Alternativene stod mellom å programmere i Python eller C#. Begge har ulike fordeler og ulemper som var med på å bestemme hva vi skulle går for. Alle på gruppen har tidligere erfaring med C# fra flere emner i bachelorutdanningen, slik at flere på gruppen føler seg mest komfortabel med å bruke dette språket. Derimot ville det være enklere og samtidig mer oversiktlig å jobbe med Python da dette er språket vi bruker for å programmere RPi-en. Begge språk er objekt-orienterte programmeringsspråk, som tilsier at det er store likheter i måten en programmerer på. Som en liten ekstra utfordring og mulighet til å lære flere programmeringsspråk, gikk derfor valget på å bruke Python også ved utvikling av GUI-løsningen. Når det gjelder utviklingsverktøy har vi også her valgt å bruke PyCharm.

### 4.5.2 Tkinter

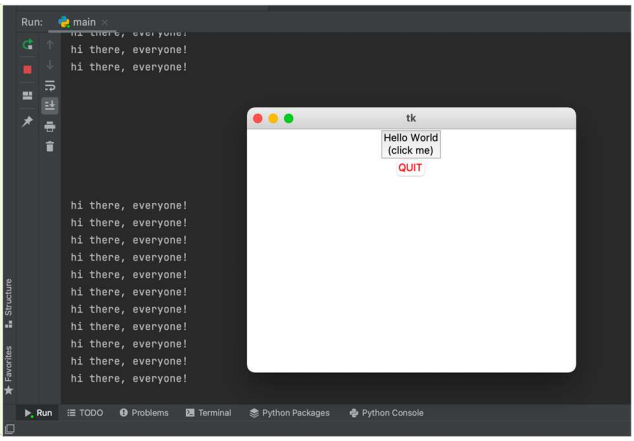

*Figur 33 - Tkinter logo [18]*

Tkinter er Python sin «standard» GUI-pakke og er kompatibel med de fleste Unix plattformer, samt Windows [19]. Tkinter er ikke den eneste GUI-programmeringspakken for Python, men det er den mest brukte. Siden Tkinter er så mye brukt når det kommer til å programmere GUI-er i Python, finnes det utallige dokumenter og videoer tilgjengelige som var til stor hjelp da gruppens medlemmer i starten skulle lære seg det grunnleggende.

Tkinter har flere sterke sider som var til hjelp da gruppen skulle bestemme seg for hvilken GUI-pakke som skulle bli brukt. Den er kryssplattform som tilsier at samme kode vil fungere på Windows, MacOS og Linux. Dette har fungert godt for vår gruppe da det gjør det mulig for alle å jobbe på samme kode til selv om noen bruker forskjellig operativsystem. Utseende er derimot tilpasset til hvilke operativsystem som brukes, slik at programmet ser ut som et Windows-program på Windows, og samtidig ser ut som et Mac-program når samme kode kjøres på MacOS [20].

Selv om Tkinter har mange gode sider, har den også noen mindre gode sider. Den største er kanskje at program som er bygget ved hjelp av Tkinter ofte ser utdaterte ut. Dette har dog ikke vært noe problem for gruppen da prioriteringen har vært å bygge et fungerende og robust program som er kryssplattform.

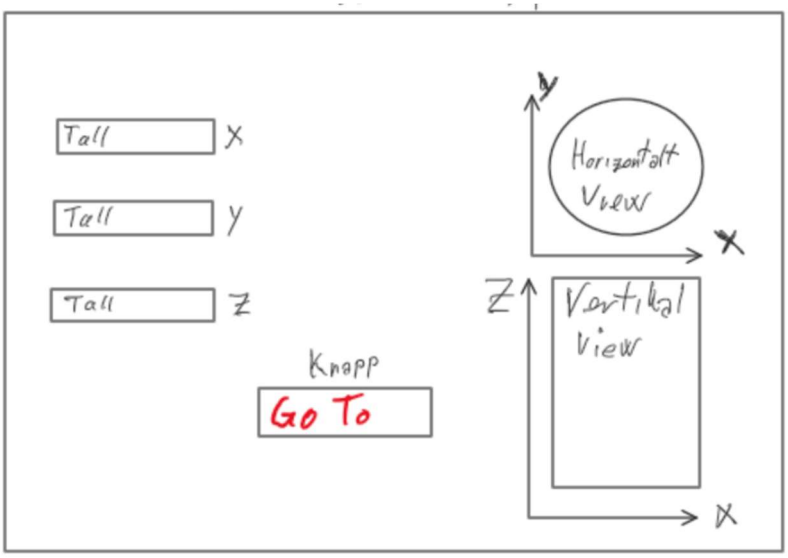
```
import tkinter as tk

class Application(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.master = master
        self.pack()
        self.create_widgets()

    def create_widgets(self):
        self.hi_there = tk.Button(self)
        self.hi_there["text"] = "Hello World\n(click me)"
        self.hi_there["command"] = self.say_hi
        self.hi_there.pack(side="top")

        self.quit = tk.Button(self, text="QUIT", fg="red",
                              command=self.master.destroy)
        self.quit.pack(side="bottom")

    def say_hi(self):
        print("hi there, everyone!")

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```




*Figur 34 - Enkelt eksempel på kode som bruker Tkinter [19], og hvordan det ser ut når programmet kjører*

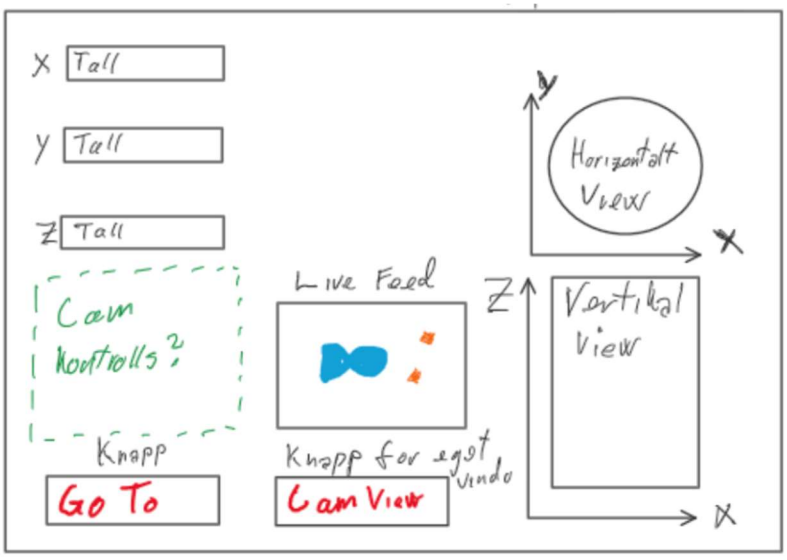
### 4.5.3 Design og utforming

GUI applikasjonen og dens design og utforming er en prosess som har vedvart siden dag én. Gruppen gikk sammen for å skissere et tidlig utkast av hvordan det var tenkt at GUI-en ville se ut, samtidig som den hadde de funksjonene som var planlagt. Første utkast med de mest grunnleggende funksjonene, som mulighet til å legge inn x-, y- og z-koordinater, samt visuell presentasjon av fiskemerden fra et horisontalt og vertikalt standpunkt, så slik ut:



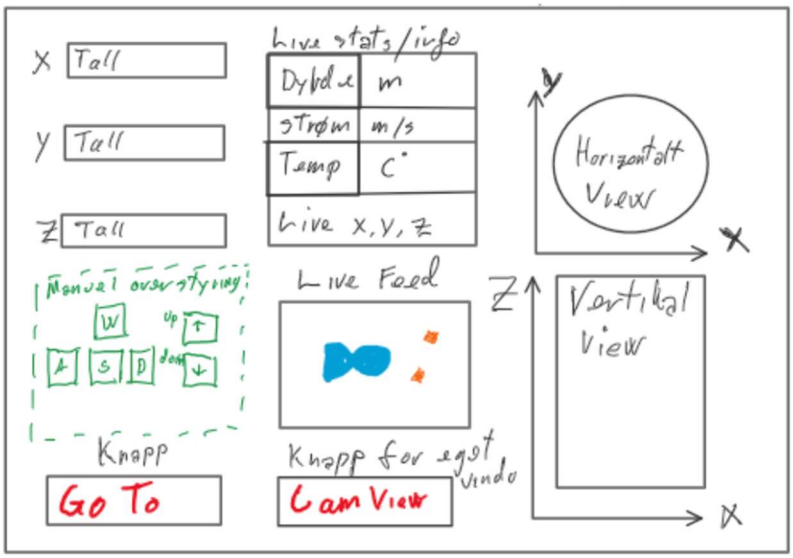

*Figur 35 - Første utkast av GUI-skissen*

