Sensur av hovedoppgaver

Universitetet i Sørøst-Norge
Fakultet for teknologi og maritime fag

Prosjektnummer: **2018-17**
For studieåret: **2017/2018**
Emnekode: **SFHO3201**

Prosjektnavn:
Automatisering av kalibreringsjigg
Calibration of RangeFinder (COR)

Utført i samarbeid med: Miros AS

Ekstern veileder: Jonas Røstad

Sammendrag:
Prosjektet vårt omhandler automatisering av en kalibreringsjigg som blir brukt til å kalibrere en radarbasert sensor med navnet RangeFinder SM-140. Oppgaven er gitt av bedriften Miros AS, prosjektet vil effektivisere kalibreringsprosessen.

Stikkord:
- Radarbaserte systemer
- Automatisering
- Posisjonering

Tilgjengelig: JA

Prosjekt deltagere og karakter:

Navn	Karakter
Deivydas Kazokas	
Tuan Anh Trinh	
Thomas Trinh	
Ecir Ali Doganci	
Vebjørn Sveva	
Vetle Damtjernhaug Aasland	

Dato: 18. Mai 2018

_____	_____	_____
Kiran Raja	Karoline Moholth	Jonas Røstad
Intern Veileder	Intern Sensor	Ekstern Sensor

Bacheloroppgave 2018
Gruppe 17

COR

Calibration Of Rangefinder

Prosjektmedlemmer	**Veiledere og Sensorer**
Tuan Anh Trinh Deivydas Kazokas Thomas Trinh Vebjørn Sveva Vetle Aasland Ecir Ali Doganci	Karoline Moholth Kiran Raja Jonas Røstad
Versjon: 3.0	Dato: 21.05.2018

Universitetet i Sørøst-Norge

MIROS

INNHOLD

1 INTRODUKSJON

Dokumenthistorie			
Versjon	**Dato**	**Endret av**	**Aktivitet**
0.1	22/01/18	TAT	Dokument opprettet
0.2	23/01/18	TAT	Ordliste
0.3	24/01/18	TAT	Medlemmer, Gruppeoversikt
0.4	25/01/18	Alle	Ansvarsområder
0.5	06/02/18	VS	Revidert for publisering
1.0	07/02/18	TAT	Publisert i rapport v1.0
1.1	05/04/18	Alle	Oppdatering av ordliste
1.2	05/04/18	VS	Revidert for publisering
2.0	06/04/18	TT	Publisert i rapport v2.0
2.1	19/05/18	TAT	Oppdatering av ordliste
3.0	21/05/18	TT	Publisert i rapport v3.0

Tabell 1.1: Dokumenthistorie for introduksjon

1.1 Innledning

Som avgangsstudenter fra Universitet i Sør-Øst Norge, Campus Kongsberg, har vi denne våren blitt tildelt en bacheloroppgave som teller 20 studiepoeng. Bacheloroppgaven skal gi studentene en smakebit på hvordan hverdagen som ingeniør i en bedrift eller et selskap er.

Bacheloroppgaven er delt opp i tre hoveddeler. Hver del avgjør gjennomføringsgraden av prosjektet, og gir et vurderingsgrunnlag av hva hvert gruppemedlem har bidratt med gjennom prosessen. De tre nevnte delene er presentasjon, dokumentasjon og selve produktet.

Vår bacheloroppgave omhandler kalibrering av Miros AS sin RangeFinder(SM-140) [35]. I dag blir denne prosessen gjort manuelt ved at en operatør flytter på en reflektorvegg fram og tilbake til ønsket posisjon. Denne operasjonen har Miros AS lyst til å modifisere slik at dette blir omgjort til et fullstendig autonomt system. Dette bidrar til å få mer nøyaktige og tidsbesparende målinger, samtidig som det vil minske kostnader for bedriften.

1.2 Ordliste

TAT	Tuan Anh Trinh
DK	Deivydas Kazokas
EAD	Ecir Ali Doganci
VS	Vebjørn Sveva
VA	Vetle Aasland
TT	Thomas Trinh
RF	Range Finder
COR	Calibration of RangeFinder
UP	Unified Prosess
VDC	Voltage Direct Current
VAC	Voltage Alternating Current
ppr	Pulse per revolusjon
RPM	Round per revolusjon
AGM	Absorbent Glass Mat
Li-Ion	Litium-ionbatteri
Li-Poly	Litium-polymerbatteri
NiMh	Nickel-metal hydride batteri
NiCd	Nickel-cadmium batteri
BMS	Battery management system
Pb	Metallisk bly
W	Watt
Ah	Ampere per timer
Mbps	Megabyte per sekund
GHz	Gigahertz
BLE	Bluetooth low energy
CPU	Central Processing Unit
GPU	Graphical Processing Unit

RAM	Random Access Memory
ROM	Read Only Memory
SBC	Single Board Computer
SoC	System on Chip
HDMI	High Definition Multimedia Interface
USB	Universal Serial Bus
UART	Universal Asynchronous Reciever-Transmitter
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
DAC	Digital-to-Analog-Converter
ADC	Analog-to-Digital-Converter
PC	Personal Computer
MSP	Motor Setup Program
IoT	Internet of Things
MQTT	Message Queing Telemetry Transport
UART	Universal Asynchronous receiver/transmitter
JSON	JavaScript Object notation
PWM	Pulse width modulation
SQL	Structured Query Language
Nm	Newton meter
RxA	receiver i tilkobling A
RxB	receiver i tilkobling B
SSH	secure shell
SFTP	secure File Transfer Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol
QoS	quality of service
SSL	Secure Sockets layer
TLS	Transport layer Security

Tabell 1.2: Ordliste

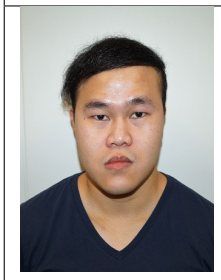
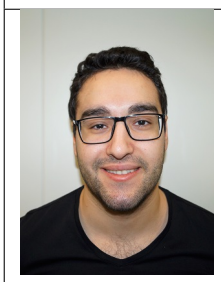
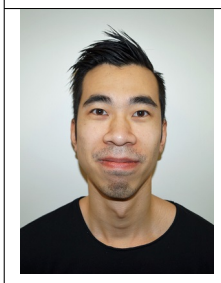
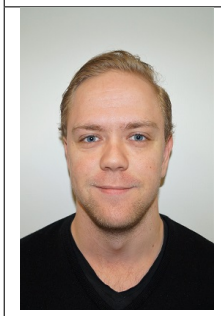
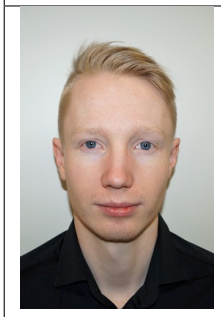
1.3 Gruppeinformasjon

Bachelorgruppe 17 består av fire elektrostudenter og to datastudenter.

- Datastudenter har bakgrunn innenfor Embedded Systems[13].
- Elektrostudenter har bakgrunn innenfor kybernetikk og mekatronikk [19].

Gruppen har fastsatte møter hver mandag og fredag. Under mandagsmøtet går vi gjennom hva som skal gjøres den kommende uken og hvilke arbeidsoppgaver som er utført eller trenger mer tid. Det blir også gjort en gjennomgang av dokumenter som skal sendes inn eller lastes opp på Canvas. Under fredagsmøtet går vi over hva som har blitt gjort i løpet av uken, og planlegger neste ukes arbeid og oppgavefordeling. Timelister for ukens arbeid oppdateres fredager.

For oversikt over arbeidsprosessen bruker vi Dropbox som lagringsplattform. Bruk av Dropbox gjør det lettere å kvalitetsikre dokumentasjonen og ta backup hvis noe går galt. Alt av dokumentasjon blir laget via Share-Latex der flere kan være tilkoblet og jobbe samtidig. Grunnen til at vi bruker Latex fremfor Word, er fordi Latex er mer oversiktlig med større dokumenter, og vi får et mer proffesjonelt dokument.

Foto	Informasjon	Ansvarsområde
	Navn: Deivydas Kazokas E-post: deivydas.kazokas@gmail.com Mobilnr.: 96899900 Retning: Kybernetikk og mekatronikk	**Prosjektleder**
	Navn: Tuan Anh Trinh E-post: tuananhtrinh101@gmailcom Mobilnr.: 45291465 Retning: Kybernetikk og mekatronikk	**Test og verfikasjon**
	Navn: Ecir Ali Doganci E-post: ecir527@gmail.com Mobilnr.: 90841087 Retning: Kybernetikk og mekatronikk	**Dokumentasjon**
	Navn: Thomas Trinh E-post: Cmt_thomas@hotmail.com Mobilnr.: 90368162 Retning: Kybernetikk og mekatronikk	**Risiko og krav**
	Navn: Vebjørn Sveva E-post: Vebsve@hotmail.com Mobilnr.: 47386249 Retning: Embedded Systems	**Web og design**
	Navn: Vetle Aasland E-post: vetle_aasland@hotmail.com Mobilnr.: 95729060 Retning: Embedded Systems	**Programmering**

Tabell 1.3: Gruppe oversikt

1.4 Ansvarsområder

Prosjektleder	• Ansvarlig for prosjektets fremgang, holde oversikt over gruppemedlemmenes arbeidsoppgaver. • Kommunikasjon med bedriften, veiledere og sensorer.
Test og verifikasjon	• Testansvarlig har ansvar for at resultater av utførte tester eller simulasjoner oppfyller kravspesifikasjon. • Verifikasjonansvarlig har som oppgave å få klarhet i om konsept eller ide har relevans i prosjektet.
Dokumentasjon	• Dokmuentasjonansvalig har hovedansvar for den fullstendige dokumentasjonen som blir levert. • Dette innebærer å lese gjennom og finpusse dokumentasjon som skal leveres slik at den ser ordentlig ut til innlevering.
Risiko og krav	• Risikoansvarlig har som ansvar å evaluere problematiske hendelser som kan inntreffe gjennom prosjektet, samt føre en risikovurdering av disse. • Kravansvarlig har ansvar for at kravspesifikasjonen blir oppdatert og gjennomført gjennom prosjektet.
Web og design	• Webansvarlig innebærer et ansvar for å opprette og oppdatere prosjektgruppens hjemmeside. • Designansvarlig innebærer ett ansvar for systemets brukergrensesnitt, samt ulike diagrammer som visualiserer og beskriver funksjonalitet.
Programmering	• Har ansvaret for koding, og at programmer passer til systemet. • Ansvar for at software blir modulbasert.

Tabell 1.4: Ansvarområde

2 VISJONSDOKUMENT

Dokumenthistorie			
Versjon	**Dato**	**Endret av**	**Aktivitet**
0.1	22/01/18	EAD	Dokument opprettet
0.2	23/01/18	TAT	Problemstilling
0.3	23/01/18	TAT	Oppgavedefinisjon
0.4	25/01/18	EAD	Interessenter
0.5	30/01/18	EAD	Konsepter
0.6	01/02/18	EAD	Visjonsdokument
0.7	06/02/18	VS	Revidert for publisering
1.0	07/02/18	EAD	Publisert i rapport v1.0
1.1	27/02/18	VS	Markedsanalyse
1.2	03/03/18	TT	Finansiering
1.3	25/03/18	EAD	Endelige konseptet
1.4	04/04/18	VS	Revidert for publisering
2.0	06/04/18	EAD	Publisert i rapport v2.0
2.1	18/05/18	TT	Justert Markedsanalyse og Finansiering
2.2	20/05/18	VA	Revidert for publisering
3.0	21/05/18	TT	Publisert i rapport v3.0

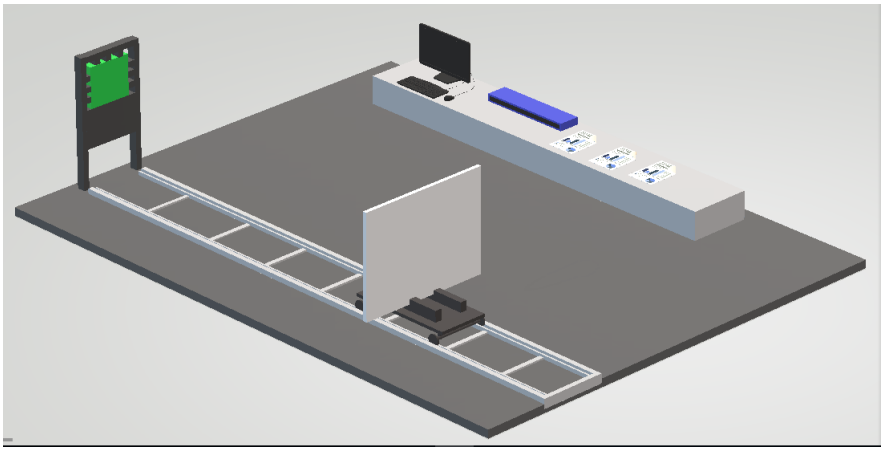
Tabell 2.1: Dokumenthistorie for visjonsdokument

2.1 Innledning

Hensikten med visjonsdokumentet er å få en oversikt over hva oppgaven går ut på, hvordan vi skal løse problemstillingen og overfladisk redegjøre ansvarsområder for interessenter og brukere. [29]

2.2 Problemstilling

RangeFinderen er et produkt utviklet av Miros AS [22] som er en radarbasert avstands- og bølgemåler. RF har en nøyaktighet på +/- 5mm, noe som krever høy kalibrering av nøyaktighet. Ut fra dagens kalibreringsjigg blir denne prosessen gjort manuelt ved å flytte på en reflektorvogn til forskjellige punkter hvor disse blir sammenlignet med referansepunkter. Hvis RF viser et avvik i fem millimeter eller mer fra oppmålt verdi, regnes det som en feilkalibrering og kalibrering utføres på nytt. Denne kalibreringsprosessen tar opptil tre timer for hver kjøring og krever mye tid for operatøren.



Figur 2.1: Dagens kaliberingjigg

2.3 Oppgavedefinisjon

Oppgaven som vi har fått fra Miros AS er å automatisere en kalibreringsjigg basert på samme kalibreringsmetode. Den autonome kalibreringsjiggen skal automatisere kjøring av en kalibreringsrunde der reflektorvognen flyttes til ti forskjellige målepunkter. For hvert avstandspunkt skal det bli tatt en avstandsmåling fra RF, og gitt verdi skal sammenlignes med referanseavstand til reflektorvognen. Etter at en kalibreringsrunde er utført og avviksverdien er større enn lovlig grenseverdi, skal en ny kalibreringsrunde tas. Dette blir gjort helt fram til avviksverdien er innenfor lovlig grenseverdi. Når avviksverdi er innenfor grenseverdien vil systemet automatisk generere en kalibreringsrapport som blir lastet opp til en server og lokal database, i tillegg til at det skal skrives ut en fysisk kopi. Oppgaven blir delt i tre separate deler der hver del krever ulike løsningsmetoder. De tre delene er som følger: Brukergrensesnitt, signaloverføring/spenningskilde og reflektorvogn.

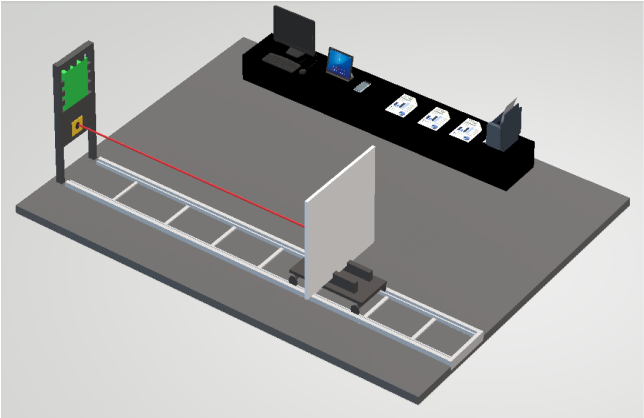
Brukergrensesnitt: Systemet skal styres av et Windows basert software - dette kan være web eller applikasjon. Hvis ønskelig kan det legges til krav om at det skal kunne styres av nettbrett og mobil. Programflyt må etableres og skisse til GUI må frembringes før implementering.

Signaloverføring og spenningskilde: RF kommuniserer serielt på RS232 og bruker 24V DC som spenningskilde. Signaloverføring innebærer å lage en frittstående modul som kan kommunisere og motta data fra RF, denne modulen skal kunne brukes til eksisterende software.

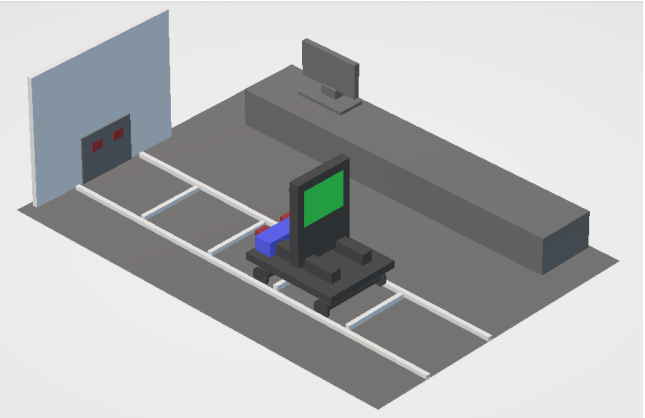
Reflektorvogn: Skal autonomt kunne beveges til ti bestemte posisjoner med høy nøyaktighet.

2.4 Konsepter

Konseptene er en omvisning og gjennomgang av tanker rundt hvordan oppgaven kan løses. Tre hoveddeler er som tidligere nevnt nødvendig for at systemet skal fungere fullstendig autonomt, siste del om kommunikasjon med datamaskin har vi foreløpig ikke utredet konsepter for. Første del er basert på avstandsmåling, der to komponenter skal vise avstandsverdi for bruk til referansepunkter. Den andre delen omhandler motorer vi har tenkt å bruke etterhvert.

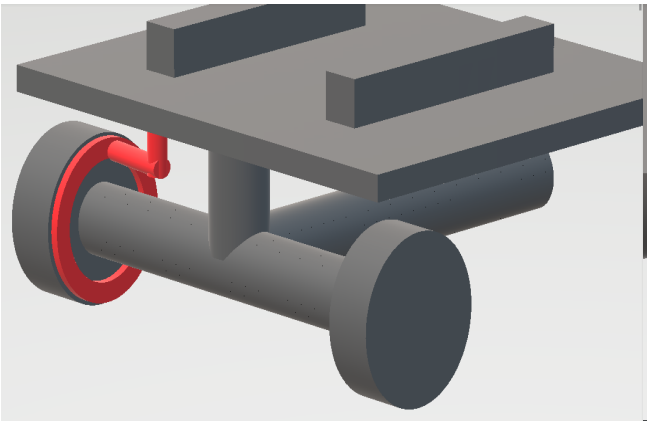


Figur 2.2: Konsept med laser

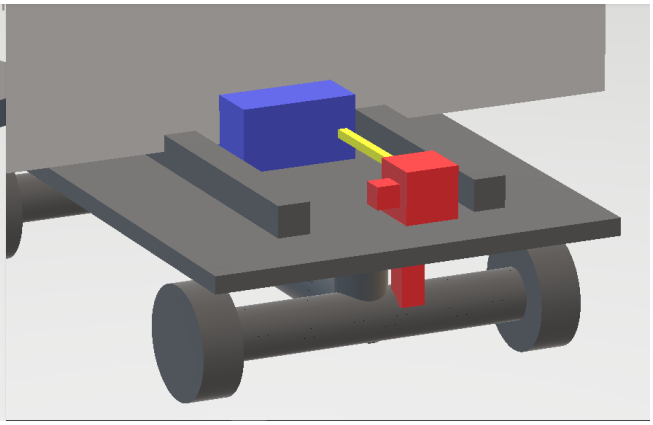


Figur 2.3: Konsept med målebånd

Figur 2.2 Et konsept basert på en lasermåler som viser avstandsverdien fra reflektorvegg og avvik fra referansepunkt.
Figur 2.3 Et konsept basert på et digitalt målebånd som har et display for avstandsverdi.