Ettersom det ble bestemt at GUI-en skulle ha flere funksjoner, som mulighet for å få direktesending fra kameraet og eventuelt manuell styring med tastetrykk, samtidig med forsøk på å fylle eventuelle tomrom, ble neste utkast seende slik ut:




*Figur 36 - Andre utkast av GUI-skissen*

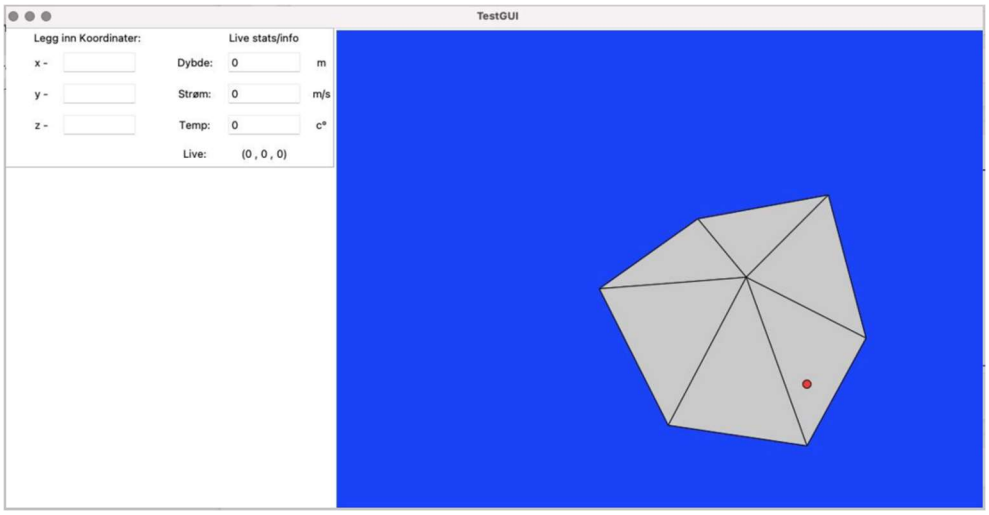
Deretter ble det bestemt at den også skulle ha mulighet for å kunne lese og overvåke dybden på kameraet, strøm og temperatur i vannet, samt den nåværende plasseringen til kameraet, ble tredje utkast seende slik ut:




***Figur 37 - Tredje utkast av GUI-skissen***

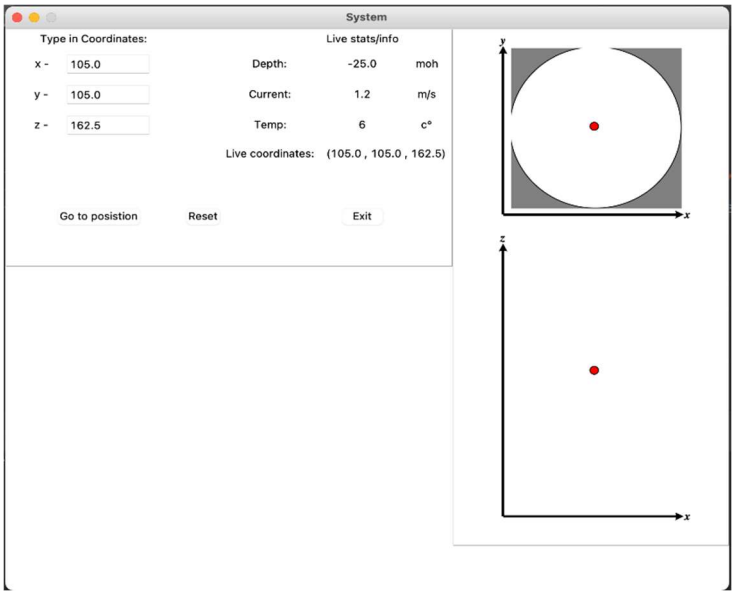
Det tredje og siste utkastet var det som ble brukt som mal da arbeidet satte i gang med å programmere og designe GUI-en. Gjennom hele prosessen ble det gjort mange endringer på utseende og funksjon da det ble bestemt å fjerne en del funksjoner som gruppen følte var unødvendige eller ble for mye arbeid.

På et tidspunkt ønsket gruppen å implementere en interaktiv tre-dimensjonal representasjon av fiskemerden, slik at det skulle være mer intuitivt for operatøren å kunne flytte rundt på kameraet og plassere det der det måtte ønskes. Til dette ble det brukt et ferdiglaget 3-D «engine» program som grunnlag, som var satt sammen med den GUI-løsningen vi hadde på det tidspunktet [10]. Dette viste seg å være alt for omfattende, da mangel på kunnskap og erfaring med tre-dimensjonal modellering, programmering og design gjorde at vi brukte mye tid på lite resultat. Dette alternativet ble så skrotet, men ikke uten at gruppen tok med seg verdifull erfaring, som videre var med på å forme sluttproduktet. Spesielt muligheten for operatøren å kunne flytte kamera med datamusen var noe gruppen fortsatt ville prioritere.




*Figur 38 - Forslag på tre-dimensjonalt brukergrensesnitt*

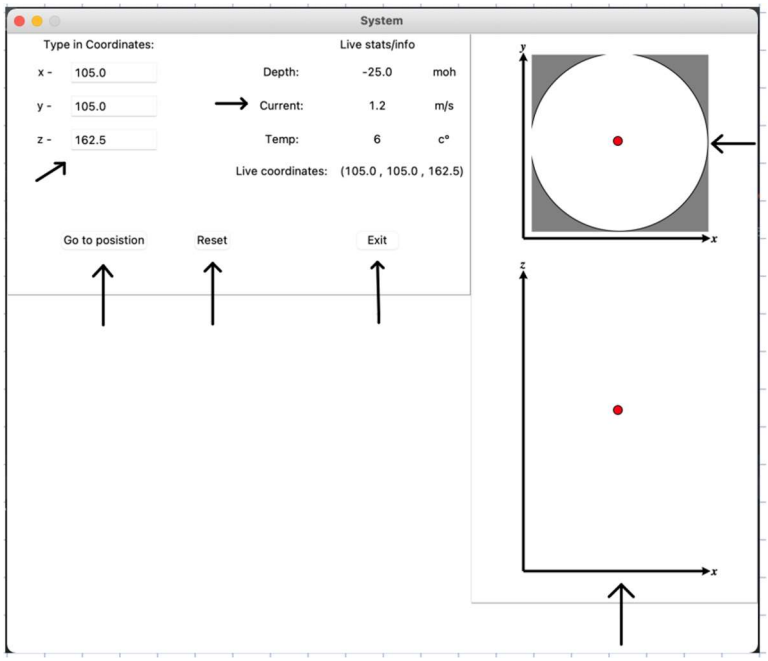
Videre gikk gruppen tilbake til den originale to-dimensjonale representasjonen av fiskemerden, bare at nå var det mulig å kunne flytte kamera rundt på grafikken som en vil, samtidig som man har mulighet til å skrive inn ønskede koordinater.




*Figur 39 - Endelig utgave av GUI*

### 4.5.4 Funksjon

I denne delen av rapporten er det gitt et forsøk på å forklare hvordan selve applikasjonen fungerer, og hvordan de ulike delene henger sammen. Pilene på figuren under representerer de ulike funksjonene til applikasjonen.




*Figur 40 - Forklaring av GUI*

I oppføringsboksene under «Type in Coordinates» er det mulig for bruker å skrive inn ønskede koordinater i x-, y-, og z-retning. Disse verdiene blir også oppdatert dersom man flytter på de røde ballene på grafikken til høyre i bildet.

Under «Live stats/info» kan bruker få oversikt over informasjon fra ulike sensorer i fiskemerden. Dette er sensorer som gir informasjon om dybden til kameramodulen, styrken på undervannsstrømmene og temperaturen i vannet. Dette er per nå ikke reelle verdier, da vi ikke har fått muligheten til å sette opp eller teste sensorer i et miljø som lar oss utføre disse testene. Dette er noe som enkelt vil kunne gjennomføres på et senere tidspunkt.
Nederst finner vi koordinatene til kameraet i sanntid, representert som (x, y, z).

«Go to position» er en knapp som tar koordinatene fra oppføringsboksene og sender dem til RPi-en, som da flytter kameramodulen til ønsket posisjon.

«Reset» er en knapp som gjør det mulig å kjapt kunne flytte kameramodulen til startposisjon som vi her har satt til å være i midten av fiskemerden. Denne har vi nyttet en del under testing av prototypen for å unngå at den kjører seg fast eller kommer for langt utenfor ønsket område.

«Exit» er som navnet tilsier ganske enkelt en knapp for å kunne avslutte programmet på riktig måte. Denne sørger for å avslutte tilkoblingen mellom GUI programmet og programmet som kjører på mikrokontrollen før GUI programmet avsluttes.

Oppe i høyre hjørne i applikasjonen er en grafikk over fiskemerden med et «ovenfra og ned» perspektiv. Det røde punktet representerer plasseringen til kameraet i x- og y-retning. Denne grafikken er interaktiv, som tilsier at bruker har mulighet til å bruke datamusen til å flytte rundt på punktet for å velge plasseringen til kameramodulen. Plasseringen er linket til oppføringsboksene slik at bruker også har oversikt over hvilke koordinater som tilhører den posisjonen punktet er i.

Nede i høyre hjørne i applikasjonen er en grafikk som har mye til felles med grafikken over, bare at vi denne gangen ser fiskemerden fra siden. Det røde punktet representerer plasseringen til kameraet i x- og z-retning. x-retningen er linket til x-retningen i grafikken over, og bruker har kun mulighet til å flytte det røde punktet i z-retning. z-retningen representerer her dybden kameramodulen er i vannet.

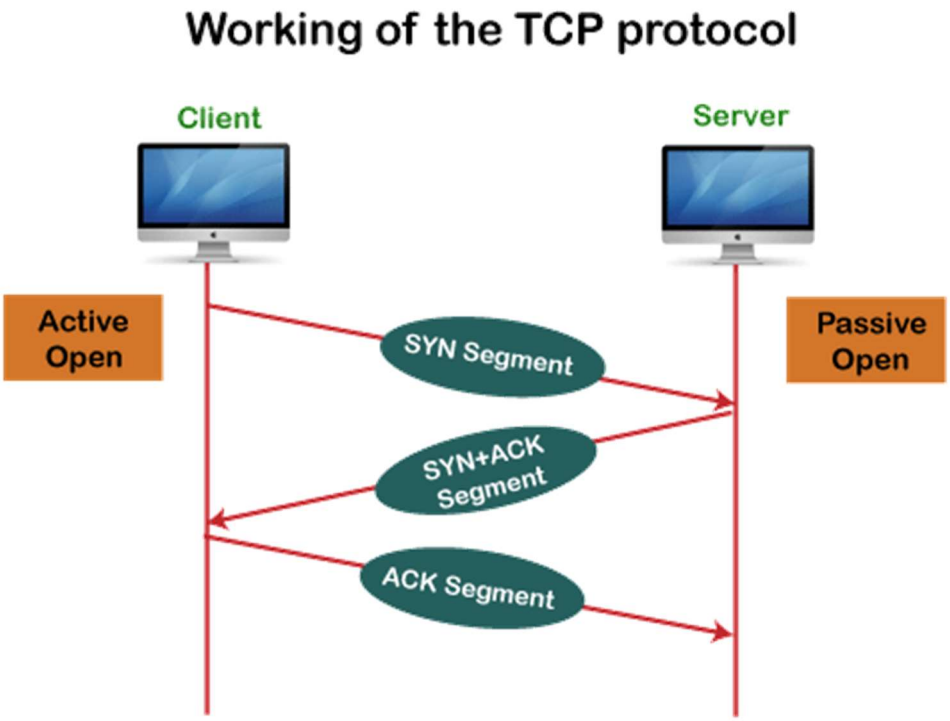
Tomrommet nederst til venstre i programmet var forbeholdt muligheten til å sette opp en direktestrøm fra kameraet. Dette var noe som gruppen ville arbeide med til slutt dersom det lot seg gjennomføre tidsmessig.

## 4.6 Server/Klient

### 4.6.1 Valg av løsning

For å kommunisere mellom RPi og PC-en som kjører grensesnittet for å kontrollere merdkameraet, så er det valgt å kode sockets for å overføre informasjon. Løsningen er valgt ettersom det er en dominerende måte å kommunisere over nettverk, med mye ressurser allerede tilgjengelig. [21]

Når socketen opprettes på server-siden, så kjøres denne i en egen tråd. Dette er en måte for serveren å holde seg aktiv og lytte etter innkommende forbindelser for å opprette socketen. Da kan man overføre informasjon uten at den delen av koden som vises for brukeren, og håndterer inndata, blir uresponsiv.




*Figur 41: Illustrasjon av TCP-Handshake [22]*

Valget for å overføre informasjon falt på Transmission Control Protocol (TCP). TCP er en sikker måte å overføre pakker over nettverk uten pakketap. [23] Det sies mer om dette i Kap. 4.6.3.

Som illustrert i figuren over, så fungerer dette med en handshake mellom server og klient. At klienten er «Active Open» vil si at den sender forespørsel om å opprette forbindelse på kommando. At serveren er «Passive Open» vil si at den alltid lytter etter forespørsler fra mulige klienter.

### 4.6.2 Posisjonsobjekt

Videre vil begrepet «posisjonsobjekt» bli brukt jevnlig. Måten operatøren sender en instruks på er ved å legge koordinater inn i et objekt kalt «Position», og overføre dette til RPi-en.

Det er posisjons-objektet som blir tolket av RPi-en, og gjør dette om til verdier som motorene kan bruke for å bevege seg.

Objekt er et navn som brukes for klasser. Når man koder en klasse, så oppretter man et nytt objekt, og disse objektene kan ha definerte egenskaper og verdier. [24]

```
class Position:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def __call__(self):
        return self
```

*Figur 42: Utklipp av kode for posisjonsobjekt*

### 4.6.3 Sockets

Socket-kommunikasjonen bruker Transmission Control Protocol (TCP). RPi-en er satt som klient, som skal ta imot et Posisjons-objekt fra serveren. Serveren kjører en socket på en egen tråd, som daemon(dette sørger for at prosessen kan avsluttes uten å vente på at en løkke skal avsluttes). [25]

TCP benyttes over User Datagram Protocol(UDP) fordi det er kritisk at den serialiserte bytestrømmen som inneholder Posisjons-objektet ikke har pakketap. [26] Manglende overføring av informasjon her kan få konsekvenser for bevegelsen av motoren.

### 4.6.4 Klient

RPi-en kjører en klient på systemet sitt som sender forespørsel på port 50007 til serveren for å opprette en forbindelse. [27] [28] Når forbindelsen er opprettet, så vil klienten være klar til å ta imot posisjons-objekt fra serveren for å styre motoren.

Klienten vil etter en timeout sende en ny tilkoblings-forespørsel til serveren, og vil med dette alltid ha en forbindelse hvis det er en server som lytter.

```
while True:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(20)
    try:
        s.connect((HOST, PORT))
        print("Connected")
        data = b""
        while True:
            packet = s.recv(4096)
            if not packet:
                break
            data += packet
            p = dill.loads(data)
            pos = Position(p.x, p.y, p.z)
            position = [p.x, p.y, p.z]
            m.move_to_position(position)
            print("Camera position: [" + str(p.x) + ", " + str(p.y) + ", " + str(p.z) + "]")
            data = b""
```

*Figur 43: Utklipp av Klientkode*

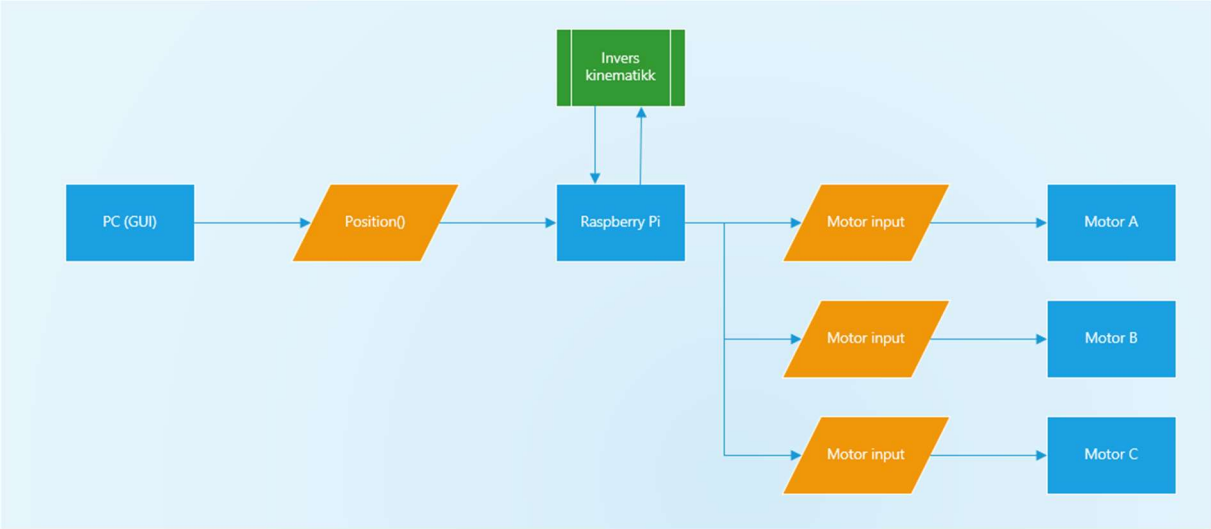
### 4.6.5 Server

På PC-en, som fungerer som et operatørpanel, kjører serveren. Serveren lytter etter innkommende forbindelser, og vil koble seg til når den får en forespørsel på port 50007. Det er serveren som sender instruksene til RPi-en i form av posisjons-objektet. Serveren starter i sin egen tråd når programmet på PC-en kjøres.