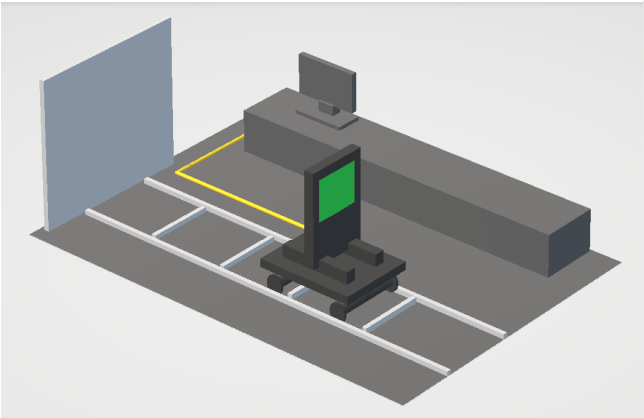


Figur 2.4: Konsept med Incremental encoder

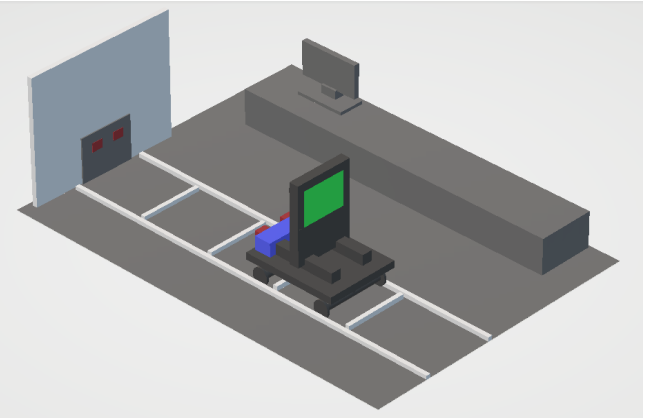


Figur 2.5: Konsept med motor

Figur 2.4 Et konsept basert på en incremental encoder som viser avstandsverdi ut fra hjulets omdreiningstall.
Figur 2.5 Et konsept basert på en incremental encoder med instrumentet, alle komponenter inkludert på vognen, utenom RF.



Figur 2.6: Konsept med RangeFinder



Figur 2.7: Konsept med RangeFinder (batteri)

Figur 2.6 Et konsept basert på at RangeFinder skal være på vognen, med bruk av strømledning.
Figur 2.7 Et konsept basert på at RangeFinder skal være på vognen, med bruk av batteri.

2.4.1 Endelige konsept

For å fremstille det endelige konseptet har vi valgt å illustrere dette i form av en animasjon som viser hvor komponentene fysisk ligger i vogna. Dette hjelper oss med å få en oversikt over hvilke komponenter som er i bruk og hvor i systemet disse skal fungere. Som vi har bestemt skal vi bytte plass på RF og reflektorveggen. Fordeler med dette er at det blir mindre bevegelig del på veibanen. Programmet vi har brukt til å illustrere dette konseptet er SolidWorks. Animasjonen er tilgjengelig på nettsiden og minnepenn.

Det skal plasseres fire ultrasonic sensorer på vogna, en på hvert hjørne, for å bedre sikkerheten. Disse overvåker veibanen slik at det ikke skal kunne kjøres over noe som kan påvirke nøyaktighet eller skape andre farer.

DC motor-komponenten er plassert fremst på vogna og skal styre den frem og tilbake i veibanen. Denne skal fungere autonomt ut fra verdiene som tas inn når operatøren starter en målerunde.

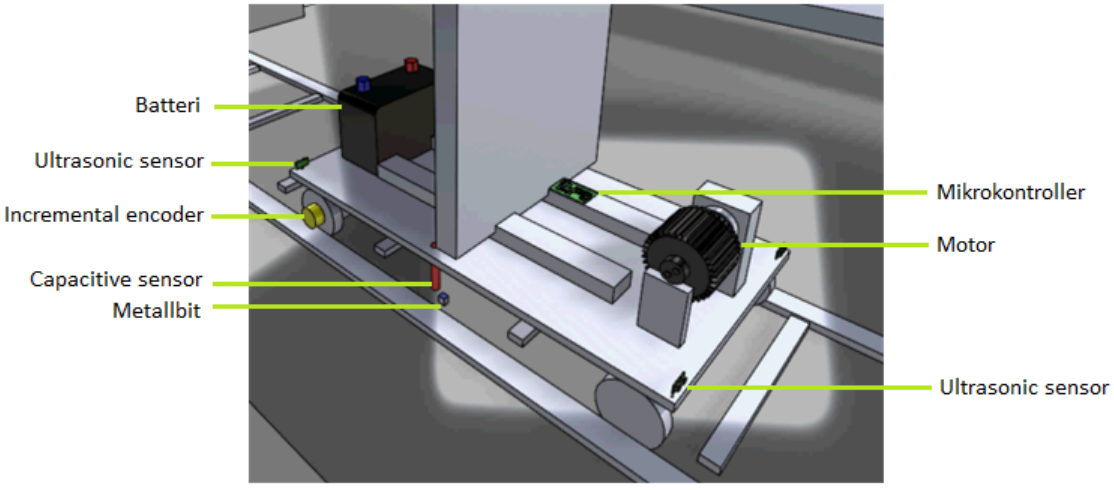
Mikrokontrolleren (Raspberry pi) er plassert midt på vogna. Raspberry pi skal ta inn signaler, bearbeide disse, og videresende de til og fra komponentene.

Incremental encoder ligger på det ene bakhjulet og teller omdreiningene. Siden hjulene bak ikke er drivende hjul slipper vi å tenke på spinn som kan være et problem med tanke på nøyaktigheten.

Capacitive sensor ligger på en vertikal linje med RF under selve vogna. Sensoren har som funksjon å telle metallbiter som er festet utover veibanen med halvmeter mellomrom på hver. For hvert signal som gis fra denne sensoren er det da kjørt en halv meter.

12V batteri skal virke som spenningskilde for hele systemet, og skal plasseres bakerst på vogna. Grunnen til det er at den må lades, så ladekontaktene skal ligge under på linje med reflektorveggen.

Noen bilder av animasjonen som viser plassering av komponenter.



Figur 2.8: Endelig konsept

2.5 Markedsanalyse

For at vi skal være mer konkurransedyktige ute på markedet så må vårt produkt være best innenfor sitt felt. Et autonomt system vil føre til at Miros AS, som fra før av er verdensledende innenfor radarteknologi offshore, vil dra enda flere fordeler på markedet. Vårt produkt legger mest vekt på presisjon, det vil si en svært lav feilmargin. Ved å bruke incremental encoder og proximity sensor får vi målt avstanden veldig nøyaktig. Siden produktet skal fungere autonomt og raskt, vil det bli spart mye tid og ressurser. Produktet er autonomt og benytter ultrasonic sensor for å forebygge uhell, så risikoen for skader er minimal. Produktet kan styres manuelt hvis det er ønskelig, og man trenger ikke høy kompetanse for å bruke produktet vårt.

2.6 Finansiering

En finansiell vurdering som tar hensyn til utgifter og inntekter er viktig i et prosjekt som dette. Et automatisert system hadde ikke vært et fordelaktig system uten en finansiell gevinst på sikt.

Budsjettet vi har blitt tildelt av Miros ligger på 50 000 NOK. Hvor stor del av denne summen som blir benyttet er fortsatt usikkert, men i utgangspunktet skal vurdering av inntekter og fortjeneste tas i forhold til det maksimale utgiftsbeløpet som er satt. Prosjektet vårt vil ha en kontantstrøm lik de fleste prosjekter, hvor utgifter tidlig i prosjektet skal føre til inntekter over tid etter at det nye systemet har blitt tatt i bruk over en periode. Vår vurdering av det endelige systemet tilsier at Miros vil få gevinst ut av prosjektet vårt i løpet av relativt liten tid. Utgifter som bestilling av motoriske deler vil etter planen forekomme i elaboration- og constructionfasen. Inntekter vil begynne å forekomme i løpet av transisjonsfasen da vi etter planen skal ha gjennomført nødvendig testing av systemet slik at bedriften kan starte å bruke det.

Vi har kommet frem til at et ønskelig produkt skal tjene inn utgiftene etter at 13 radarer er kalibrert. Tiden dette tar avhenger av hvor mange som blir kalibrert daglig, men gevinst bør forekomme kun i løpet av noen få arbeidsdager. Denne beregningen er tatt ut fra et estimat som tilsier at det vil spares omtrentlig 4 000 NOK per gjennomførte kalibreringsrunde. Dette betinger at systemet fungerer optimalt uten uventede utgifter og tidskrevende vedlikehold og/eller forbedringer. Noen utgifter må forventes også etter produktet tas i bruk, som vedlikehold og eventuelt innleie av personell, men selv med disse utgiftene så vil systemet ha klar økonomisk gevinst.

Effektivitet i forhold til kjøretid er vesentlig i dette tilfellet, og i prosjektet vårt er det satt et målbart krav til en times kjøretid per kalibreringsrunde. Dette er mer enn halvert kjøretid, noe som vil være avgjørende for økning av inntekter. Et autonomt system vil også sette mindre krav til kunnskap hos operatøren, noe som vil senke fremtidens systems utgifter i forhold til det nåværende systemet. Ønskelig skal systemet vårt være modulbasert, noe som vil si at elektromekaniske deler samt programvaredeler skal kunne gjenbrukes i andre systemer med andre funksjoner. Selv om dette ikke har en direkte innvirkning på dette prosjektet, så kan det ha en stor innvirkning på bedriftens fremtidige prosjekter, både i forhold til tid og kostnader.

Et system med god kvalitet og repeterbarhet sikrer en indirekte, men langsiktig økonomisk gevinst. Den langsiktige økonomiske gevinsten oppnås ved at et funksjonelt system har mindre behov for garanti og teknisk support, og dermed mindre kostnader. Merkenavnet Miros/RangeFinder vil med et godt system på sikt styrkes på markedet.

2.7 Interessenter

Ansvaret interessentene har rettet mot prosjektet er å kontrollere om oppgaven oppfyller behovene. Dette henviser til at interessentene skal vite hva produktet innebærer og hvordan det er satt opp. Ut fra dette har de en oversikt over hva som skal gjøres med videre med problemstillingen. Interessenten til produktet vårt er Miros AS i første omgang, da de skal bruke produktet videre for kalibrering av RangeFinder'en deres.

Figur 2.9: Interessenter

3 PROSESSDOKUMENT

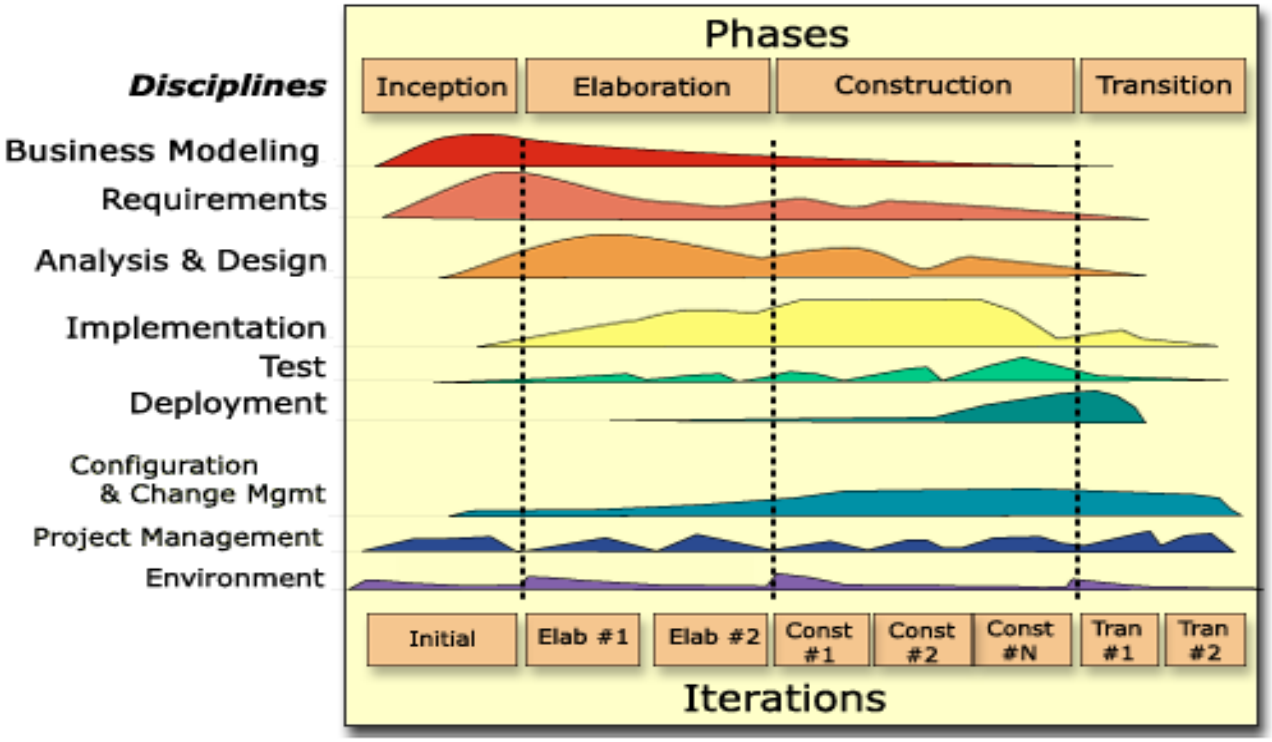
Dokumenthistorie			
Versjon	**Dato**	**Endret av**	**Aktivitet**
0.1	22/01/18	DK	Dokument opprettet
0.2	22/01/18	DK	Prosjektplan og modell
0.3	23/01/18	DK	Oppretting av gantt diagrammet
0.4	25/01/18	DK	Oppretting av prosessdokument
0.5	24/01/18	DK	Dokumentert iterasjon 1.0
0.6	06/02/18	DK	Dokumentert iterasjon 1.1
0.7	06/02/18	VS	Revidert for publisering
1.0	07/02/18	TT	Publisert i rapport v1.0
1.1	21/02/18	DK	Dokumentert iterasjon 2.0
1.2	11/03/18	DK	Dokumentert iterasjon 2.1
1.3	25/03/18	DK	Dokumentert iterasjon 2.2
1.4	05/04/18	DK	Dokumentert iterasjon 2.3
1.5	05/04/18	DK	Oppdatert gantt diagramet
1.6	05/04/18	VS	Revidert for publisering
2.0	06/04/18	TT	Publisert i rapport v2.0
2.1	26/04/18	DK	Dokumentert iterasjon 3.0
2.2	07/05/18	DK	Dokumentert iterasjon 3.1
2.3	20/05/18	DK	Dokumentert iterasjon 3.2
2.4	21/05/18	VS	Revidert for publisering
3.0	21/05/18	TT	Publisert i rapport v3.0

Tabell 3.1: Dokumenthistorie for prosessdokument

3.1 Unified Process

For å få en god oversikt over prosjektet har vi valgt Unified Process [49] som vår prosessmodell. Denne prosessmodellen gir oss klar oversikt over retningslinjer som skal følges i prosjektet.

Unified Process er basert på en iterativ og trinnvis utviklingsramme. Gjennom iterasjoner økes gradvis generell forståelse av prosjektet på vei mot en effektiv løsning. Modellen er drevet av viktige elementer som use-caser og disse bør gjenspeiles i hver fase av prosessen, slik at sluttproduktet møter kundenes behov. Use-casene hjelper oss med å definere funksjonelle og ikke funksjonelle krav som tas i bruk videre i prosessen. Videre er modellen drevet av jevnlig risk analyse, slik at de mest kritiske risikoene kan angripes tidlig i hver fase. Dette fører til en mer stabil og raskere utvikling av produkt mot slutten av prosjektet.



Figur 3.1: Unified prosess tidslinje [50]

Unified Process består av disipliner som businessmodell, krav, analyse og design, implementasjon, testing, utvikling og prosjektstyring.

• Businessmodell er fokusert på forståelse av organisasjonen, dens prosesser og behov. Det er noen faktorer som må forstås før produktet skal tas i betraktning for en utviklingsprosess. Noen av de kan være misjonserklæring, bedriftens regler og bedriftens visjoner.

• Kravdefinisjon er basert på use-casene.

• Analyse og design blir utført i forhold til tenkt løsning. Dette blir gjort for å analysere og forstå systemets krav.

• Implementasjon inneholder enhetstesting, koding og integrasjon av programvare.

• Testing og utvikling er basert på forsikring av produktkvalitet og inneholder testplanlegging og utførelse samt rapportering av feil som oppstår underveis. Godkjennelse av marginer for testing må også tas i betraktning.

• Prosjektstyring er fokusert på essensielle aktiviteter som styring av personell, samordning for interessenter, risikostyring av prosjektet, tidsestimering og planlegging av prosjektet og dens iterasjoner.

Unified Process er sammensatt av fire hovedfaser.

Innledning (Inception) fase definerer delen av prosjektet som omhandler hva vi skal utvikle og hvorfor.

• Første businessmodell skal inneholde kontekst, suksessanalyse, budsjett og estimerte kostnader samt markedsanalyse.

• Et visjonsdokument skal opprettes. Dette inneholder problemstilling, analyse av interessenter, første skissering av oppgaveløsning og analyse av forutsetninger og rammeverk.

• Utrede første og mest generelle kravspesifikasjon.

• Utvikle første risikovurdering.

• Utvikle første utgave av prosjektplan.

Milepeler i innledningsfasen bør inneholde enighet i krav som utledes fra use-casene, interessentens enighet i produktets visjon samt fullført businessplan.

Utforming (Elaboration) fase definerer delen av prosjektet som beskriver hvordan vi skal utvikle produktet. Detaljert og teknisk analyse utføres i denne fasen.

- Identifisering av alle aktører og dens use-caser.
- Identifisering av funksjonelle samt ikke-funksjonelle krav.
- Businessplan og høyprioritets-risikoer forebygges og klargjøres.
- Utvikling av en mer detaljert prosjektplan som viser iterasjoner og evaluering av disse.
- Valg av fysiske komponenter.
- Preliminær utvikling av programvaren.
- Sørge for at alt av nødvendig verktøy, prosesser og retningslinjer er definert før konstruksjonsfasen påbegynnes.

Milepeler i utformingsfasen bør inneholde godkjennelse av produktets visjon og arkitektur samtidig som identifisering og løsning for kritiske risikoer skal være utført.

Konstruksjon (Construction) fase inneholder programvareutvikling, integrasjon og testing.

- Programvaren er ferdigutviklet, integrert og testet.
- Fysiske komponenter er bestilt, integrert og testet.
- Revisjon og verifikasjon av kravene.
- Forsikre at produktet oppfyller kundens behov og estimeringer.
- Testing av integrert system.
- Unified Modelling Language (UML) diagrammer som: aktivitetsdiagram, sekvensdiagram, collaborationdiagram, statechart diagram og interaction overview diagram benyttes i denne prosessen.
- Oppnå tilstrekkelig kvalitet så raskt og effektivt som mulig.
- Utvikle brukermanualer og nødvendig støttedokumentasjon.

Milepeler i konstruksjonsfasen bør inneholde fullførelse av produktets stabilitet og grad av fullstendighet. Enighet med alle interessenter om at overgang til utvikling av det endelige produktet kan foretas. Sammenligning av estimerte og faktiske utgifter.

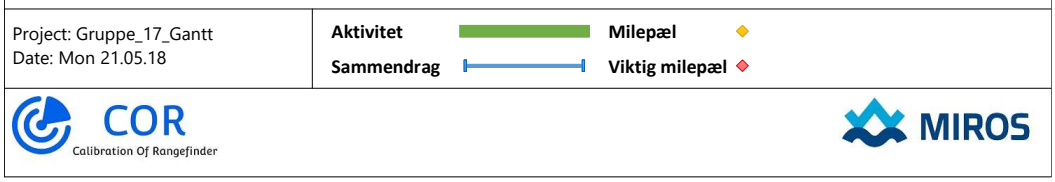
Overgang (Transition) fase inneholder brukerutnyttelse av produktet. Potensielle problemer identifiseres.

- Potensielle problemer er definert og rettet opp av utviklere.
- Opplæring av operatører og vedlikeholdspersonell.
- Inneholder sluttbrukergodkjennelse for produktet.

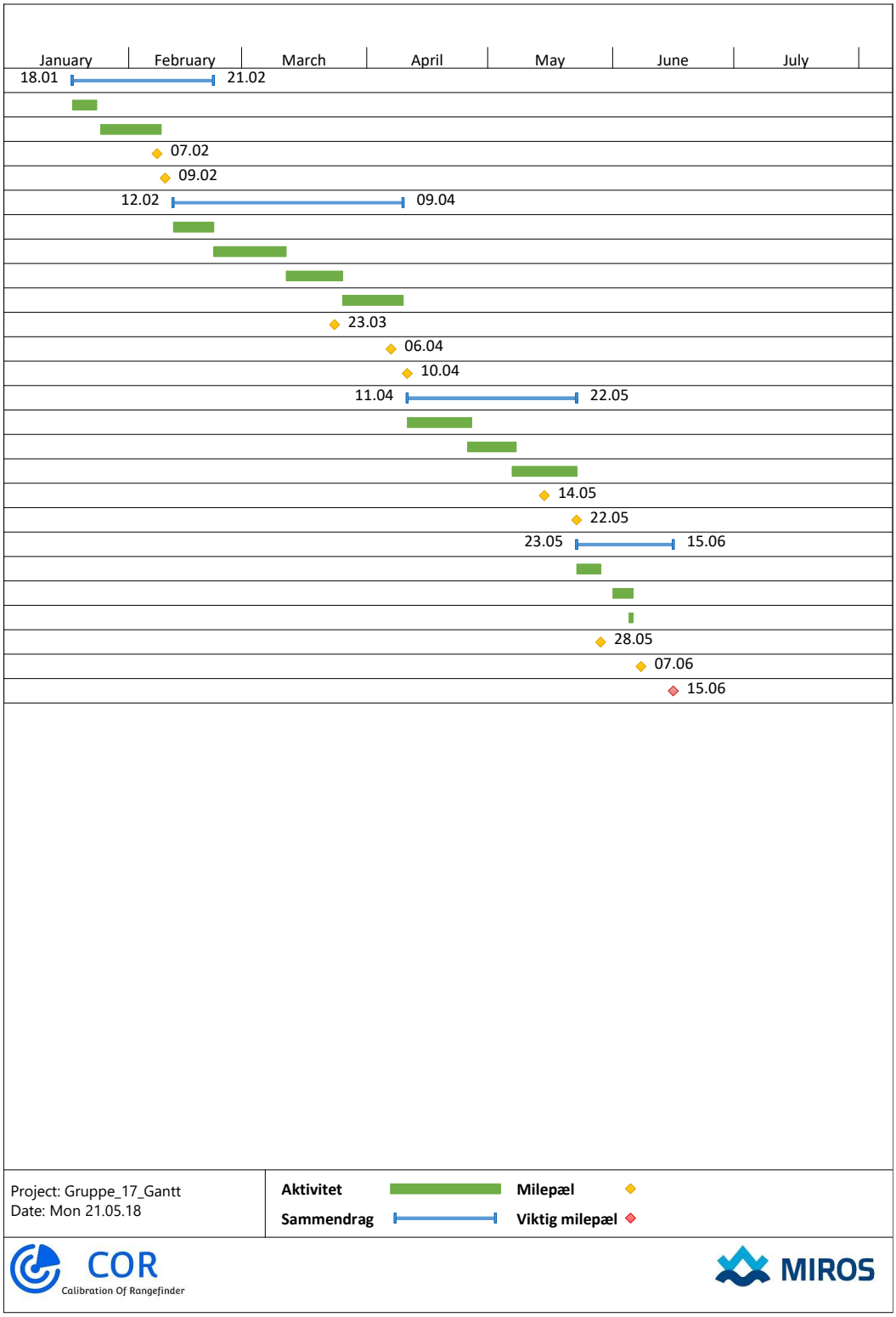
Milepeler i overgangsfasen bør inneholde brukerens fornøyelse med det ferdige produktet.

3.2 Gantt-diagram

NR	Aktivitet	Start	Slutt	Varighet
1	**1. Innledning (Inception)**	18 Jan	21 Feb	25 days
2	Iterasjon 1.0	18 Jan	23 Jan	4 days
3	Iterasjon 1.1	25 Jan	08 Feb	11 days
4	Dokumentlevering	07 Feb	07 Feb	1 day
5	Første presentasjon	09 Feb	09 Feb	1 day
6	**2. Utforming (Elaboration)**	12 Feb	09 Apr	41 days
7	Iterasjon 2.0	12 Feb	21 Feb	8 days
8	Iterasjon 2.1	22 Feb	11 Mar	13 days
9	Iterasjon 2.2	12 Mar	25 Mar	11 days
10	Iterasjon 2.3	26 Mar	09 Apr	11 days
11	Valg av konsept	23 Mar	23 Mar	1 day
12	Innleveringsfrist for dokumentasjon	06 Apr	06 Apr	1 day
13	Andre presentasjon	10 Apr	10 Apr	1 day
14	**3. Construction (Bygging)**	11 Apr	22 May	30 days
15	Iterasjon 3.0	11 Apr	26 Apr	12 days
16	Iterasjon 3.1	26 Apr	07 May	8 days
17	Iterasjon 3.2	07 May	22 May	12 days
18	Dokumentgodkjenning fra Miros AS	11 May	14 May	2 days
19	Innleveringsfrist for dokumentasjon	21 May	22 May	2 days
20	**4. Transition (Overgang)**	23 May	15 Jun	18 days
21	Plakat for prosjektet	23 May	28 May	4 days
22	Brukermanual for produktet	01 Jun	05 Jun	3 days
23	Endelig dokumentlevering til Miros AS	05 Jun	05 Jun	1 day
24	Plakatinnlevering	28 May	28 May	1 day
25	Tredje presentasjon	07 Jun	07 Jun	1 day
26	Vitnemålutdeling	15 Jun	15 Jun	1 day



Tabell 3.2: Gantt diagram


January
February
March
April
May
June
July
18.01
21.02
07.02
09.02
12.02
09.04
23.03
06.04
10.04
11.04
22.05
14.05
22.05
23.05
15.06
28.05
07.06
15.06
Project: Gruppe_17_Gantt
Date: Mon 21.05.18
Aktivitet
Sammendrag
Milepæl
Viktig milepæl
COR
Calibration Of Rangefinder
MIROS

3.3 Iterasjonsdokument

Iterasjonsdokumentet tar for seg aktiviteter og mål for hver iterasjon gjennom prosjektets faser. Antall iterasjoner avhenger av fasens lengde i forhold til prosjektmodellen vår. Lengden til hver iterasjon er estimert til å ha en varighet på omtrent to uker.

3.3.1 Iterasjon 1.0

1. Hovedmål

Høydepunkter for iterasjon

- Lage en klar oppgavebeskrivelse.
- Lage oversikt over gruppa og en generell risk analyse.
- Velge en prosessmodell og lage en preliminær prosjektplan.
- Visualisere ideer og lage et visjonsdokument.

2. Gjennomføring

Første iterasjon begynte med prosjektets planlegging. For å forenkle planleggingen valgte vi å bruke Unified Process, siden denne modellen er basert på iterasjoner og trinnvis utviklingsramme, noe som vil gi oss en klar oversikt over oppgaver som skal utføres i prosessen. Etter valg av modellen ble preliminær prosjektplan satt på plass og visualisert ved hjelp av Gantt diagrammet. Prosjektplanen ga oss mulighet til å velge passende oppgaver for hvert gruppemedlem.

Videre har vi analysert gitt oppgavebeskrivelse fra bedriften og utfallet ble noen designskisser for konsepter der ideene til gruppa ble visualisert i 2D.

Første utkast av use-case diagrammet hjalp oss med å bli enig om hovedfunksjoner operatøren skal kunne utføre. Deretter ble første versjon av testplantabell opprettet.

Dokumenter som ble opprettet iløpet av iterasjon

- Prosjektmodell
- Prosjektplan - første utkast av Gantt diagrammet
- Oppgavebeskrivelse fra bedriften
- Konsepter - Laget i 2D
- Use-case diagrammet - første utkast
- Testplan - første utkast

Iterasjon 1.0			
Aktivitet	**Start**	**Slutt**	**Varighet**
Prosjektmodell (Unified Process)	18. jan.	19. jan.	1
Prosjektplan (første utkast)	22. jan.	6. feb.	15
Oppgave beskrivelse	22. jan.	24. jan.	2
Konsepter	22. jan.	30. jan.	8
Unified modeling language (UML)	22. jan.	26. jan.	4
Testplan v1	22. jan.	30. jan.	8

Tabell 3.3: Tidstabell for iterasjon 1.0

Det meste av fokuset gikk til prosjektets planlegging og visualisering av konseptene

3.3.2 Iterasjon 1.1

1. Hovedmål

Høydepunkter for iterasjon 1.1

- Løsning på problemstilling
- Lage en ordliste med forkortelser til første rapport
- Utdype gitte krav
- Lage en businessplan
- Legge all dokumentasjon i første rapport

2. Gjennomføring

Iterasjonen begynte med oppretting av visjonsdokumentet, dette dokumentet gir oss en oversikt om hva oppgaven går ut på og hvordan vi kan løse problemstillingen.

Ordlistedokumentet ble opprettet med forkortelser som kan finne sted i første rapport.

Bedriften har gitt oss fire store hovedkrav som vi delte opp i mindre målbare krav.

Vi kom fram til at utreding av hovedkrav vil gi systemet forbedret funksjonalitet. Kravene ble lagt inn i en tabell og visualisert ved hjelp av Visio diagrammer. Vi sorterte og grupperte kravene for enklere sporbarhet. Deretter ble kravspesifikasjonsdokumentet opprettet.

Første utkast til vår egen webside ble produsert.

For at produktet skal være attraktivt for vår oppdragsgiver og mulige fremtidsklienter måtte vi lage første utkast av en businessplan. Planen omtaler budsjettet og gevinster som oppnås med produktet vårt.

Dokumenter som ble opprettet iløpet av iterasjon

- Ordliste
- Kravspesifikasjon
- Businessplan
- Første Rapport

3.1 Evaluering

Iterasjon 1.1			
Aktivitet	**Start**	**Slutt**	**Varighet**
Visjonsdokument	25. jan.	1. feb.	7
Ordliste	29. jan.	30. jan.	1
Kravspesifikasjoner (første utkast)	29. jan.	2. feb.	4
Web-design (første utkast)	29. jan.	8. feb.	10
Businessplan (første utkast)	31. jan.	5. feb.	5
Første presentasjon	5. feb.	8. feb.	3

Tabell 3.4: Tidstabell for iterasjon 1.1

Hovedfokuset under denne iterasjonen gikk til kravspesifikasjon og første rapport. Utredelse av krav følte vi at var en veldig viktig del av prosjektet. Konsultasjon med oppdragsgiver var veldig nyttig i forhold til utredelsen.

Første utkast av websiden ble produsert men siden vi ikke fikk et domene fra skolen så måtte vi fokusere på resten av aktivitetene.

På slutten av iterasjonen satt vi all dokumentasjon sammen til den første rapporten. Deretter ble tiden satt av til øving på første presentasjon.

3.3.3 Iterasjon 2.0

1. Hovedmål

Høydepunkter for iterasjon

- Videre analyse av krav
- Oppdatere prosjektplan
- Utfylle use-casene
- Animere konseptene i 3D
- Lage et systemarkitekturdiagram
- Lage et sekvensdiagram

2. Gjennomføring

Iterasjonen begynte med fortsettelse av kravanalyse. Kravene ble sortert i funksjonelle og ikke funksjonelle krav. Noen av kravene ble satt til underkrav slik at vi fikk færre hovedkrav. Videre har vi laget en backupløsning for produktet i form av et Visio diagram.

Use-case diagrammet ble oppdatert til et større, mer fullstendig diagram med flere funksjoner. Videre ble det utviklet et sekvensdiagram som ga oss en bedre oversikt over hvilke steg programmet skal utføre gjennom de ulike fasene i prosessen. Sekvensdiagrammet hjalp oss med å bli enige om rekkefølgen av systemets operasjoner slik at ingen har andre meninger om hvordan systemet skal fungere.

Systemarkitekturen visualiserte funksjoner til separate komponenter og dens avhengighet av andre komponenter.

Konseptene ble animert i 3D ved hjelp av SolidWorks programvare.

Utvalgte komponenter for målemetoder og mikrokontrollere ble lagt inn på separate pughmatriser der en tydelig forskjell mellom fordeler og ulemper ble fremvist.

Dokumenter som ble opprettet iløpet av iterasjon

- Kravtabell
- Use-case
- Oppdatert prosjektplan
- Riskanalyse for krav (backup plan)
- Systemarkitektur
- Sekvensdiagram
- Pughmatrise for målemetoder
- Pughmatrise for mikrokontroller

Iterasjon 2.0			
Aktivitet	**Start**	**Slutt**	**Varighet**
Kravanalyse og spesifikasjoner (funksjonelle eller ikke)	12. feb.	19. feb.	7
Use-case utfylling (minst 80% ferdig)	14. feb.	15. feb.	1
Konsepter animert i 3D (første utkast)	14. feb.	23. feb.	9
Oppdatering av prosjektplan	14. feb.	21. feb.	7
Pugh for målemetoder (3-4 typer)	14. feb.	22. feb.	8
Riskanalyse for krav (backup plan)	15. feb.	16. feb.	1
System arkitektur	19. feb.	22. feb.	3
Sekvensdiagrammer	19. feb.	26. feb.	7
Pugh for mikrokontroller	19. feb.	2. mar.	11

Tabell 3.5: Tidstabell for iterasjon 2.0

Utfordring med konseptanimering i 3D var at ingen i gruppa vår hadde noe erfaring med slike programmer. Et medlem fra gruppa tok på seg jobben med SolidWorks-programmet og vi fikk et imponerende første utkast av animasjon for endelig konsept.

Gruppa hadde en del diskusjoner og teorier om målemetoder. Løsningen på dette problemet var utvikling av pughmatriser, noe som viste seg å være et godt evalueringsverktøy for valg av målekomponenter.

Mikrokontroller/Mikroprosessor viste seg å være en stor utfordring siden en slik komponent skal være muskelen og hjernen til hele systemet. Valget sto mellom tre komponenter; Raspberry Pi 3, Arduino Mega og BeagleBone Black. En pughmatrise for disse tre komponentene viste seg å bidra som et nyttig hjelpeverktøy. Vi valgte en mer utfordrende men lærerik vei, nemlig å bruke Raspberry Pi 3 som hjernen vår. Ulempen med valget vårt var at Raspberry Pi er avhengig av et operativsystem og kontrollpinner må konfigureres og kodes med Python som programmeringsspråk. Heldigvis har Windows utviklet et operativsystem tilpasset Raspberry Pi, dermed kan vi bruke Visual Studios med Python utvidelse som plattform for koding.

3.3.4 Iterasjon 2.1

1. Hovedmål

Høydepunkter for iterasjon

- Valg av gjenstående komponenter
- Videreutvikle eksisterende businessplan
- Kjøpsliste for komponenter
- Skissere grafisk brukergrensesnitt

2. Gjennomføring

Fra iterasjon 2.0 fortsatte elektrostudentene med komponentvalg. Pughmatriser for flesteparten av gjenstående komponenter ble utviklet. Totalt energibudsjett ble beregnet for videre valg av spenningskilde.

Utdypning av eksisterende businessplan og første utkast for grafisk brukergrensesnitt ble produsert under iterasjon 2.1.

Dokumenter som ble opprettet iløpet av iterasjon

- Businessplan (oppdatert versjon)
- Grafisk brukergrensesnitt (første versjon)
- Pughmatrise for kommunikasjon
- Pughmatrise for motor
- Energibudsjett
- Pughmatrise for batteri
- Bill Of Materials
- Utstyrsliste (første versjon)

Iterasjon 2.1			
Aktivitet	**Start**	**Slutt**	**Varighet**
Pugh for kommunikasjon	22. feb.	23. feb.	1
Pugh for motor	22. feb.	28. feb.	6
Businessplan v2	26. feb.	2. mar.	4
Utstyrsliste (Spesifikasjoner)	28. feb.	1. mar.	1
Graphical User interface V.1	5. mar.	9. mar.	4
Pugh for batteri	7. mar.	12. mar.	5
Energi budsjett	8. mar.	9. mar.	1
Bill of material	7. mar.	9. mar.	2
DC omformere	8. mar.	9. mar.	1
Sikkerhetssensor	8. mar.	9. mar.	1
Batterilader	8. mar.	9. mar.	1

Tabell 3.6: Tidstabell for iterasjon 2.1

Hovedfokuset hos elektorstudentene under denne iterasjonen gikk til motor- og batterivalg. Vi har sett på mange forskjellige motorer og konklusjonen var at en spesifikk motor produsert fra et selskap som heter Teknik passet oss best. Motoren blir produsert i USA og det kan være utfordring med leveringstiden.

Etter at alt av komponentvalg ble fullført så måtte vi beregne hvor stor spenningskilde vi trenger til å styre systemet. Vi bestemte at spenningskilden skal ha nok energi til systemet for en hel arbeidsdag.

Batterivalget ble en utfordring, siden økonomibudsjettet er begrenset. Noen av batteriene vi har sett på er litiumbatterier. Litiumbatteriene er teknologisk avanserte med lav vekt og størrelse i forhold til mer konvensjonelle batterier som blybatterier. Valget vårt endte på blybatterier på grunn av tilgjengelighet og pris. Ulempen er vekt og størrelse, men det vil ikke ha så stor innvirkning på vårt prosjekt.

3.3.5 Iterasjon 2.2

1. Hovedmål

Høydepunkter for iterasjon

- Lage andre versjon av brukergrensesnitt
- Lage testplan
- Utfylle sekvensdiagrammer
- Animere endelig konsept i 3D
- Oppdatere utstyrsliste
- Oppdatere prosjektplan
- Lage en backupløsning
- Lage et klassediagram

2. Gjennomføring

Detaljert testplan og beskrivelse ble produsert under denne iterasjonen.

Klassediagram ble produsert for å få et bedre innblikk i inndeling av moduler og hvordan disse skal kommunisere for å oppnå en best mulig programvare og autonomt system.

Utstyrsliste ble oppdatert med tilhørende bilder, med beskrivelse av funksjonalitet og bruksområde til valgte komponenter.

Backupløsning ble produsert ved hjelp av Visio. Diagrammet visualiserer primær og sekundær løsning til systemet i forhold til noen gitte krav.

Dokumenter som ble opprettet iløpet av iterasjon

- Testplan og beskrivelse
- Ladekontakt
- Klassediagram
- Iterasjon- og evalueringsdokumenter
- Gantt (Oppdatert versjon)
- Backupløsning (I forhold til krav)
- Sekvensdiagram for målingsrunde
- Sekvensdiagram for sammenligning av målingsresultater
- Sekvensdiagram for innlogging
- Utstyrsliste (Oppdatert versjon)

Iterasjon 2.2			
Aktivitet	**Start**	**Slutt**	**Varighet**
Testanalyse (Testplan og beskrivelse)	12. mar.	15. mar.	3
Ladekontakt (Magnetisk)	12. mar.	13. mar.	1
Klassediagram (Program arkitektur)	12. mar.	23. mar.	11
Grafisk brukergrensesnitt (andre versjon)	12. mar.	8. apr.	27
Endelig konsept i 3D	14. mar.	21. mar.	7
Prosjektplan (Iterasjoner og evaluering, gantt)	15. mar.	23. mar.	8
Buckupløsning (I forhold til krav)	16. mar.	21. mar.	5
Sekvensdiagram for målingsrunde	19. mar.	22. mar.	3
Sekvensdiagram for sammenligning av målingsresultater	19. mar.	22. mar.	3
Sekvensdiagram for innlogging	22. mar.	23. mar.	1
Utstyrsliste (Oppdatert versjon)	23. mar.	25. mar.	2

Tabell 3.7: Tidstabell for iterasjon 2.2

Utfordringen gjennom denne iterasjonen ble produksjon av grafisk brukergrensesnitt. Ansvarlig datastudent har vært uheldig og fikk fullstendig pc krasj. Siden skolens datamaskiner ikke er utstyrt med Visual Studios så måtte vår datastudent anskaffe en annen datamaskin.

Iterasjon 2.3 blir satt til dokumentredigering og øving til andre presentasjon.

3.3.6 Iterasjon 2.3

1. Hovedmål

Høydepunkter for iterasjon

- Ferdigstille krav
- Bestilling av valgte komponenter
- Websideutvikling
- Rettskriving av dokumentasjon
- Forberedelse til andre presentasjon

2. Gjennomføring

Iterasjonen begynte med bestilling av valgte komponenter slik at vi kan forberede oss til konstruksjonsfasen.

Dokumenter som ble opprettet i løpet av iterasjon

- Rapport v2.0

3. Evaluering

Iterasjon 2.3			
Aktivitet	**Start**	**Slutt**	**Varighet**
Ferdigstille krav (forklaring)	26.mar	29.mar	3
Web-design (andre versjon)	3. apr.	6. apr.	3
Dokumentredigering/forklaring	26. mar.	6. apr.	11

Tabell 3.8: Tidstabell for iterasjon 2.3

Iterasjon 2.3 ble satt til dokumentredigering og øving til andre presentasjon.

3.3.7 Iterasjon 3.0

1. Hovedmål

Høydepunkter for iterasjon

- Visualisere komponentvalg
- Lage skripter for valgte komponenter
- Lage skript for rapportgenerering og utregningsmodul
- Utforske muligheter for Web-basert brukergrensesnitt

2. Gjennomføring
Detaljert komponentdiagram og beskrivelse ble produsert under denne iterasjonen.

Skripter for valgte komponenter ble laget av elektrostudenter på Visual Studio Code med Python som kodespråk. Videre ble valgte komponenter testet.

Skripter for rapportgenerering og utregningsmodul ble laget av datastudenter på Visual Studio med C# som kodespråk.

Utforskning og installasjon av Node-RED på Raspberry Pi ble utført. Etter vår mening er dette det første steget mot IoT i prosjektet.

Azure-tjenestene ble utforsket med oppretting av bruker samt testing av relevante tjenester som IoT Hub og Storage.