```
def startserver(self):
    host = ""
    port = 50007

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((host, port))
    print("socket binded to port.")

    s.listen(5)
    print("Socket is listening.")
    while True:
        # establish connection with client
        self.conn, self.addr = s.accept()
        print('Connected to :', self.addr[0], ':', self.addr[1])
    # This is the function which sends the position over the socket.
```

*Figur 44: Utklipp av serverkode*

### 4.6.6 Funksjon




*Figur 45: Blokkskjema av programfunksjon*

1. PC-en er i drift, og denne kjører serveren. Serveren vil alltid lytte på port 50007 når programmet kjører.

2. RPi-en startes. Her kjøres klienten. Klienten forespør å koble seg til port 50007, med serveren sin IP.

3. Det opprettes forbindelse, og posisjons-objekt for å styre motorene kan sendes over denne forbindelsen.

4. Timeout på klient på 20 sekunder sørger for at klienten ikke oversvømmer forbindelsen med forespørsler.

5. Når det gis kommando fra GUI-en, serialiseres et posisjons-objekt, og sendes til RPi-en. Dette posisjonsobjektet vil også oppdatere posisjonen i GUI-en med det samme.

6. Posisjonen mottas av klienten på RPi-en, og blir deserialisert til et posisjonsobjekt.

7. Posisjonsobjektet blir satt inn i en metode som gjør objektet om til koordinater og til slutt verdier som motorene kan tolke.

# 5 Testing

## 5.1 GUI testing

Test av GUI er en del av en naturlig prosess som foregår samtidig med utviklingen av programmet. For hver ny funksjon som blir lagt til er det naturlig å teste denne før en går videre. Funksjoner som at bruker ikke skal kunne skrive noe annet enn koordinater i oppføringsboksene, eller at bruker kan flytte de røde punktene utenfor grafikken, er noen eksempler på tester som er utført samtidig med utviklingen.

Da gruppen følte at GUI-en hadde de viktigste funksjonene på plass, var det tid for å teste de opp mot prototypen.

## 5.2 Beskrivelse av Prototype testing

For at produktet skal være robust nok til å kunne tas i bruk, så er det kritisk at det går gjennom en testfase.

### 5.2.1 Test av samkjøring mellom GUI, RPi og merd-protype

Testing av den sammensatte protypen har vært viktig for dette prosjektet. Målet ved testene vi har utviklet er ment til presentere hva denne prototypen er i stand til å gjøre og dermed kunne vurdere om det vi har utviklet fra ide til prototype faktisk fungerer.

Det er en rekke aspekter som med denne prototypen som dikterer dens oppførsel. Vi har valgt å fokusere på motorstyringens PWM signal, det fordi at å kalibrere det med riktig PWM signal er kritisk for å motorene skal kunne heve og senke på modulen på en kontrollert måte. Andre aspekter med prototypen som dens fysiske størrelse, motorplassering, taulenge osv. representerer vesentlige endringer for denne prototypen noe som tiden ikke strekker til for dette prosjektet.

### 5.2.2 Test av nettverkskommunikasjon

Når koden for nettverkskommunikasjonen var kommet på plass, så var det en del problem som presenterte seg når det skulle testes. I underkapitlene beskrives problemene og løsningene som ble utarbeidet i etterkant.

Etter forbedringer i server/klient-koden, så er det oppnådd en robusthet som skal kunne la forbindelsen opprettholdes uten avbrudd, gitt at serveren og klienten faktisk kjøres.

#### *5.2.2.1 Klient*

Hovedproblemet med klienten var en mulig situasjon hvor RPi-en skulle starte på nytt eller miste forbindelsen. I utgangspunktet hadde den ikke noe kode som ba klienten koble seg til serveren igjen. Dette ville resultert i et utfall hvor operatører ville vært nødt til å gå ut til merden for å manuelt starte oppkoblingen.

Dette testet vi ved å starte RPi-en på nytt og terminere tilkoblingen manuelt. Resultatet ble at tilkoblingen ble lukket uten å bli gjenopprettet.

Som konsekvens av dette så skrev vi kode som definerte en «timeout» på 20 sekund. Etter dette vil så klienten prøve å koble seg til serveren igjen. Dette gjør den med intervaller på 1 sekund frem til tilkoblingen blir opprettet.

Et annet aspekt dette løste var at en tidligere løsning ville kontinuerlig prøve å koble seg til serveren, men dette gjorde at det ble stor trafikk på forbindelsen, i størrelsesorden Megabytes/sekund.

#### *5.2.2.2 Server*

Servertestingen var mindre omfattende. En del av serveren sin funksjon er å lytte etter innkommende forbindelser. Siden dette skjer passivt, så er det ikke noe kode som serveren aktivt må kjøre.

Det som dukket opp her var behovet for å sørge for at serveren kunne kjøre uavhengig parallelt med GUI-en. Tidlige tester gjorde at GUI-en ikke var responsiv når serveren kjørte. Dette løste vi ved å kjøre serveren i egen tråd i programmet.

### 5.2.3 Test I, Heving, senking av modul og kalibrering av PWM-signal

#### *5.2.3.1 Prinsipp*

Den først delen av testen omhandlet kalibrering av prototypen sitt PWM signal. PWM signalet er inndelt i en opp og ned hastighet. En god kalibrering vil være at motorene har nok hastighet og kraft til å løfte modulen til ønskede koordinater, men samtidig en lav nok hastighet slik at modulen kan senkes kontrollert til ønskede koordinater.

#### *5.2.3.2 Metode*

For å se om vi har oppnådd en god kalibrering av PWM-signalet vil vi teste prototypens oppførsel når motorene løfter modulen rett opp fra en gitt koordinat f.eks. (x = 292, y = 292, z = 650) til (292, 292, 100) og tilbake til (292, 292, 650). Dette representer at modulen er helt nede i senter av prototypen. Modulen blir så hevet til topp og går tilbake til bunnen. Dette vil kunne gi oss en visuell representasjon på om motorene fortsatt er synkronisert eller om avvik har forekommet.

#### *5.2.3.3 Resultat*

Tabellen under viser resultat av denne testen

| | X | Y | Z | OK/IKKE OK | PWM opp | PWM ned | Beskrivelse |
|---|---|---|---|---|---|---|---|
| Test 1 | 292 | 292 | 650 | OK | 0.15 | 0.08 | Start punkt |
| Test 2 | 292 | 392 | 100 | IKKE OK | 0.15 | 0.08 | Problemer med å nå målet |
| Test 3 | 292 | 292 | 650 | IKKE OK | 0.18 | 0.08 | Usynkron |
| Test 4 | 292 | 292 | 100 | OK | 0.18 | 0.1 | |
| Test 5 | 292 | 292 | 650 | IKKE OK | 0.18 | 0.1 | Usynkron |
| Test 6 | 292 | 292 | 100 | IKKE OK | 0.17 | 0.1 | Usynkron |
| Test 7 | 292 | 292 | 650 | IKKE OK | 0.17 | 0.1 | Usynkron |
| Test 8 | 292 | 292 | 100 | OK | 0.16 | 0.1 | |
| Test 9 | 292 | 292 | 650 | IKKE OK | 0.16 | 0.1 | Usynkron |
| Test 10 | 292 | 292 | 100 | OK | 0.16 | 0.09 | |
| Test 11 | 292 | 292 | 650 | IKKE OK | 0.16 | 0.09 | Strammet inn |
| Test 12 | 292 | 292 | 100 | OK | 0.16 | 0.09 | |
| Test 13 | 292 | 292 | 650 | IKKE OK | 0.16 | 0.09 | Sklidd på motor-c sensor |
| Test 14 | 292 | 292 | 100 | OK | 0.16 | 0.09 | |
| Test 15 | 292 | 292 | 650 | IKKE OK | 0.16 | 0.09 | Usynkron |

***Figur 46 Tabell av resultat av Test I***

Vi valgte å dele dette resultatet opp for skille mellom når modulen løftes opp og når modulen senkes ned. Dette er fordi det var et klart skille mellom protypens faktiske oppførsel og det som var ønsket resultat.