Dokumenter som ble opprettet iløpet av iterasjon

- Komponentdiagram
- Utregningsmodul til RangeFinder
- Skript for motorstyring
- Skript for encoder og proximity sensorer
- Skript for ultrasonic sensor
- Web-basert brukergrensesnitt
- Skript for rapportgenerering

Iterasjon 3.0			
Aktivitet	**Start**	**Slutt**	**Varighet**
Komponentdiagram	11. apr.	13. apr.	2
Klargjøring av Raspberry Pi	12. apr.	14. apr.	2
Kalkuleringskode til Rangefinder (Utregningsmodul)	13. apr.	20. apr.	7
Pythonkode for motorstyring	16. apr.	26. apr.	10
Pythonkode for Encoder og Proximity sensor	16. apr.	26. apr.	10
Pythonkode for Ultrasonic sensor	16. apr.	20. apr.	4
Node-Red oppsett for Rpi	18. apr.	19. apr.	1
Azure utforskning	18. apr.	20. apr.	2
Web-Interface (UI flyttes til sky)	18. apr.	26. apr.	8
Azure IoT Hub og Storage	19. apr.	26. apr.	7
Rapportgenerering	19. apr.	7. mai.	18

Tabell 3.9: Tidstabell for iterasjon 3.0

Utfordringen i denne iterasjonen for elektrostudenter var å lage skripter i Python for nødvendige komponenter. Videre ble Raspberry Pi konfigurert og klargjort for Node-RED bruk. Azure-tjenestene ble utforsket, og for å involvere mest mulig IoT-løsninger i dette prosjektet skulle brukergrensesnittet flyttes til en skybasert tjeneste. Etter en viss tid viste det seg at denne løsningen var ganske komplisert, og på grunn av tidsmangel for å fullføre overgangen måtte gruppa fokusere på tidligere løsning, nemlig en applikasjon for brukergrensesnittet.

3.3.8 Iterasjon 3.1

1. Hovedmål

Høydepunkter for iterasjon

- Lage deler i 2D og 3D
- Node-RED med MQTT
- Kretstegning for valgte komponenter
- Oppdatering av brukergrensesnittet
- Montering av valgte komponenter hos oppdragsgiver

2. Gjennomføring
Iterasjonen begynte med 3D-tegning av deler i SolidWorks. Siden det er ingen maskinstudenter i gruppa vår så måtte en elektrostudent ta for seg jobben. Heldigvis har et medlem i gruppen har vært borti SolidWorks tidligere. Videre ble det produsert en del i 2D for laserkutting, delen ble brukt til å feste motoren til kalibreringsvogn.

Videre utforskning av Node-RED viste seg å være nyttig. Ved bruk av MQTT node på Node-RED kunne vi enkelt sende og motta meldinger mellom Raspberry Pi og PC.

Forbedringer i skripter til valgte komponenter har følget hele veien fra iterasjon 3.0.

Brukergrensesnittet ble delt i to deler, der designdelen ble utført først og funksjoner for brukergrensesnittet deretter.

Dokumenter som ble opprettet iløpet av iterasjon

- 3D tegninger av deler for 3D printing i SolidWorks
- Kretstegning for valgte komponenter
- Brukergrensesnitt (design)
- 2D tegninger av deler for laserkutting i SolidWorks
- Brukergrensesnitt (funksjoner)

Iterasjon 3.1			
Aktivitet	**Start**	**Slutt**	**Varighet**
Lage 3D deler i SolidWorks	26. apr.	30. apr.	4
Node-Red med MQTT	30. apr.	4. mai.	4
Kretstegning for Motor, Proximity og Encoder	30. apr.	2. mai.	2
Forbedring i målemetode	30. apr.	14. mai.	14
Endelig User Interface – Design	30. apr.	4. mai.	4
Lage 2D deler i SolidWorks for laserkutting	3. mai.	4. mai.	1
SQL database	3. mai.	10. mai.	7
Endelig User Interface - Funksjoner	4. mai.	14. mai.	10
Montering av fysiske deler hos bedriften	4. mai.	14. mai.	10

Tabell 3.10: Tidstabell for iterasjon 3.1

Datastudenter måtte forkaste Web-basert løsning på grunn av tidsmangel, det vil si at brukergrensesnittet skal beholdes som PC-applikasjon.

Utfordring for elektrostudentene gjennom denne iterasjonen var montering av valgte komponenter hos oppdragsgiver. En del justeringer av eksisterende system måtte utføres. Videre viste det seg at 3D printede deler var for svake til å håndtere kraften som ble påført fra motoren til vognen. Løsningen til dette var modifikasjon av eksisterende deler som bedriften har hatt liggende.

3.3.9 Iterasjon 3.2

1. Hovedmål

Høydepunkter for iterasjon

- Nye deler i 3D
- MQTT utforskning
- Kontroll av vognen med Node-RED
- Kommunikasjon mellom RangeFinderen og Raspberry Pi
- Kalibrering av vogna
- Forbedring i brukergrensesnittet med SQL
- Utførelse og dokumentasjon av tester
- Rapportskriving og retting
- Dokumentlevering

2. Gjennomføring
Iterasjonen begynte med fortsettelse fra iterasjon 3.1 i forhold til komponentmontering, SQL databaseimplementering i brukergrensesnittet og forbedring i måleskript.

Oppdatering av brukergrensesnitt ble også gjennomført, med nye funksjoner for endring av målepunkter og manuell overstyring. Kalkuleringsmodul ble ferdigstilt og koblet inn i brukergrensesnittet sammen med rapportmodulen.

Etter delene som var printet i 3D gikk i stykker måtte gruppen finne på en annen løsning for å feste motorens rotor til vognen. Som nevnt i iterasjon 3.1 har eksisterende deler og verktøy blitt tatt i bruk. Videre så har vi forbedret 3D tegning for motorfeste slik at delene skal holde beregnet kraft.

Eksisterende kommunikasjonsmetode mellom RangeFinderen og PC har blitt utforsket.

MQTT har blitt tatt i bruk med Node-RED og PC, mens skript som ble laget i C# for MQTT hadde vansker å koble seg til bestemt server.

Tester har blitt utført og dokumentert.

Dokumenter som ble opprettet iløpet av iterasjon

- Forbedret deler i 3D
- MQTT bruk
- Node-RED nodene og implementasjon i prosjektet
- Utførte testtabeller
- Raspberry Pi oppsetting for UART pinner
- Forbedring i utregningsmodul
- Rapport v3

Iterasjon 3.2			
Aktivitet	**Start**	**Slutt**	**Varighet**
Forbedring i 3D deler	7. mai.	8. mai.	1
MQTT	7. mai.	11. mai.	4
Node-RED	7. mai.	14. mai.	7
Kalibrering av vogna	8. mai.	14. mai.	6
Testutførelse og godkjenning	9. mai.	16. mai.	3
Kommunikasjon mellom RF og HAT	11. mai.	12. mai.	1
Forbedring i kode for utregningsmodul	11. mai.	14. mai.	3
Rapportskriving og retting	14.mai	21.mai	7
Dokumentlevering	21.mai	22.mai	1

Tabell 3.11: Tidstabell for iterasjon 3.2

Som nevnt ovenfor hadde vi vansker med å koble oss til bestemt server med C skript og istedenfor MQTT har vi gått over til lagring av kommandoer fra brukergrensesnittet i et tekstfilformat. Som følge av dette har vi også mulighet til å sette inn talverdier fra et tekstdokument inn i utregningsmodulen. For overføring av slike filer har vi brukt skybasert lagringstjeneste, Dropbox. Filene blir lastet ned av Raspberry Pi og konvertert til JSON format for utførelse av bestemte funksjoner i Python skripter.

Kommunikasjon og lesing av data fra RangeFinderen viste seg til å være stor utfordring. Vi har prøvd oss frem med eksisterende kommunikasjonsmetode, nemlig Moxa sis USB til RS 232/422 adapter uten lykke. For å klare og oppfylle kravet til kommunikasjon mellom RangeFinderen og Rasspberry Pi måtte vi ta reserveløsning i bruk. Vi bestilte en komponent som heter RS 422/485 HAT. Etter oppsetting for UART pinner i Raspberry Pi kunne vi lese signalene fra RangeFinderen.

Siste uke av Iterasjon 3.2 ble satt til dokumentredigering og rapportskriving.

4 RISIKOANALYSE

Dokumenthistorie			
Versjon	**Dato**	**Endret av**	**Aktivitet**
0.1	22/02/18	TT	Dokument opprettet
0.2	24/02/18	TT	Prioritering, konsekvens, sannsynlighet
0.3	25/02/18	TT	Risikotabell, risikomatrise
0.4	27/02/18	TT	Justering av tabeller
0.5	06/02/18	TT	Revidert for publisering
1.0	07/02/18	TT	Publisert i rapport v1.0
2.0	06/04/18	TT	Publisert i rapport v2.0
2.1	20/04/18	TT	Inntrufne hendelser
2.2	21/04/18	VS	Revidert for publisering
3.0	21/04/18	TT	Publisert i rapport v3.0

Tabell 4.1: Dokumenthistorie for risikoanalyse

4.1 Innledning

Formålet med en risikoanalyse er å identifisere risikoen ved hver enkelt hendelse som kan forekomme gjennom prosjektet. En risiko som inntreffer kan påvirke prosjektets delmål, som kan føre til at vi må bruke mer tid enn planlagt på prosesser og dermed forsinke prosjektets fremgang. Gjennom risikoanalyse av mulige hendelser, får vi et klarere bilde av hva som kan gå galt. Da er man bedre forberedt til å unngå uønskede hendelser eller begrense skader hvis de inntreffer.

4.2 Sannsynlighet

Tabellen viser hendelsenes sannsynlighet for en gitt tidsperiode.

Sannsynlighet	**Beskrivelse**	**Verdi**
Svært liten	Mindre enn en hendelse hver tusende time	1
Liten	Ca. en hendelse hver tusende time	2
Middels	Ca. en hendelse hver hundrende time	3
Stor	Ca. en hendelse hver tiende time	4
Svært stor	Mer enn en hendelse hver tiende time	5

Tabell 4.2: Risikosannsynlighet

4.3 Konsekvens

Hendelser har ulike konsekvenser for prosjektets fremgang, dette er vist i tabellen nedenfor.

Konsekvens	Beskrivelse	Verdi
Svært liten	Ingen påvirkning på prosjektet	1
Liten	Prosjektet påvirkes i liten grad	2
Middels	Prosjektet bremses, noe tiltak bør settes inn	3
Stor	Prosjektet stagnerer, tiltak er nødvendig	4
Svært stor	Prosjektet er i kritisk tilstand, tiltak må iverksettes øyeblikkelig	5

Tabell 4.3: Risiko konsekvens

4.4 Risikomatrise

Risikoverdien er basert på sannsynlighetsverdi multiplisert med konsekvensverdi. Denne verdien vil fortelle oss hvor kritisk denne risikoen er.

	Konsekvens				
Sannsynlighet	**Svært liten**	**Liten**	**Middels**	**Stor**	**Svært stor**
Svært liten	1	2	3	4	5
Liten	2	4	6	8	10
Middels	3	6	9	12	15
Stor	4	8	12	16	20
Svært stor	5	10	15	20	25

Tabell 4.4: Risiko matrise

Risikoen er delt inn i tre faser: lav risiko, middels risiko og høy risiko. Ut i fra dette kan vi se hva handlingsforløpet vårt blir ved hver hendelse.

Lav risiko	Risiko er lav, tiltak er ikke nødvendig
Middels risiko	Risiko er middels, tiltak vurderes i enkelte tilfeller
Høy risiko	Risiko er høy, tiltak må settes in

4.5 Risikotabell

Tabellen viser de forskjellige hendelsene, beregnet risikoverdi, konsekvens og hvilket tiltak som må gjøres. S står her for sannsynlighetsgrad og K for konsekvensgrad.

Hendelse	S	K	Sum	Konsekvens	Tiltak
Generell					
Veileder må avbryte samarbeid	1	3	3	Mister viktig samarbeidspartner	Kommunisere med skolen om løsning
Medlem trekker seg	1	5	5	Større arbeidsmengde for resten	Håndere, planlegge på nytt
Bedrift går konkurs	1	5	5	Mister oppdragsgiver & oppgave	Kommunisere med skolen om løsning
Langvarig sykdom	1	5	5	Arbeid må gjøres av andre	Se på alternative løsninger
Konfidensiell informasjon	1	5	5	Bedriften avbryter kontrakten	Avtale om offentliggjøring av informasjon
Kortvarig sykdom	3	2	6	Arbeid må gjøres av andre	Håndtere
Dårlig kommunikasjon	2	3	6	Overlapping av arbeid, lav gruppekjemi	Ta opp problemer
Lav motivasjon	2	3	6	Lav effektivitet, dårlig oppmøte	Støtte hverandre
For sent til møter	3	2	6	Begrenset effektivitet & kommunikasjon	Ta opp hvis det skjer jevnlig
Skjev arbeidsfordeling	3	3	9	Ujevn arbeidsmengde, problem med frister	Ta opp, fordele på nytt
Overholder ikke tidsfrister	2	5	10	Dårligere resultat, nedsatt vurdering	Begrunne, lære av feil
Dårlig validering	2	5	10	Uønsket resultat ift. oppdragsgiver	Gjøre nødvendige/mulige endringer
Dårlig planlegging	3	4	12	Ujevne arbeidsdager, dårlig kommunikasjon	Bruke nok tid til planlegging
Kunnskapsmangel	4	3	12	Mindre effektivitet	Research
Teknisk					
Sen levering av komponenter	2	3	6	Tilhørende arbeid blir satt på vent	Bestille fra flere steder
Utstyrskade under testing	2	4	8	Tilhørende arbeid blir satt på vent	Be om nytt utstyr fra bedriften
PC problemer	2	4	8	Treg ytelse, lite fremgang i prosjektet	Antivirus, rense harddisk jevnlig
Defekte komponenter	3	3	9	Tilhørende arbeid blir satt på vent	Bestille nytt komponent
Tap av dokument	2	5	10	Dobbeltarbeid, mister oversikt	Backup av dokumenter

Tabell 4.5: Risikotabell

4.6 Inntrufne hendelser

Gjennom prosjektet har ulike hendelser som er listet opp i risikotabellen inntruffet. Noen av hendelsene hadde påvirkning på prosjektets fremgang og svekket prosjektets totale helhet. Blant annet førte dette til at vi fikk begrenset med tid til å gjøre systemet optimalt slik vi ønsket, og man ser at det ikke ble lagt inn nok tid til feilmarginer for å oppnå det ønskede resultatet.

Disse hendelsene påvirket prosjektets fremgang:

- **Kunnskapsmangel**

Vi oppdaget underveis i arbeidet at flere av komponenter og programmer som krevde mer research enn antatt. Vi måtte opparbeide oss nok kunnskap før vi kunne ta de i bruk i systemet, slik at de ble anvendt på riktig måte. Tekniske løsninger måtte endres for å kunne få et funksjonelt produkt.

- **Dårlig planlegging**

Etter hvert som prosjektet gikk fremover hopet det seg opp flere og flere uforutsette problemstillinger som vi måtte ta stilling til. Det ble brukt mer tid på enkelte oppgaver enn planlagt. Nye oppgaver ble utsatt, på bakgrunn av at tidligere oppgaver krevde mer tid enn antatt. Vi havnet dermed på etterskudd og fant oss i en tidsklemme på slutten.

- **Sen levering av komponenter**

De bestilte komponentene som kreves for å kunne oppfylle produktet, ble ikke levert i tide. Dette forårsaket at vi ikke klarte å oppfylle kravet K-5.1.

- **Utstyr skade under testing**

Utstyret som ble brukt til å feste motoren til beltet på vogna ble ødelagt flere ganger. Det er utstyr som har blitt printet av en 3D-printer. Det tok mye tid før vi fikk printet ut nye deler, som gjorde at en del arbeid ble satt på vent.

Underveis i prosjektet var det også mindre risikable hendelser som inntraff. F.eks kortvarig sykdom hos flere medlemmer, tekniske problemer med PC og forsinket oppmøte. Disse hadde ikke store konsekvenser for prosjektets fremgang

5 KRAV

Dokumenthistorie			
Versjon	**Dato**	**Endret av**	**Aktivitet**
0.1	01/02/18	TT	Dokument opprettet
0.2	02/02/18	VS	Kravanalyse
0.3	02/02/18	TT	Gruppering, prioritering
0.4	06/02/18	VS	Revidert for publisering
1.0	07/02/18	TT	Publisert i rapport v1.0
1.1	14/02/18	TT	Kravendring
1.2	15/02/18	TT	Innledning, utredning
1.3	16/02/18	TT	Tabeller
1.4	19/02/18	DK	Visio oversikt
1.5	07/03/18	VS	Funkjonell/ikke-funksjonell
1.6	15/03/18	DK	Backup løsning
1.7	23/03/18	TT	Hovedkrav, underkrav
1.8	04/04/18	TT	Revidert for publisering
2.0	06/04/18	TT	Publisert i rapport v2.0
3.0	21/05/18	TT	Publisert i rapport v3.0

Tabell 5.1: Dokumenthistorie for krav

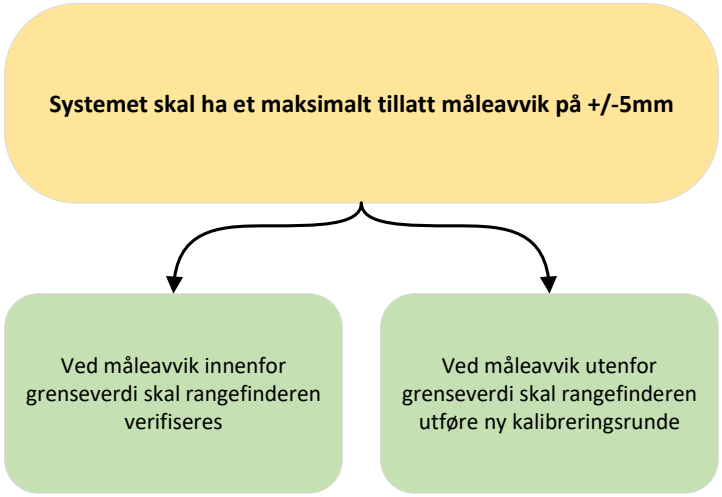
5.1 Innledning

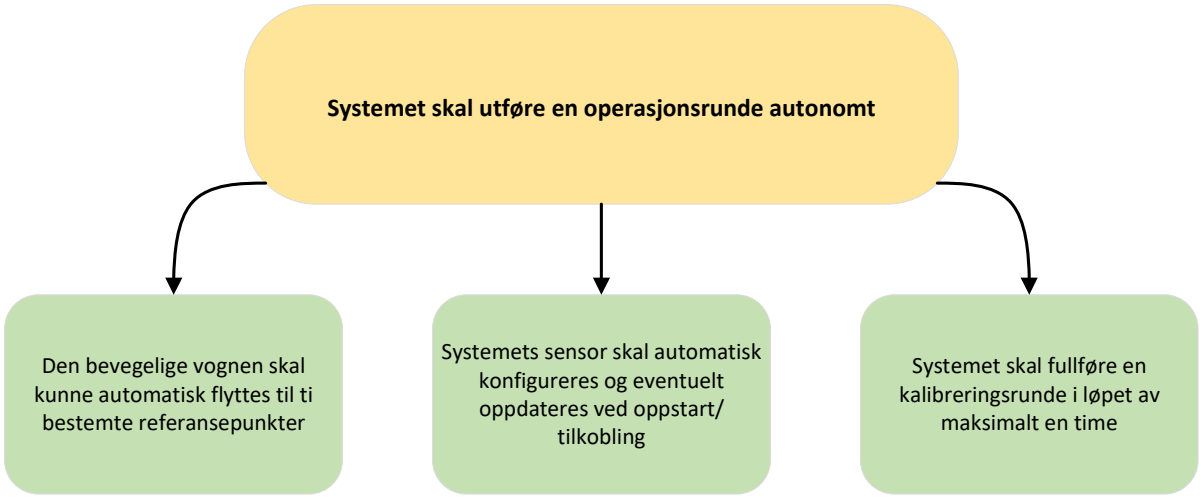
For at en gruppe ingeniørstudenter med ulike fagfelt skal fungere gjennom et prosjekt med så stort omfang, så er målbare krav å jobbe mot helt avgjørende. Dette dokumentet skal gi en oversikt over kravene vi har mottatt fra oppdragsgiver MIROS AS og noen krav vi har utredet selv da vi mener disse vil gi systemet forbedret funksjonalitet.