| | X | Y | Z | OK/IKKE OK | PWM opp | Beskrivelse |
|---|---|---|---|---|---|---|
| Test 1 | 292 | 292 | 650 | | 0.15 | Start punkt |
| Test 2 | 292 | 392 | 100 | IKKE OK | 0.15 | Problemer med å nå målet |
| Test 4 | 292 | 292 | 100 | OK | 0.18 | Nådde målet men går for fort |
| Test 6 | 292 | 292 | 100 | IKKE OK | 0.17 | Usynkron |
| Test 8 | 292 | 292 | 100 | OK | 0.16 | |
| Test 10 | 292 | 292 | 100 | OK | 0.16 | |
| Test 12 | 292 | 292 | 100 | OK | 0.16 | |
| Test 14 | 292 | 292 | 100 | OK | 0.16 | |

***Figur 47 Utsnitt fra figur 46, heving av modul***

Tabellen over er resultat av når motoren løfter modulen fra senterbunnen (292, 292, 650) av prototypen til sentertopp (292, 292, 100) med ulik PWM hastigheter. Resultatet av testen viste at en PWM hastighet på 0.16 opp er ganske så ideelt for denne prototypen.

| | X | Y | Z | OK/IKKE OK | PWM ned | Beskrivelse |
|---|---|---|---|---|---|---|
| Test 1 | 292 | 292 | 650 | | 0.08 | Start punkt |
| Test 3 | 292 | 292 | 650 | IKKE OK | 0.08 | Usynkron |
| Test 5 | 292 | 292 | 650 | IKKE OK | 0.10 | Usynkron |
| Test 7 | 292 | 292 | 650 | IKKE OK | 0.10 | Usynkron |
| Test 9 | 292 | 292 | 650 | IKKE OK | 0.10 | Usynkron |
| Test 11 | 292 | 292 | 650 | IKKE OK | 0.09 | motorene feste er slakt |
| Test 13 | 292 | 292 | 650 | IKKE OK | 0.09 | Sklidd på motor-c sensor |
| Test 15 | 292 | 292 | 650 | IKKE OK | 0.09 | Usynkron |

***Figur 48 Utsnitt av figur 46, senking av modul***

Tabellen over er resultat av testen når motoren senker modulen fra sentertopp (292, 292, 100) av prototypen til senterbunnen (292, 292, 650) med ulik PWM hastigheter. Det ble oppdaget at når modulen skulle senkes var den veldig utsatt for å ikke være synkronisert lenger. I en senere testing kom det fram at vi hadde løftet modulen for høyt i denne delen av testen. Dermed ble det avgjort at PWM hastigheten på 0.09 ned fungerte ganske så bra.

### 5.2.4 Test II, Test av yttergrenser

#### *5.2.4.1 Prinsipp*

Den andre delen ut av kalibreringstesten var å finne ut av hva yttergrensene til hvor prototypen klarer å plassere modulen med gitte koordinater. Dette vil kunne gi oss en forståelse for prototypens operative område den kan flytte modulen uten at den utvikler uønsket oppførsel.

#### *5.2.4.2 Metode*

For å finne yttergrensene til protypen så bestemte vi oss for å ha modulen midt i prototypen (292, 292, 325), deretter kjøre den ut til det som er maks og minimum koordinater på en akse. Hvis prototypen gjør noe som er uønsket så vil vi justere koordinatene deretter for å oppnå ønsket oppførsel. Dette vil i så bli definert som en yttergrense for prototypens evne til å flytte modulen.

#### *5.2.4.3 Resultat*

Under ligger det en tabell av resultatet av denne testen:

| | X | Y | Z | OK/IKKE OK | Beskrivelse | | |
|---|---|---|---|---|---|---|---|
| Test 1 | 292 | 160 | 325 | | Tau i fare for hoppe av | | |
| Test 2 | 292 | 170 | 325 | | Tau slakt | | |
| Test 3 | 292 | 178 | 325 | | Tau slakt | | |
| Test 4 | 292 | 195 | 325 | OK | | | Yttergrense test Y-aksen |
| Test 5 | 292 | 389 | 325 | | Tau slakt | | |
| Test 6 | 292 | 170 | 325 | | Tau slakt | | |
| Test 7 | 292 | 473 | 325 | IKKE OK | Tau hoppet av | | |
| Test 8 | 292 | 292 | 325 | IKKE OK | Gikk ikke tilbake synkront | | |
| Test 9 | 389 | 292 | 325 | IKKE OK | Tau hoppet av | | |
| Test 10 | 381 | 292 | 325 | | | | |
| Test 11 | 197 | 292 | 325 | | | | |
| Test 12 | 167 | 292 | 325 | | | | Yttergrense test X-aksen |
| Test 13 | 153 | 292 | 325 | | | | |
| Test 14 | 125 | 292 | 325 | | | | |
| Test 15 | 195 | 195 | 325 | | | | |
| Test 16 | 195 | 292 | 325 | | | | |
| Test 17 | 292 | 455 | 325 | | | | Yttergrense test X og Y aksene |
| Test 18 | 292 | 381 | 325 | | | | |
| Test 19 | 381 | 381 | 325 | IKKE OK | Tau hoppet av | | |
| Test 20 | 375 | 375 | 325 | | | | |

***Figur 49 Tabell resultat av Test II***

Resultatet av denne del av testen gav en rekke resultater for prototypens oppførsel og dens evne til å flytte modulen rundt i prototypen. Det ble oppdaget at prototypens oppførsel er svært avhengig av koordinatene som er gitt for å flytte modulen. Det kom også fram at hvis modulen flyttes for langt vekk fra en motor vil tauet som går over sensorhjulet falle av og dermed ikke lenger kunne sette modulen til riktige koordinater. En yttergrense på minimum 195 og maksimum 375 i x, y planet viste seg å være innenfor toleransene til hvor modulen kan flyttes til.

Testen avslørte også hvordan plassering av motorene på prototypen påvirker dens evne til å flytte modulen rundt i prototypen. Hvis modulen kjøres mot det som blir rett under en av motorene til prototypen, uavhengig av hvilken verdi det vil ha på X- eller Y-aksen så har den et mye større operativt område enn om den ville ha prøvd å flytte modulen til en yttergrense mellom to motorer.

### 5.2.5 Test III, Test av oppførsel i topp og bunn

#### *5.2.5.1 Prinsipp*

Oppførselen til prototypen ved manøvrering av modulen i topp og bunn av prototypen sees på som et aspekt som krever sin egen del av testingen. Dette kommer av at protypens evne til å flytte modulen er avhengig av belasting på tauet som er festet i modulen. I topp vil tauene være stramme, og vi står i fare for at belastingen på en rekke komponenter vil gi oss uønsket oppførsel som følge av dette og ikke klare å flytte modulen til ønskede koordinater. Hvis modulen er i bunnen av prototypen risikerer vi at tauene er så slakke at de vil ikke registrere endringer i tauene sin lengde og dermed få uønsket oppførsel fra prototypen. Det å finne og etablere en yttergrense for Z-aksen til prototypen er kritisk for prototypens operasjonelle funksjonalitet.

#### *5.2.5.2 Metode*

Vi vil kjøre modulen i topp av prototypen, deretter vil vi gi koordinater som tilsvarer yttergrenser for x- og y-aksene for så dokumentere prototypens evne til å flytte modulen på den gitte høyden. Hvis den ikke klarer å flytte modulen til det som er ønskede koordinater regner vi testen som ikke ok.

#### *5.2.5.3 Resultat*

Under ligger det en tabell av resultatet av denne testen:

| | X | Y | Z | OK/IKKE OK | Beskrivelse | | |
|---|---|---|---|---|---|---|---|
| test 47 | 375 | 375 | 650 | IKKE OK | maks tau lengde | | |
| test 48 | 375 | 375 | 600 | IKKE OK | maks tau lengde | | Testing i bunnen av prototypen |
| test 49 | 375 | 375 | 550 | | | | |
| test 50 | 195 | 195 | 550 | OK | | | |
| test 51 | 195 | 195 | 100 | IKKE OK | Tau for stramt | | |
| test 52 | 195 | 195 | 150 | OK | | | |
| test 53 | 375 | 375 | 150 | | | | Testing i toppen av prototypen |
| test 54 | 375 | 195 | 150 | OK | | | |
| test 55 | 195 | 375 | 150 | OK | | | |

***Figur 50 Tabell av resultat av Test III***

Prototypens oppførsel og evne til å flytte rundt modulen i topp og bunn var som forventet vesentlig påvirket av hvor høyt eller lavt den er plassert i prototypen. Yttergrense til Z-aksen ble konkludert til være Z=150 som maksimumhøyde og Z=550 som minimumshøyden. Utenfor dette området så er det en rekke forskjellige aspekter med prototypen som gir uønsket oppførsel. Hvis en setter modulen lavere enn Z=550 så er det en fare for at tauet blir spolt helt ut. Videre hvis modulen manøvreres i området lavere enn Z=550 så er det også en risiko for at modulen kolliderer med deler av prototypens struktur. Hvis prototypen prøver å manøvrere modulen i området høyere enn Z=150, så vil det kunne oppstå problemer med at belastingen på tauet og prototypen overstiger motorens evne til å flytte eller holde modulen i ro på ønskede koordinater og plassering.

# 6 Diskusjon

## 6.1 Mål og resultat

Etter at vi nå er kommet i mål med et produkt, så har vi stilt oss noen spørsmål om resultatet. Vi har laget en løsning som har automatisert bevegelse av en gjenstand i en fiskemerd. Vi har oppnådd målet om å kunne taste inn koordinater eller visuelt bevege kameraet, som igjen overfører informasjonen om posisjonen til merden, og beveger motorene.

Denne prototypen fungerer som et springbrett for det som eventuelt kan brukes i et fullskalert anlegg. Det er ikke noe umiddelbar praktisk funksjon som kan overføres til faktiske produksjonsanlegg, men mye av lærdommen og prinsippene kan overføres når en vil skalere opp til et større anlegg.

Ut ifra testene som ble gjennomført ble det klargjort for oss at prototypen har en rekke svakheter med hvordan motorene er plassert og hvordan motoren flytter på kameramodellen. Men testene som et kalibreringsverktøy viste seg å gi gode resultater på hvor prototypen er i stand til å pålitelig flytte modulen innenfor rammene som ble satt som yttergrenser.

## 6.2 Forslag til Forbedring av GUI Løsning

Forbedringer når det kommer til valg av GUI-løsning, er det en del ting gruppen ville gjort annerledes i dag. Siden alle gruppemedlemmer har mer erfaring med å designe og programmere GUI-løsninger i C# enn i Python, er dette noe som hadde redusert tidsbruken noe, samt øke sannsynlighet for å levere en mer solid løsning. En annen ting hadde vært å bruke en annen GUI-pakke enn Tkinter, for å gi GUI-en et mer moderne utseende.

Andre forbedringer med GUI-en er å implementere de ulike funksjonene som gruppen satte som mål i starten. Funksjoner som å legge inn en direktestrøm fra kameraet i GUI-vinduet, hente inn data fra sensorer i fiskemerden for å vise til bruker, og lage en løsning for å kunne styre flere fiskemerder.

Når det kommer til muligheter for å bruke denne løsningen i et fullskalert anlegg, var planen å lage til en komplett oversikt over direktestrømmer fra kameraene i de ulike merdene. Bruker har da mulighet til å se direktestrøm fra alle kameraene samtidig, og mulighet til å klikke inn på de enkelte direktestrømmene for å åpne GUI-en slik at en derifra kan flytte kameraene til ønsket posisjon.

## 6.3 Forslag til forbedring Server/Klient løsning

Server og klient fungerer bra med dagens løsning. Det som kan tenkes å forbedre er å lage en løsning som tar høyde for flere klienter, eller merder, slik at serveren kan opprettholde flere forbindelser samtidig på forskjellige tråder. Dette vil si at hvis en implementerer kode som lager en ny tråd for flere innkommende forbindelser, så vil en kunne støtte styring av flere merder i samme anlegg.

Med en mer omfattende løsning så ville det være naturlig å implementere OPC-Unified Architecture(OPC-UA). [29] Dette er en protokoll utviklet for å bruke i sammenheng med servere/klienter, men gir en del tilleggsfunksjoner, og men for denne prototypen, så er dagens måte å gjøre det på tilstrekkelig. I en fullskala løsning, så ville vi tatt i bruk denne arkitekturen.

## 6.4 Forslag til forbedring av Prototypen

Under testingen ble det oppdaget at tauet kan hoppe ut av sensorhjulene om kameraet blir kjørt til enkelte posisjoner. Om kameramodulen blir kjørt til enkelte ytterposisjoner vil tauet også kunne bli slakt, noe som fører til at bevegelser ikke blir registrert av sensorhjulene.

Enkelte brå bevegelser gjør at tauet kan glippe på sensorhjulet.

Alle disse problemene kan forbedres ved å for eksempel sette på et ekstra motorisert hjul som presser tauet mot sensorhjulet. Dette vil minimere sjansen for at tauet glipper på grunn av brå bevegelser, i tillegg til at det sørger for at tauet blir ordentlig spolt ut selv om det er slakk på det.

Dersom hele festebraketten til spolen/sensorhjulet også kan rotere vil vinkelen mellom tauet og sensorhjulet bli mindre, og sjansen for at den hopper av vil minke.

# 7 Konklusjon

## 7.1 Rapport

Modellen vi har laget representerer samlet en prototype av et kamerastyringssystem til en generisk fiskemerd. Samt tilhørende GUI applikasjon som er ment som brukerutstyr for en operatør. GUI-applikasjonen er ment til være et interaktivt verktøy for en operatør til å flytte en kameramodul inni en fiskemerd med fokus på nøyaktighet og pålitelighet for å nå angitte koordinater. Begge prototypene kommuniser med hverandre ved hjelp av en server/klient løsning som oppretter tilkobling mellom GUI applikasjonen og merd prototypen. Denne oppkoblingen gjør det mulig å sende koordinater mellom GUI-en og prototypen for å så flytte modulen til gitte koordinater.

Samlet sett ser vi på dette prosjektet som et «Proof-of-Concept» og som et vellykket prosjekt. Det er også et prosjekt hvor det er rom for utbedringer. Det er en rekke utbedringer og tilleggsfunksjoner som kan løfte nivået til prototypen. Den mest konkrete forbedring som kan gjennomføres vil være en annen løsning eller utvidelse av motorene og anordning som motorene er festet på. Det ser vi vil øke prototypens ytelse og kapabilitet betraktelig. Andre forbedringer som å implementere OPC-UA protokollen, generelle forbedringer på GUI-applikasjonen og implementering av videostrømming er det vi ser på som gode muligheter for en annen gruppe til bygge videre på, i dette prosjektet.

## 7.2 Arbeidsløp

Ved begynnelsen av dette prosjekt hadde ingen av prosjektets gruppemedlemmer erfaring med å arbeide med ABB, oppdrettsanlegg eller røkting. Gjennom den innledende samtalen med ABB ble det forespeilet at de ville ha en ny løsning når det kom til automatisering av deres akvakultursystemer.

Derfor begynte dette prosjektet på ganske så bar bakke og i samtaler med ABB så ble det avgjort at denne oppgaven og dens prosjekt kom til å være ganske åpen, både i hva dette prosjektet skal være og hvordan gruppen velger å jobbe med den.

Gruppens medlemmer består utelukkende av Kommunikasjonssystem studenter, men med ulik bakgrunn fra før studietiden. Gjennom dette prosjektet har vi som gruppe og enkeltpersoner utviklet oss og fått en bredere og dypere erfaring fra det å jobbe med et prosjekt som har en rekke ulike aspekter og utfordringer.

ABB har gitt gruppen positive tilbakemeldinger rundt midtveis-evalueringen, men vi har for så vidt arbeidet helt autonomt og selvstendig. Det har ikke nødvendigvis vært helt optimalt, men i skrivende stund for denne rapporten har Covid-19 vært et vedvarende problem for dette prosjektet, gruppens muligheter til å møte fysisk med hverandre og ha løpende kontakt med ABB.

Gjennom dette prosjektet har gruppen hatt et så godt samarbeid som det lot seg gjøre med henhold til Covid-19 situasjonen som har vært et hinder for økt og tettere samarbeid. Pandemien har vært lite stabil og for å ivareta smittevern i tråd med myndighetene sine råd og anbefalinger har vi som gruppe vært på føre var prinsippet når kom til å ivareta hverandres helse.

Dette prosjektet og skriving av denne rapporten har gitt oss som gruppe en god og bred innsikt i det å bygge opp et prosjekt fra bunnen og dette er erfaring vi kommer til å ta med oss videre.

# 8 Referanser


[1] ABB Ltd., «ABB,» [Internett]. Available: https://new.abb.com/no/om-oss/var-virksomhet. [Funnet 5 Juni 2021].

[2] Imenco, «Imenco,» 6 Juni 2021. [Internett]. Available: https://imenco.no/nb/product/komplett-kamerasystem.