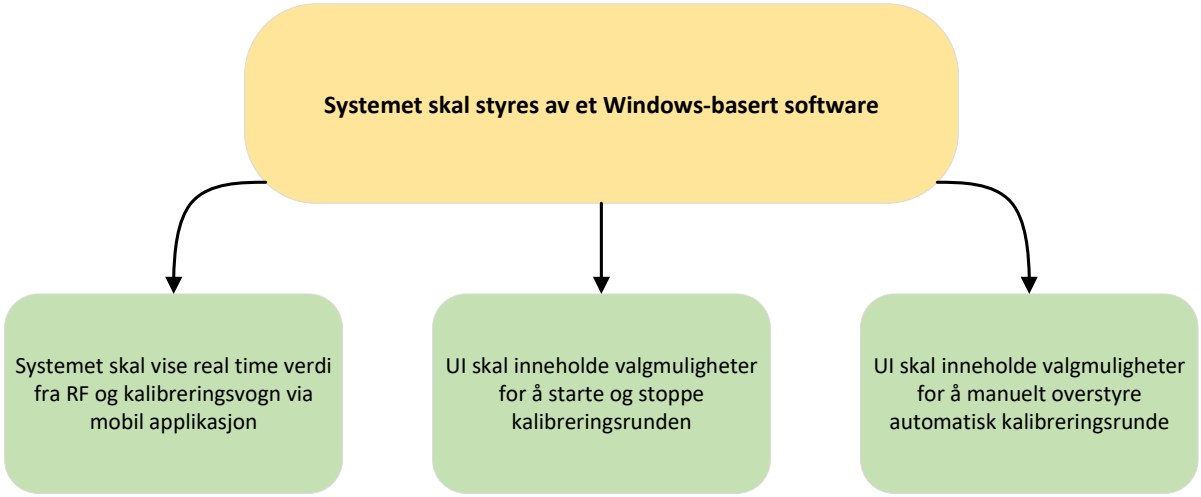
5.2 Utreding

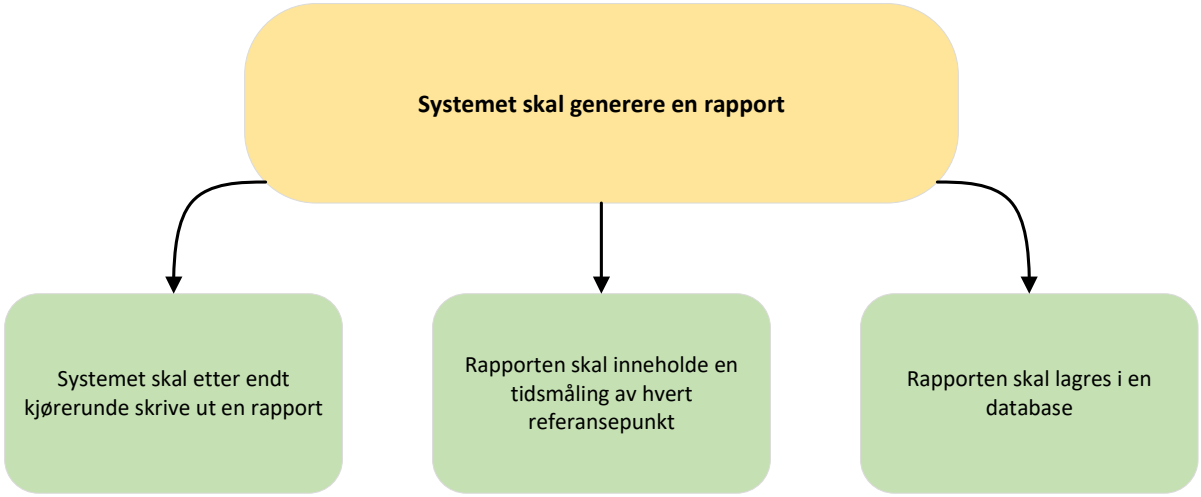
Når vi mottok oppgavedefinisjonen og kravene fra oppdragsgiver, bestemte vi for å dele dem opp i hovedkrav og underkrav. Underkravene bygges opp under hovedkravene, og gir en klar oversikt over hovedkravets omfang. Hovedkravet kan bli oppfylt selv om ikke alle underkravene er oppfylt. Om dette er tilfelle vil vi ikke anse at hovedkravet er optimalt oppfylt, med ønsket funksjonalitet og kvalitet.

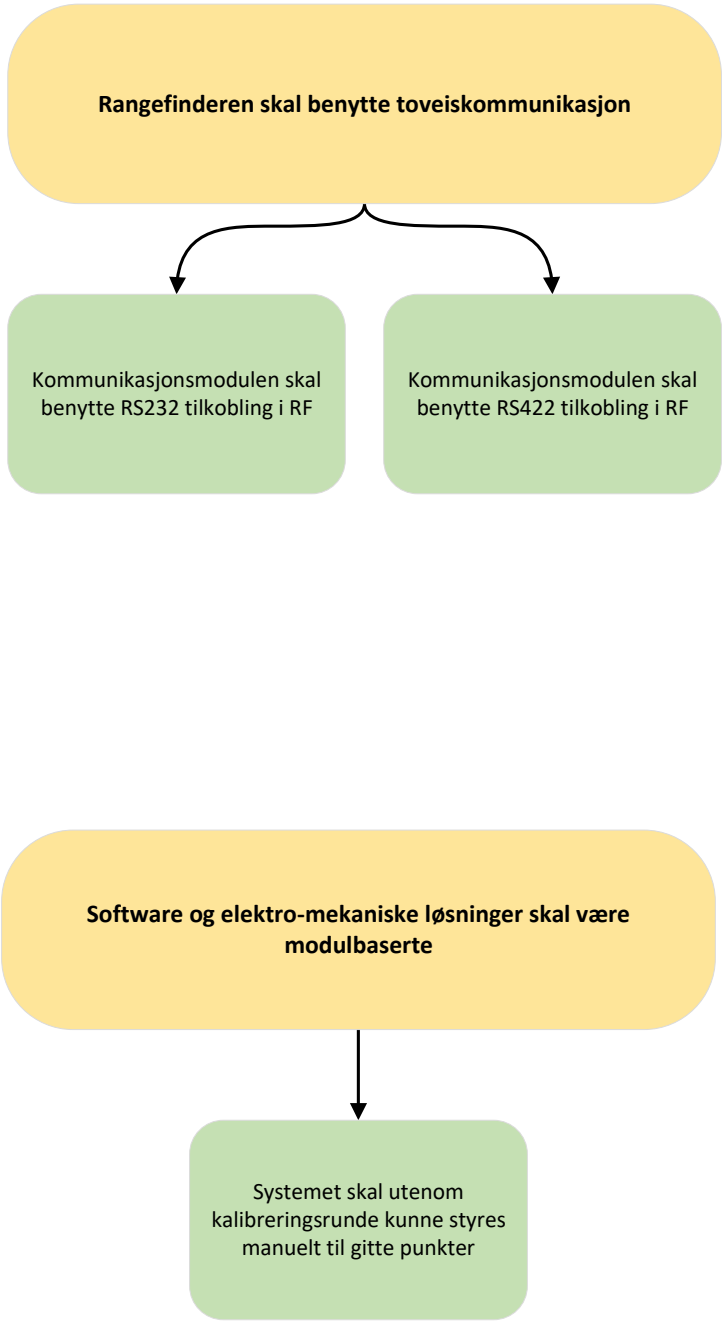
Fremgangsmåten vi benyttet for å løse denne oppgaven gikk ut på å utvikle et Visio-diagram for hvert hovedkrav. Vi har 6 hovedkrav og 14 underkrav. Piler mellom boksene i diagrammene beskriver direkte relasjoner mellom hovedkrav og underkrav.


Systemet skal ha et maksimalt tillatt måleavvik på +/-5mm
Ved måleavvik innenfor grenseverdi skal rangefinderen verifiseres
Ved måleavvik utenfor grenseverdi skal rangefinderen utføre ny kalibreringsrunde


Systemet skal utføre en operasjonsrunde autonomt
Den bevegelige vognen skal kunne automatisk flyttes til ti bestemte referansepunkter
Systemets sensor skal automatisk konfigureres og eventuelt oppdateres ved oppstart/ tilkobling
Systemet skal fullføre en kalibreringsrunde i løpet av maksimalt en time


Systemet skal styres av et Windows-basert software
Systemet skal vise real time verdi fra RF og kalibreringsvogn via mobil applikasjon
UI skal inneholde valgmuligheter for å starte og stoppe kalibreringsrunden
UI skal inneholde valgmuligheter for å manuelt overstyre automatisk kalibreringsrunde


Systemet skal generere en rapport
Systemet skal etter endt kjørerunde skrive ut en rapport
Rapporten skal inneholde en tidsmåling av hvert referansepunkt
Rapporten skal lagres i en database



Figur 5.1: Kravoversikt

5.3 Prioritet

Prioritet er svært viktig når krav skal analyseres. Ved grundig drøfting av kravprioritet vil vi få en klarere oversikt over de mer tekniske risikofaktorene og få et klarere skille mellom absolutt nødvendig funksjonalitet og ønsket funksjonalitet.

Prioritet til krav har vi valgt å dele inn i tre grupper. Disse gruppene er navngitt A, B og C for å gjøre det enklest mulig for gruppen å holde oversikt.

Krav med prioritet A ser vi på som høyst essensielle for prosjektet vårt. En løsning uten å ha oppfylt alle krav med A-prioritet kommer til å ha store mangler og vil høyst sannsynlig ikke bli regnet som funksjonelt. Risikovurdering må tas nøye og det må handles tidlig deretter i forhold til disse kravene.

Krav med prioritet B er viktige elementer som bør være med i den endelige løsningen. Produktet vil kunne fungere uten et av disse kravene, men løsningen vil hemmes og andre deler av systemet vil ikke nødvendigvis fungere like godt uten.

Krav med prioritet C er krav vi ønsker å ha med i systemet vårt dersom tid og ressurser strekker til. Disse er ønskelig for å oppnå optimalt resultat, men vil ikke prioriteres med mindre vi har god nok kontroll på A- og B-kravene våre. En løsning uten C-krav bør ikke ha for stor konsekvens for systemet.

Prioritet	Beskrivelse
A	Krav som er nødvendig for et funksjonelt system utifra oppgavedefinisjon
B	Krav som bør være med for at systemet skal oppnå ønsket kvalitet
C	Krav som kan være med for at systemet skal oppnå optimal kvalitet

Tabell 5.2: Kravprioritet

5.4 Funksjonell/ikke-funksjonell

Funksjonelle krav

Funksjonelle krav beskriver i korte trekk hva et system skal gjøre. Kravene skal beskrive funksjonen som skal utføres ut fra en gitt situasjon. «Systemet skal generere en rapport» er et krav som beskriver en funksjon som utføres etter det bevegelige systemet har fullført målerunden sin. Det er vesentlig at de funksjonelle kravene blir oppfylt eller eventuelt følges opp med en gjennomarbeidet plan b.

Ikke-funksjonelle krav

Ikke-funksjonelle krav beskriver hvordan systemet utfører funksjoner. Disse kravene skal kvalitetssikre funksjoner ved å spesifisere systemets oppførsel under utførelse. «Systemet skal ha et maksimalt tillatt måleavvik på +/- 5mm» er et ikke-funksjonelt krav. Her settes en definert grenseverdi som skal kvalitetssikre systemets operasjoner.

5.5 Krav tabell

Tabellene nedenfor viser de forskjellige hovedkravene og underkravene, beskrivelse av dem, krav ID, prioriteten og interessenten. Det er et kommentarfelt for hvert krav der vi kan tilføye en kommentar om nødvendig.

ID	Prioritet	Hovedkrav	Funksjonell/ ikke funksjonell	Interessent
K - 1	A	Rangefinderen skal ha et maksimalt tillatt måleavvik på +/-5mm	Ikke-Funksjonell	Miros
		Underkrav		
K - 1.1	A	Ved måleavvik innenfor grenseverdi skal rangefinderen verifiseres	Funksjonell	Miros
K - 1.2	A	Ved måleavvik utenfor grenseverdi skal rangefinderen utføre ny kalibreringsrunde	Funksjonell	Miros
		Kommentar		

ID	Prioritet	Hovedkrav	Funksjonell/ ikke funksjonell	Interessent
K - 2	A	Systemet skal utføre en operasjonsrunde autonomt	Ikke-Funksjonell	Miros
		Underkrav		
K - 2.1	A	Den bevegelige vognen skal kunne automatisk flyttes til ti bestemte referansepunkter	Funksjonell	Miros
K - 2.2	B	Systemets sensor skal automatisk konfigureres og eventuelt oppdateres ved oppstart/tilkobling	Ikke-funksjonell	Miros
K - 2.3	B	Systemet skal fullføre en kalibreringsrunde i løpet av maksimalt en time	Ikke-funksjonell	Miros
		Kommentar		

ID	Prioritet	Hovedkrav	Funksjonell/ ikke funksjonell	Interessent
K - 3	A	Systemet skal styres av et Windows-basert software	Funksjonell	Miros
		Underkrav		
K - 3.1	C	Systemet skal vise real time verdi fra RF og kalibreringsvogn via mobil applikasjon	Ikke-funksjonell	COR
K - 3.2	A	UI skal inneholde valgmuligheter for å starte og stoppe kalibreringsrunden	Ikke-funksjonell	Miros
K - 3.3	B	UI skal inneholde valgmuligheter for å manuelt overstyre automatisk kalibreringsrunde	Ikke-funksjonell	Miros
		Kommentar		
K - 3	Windows basert software : Windows 10 IOT			

ID	Prioritet	Hovedkrav	Funksjonell/ ikke funksjonell	Interessent
K - 4	A	Systemet skal generere en rapport	Funksjonell	Miros
		Underkrav		
K - 4.1	B	Systemet skal etter endt kjørerunde skrive ut en rapport	Funksjonell	Miros
K - 4.2	B	Rapporten skal lagres i en database	Funksjonell	Miros
K - 4.3	C	Rapporten skal inneholde en tidsmåling av hvert referansepunkt	Ikke-funksjonell	COR
		Kommentar		
K - 4	Rapporten skal inneholde: målerunde(verdi fra RF og kalibreringsvogn), RF og kalebreringsvogn sammenlikningsgraf, offset og gain, grafisk-avviksverdi før og etter kalibreringsrunde			
K - 4.2	Database: MS-Azure og lokal harddisk			

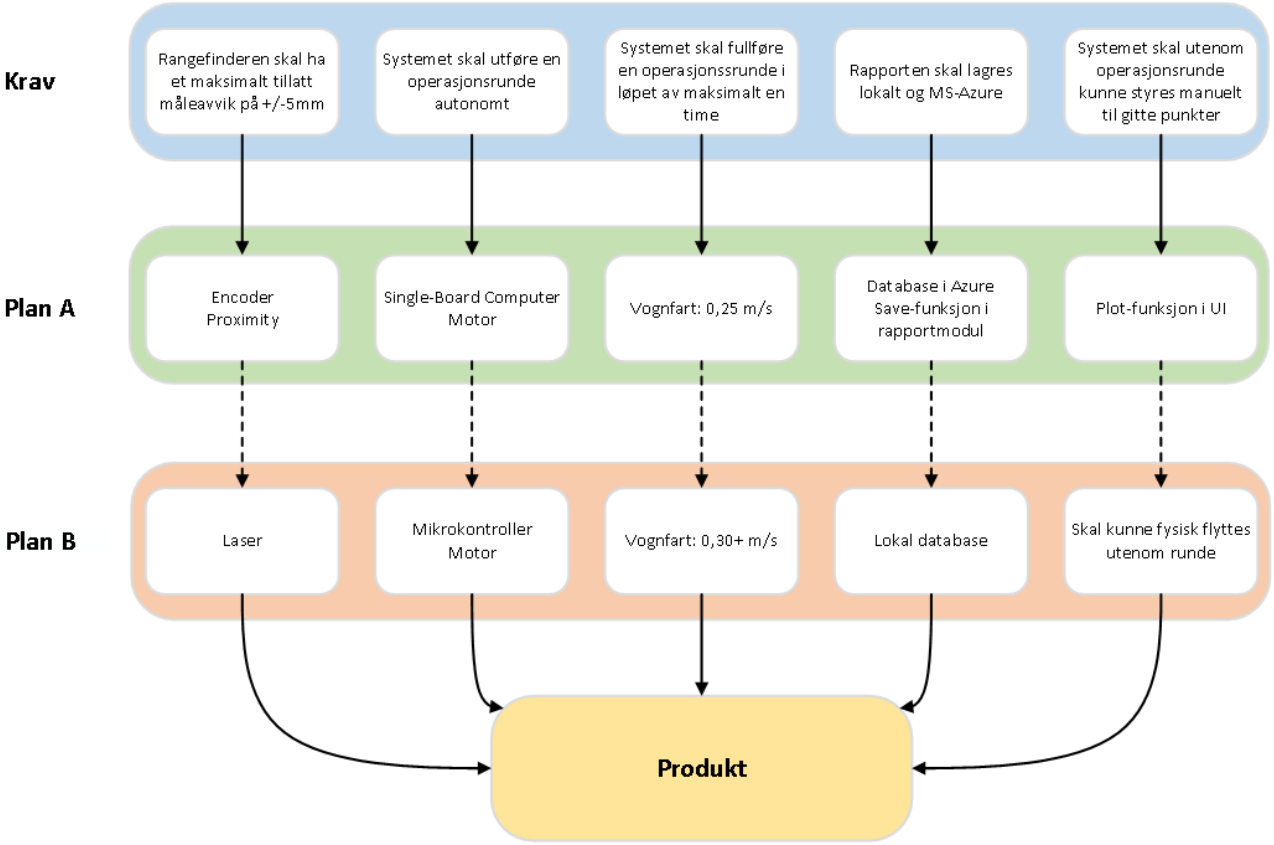
ID	Prioritet	Hovedkrav	Funksjonell/ ikke funksjonell	Interessent
K - 5	A	Rangefinderen skal benytte toveiskommunikasjon	Ikke-funksjonell	Miros
		Underkrav		
K - 5.1	A	Kommunikasjonsmodulen skal benytte RS232 tilkobling i RF	Ikke-funksjonell	Miros
K - 5.2	A	Kommunikasjonsmodulen skal benytte RS422 tilkobling i RF	Ikke-funksjonell	Miros
		Kommentar		

ID	Prioritet	Hovedkrav	Funksjonell/ ikke funksjonell	Interessent
K - 6	B	Software og elektro-mekaniske løsninger skal være modulbaserte	Ikke-funksjonell	Miros
		Underkrav		
K - 6.1	A	Systemet skal utenom kalibreringsrunde kunne styres manuelt til gitte punkter	Funksjonell	Miros
		Kommentar		

Tabell 5.3: Kravtabell

5.6 Backup løsning

Enkelte krav har høyere vanskelighetsgrad enn andre, og kommer til å kreve mer tid og kompetanse for å fullføre. Hvis vi ikke klarer å oppfylle kravet med metoden som vi har planlagt, vil det skape problemer for oss. Derfor har vi laget en "plan b" som vist i fig 5.2 for hvordan vi skal fullføre kravet. Dette er for å forsikre oss om at kravene blir fullført på en eller annen måte, slik at systemet fungerer slik oppdragsgiveren ønsker.



Figur 5.2: backup løsning

6 ARKITEKTUR

Dokumenthistorie			
Versjon	**Dato**	**Endret av**	**Aktivitet**
0.1	05/02/18	VS	Dokument opprettet
0.2	06/02/18	VA	Usecase diagram versjon 1
1.0	07/02/18	VA	Publisert i rapport v1.0
1.1	15/02/18	VA	Usecase diagram versjon 2
1.2	26/02/18	VA	Første utkast sekvensdiagram (hele prosessen)
1.3	06/03/18	VS	Webside opplastning
1.4	08/03/18	VS	Første utkast web-design
1.5	09/03/18	VA	GUI første utkast
1.6	16/03/18	VA	Sekvensdiagram for hele prosessen fullført
1.7	22/03/18	VS	Sekvensdiagram for start av målingsrunde
1.8	22/03/18	VA	Sekvensdiagram for sammenligning av måleresultater
1.9	23/03/18	VS	Sekvensdiagram for innlogging
1.10	23/03/18	VS	Klassediagram første utkast
1.11	04/04/18	VS	Revidert før publisering
2.0	06/04/18	VS	Publisert i rapport v2.0
2.1	16/04/18	VS	La til komponentdiagram
2.3	21/05/18	VS	Revidert før publisering
3.0	21/05/18	VS	Publisert i rapport v3.0

Tabell 6.1: Dokumenthistorie for arkitektur

6.1 Innledning

Systemet vi utvikler skal fungere autonomt ved hjelp av en Windows-basert programvare. Programvaredelen tar for seg alt fra brukergrensesnittet som målerunden settes i gang fra, til autonom styring av vogna, måling av punkter og lagring av resultater.

Dette kapittelet vil ta for seg planleggingen som foregikk i elaborationfasen for implementasjon av disse delene. UML-diagrammer er utviklet for å skape visuelle skisser. Use case diagrammer utarbeides ut fra hvilke prosesser det er tenkt at operatøren skal kunne sette i gang. For hver av disse prosessene utvikles det da sekvensdiagrammer som viser hvordan disse prosessene utvikler seg steg for steg i programvaren. Et klassediagram utvikles videre for å vise hvordan det er tenkt at programvarens moduler skal deles inn og kommunisere med hverandre.