[3] Python Software Foundation, «Python logo,» [Internett]. Available: https://www.python.org/community/logos/. [Funnet 06 06 2021].

[4] G. v. Rossum, «Python,» Python Software Foundation, Mai 1996. [Internett]. Available: https://www.python.org/doc/essays/foreword/. [Funnet 7 Juni 2021].

[5] Python Software Foundation, «About Python,» [Internett]. Available: https://www.python.org/about/. [Funnet 07 Juni 2021].

[6] Jetbrains, «Jetbrains,» [Internett]. Available: https://www.jetbrains.com/company/brand/logos/. [Funnet 06 06 2021].

[7] JetBrains s.r.o, «PyCharm Features,» JetBrains, [Internett]. Available: https://www.jetbrains.com/pycharm/features/. [Funnet 7 Juni 2021].

[8] Raspberry Pi, «Raspberry Pi,» [Internett]. Available: https://www.raspberrypi.org/trademark-rules/ . [Funnet 06 06 2021].

[9] Raspberry Pi Foundation, «About Rasberry Pi,» [Internett]. Available: https://www.raspberrypi.org/about/. [Funnet 2021 Juni 2021].

[10] H. Haefliger, «Github,» 23 Februar 2021. [Internett]. Available: https://github.com/hnhaefliger/pyEngine3D. [Funnet 27 Mars 2021].

[11] Autodesk, «Autodesk,» [Internett]. Available: https://www.autodesk.com/products/fusion-360/blog/fusion-360-extensions/fusion-logo/ . [Funnet 06 06 2021].

[12] Microsoft Teams, «Microsoft,» [Internett]. Available: https://www.microsoft.com/nb-no/microsoft-teams/log-in. [Funnet 6 Juni 2021].

[13] «Zoom,» [Internett]. Available: https://zoom.us/. [Funnet 8 Juni 2021].

[14] PRUSA Research by Josef Prusa, «PRUSA,» [Internett]. Available: https://www.prusa3d.com/. [Funnet 06 06 2021].

[15] Wanhao, «Wanhao,» [Internett]. Available: https://www.wanhao3dprinter.com/Index.html. [Funnet 06 06 2021].

[16] ALLEGRO microsystems, «ALLEGRO microsystems,» [Internett]. Available: https://www.allegromicro.com/en/insights-and-innovations/technical-documents/hall-effect-sensor-ic-publications/position-and-level-sensing-using-hall-effect-sensing-technology . [Funnet 06 06 2021].

[17] D. J. Ben Nuttall, «GPIO Zero,» [Internett]. Available: https://gpiozero.readthedocs.io/en/stable/ . [Funnet 18 02 2021].

[18] «Tkinter logo,» [Internett]. Available: https://www.google.com/url?sa=i&url=https%3A%2F%2Fiot4beginners.com%2Flook-and-feel-customization-on-tkinter-python-gui%2F&psig=AOvVaw2yIwTCtwAG4i8F4o441oMv&ust=1623146710134000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCMj64umihfECFQAAAAAdAAAAABAI. [Funnet 28 Mai 2021].

[19] Python Software Foundation, «Graphical User Interfaces with Tk,» [Internett]. Available: https://docs.python.org/3/library/tkinter.html. [Funnet 28 Mai 2021].

[20] D. Amos, «Real Python,» [Internett]. Available: https://realpython.com/python-gui-tkinter/. [Funnet 28 Mai 2021].

[21] I. Ames, «Medium,» [Internett]. Available: https://medium.com/software-engineering-roundup/tcp-ip-protocol-suite-ec7ed4888d5d. [Funnet 8 Juni 2021].

[22] Javatpoint, «Javatpoint,» [Internett]. Available: https://www.javatpoint.com/tcp. [Funnet 7 Juni 2021].

[23] M. Cook, «Lifesize,» [Internett]. Available: https://www.lifesize.com/en/blog/tcp-vs-udp/. [Funnet 8 Juni 2021].

[24] Python, «Classes,» [Internett]. Available: https://docs.python.org/3/tutorial/classes.html. [Funnet 8 Juni 2021].

[25] Oracle, «Docs Oracle,» [Internett]. Available: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html. [Funnet 6 Juni 2021].

[26] Wikipedia, «Wikipedia,» Wikimedia, [Internett]. Available: https://en.wikipedia.org/wiki/Bitstream. [Funnet 6 Juni 2021].

[27] Python, «Sockets,» [Internett]. Available: https://docs.python.org/3/howto/sockets.html. [Funnet 6 Juni 2021].

[28] WhatIsMyIpAddress, «WhatIsMyIpAddress,» [Internett]. Available: https://whatismyipaddress.com/port. [Funnet 6 Juni 2021].

[29] OPC Foundation, «OPC Foundation,» [Internett]. Available: https://opcfoundation.org/about/opc-technologies/opc-ua/. [Funnet 2021 Juni 11].

[30] D. Radigan, «Atlassian,» [Internett]. Available: https://www.atlassian.com/agile/kanban. [Funnet 2021 Juni 2021].

[31] S. M. Nes, «Visma,» [Internett]. Available: https://www.visma.no/blogg/en-kort-introduksjon-til-scrum/. [Funnet 6 Juni 2021].

[32] M. Rehkopf, «Atlassian,» [Internett]. Available: https://www.atlassian.com/agile/kanban/boards. [Funnet 6 Juni 2021].

[33] V. Mohanan, «Unsplash,» [Internett]. Available: https://unsplash.com/photos/rZKdS0wI8Ks. [Funnet 06 06 2021].

# Appendiks A Forkortelser og ordforklaringer

| | |
|---|---|
| Bytestream | En rekke bytes/bit for å overføre informasjon |
| C# | Programmeringsspråk |
| DC | Direct Current / Likestrøm |
| Deserialisering | Henter informasjon fra en bytestream og rekonstruerer til f.eks. et objekt |
| Extruder | Del av 3D-printer som presser materialet videre inn i skrivehodet til printeren. |
| Gcode | En fil med instruksjoner som 3d-printere, CNC-maskiner og lignende bruker til å vite hvordan de skal bevege seg. |
| GUI | Graphical User Interface / Brukergrensesnitt |
| Hall Effect Sensor | En type sensor som måler endringer i magnetfelt. |
| HW | Hardware / Maskinvare |
| IDE | Integrated Development Environment / Uviklermiljø for programmering |
| Klient | Kommuniserer med server over nett |
| MOSFET | Metal-oxide-semiconducter Field-effect-transistor / Transistortype |
| Objekt | En kodetype bestående av variabler |
| OPC-UA | OPC Unified Architecture |
| PC | Personal Computer / Datamaskin |
| Pulse Width Modulation | En måte å styre spenning ved å sende en firkant signal hvor man endrer perioden i signalet. |
| PWM | Pulse Width Modulation / Pulsbreddemodulasjon |
| Python | Programmeringsspråk |
| Rotary Encoder | En type sensor som sender ut pulser når den roteres. |
| RPi | Raspberry Pi |
| Serialisering | Gjør noe om til en bytestream |
| Server | Kommuniserer med klient over nett |
| Single-Board | Alt på ett kretskort (Beskrivelse av en type datamaskin) |
| Slicer | Et program som gjør en 3d modell om til gcode som en 3d-printer kan forstå. |
| SW | Software / Programvare |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |

# Appendiks B Prosjektledelse og styring

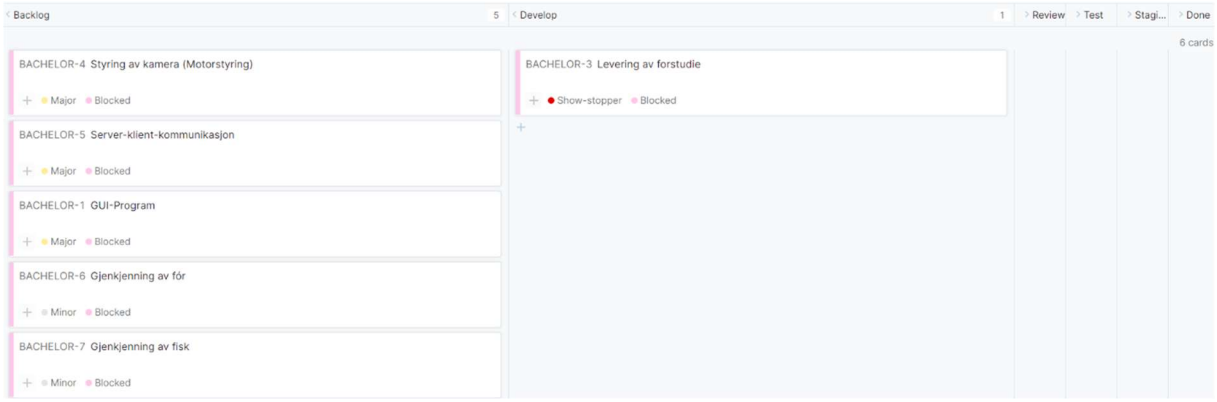
## B.1 Prosjektorganisasjon

På grunn av prosjektformen hos oss, så er det en løst organisert gruppe. Vi har et likt sett med kompetanse, og det er derfor uklokt å delegere noen til spesifikke arbeidsoppgaver. Eneste løse rollen er et gruppemedlem som har brorparten av kommunikasjonen med ABB (arbeidsgiver).