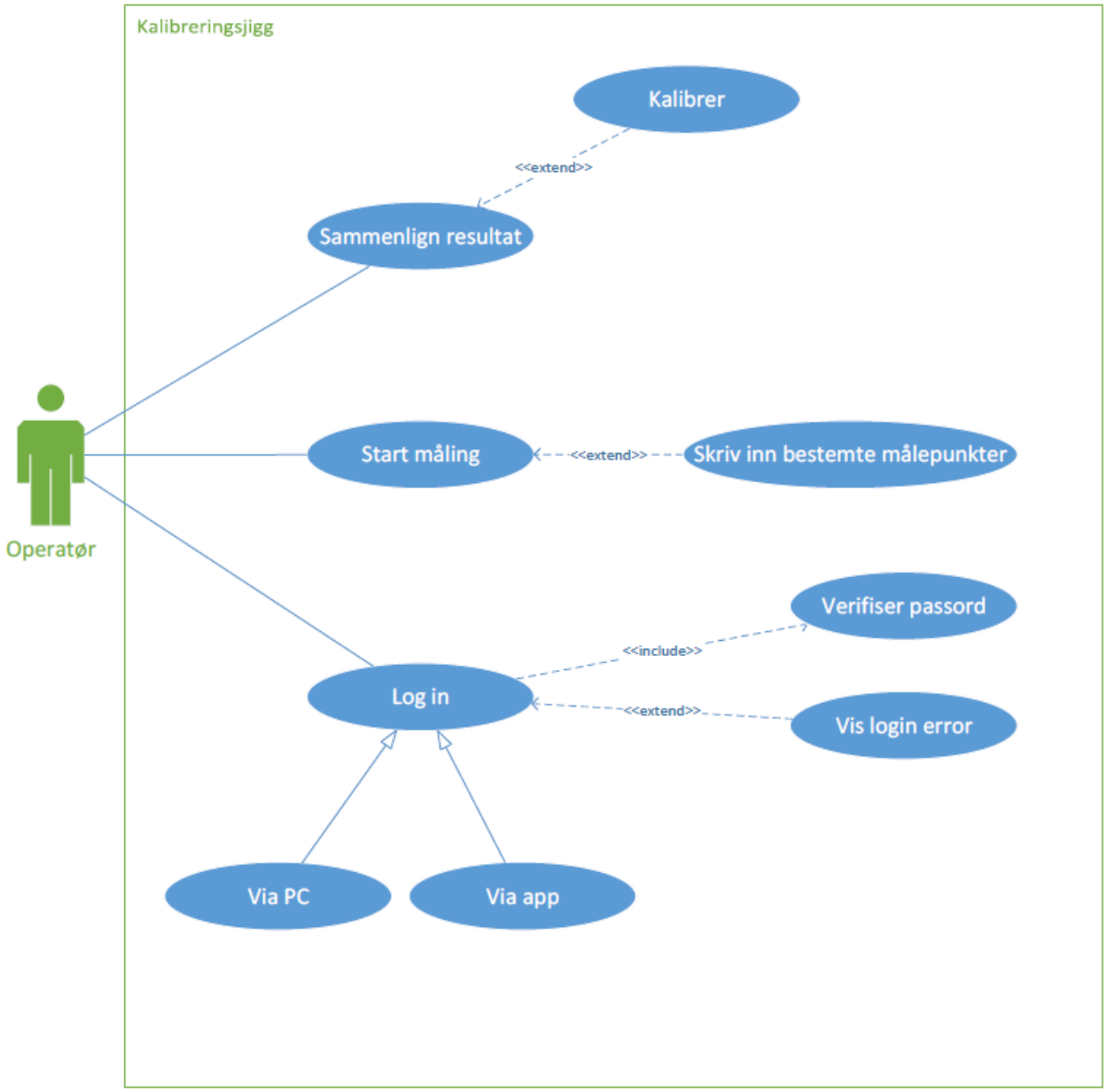
En skisse for tenkt brukergrensesnitt finnes også her, samt en kort forklaring av prosjektgruppens nettside og dens innhold.

6.2 Use case

Vi har utviklet et Use case diagram som vist i fig 6.1, for å skaffe en oversikt over hvilke funksjoner operatøren kan sette igang. Operatøren skal kunne starte en målerunde med analyse av radaren. Målerunden vil bekrefte eller avkrefte at radaren er nøyaktig nok i forhold til den mer nøyaktige normalen som er satt opp. Hvis resultatet er nøyaktig nok vil den bli godkjent og lagret til minnet, samt lastes opp til en server. Hvis målingen er for langt unna gitt normal, må radaren bli kalibrert og deretter sammenlignet med normalen igjen.

Under analysefasen vil radaren måle ti statiske distanser, men operatøren kan også skrive inn nye separate distansepunkter som radaren må måle hvis det er ønskelig.

Operatøren har også mulighet til å avbryte nåværende målerunde dersom det ses på som nødvendig. [42].



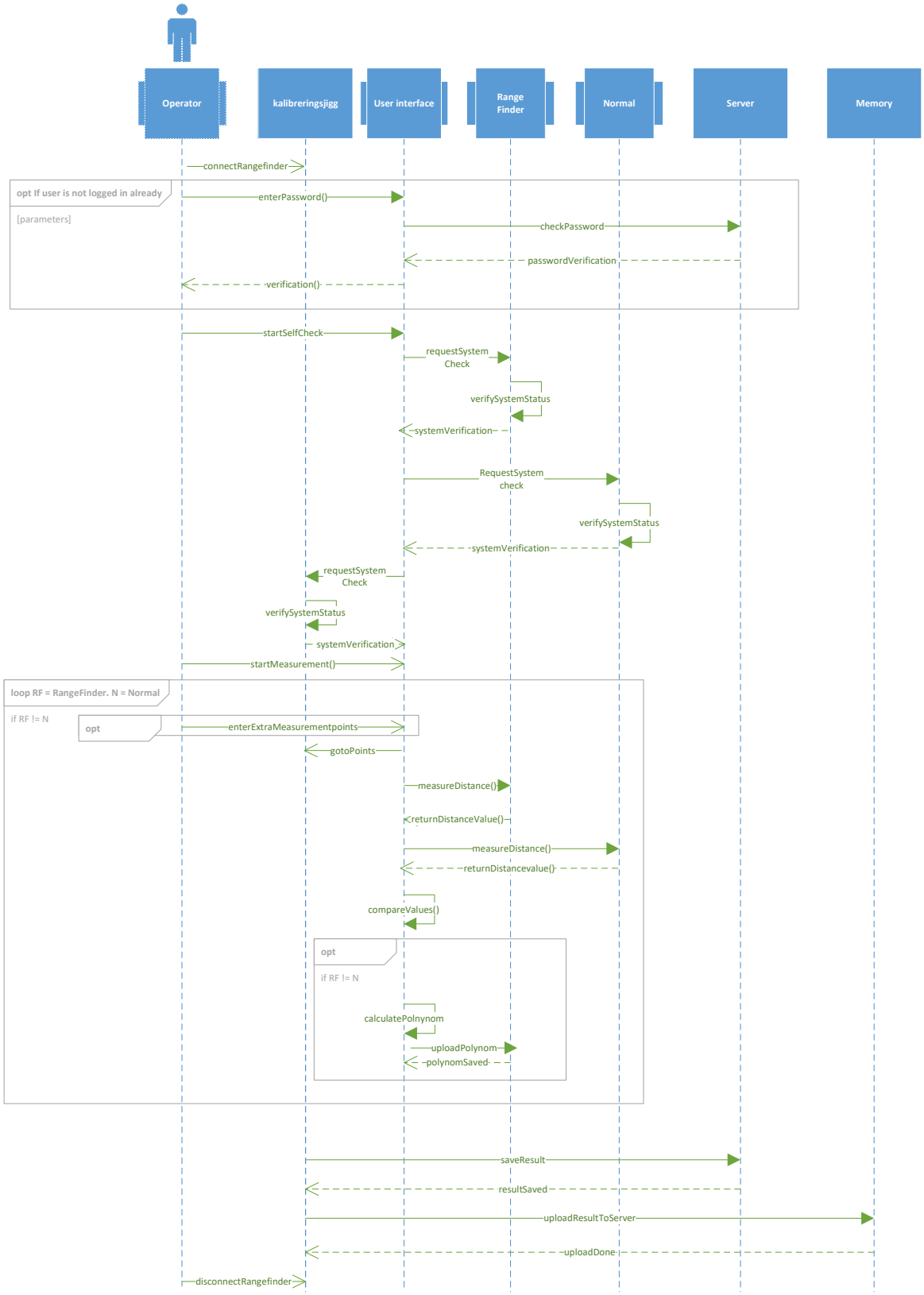
Figur 6.1: Use case

6.3 Sekvensdiagram

Et overordnet sekvensdiagram for planlegging av hele prosessen er laget som vist i fig 6.2. Dette diagrammet tar for seg alle systemets prosesser fra oppstart til fullført operasjonsrunde.

Ved operasjonsstart kobles RangeFinderen til riggen. Deretter blir brukeren bedt om å logge på hvis han allerede ikke er det. Passord og brukernavn blir sammenlignet med eksisterende passord som ligger i databasen. Hvis passord og brukernavn eksisterer får operatøren vite at han er logget inn og dermed har tilgang til programmets funksjoner. Når operatøren ber om å sette igang en målerunde gjennom grensesnittet, vil sensorens programvare starte en selfcheck for å være sikker på at programvaren fugerer slik den skal. Etter dette sendes samme etterspørsel til normalen, og så til slutt til selve kalibreringsjiggen. Etter at selfcheck av alt utstyr er ferdig, gjennomføres operatørens forespørsel om å starte oppmålingsfasen.

Deretter går programmet inn i en loop som sier at hvis Rangefinders verdier ikke ligger på +-5mm i forhold til den samme verdien som normalen, så vil oppmålingsfasen starte på nytt. Før jiggen begynner å gå mot punktene har brukeren mulighet til å skive inn nye punkter, eller endre på de ti standardpunktene som er gitt. Deretter vil riggen gå mot de valgte punktene og starte oppmåling mot både RangeFinder og normalen. Etter at alle punkter er målt opp vil verdiene til RangeFinder bli sammenlignet med Normalen. Hvis verdiene ikke stemmer med hverandre vil programmet starte kalibrering av RangeFinder, og deretter begynne med oppmålingsfasen på nytt. Hvis verdien ligger innenfor grenseverdien vi ble gitt, vil programmet lagre rapporten i databasen, og til minnet.



Figur 6.2: Sekvensdiagram

6.3.1 Målingsrunde

Sekvensdiagrammet som vist på fig 6.3 beskriver operasjonene systemet skal foreta seg når operatøren starter en ny målerunde.

• Foruten operatøren som aktor, så er sekvensdiagrammets benyttede objekter følgende: Grensesnittet til dataprogrammet, mikrokontrolleren Raspberry Pi, avstandssensoren Rangefinder, og motoren som styrer den fysiske bevegelsen.

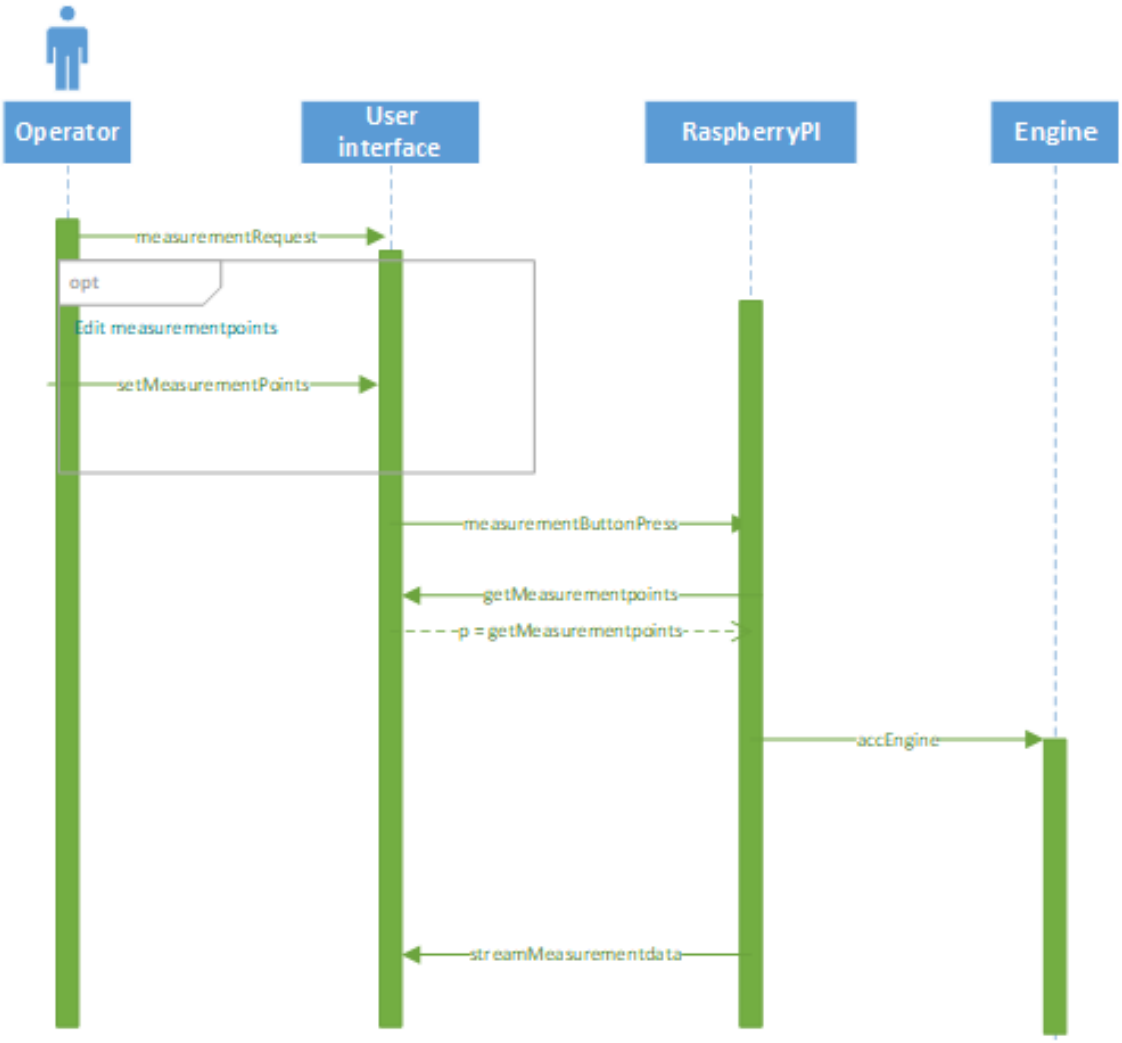
• Operatøren skal starte målerunden fra grensesnittet. Forespørsel fra operatøren til grensesnittet informerer om at operatøren skal starte en målerunde via dette grensesnittet.

• Det meste av «message-passing» foregår mellom brukergrensesnitt og systemets Raspberry Pi. Det trykkes på en knapp som sier ifra til Raspberry PI at en ny målesekvens skal iverksettes. Mikrokontrolleren tar da inn målepunktene fra brukergrensesnittet og bearbeider disse parameterne.

• Raspberry PI setter i gang en systemsjekk i Rangefinderen, der sensoren konfigureres og sjekker etter oppdateringer. Når sensoren er konfigurert og klar, så gir den en respons tilbake. Raspberry PI gir da motoren beskjed om å sette i gang bevegelse mot første målepunkt.

• Sekvensdiagrammet inneholder en «optional fragment», som beskriver en operasjon som oppstår dersom operatøren ønsker det. Systemet har ti gitte standardpunkter som den tar inn og kjører til, men det skal også være mulig å endre disse punktene. Operatøren skriver da disse punktene inn i brukergrensesnittet som oppdateres med de nye punktene.

• Når målerunden er satt i gang vil Raspberry PI overføre informasjonsstrømmer over til grensesnittet slik at operatøren kan overvåke prosessen under kjøretiden.



Figur 6.3: Målingsrunde

6.3.2 Sammenligning av måleresultat

Sekvensdiagrammet som vist i fig 6.4 beskriver operasjonene systemet skal foreta seg når måleresultater skal sammenlignes.

• Under sammenligningsprosessen sender brukergrensesnittet en forespørsel til måleprogrammet om å hente data fra RangeFinder(rf) og Incremental encoder (IE).

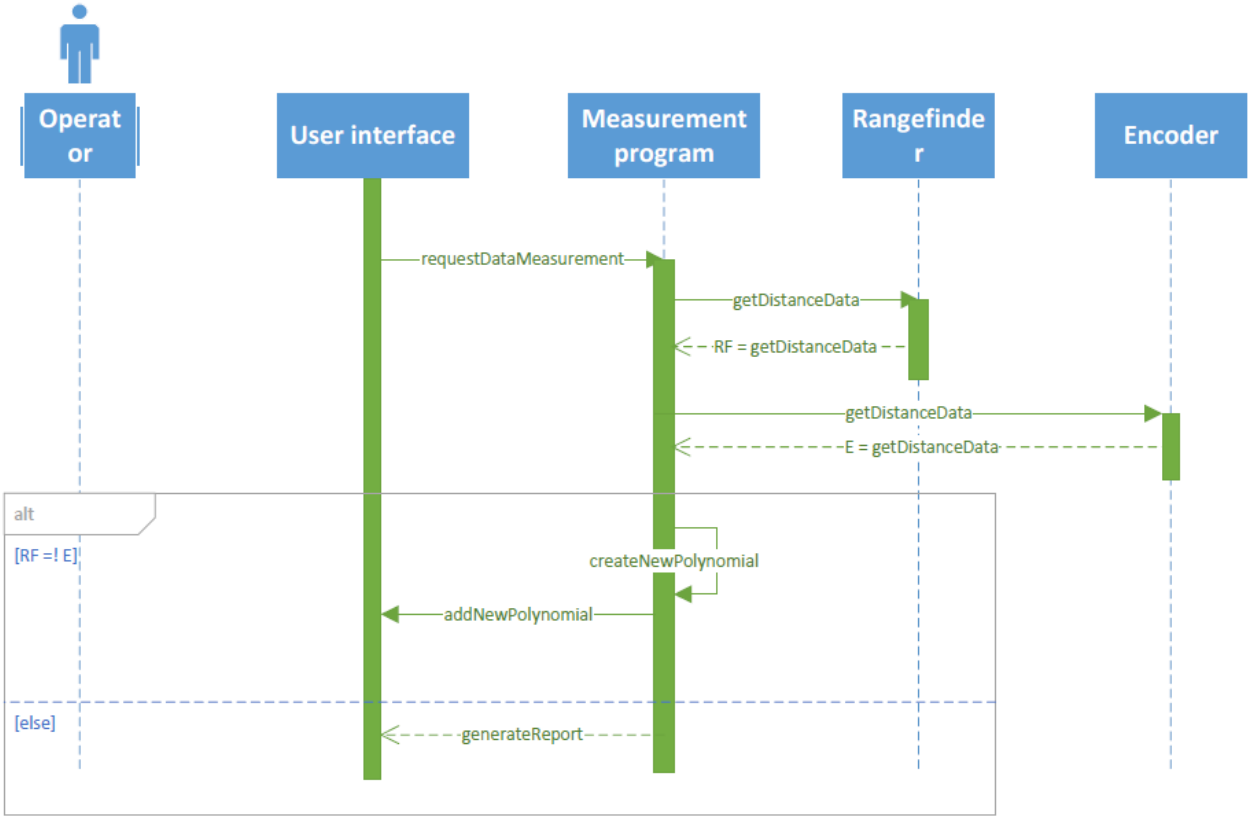
• Måleprogrammet får distansedata tilbake fra RangeFinder og gir verdien navnet RF, deretter hentes distansedataen fra Incremental Encoder, og gir den navnet E.

• Så sammenlignes verdiene i programmet med en IF setning [RF != E].

• Hvis verdiene ikke ligger innenfor den maksimale avviksverdien, må et nytt polynom bli generert. Dette blir gjort ved å først regne ut korreksjonsfaktoren, som regnes ut slik: $\frac{1}{a}$ der a er stigningstallet ($a = \frac{(x10-\ldots x1)}{(y10-\ldots y1)}$) til målingen. Deretter blir korreksjonsfaktoren multiplisert med gain (standard gain til rf er 374,383) slik at du får ett nytt gain. Korreksjonsfaktoren blir også multiplisert med Rangefinders verdier for å skaffe gain-korrigerte distanseverdier. Så blir et nytt skjæringspunkt regnet ut med de nye gain-korrigerte distanseverdiene til RangeFinderen $a = \frac{(x10-\ldots x1)}{(y10-\ldots y1)}$. Etter på blir gain-korrigerte RangeFinderverdier subtrahert med det nye stigningstallet som gir oss distanseverdiene etter gain- og offsetkorreksjon. Disse verdiene blir deretter subtrahert med E som vil gi oss avviksverdien til den kalibrerte RangeFinderen.

• Etter dette burde Rangefinder ligge innenfor den maksimale avviksverdien.

• Hvis verdiene ligger innenfor den maksimale avviksverdien, vil måleprogrammet sende melding til grensesnittet at den skal generere en rapport.