Med tanke på arbeidsoppgaver i prosjektet, så har det vært grunnleggende inndelt. Andreas Håvardsholm Brekken har hatt ansvar for den fysiske modellen, motorstyring og RPi-programmeringen, Runar Grøneng har hatt ansvar for server/klient-programmeringen, og Andreas Gjervik Aaland og Jan Einar Husa har hatt ansvar for programmering av brukergrensesnittet.

## B.2 Prosjektform

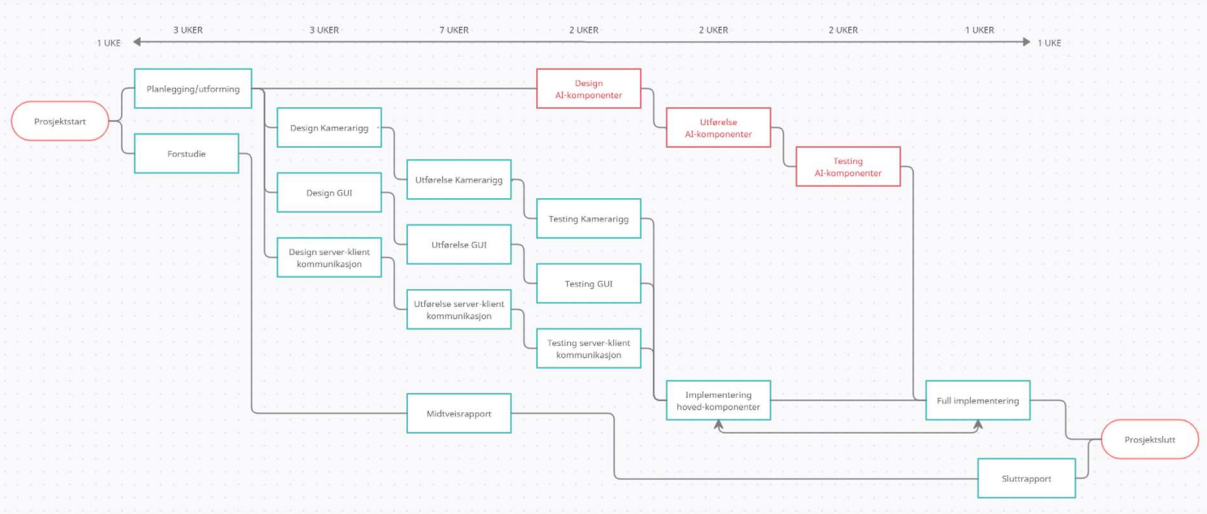
Vi har en hybrid prosjektform med Agile og Waterfall. Det brukes Agile "Kanban" [30] som agile prosjektform. [30] Dette hjelper oss som et lite team til å ha fleksibilitet i hvor og hva vi jobber med. Istedenfor å sette konkrete tidsfrister, deler vi prosjektet opp i kort og bolker som må gjøres. Dette gjør også at møter ikke er nødvendig like ofte som for eksempel Scrum. [31]




***Figur 51: Kanban-board***

## B.3 Fremdriftsplan

Fremdriftsplanen er løpende og fleksibel i form av en oppslagstavle. [32] Det som er den tellende tidsfristen, er innlevering av det viktigste prosjektkomponentene og den endelige rapporten. I løpet av prosjektet vil det tas en avgjørelse om det kan legges til flere av ønskede funksjoner avhengig av fremdriften til hovedfunksjonene. Siden vi har en hybrid av Waterfall og Agile, så tillater det oss å ha en overordnet fremdriftsplan (Figur 53). Denne sier i grove trekk hvor vi bør befinne oss i løpet av prosjektet.




*Figur 52: Fremdriftsplan*

## B.4 Risikoliste

| | |
|---|---|
| **Risiko** | Prosjektet blir ikke ferdig innen tidsfristen |
| **Konsekvens** | Ikke bestått/Utsettelse av frist / Levere forbedret oppgave påfølgende semester |
| **Sannsynlighet** | Lav |
| **Alvorlighet** | Høy |
| **Tiltak** | Opprettholde tidsfrister |

| | |
|---|---|
| **Risiko** | Misforståelse av oppgaven |
| **Konsekvens** | Forsinkelse eller utvikle feil produkt/løsning |
| **Sannsynlighet** | Lav |
| **Alvorlighet** | Høy |
| **Tiltak** | Forholde seg til kravspesifikasjonen, god kommunikasjon med oppdragsgivere |

| | |
|---|---|
| **Risiko** | Sykdom / frafall |
| **Konsekvens** | Redusert effektivitet, enkelte gruppemedlemmer blir gjerne overarbeidet, risikere å miste tidsfristen |
| **Sannsynlighet** | Middels |
| **Alvorlighet** | Høy |
| **Tiltak** | Holde seg aktiv, opprettholde god hygiene, holde avstand, være motivert |

| | |
|---|---|
| **Risiko** | Enkelte medlemmer i teamet tar ikke oppgaven seriøst/har dårlig motivasjon |
| **Konsekvens** | Forsinkelse. Lav kvalitet på arbeidet |
| **Sannsynlighet** | Lav |
| **Alvorlighet** | Høy |
| **Tiltak** | God kommunikasjon i teamet, muntre hverandre opp/hjelpe hverandre |

| | |
|---|---|
| **Risiko** | Leveringsproblemer på utstyr som er bestilt til prosjektet |
| **Konsekvens** | Forsinkelse. |
| **Sannsynlighet** | Lav |
| **Alvorlighet** | Middels |
| **Tiltak** | Sende bestillinger så tidlig som mulig |

| | |
|---|---|
| **Risiko** | Maskinvarefeil / Harddisk - kræsj |
| **Konsekvens** | Tap av data |
| **Sannsynlighet** | Lav |
| **Alvorlighet** | Høy |
| **Tiltak** | Backup / online lagring |

# Appendiks C Brukerdokumentasjon

Dokumentasjon som er relevant for brukeren ligger i vedlegg.

# Appendiks D Kildekode, Bill Of Materials mm

**Budsjett:**

## Budsjett for Bachelor oppgave 2020

| | |
|---|---|
| **Oppgave nr** | 27 |
| **Antall studenter** | 4 |
| **Bedrift** | ABB |

| | | Hvem skal betale ? | | | |
|---|---|---|---|---|---|
| **#** | **Beskrivelse av utgift** | **Studenter** | **Bedrift** | **HVL** | |
| 1 | Raspberry Pi-kit | | 1400 | | |
| 2 | Ekstra mikrokontroller for IO | | 400 | | |
| 3 | Servomotorer | | Skaffes av ABB | | |
| 4 | MOSFETs | | 300 | | |
| 5 | Endestoppbrytere/sensorer | | 300 | | |
| 6 | PLA(materiale til 3d-printer) | | 600 | | |
| 7 | Webkamera | | 900 | | |
| 8 | Diverse materialer til prototype | | 1000 | | |
| | **Totalt** | **0** | **4900** | **0** | |

**Regnskap:**

| | **Regnskap for bachelor oppgave** | |
|---|---|---|
| # | **Beskrivelse av utgift** | **kostnad** |
| 1 | Monteringsbånd | kr 69,90 |
| 2 | Gjengestenger | kr 399,60 |
| 3 | Muttere | kr 32,90 |
| 4 | Sekskantskruer | kr 39,90 |
| 5 | PLA | kr 259,00 |
| 6 | Rotary Encodere | kr 327,38 |
| 7 | Motorer | kr 192,63 |
| 8 | Motorkontrollere | kr 100,00 |
| 9 | Raspberry Pi | kr 690,00 |
| 10 | Minnekort | kr 189,00 |
| 11 | USB-C PSU | kr 249,90 |
| 12 | Ledninger | kr 200,00 |
| 13 | Diverse elektriske komponenter | kr 100,00 |
| | **Sum** | kr 2 850,21 |

Kildekoden til prosjektet er vedlagt i zip-fil.