Figur 6.4: Sammenligning av måleresultater

6.3.3 Innlogging

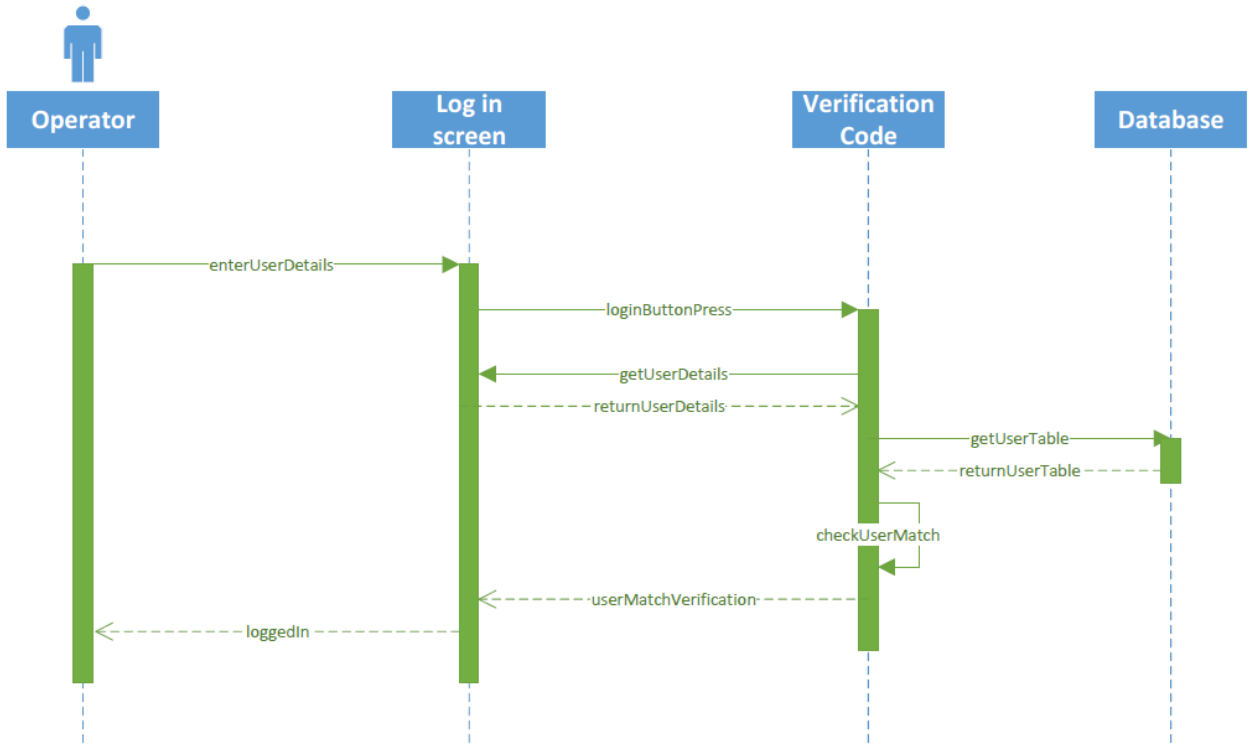
Sekvensdiagrammet som vist i fig 6.5 beskriver operasjonene systemet utfører når operatøren skal logge seg inn på brukerprogrammet. Dette er planlagt funksjonalitet som ikke er lagt inn i systemet enda.

• Operatøren er aktor, mens objektene i dette diagrammet er logg-inn vinduet, verifikasjonskoden i tillegg til databasen som holder på brukerinformasjon.

• Via logg-inn vinduet legges brukernavn og passord inn. Verdiene her sendes til verifikasjonskoden som holder på dataene.

• Verifikasjonskoden gjør nå jobben med å finne ut om det er lagt inn gyldig brukerinformasjon. En tabell med all brukerdata hentes fra databasen. Så sjekkes hver rad for treff med brukerinformasjonen som operatøren har lagt inn.

• Når det oppstår ett treff i søket så sendes det en verifikasjon til logg-inn vinduet som sender brukeren videre til programmets brukergrensesnitt.



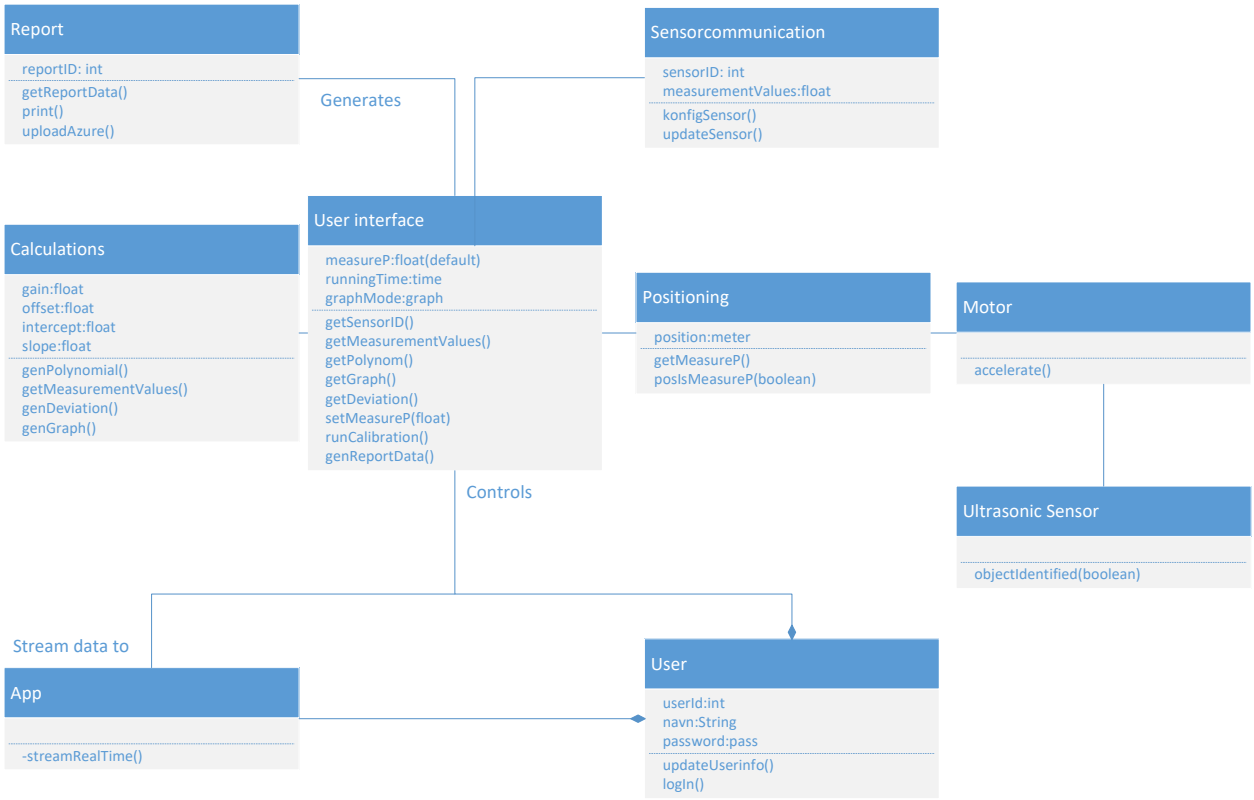
Figur 6.5: Innlogging

6.4 Class diagram

Programvaren COR skal utvikle gjennom prosjektet skal være modulbasert. Et modulbasert system vil si en programvare som deles opp i uavhengige kodedeler, kalt moduler, som kommuniserer med hverandre. I tillegg til at vi får et mer oversiktlig system som vil være lettere å vedlikeholde, så vil de uavhengige modulene kunne gjenbrukes som en del av andre relevante systemer.

Når et modulbasert system skal utvikles er det vesentlig med en grundig planlegging av moduloppdeling. Grunner til at dette er viktig er for å få en mest mulig effektiv kommunikasjonsmetode og en ryddig implementasjonsfase med et begrenset antall uforutsette problemstillinger.

Et klassediagram med en oversikt planlagt oppdelingen ble utviklet i elaborationfasen, som vist i fig 6.6. En kort beskrivelse av planlagt funksjonalitet til hver modul følger under.



Figur 6.6: Class diagram

Moduler:

User interface

UI-delen av programmet skal inneholde informasjonen som brukeren ser. Det er gjennom denne modulen operasjonsrunder startes, stoppes og verifiseres, i tillegg til innstillinger som for eksempel plassering av referansepunkter.

App

App-modulen sin oppgave er å la brukeren overføre real-time informasjon omhandlende nåværende operasjonsrunde. Denne modulen er ikke implementert ved innlevering.

User

For å få tilgang til grensesnittet enten via Appen eller programvaren på PC skal det kreves at en bruker logger inn via et innloggingsvindu. Brukermodulen vil sørge for dette, noe som vil skape et ett mer kvalitetssikkert system. Denne modulen er ikke implementert ved innlevering.

Sensor

Sensormodulens oppgave er å skape en tilkobling mellom brukerprogrammet og den unike sensoren som skal konfigureres og oppdateres før målerunde.

Positioning

Denne modulen tar referansepunktene som gis i det en kjørerunde skal startes og holder på disse punktene. Posisjoneringsmodulen skal gi motoren beskjed om når det skal akselereres og når det skal bremses ned. Den sjekker jevnlig nåværende posisjonsdata opp mot referansepunktene, stemmer de ikke overens skal motoren kjøre, stemmer de skal det stanses og måles.

Motor

Vogna skal ha en motor som autonomt skal kjøre vogna til bestemte referansepunkter. Motormodulen skal ta inn beskjed når systemet skal akselerere og stanse, og gjøre denne instruksjonen om til fysisk bevegelse.

Ultrasonic sensor

Hvis denne sensoren oppdager gjenstander i veibanen skal den overstyre posisjoneringsmodulen og øyeblikkelig gi motoren en kommando om å stanse umiddelbart.

Calculations

Denne delen av programmet skal regne ut matematiske funksjoner som skal verifisere oppmålte verdier opp mot gitte grenseverdier, altså +/-5mm. Ved hjelp av grafer og matematiske utregninger skal det ved for stort måleavvik regnes ut et nytt polynom skal gi mer nøyaktige målinger.

Report

Rapportmodulen skal etter hver fullførte kjørerunde generere en rapport med data som omhandler målerverdier og dens avvik, samt sensorinformasjon og annen nyttig info. Denne rapporten skal lastes opp til en lokal database og Microsoft Azure, samt generere en skriftlig rapport.

6.5 Webside

Gjennom prosjektets faser jobbes det med å utvikle en nettside som informerer om bachelorprosjektet vårt. Her legges det ut informasjon om prosjektoppgaven, prosjektgruppen COR, oppdragsgiver MIROS samt oppdatering på prosjektets utvikling.

Nettsiden, som vist i fig 6.8, er utviklet med html som programmeringsspråk og Notepad++ som IDE. Praktiske kodebibliotek som bootstrap har vært svært nyttig for utvikling av nettsidens design.

Kort info om de ulike sidene og deres innhold følger under.

Figur 6.7: Nettside

Hjem

Dette er områdets forside, og inneholder kort info om nettsidens formål, kort om prosjektgruppen og hva du kan vente deg å finne når du navigerer deg rundt på webområdet.

Om oss

Denne siden inneholder et portrettbilde av hvert gruppemedlem med tilhørende informasjon som navn, studieretning og hovedansvarsområdet under bachelorprosjektet.

Prosjektbeskrivelse

Her kan du lese kort om oppdragsgiveren vår, problemstillingen som legges til grunn for prosjektet, og ut fra problemstillingen beskrivelse av oppgaven vi er tildelt.

Arbeidet

På denne siden legges det ut info om prosjektets framgang og iterasjonene som har blitt gjennomført i de ulike fasene, med problemstillinger og milepæler som oppstår underveis.

Galleri

Gallerisiden skal gi et mer visuelt innblikk i hvordan arbeidsprosessen vår har vært og prosjektets omfang, gjennom interessante bilder av prosjektets ulike spektre.

I tillegg til at prosjektbeskrivelsesiden viser kort info om oppdragsgiver MIROS, så ligger logoen deres til høyre på hver side. Et klikk på denne vil føre brukeren til oppdragsgiverens hjemmeside.

7 KOMPONENTER

Dokumenthistorie			
Versjon	**Dato**	**Endret av**	**Aktivitet**
1.1	12/02/18	DK	Dokument opprettet
1.2	21/02/18	EAD	Systemarkitektur
1.3	09/03/18	TAT	Pugh for målemetoder
1.4	11/03/18	TAT	Pugh for motor
1.5	12/03/18	DK	Mikrokontroller
1.6	14/03/18	TAT	Pugh for omformer/Omformer/ Bill og materials
1.7	16/03/18	TT	Kommunikasjon
1.8	17/03/18	TT	Energiforbruk
1.9	18/03/18	EAD	Spenningskilde
1.10	20/03/18	TAT,DK	Batterilader, ladekontakt, sikkerhetssensor
1.11	21/03/18	EAD	Utstyrsliste
1.12	04/04/18	VS	Revidert for publisering
2.0	06/04/18	TT	Publisert i rapport v2.0
3.0	21/05/18	TT	Publisert i rapport v3.0

Tabell 7.1: Dokumenthistorie for komponenter

7.1 Innledning

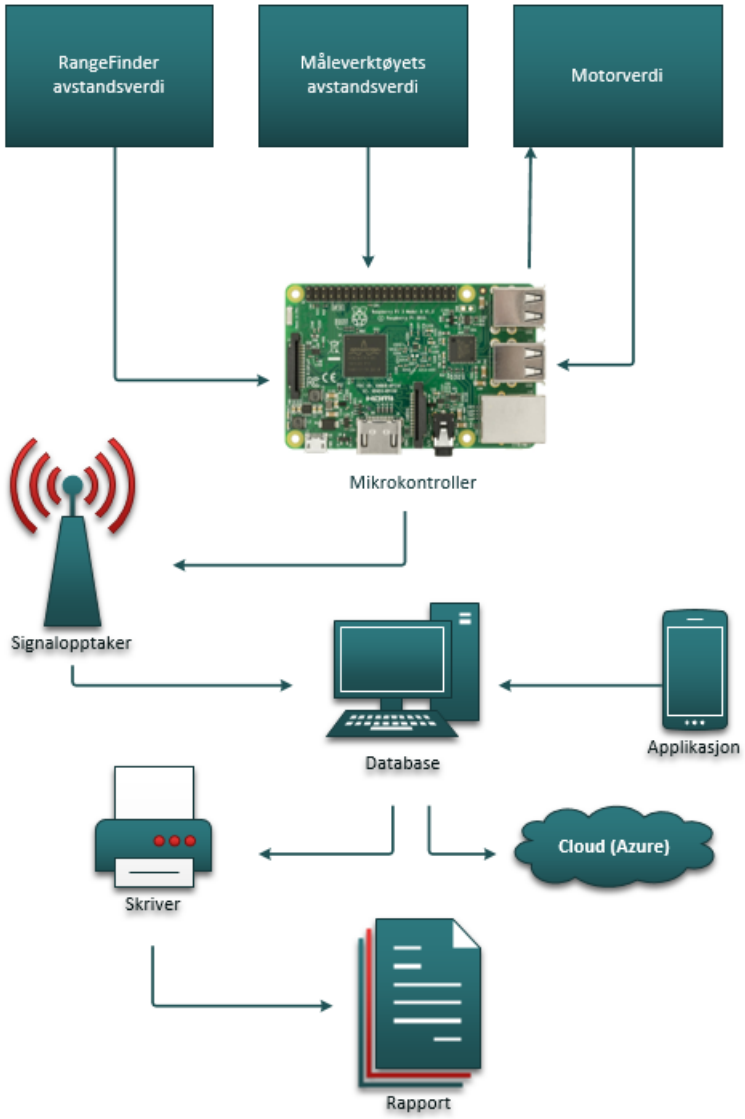
I de følgende underkapitlene viser vi de forskjellige komponentene vi har valgt til prosjektet vårt. Alle komponenter har blitt nøye vurdert og sammenlignet med en rekke valgmuligheter. En pughmatrise har blitt utviklet for hvert valgte komponent. Dette er gjort for å få en mer nøyaktig og oversiktlig vurdering med mer oversiktlige sammenligninger.

Pughmatrisens poengsum regnes ut slik:
+ multipliserer viktighetsgrad med 1.5.
= multipliserer viktighetsgrad med 1.
- multipliserer viktighetsgrad med 0.5.

7.2 Systemarkitektur

Systemarkitektur som vises i fig 7.1 er en henvisning til komponenter som virker og henger sammen i systemet. Dette gir oss en detaljert oversikt over prosesser som utføres av og mellom komponentene og komponentenes sammenheng i systemet.

I oppstartfasen er det slik at verdiene fra RF og måleverktøyet blir sendt til en mikrokontroller (Raspberry pi) som regulerer systemet. Verdien til motoren settes ut fra hvilken verdi måleverktøyet har. Motoren skal automatisk beregne hvor mange omdreininger det trengs for å kjøre nødvendig avstand. Ut fra dette skal motoren få gitt en egen verdi som sier hvor lenge den skal kjøre til gitt avstandspunkt. Disse verdiene blir sendt videre til en database som enten er en mobil enhet eller datamaskin med display. Når disse verdiene blir godkjent for den gjeldende målerunde, blir verdiene sendt videre til en skriver som skriver ut en rapport. Rapporten inneholder disse dataene som da bekrefter at RF er kalibrert.



Figur 7.1: Systemarkitektur

7.3 Motor

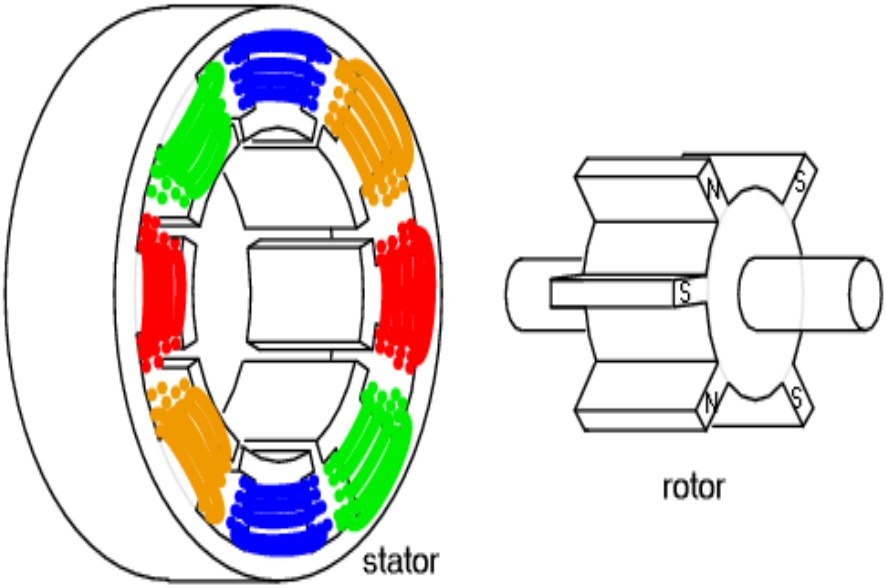
Vår bacheloroppgave innholder en bevegelig del der kalibreringsvogna skal kunne bevege seg autonomt til forskjellige referansepunkter med svært høy nøyaktighet. Ut fra kravene vi har mottatt så må kalibreringsensor ha en feilmargin på +- 5mm, det vil si kalibreringutstyr skal ha to til fire ganger mindre feilmargin en +-5mm. For å kunne oppfylle dette kravet må vi ha svært lite feil prosent på kalibreringsvogna. Det finnes mange metoder og løsninger for denne problemstillingen, siden det finnes flere type motorer som DC-motor,AC-motor, fossildrevet motor og jet- motor.

Siden kalibreringsvogna kun skal benyttes innendørs kan vi se bort fra fossildrevet motor og jet-motor på grunn av disse forurenser og bråker svær mye. Alternativene vi står igjen med er DC-motor og AC-motor. Det finnes flere ulike typer DC- og AC motorer. Innenfor DC-motorer har vi stepper, servo ,brush-less og brush-type. Innenfor AC-motorer finnes det to forskjellig typer, singel phase og tre phase motor. AC-motor er den mest brukte både til industri og privat anvendelse. Singel phase AC-motor finnes i nesten alle husholdningsutstyr som i kjøleskap, støvsugere og vifter. Tre-phase motor brukes som regel i industriområder, siden denne type motor har ekstrem kraft i forhold til motor størrelse. Tre-phase motoren krever en spenningkilde på rundt 400 VAC. Et problem med AC-motor er at vi trenger en spenningkilde på vekselspenning, og denne metoden kan by på problemer med tanke på spenningskabelen.

Siden vår oppgave krever at kalibreringsvogna skal være autonom med en spenningskabel så vil kalibreringsvogna være begrenset av kabelens lengde. Bruk av DC-motor vil være den beste alternativ for vårt system, siden vi ikke har ekstern spenningskabel hvis vi plasserer batteriet i selve kalibreringsvogna. Det finnes mange typer DC-motorer som steppermotor, brushless permanent magnet motor, hybrid stepper og servomotor. Vi tar for oss stepper og servomotor i dette prosjektet siden disse motortypene er de vi er mest kjent med. En DC-motor alene vil ikke være optimalt hvis det mangler motorkontroll. For å kontrollere hastighet og posisjon til disse motorene vil vi regulere spenningsnivå som tilføres til motoren.

7.3.1 Steppermotor

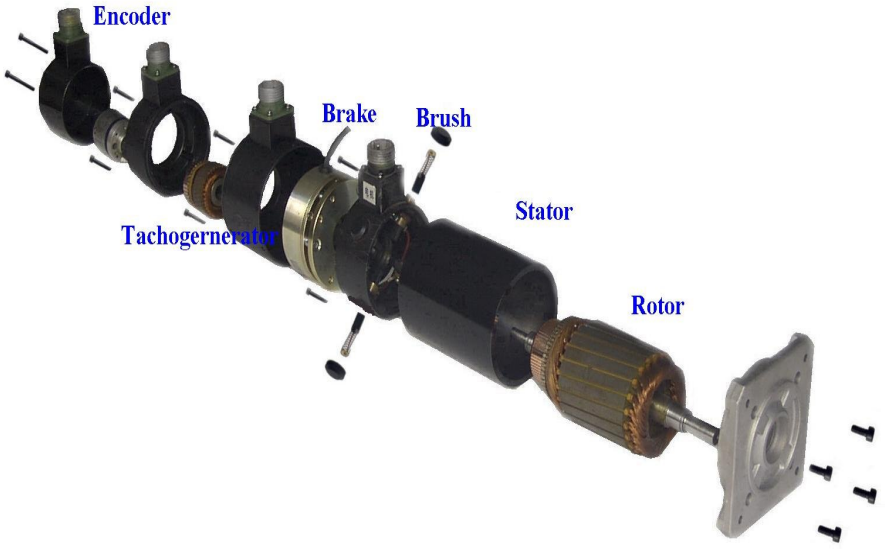
Steppermotor er en type DC-motor som beveger seg i korte, diskrete steg [45]. Steppermotor blir mest brukt til 3D printing, CNC maskiner eller andre bruksområder der det kreves høy nøyaktighet. Steppermotor består av to deler. En statordel og en rotordel som vist i fig 7.2. Statordelen består av et par poler som blir magnetisk når vi tilfører spenning til motoren. Dette får da rotordelen til å rotere til neste posisjon, rotoren er koblet til akselutgangen. Hvor nøyaktig steppermotoren er avhenger av antall statorer og rotorer som er i motoren, men steppermotoren har en typisk vinkelfeil fra utgangsaksel på 1.8° grader.
Steppermotor er ekstremt godt egnet til å kontroll av DC-motorens rotasjonhastighet, og ved lav rotasjonshastighet har steppermotor høyere torque enn en kommersiell DC-motor.
Ulempen med steppermotor er at den ikke har tilbakekobling for posisjon.



Figur 7.2: Steppermotor med stator og rotor [46].

7.3.2 Servomotor

Servomotorer er designet for bruk til robotiske- og automatiske systemer som krever høy grad av presisjon og posisjonering med høy torque. Ved bruk av servomotor som vist i fig 7.3 kreves det en motorkontroll som en steppermotor. Forskjellen mellom servomotor og steppermotor er at på servomotor er det en innebygget posisjonssensor som teller antall vinkelrotasjoner og sender det til motorkontrollen eller til kontrollsenter. Et kontrollsenter er typisk en mikroprosessor eller PLS. Siden servomotoren har posisjonssensor innebygget i motoren vil den ha lavere vinkelfeil enn steppermotor, denne vinkelfeilen ligger på rundt 0.45° grader [38].



Figur 7.3: Servo DC-motor [39].

7.3.3 ClearPath

Teknic er et Amerikansk firma som lager forskjellige motortyper. ClearPath er en av deres produktserier. ClearPath er en brushless servomotor, den har samme oppbygging som servomotorer, forskjellen mellom ClearPath og en vanlig servomotor er at ClearPath sin har en innebygget motorkontroll[7]. Siden ClearPath motor er brush-less motor vil den ha lengre levetid enn vanlig brush motor,siden oppbygging til brush-less motor ikke produserer gnist eller friksjon inni motoren ved rotasjon.

7.3.4 Torque-utregning

For å finne ut hvor mye torque som trengs for å fullføre en målerunde iløpet av en gitt tidsgrense, maksimalt en time ut fra krav K-2.3, trengs det en viss normal hastighet på kalibreringsvogna. Siden distansen på skinna som kalibreringsvogna opererer på maksimalt er 12 meter lang, trenger vi ikke alt for høyt hastighet for å oppfylle krav K-2.3. Massen til kalibreringsvogna er på 80kg med alle komponenter innsatt. Kjørehastighet er satt til 0.25m/s, noe som tilsvarer 0.9km/t. Diameter til hjulet på kalibreringsvogna er 0.06m, som tilsvarer 6cm, og radius blir på 3cm.
Sett ut fra databladet til en av DC-motorene som vi fikk tilgang til, så er max rpm på 1130 rpm, og rotor inertia på 0.00021 kg-m^2
V = 0.25m/s a = 0.05m/s T = 5s d = 0.06m r = 0.03m
Maxrpm = 1130rpm = 18.833 rev/s Rotor inertia = 0.00021 kg-m^2

$$mg = 80 * 9.81 = 785N \tag{7.1}$$

$$DN = mg * r = 785N * 0.03 = 23.55N \tag{7.2}$$

$$\omega = \frac{V}{r} = \frac{0.25m/s}{0.03m} = 8.33rad/s \tag{7.3}$$

$$\frac{\omega}{2\pi} = 1.326rev/s \tag{7.4}$$

$$Girforhold = \frac{MaxRrm}{\omega/2\pi} = \frac{18.833rev/s}{1.326rev/s} = 14.20 \tag{7.5}$$

$$\frac{DN}{Girforhold} = \frac{23.55N}{14.20} = 1.658N \tag{7.6}$$

For at kalibreringsvogna skal klare å holde en normal hastighet på 0.25m/s trenger vi 1.66 Nm. Dette er vår konstant torque.
Når kalibreringsvogna skal akselereres kreves det ekstra torque.

$$\frac{1}{14.2}^2 * m * r^2 = \frac{1}{14.2}^2 * 80kg * 0.03^2 = 3.57 * 10^{-4} \tag{7.7}$$

$$3.57*10^{-4} + Rotorinertia + Rotorinertia = 3.57*10^{-4} + 0.00021 + 0.00021 = 7.77*10^{-4} \tag{7.8}$$

$$\frac{MaxRpm * 2\pi}{5} = \frac{18.833rev/s * 2\pi}{5} = 23.66rad/s \tag{7.9}$$

$$7.77 * 10^{-4} * 23.66 = 0.018Nm \tag{7.10}$$

$$PeakTorque = 1.66Nm + 0.018Nm = 1.678Nm \tag{7.11}$$

For at kalibreringsvogna skal klare å kjøre 0.25m/s med en akselerasjon på 0.05m/s, trenger vi en motor som har en konstant torque på 1.658 Nm og Peak torque på 1.678 Nm. Siden vi ikke har en eksakt verdi på hvor mye hele kalibreringsvogna og RF veier til sammen, i tillegg til alle kompomenters vekt, går vi for litt ekstra Peak torque og konstant torque. Motoren skal ha en minimum Peak torque på 3 Nm.

7.3.5 Pughmatrise for DC-motor

For motor blir flere forskjellige kriterier viktig for oss som:

- **Dimensjon:** Motoren skal ikke være større enn 200x100x150mm.
- **Maks RPM:** Motoren skal ha maks rpm over 1000 rpm.
- **Vekt:** Motoren skal ikke veie mer enn 10kg.
- **Maks torque:** Motoren skal ha minimum maks torque på 4Nm.
- **Effektforbruk:** Motoren skal ikke bruke mer enn 400W.
- **Rotasjonsnøyaktighet:** Avvik på utgangsakselen i forhold til motoren skal være under 2° grader.
- **Pris:** Motoren skal ikke koste over 10.000 kr
- **Spenning:** Motor skal bruke DC spenning.
- **Temperatur arbeidsområde:**Motoren skal kunne operere optimal på 20° - 30° C grader.
- **Tilpasningsmulighet:** Motoren skal kunne implementeres via en mikrokontroller eller en mikroprosessor.
- **Programvare:** Motoren skal støtte programvare for motorstyring.

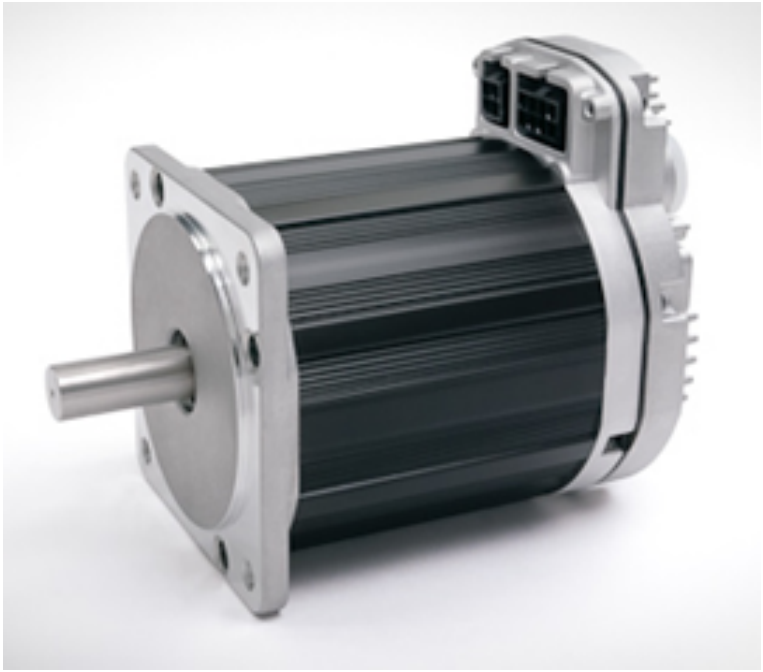
Pugh matrise for DC motor				
Criteria	Poeng	ClearPath	Servo(90mm)	Stepper
Dimmensjon	3	=	-	=
Maks RPM	3	+	-	=
Vekt	3	=	=	-
Maks torque	4	=	+	=
Effekt forbruk	4	=	=	-
Rotasjon nøyaktighet	5	+	=	-
Pris	3	=	=	=
Spenning	3	=	=	=
Arbeid temperatur område	2	=	=	=
Tilpassning mulighet	4	+	=	=
Program vare	3	+	=	=
Sum		44.5	36	31

Tabell 7.2: Pugh for DC-motor

Datablad for servo : [40]
Datablad for stepper: [44]
Datablad for ClearPath : [8]

7.3.6 Konklusjon DC motor

Ut fra pughmatrisen for DC-motorer så kommer ClearPaths motor, som vist i fig 7.4, best ut. Det er ikke mye som skiller mellom ClearPaths motor og vanlig servomotor. Grunnen til at ClearPaths motor får høyest poengsum her er fordi den inneholder flere tilpasningsmuligheter, samt at ClearPaths motor har en egen tuning-programvare som vil være nyttig til dette prosjektet. I tillegg har den en høyere rotasjonsnøyaktighet i forhold til vanlige servomotorer. Vi har valgt å bestille en Enhanced ClearPathmotor som har mindre avvik på utgangsaksel, med 0.0057°C grader.



Figur 7.4: ClearPath motor

7.4 Målemetoder

For at vi skal oppfylle krav K-1.0, som er at RF skal ha et måleavvik på +- 5mm, så må kalibreringsutstyr ha 2-4 ganger mindre tillatt måleavvik, dvs kalibreringsvogna må ha ett maksimalt avvik på +-1 til 2 mm. Dette krever ekstremt presist måleverktøy. Det finnes mange typer måleverktøy på markedet med tilstrekkelig nøyaktighet, men på rekkevidde er de begrenset. Vi har sett på ulike målemetoder som incremental encoder, laser for distansemåler med analog eller digital verdi og proximity sensor for materiell deteksjon.

7.4.1 Incremental encoder

Incremental encoder består av flere forskjellige deler, den har en aksel som vi kommer til å montere til selve hjulet på kalibreringsvogna. Incremental encoder innholder tre deler; en roterende skive, en lyskilde og en lyssensor. Den roterende skiva blir koblet til en aksel som er plassert på hjulet til kalibreringsvogna. Den roterende skiva har et mønster der annenhvert felt er hvitt og mørkt. Ved bevegelse av roterende skive blir lys slippet inn til lyssensor, da får vi ett eller flere pulssignal.
Når vi kobler incremental encoder til kalibreringsvogna så vil vi få en ekstrem nøyaktighet ved bruk av lyspuls fra encoder. Hjulet til kalibreringsvogna har en diamenter på 6 cm, vi bruker en encoder som bruker 1024 pulsslag per omdreining, da oppnår vi følgende:
D = 6 cm
Omkrets $= \pi * D = \pi * 6cm = 18.849cm = 188.849mm$
Antall puls = 1024
Pulsoppløsning:

$$\frac{Omkrets}{antallpuls} = \frac{188.849mm}{1024} = 0.18442mm \tag{7.12}$$

Ut fra ligning 7.12 oppnår vi en nøyaktighet på 0.18442 mm per puls. Encoderen har en pulsfrekvens, som er hvor fort den klarer å detektere signal. Dette er fordi encoderen kan miste pulstelling hvis rotasjonsakselen roterer for fort.

Encoderens minimale oppdateringsfrekvens ved 0.25m/s:

- Max akselrotasjon i 0.25m/s
- Fra ligning 7.3 0.25m/s med 0.03m i hjulradius = 8.33 rad/s
- Antall encoderpuls = 1024

$$\frac{radpersekund * 60}{2\pi} = \frac{8.33rad/s * 60}{2\pi} = 79.54rpm \tag{7.13}$$

$$\frac{(maxakselrotasjoni 0.25m/s) * (antallplus)}{60} = \frac{80rpm * 1024puls}{60} = 1365.33Hz \tag{7.14}$$

Fra ligning 7.14 så ser vi at vi trenger en encoder som har høy oppdateringsfrekvens. Minimale verdi er 1365.33 Hz for at vi ikke skal miste pulstelling.

7.4.2 Proximity sensor

Proximity sensor eller nærhetssensor blir brukt til å detektere gjenstander eller materiell. Nærhetssensor fungerer slik at den sender ut elektromagnetiske bølger som blir reflektert tilbake når signaler treffer gjenstander eller materiell som ligger innenfor måleområde.

Det finnes to typer nærhetssensorer; capacitive og inductive. Capacitive sensor kan detektere flere forskjellige materialer som glass, metall, gummi osv. Inductive sensor detekterer kun metalliske gjenstander. I vår pughmatrise bruker vi capacitive sensor, sånn at vi skal kunne bestemme hvilket materiell sensoren skal reagere på. Ulempen ved bruk av proximity sensor er at den har en begrenset rekkevidde, typisk fra 0.2mm til 20mm.

Ved bruk av proximity sensor plasserer vi små metall- eller aluminiumbiter med 50cm mellomrom. Ved bevegelse av kalibreringsvogna så vil det gis signal hver gang proximity sensor kjører over metall- eller aluminiumbiter, disse signalene vil legges til i vår posisjoneringsmodul.

7.4.3 Laser

Laser eller photoelectric sensor brukes til avstandsmåling med svært høy presisjon. Photoelectric sensor blir brukt mest i industrioperasjoner til å detektere gjenstanders posisjon. Sensoren er ekstremt nøyaktig, men feilavviket blir større ved lengre avstander. Ved bruk av photoelectric sensor finnes det to forskjellige utgangsverdier. Analoge utgangsverdier gir enten spenning- eller strømverdier som kan omformes til avstandsverdi. Digitale utgangsverdier gir som regel ut avstandverdi eller binærkodeverdi.

7.4.4 Pughmatrise for målemetoder

For målemetoder blir flere forskjellige kriterier viktig for oss:

- **Måleavstand:** Sensorens minimale målerekkevidde må være over 15 meter.
- **Nøyaktighet:** Sensorens maksimale måleavvik må være under +-5mm.
- **Levetid:** Sensoren skal ha en levetid over 10 000 timer.
- **Kostnad:** Prisen på den enkelte sensoren skal ikke overstige 5000kr .
- **Vekt:** Vekten på hver sensor skal være under 2kg.
- **Dimensjon:** Sensoren skal ikke være større en 6 cm i radius.
- **Sikkerhet:** Det skal ikke være behov for verneutstyr med denne typen sensor.
- **Stabilitet:** Stabilitet under kjøring dvs: vibrasjon, spenning forstyrrelse.
- **Temperatur arbeidsområde:** Sensor skal kunne operere optimalt på 20° - 30° C grader.
- **Effektforbruk:** Sensoren skal bruke under 50W.
- **Beskyttelsesgrad:** Sensoren skal kunne tåle støv og vannsprut.
- **Tilkoblingsmulighet:** Skal bruke standard tilkoblingsport, eks: M12.

Pugh matrise for måleintrument					
Kriterier	Poeng	Laser(Analog)	Laser(Digital)	Encoder(1040ppr)	Proximity
Måleavstand	4	=	=	-	+
Nøyaktighet	5	=	=	+	=
Levetid	2	=	=	+	=
Kostnald	2	=	-	+	=
Vekt	1	=	=	-	-
Dimensjon	2	=	=	+	+
Sikkerhet	4	-	-	+	+
Stabilitet (fysisk)	4	=	=	+	=
Arbeidsområde i °C	3	=	=	+	+
Effektforbruk	1	=	=	=	+
Beskyttelsesgrad	3	=	-	=	-
Tilkoblingmulighet	3	=	=	=	=
Sum		32	29.5	42.5	39

Tabell 7.3: Pughmatrise for målemetoder

7.4.5 Konklusjon for målemetoder

I tabell 7.3 så fikk Encoder og proximity sensor høyest poengsum, med 42.5 poeng på Encoder og 30 poeng på proximity.
Ut fra konseptene som vi har tenkt på så har vi valgt å kombinere encoder og proximity til måling for å oppnå best mulig avstandsmåling. Proximity sensor vil ta avstandsmålinger ved punkter hver halve meter, mens Encoderen blir brukt til finjustering mellom disse referansepunktene.

7.5 Kommunikasjon

I vårt system skal den bevegelige vognen kunne kommunisere med pc-en. Den skal kunne sende og motta data. Til dette skal vi benytte trådløs kommunikasjon, dette for å unngå å bruke ledninger som kan være kronglete når den bevegelige vognen skal kjøre frem og tilbake. Vi skal enten bruke WiFi, Bluetooth eller ZigBee. Da er det viktig for oss å se på spesifikasjonene til de forskjellige kommunikasjonsmetodene, og finne ut hvilken av dem som passer best til vårt system.

7.5.1 Wifi

Rekkevidden til WiFi er 250 meter, men hvis det skal fungere optimalt bør kommunikasjonen fungere så nærme som mulig. Datahastigheten er veldig høy i forhold til de andre kommunikasjonsmetodene, maksimal hastighet er 600Mbps og normal hastighet er 150Mbps. Latency(ping) ligger på opptil tre sekunder. Frekvensen ligger enten på 2,4Ghz eller 5Ghz. Strømforbruket til WiFi er ganske høyt.

7.5.2 Bluetooth

Rekkevidden på bluetooth classic og BLE er 100 meter, signalene er sterkere jo mindre avstand det er mellom komponentene. Datahastigheten er veldig lav i forhold til WiFi, Bluetooth har maksimal hastighet på 3Mbps og normal hastighet på 0,3Mbps. Frekvensen ligger på 2,4GHz. Latency(ping) til BLE er tilnærmet lik Wifi, mens classic har opptil 100 sekunder. Strømforbruket til BLE ligger på kun 1/10 av classic, men classic har fortsatt mye mindre strømforbruk enn WiFi.

7.5.3 ZigBee

Zigbee har lik rekkevidde som Bluetooth, altså 100 meter. Datahastigheten er veldig lav i forhold til WiFi, ZigBee har en maksimal hastighet på 0,25Mbps og en normal hastiget på 0,2Mbps. Frekvensen ligger på 2,4Ghz. Strømforbruket er ganske lavt, og latency(ping) ligger på opptil 30 sekunder.

7.5.4 Pughmatrise for kommunikasjon

For kommunikasjon blir flere forskjellige kriterier viktig for oss:

- **Rekkevidde:** Rekkevidden må være minst 20 meter
- **Datahastighet max:** Maksimal hastighet på dataoverføring
- **Datahastighet normal:** Normal hastighet på dataoverføring
- **Frekvens:** Komponentens frekvensområde
- **Strømforbruk:** Hvor mye strøm komponenten bruker
- **Latency(ping):** Forsinkelse på signaler

Pugh matrise for kommunikasjon					
Kriterier	Poeng	Wi-Fi- 802.11x	Bluetooth low energy 4.X	Bluetooth classic	ZigBee
Rekkevidde	4	=	=	=	=
Datahastighet max	2	+	=	=	=
Datahastighet normal	4	+	=	=	=
Frekvens	2	+	=	=	=
Strømforbruk	3	-	+	=	=
Latency (ping)	4	+	=	-	-
Sum		23.5	20.5	17	17

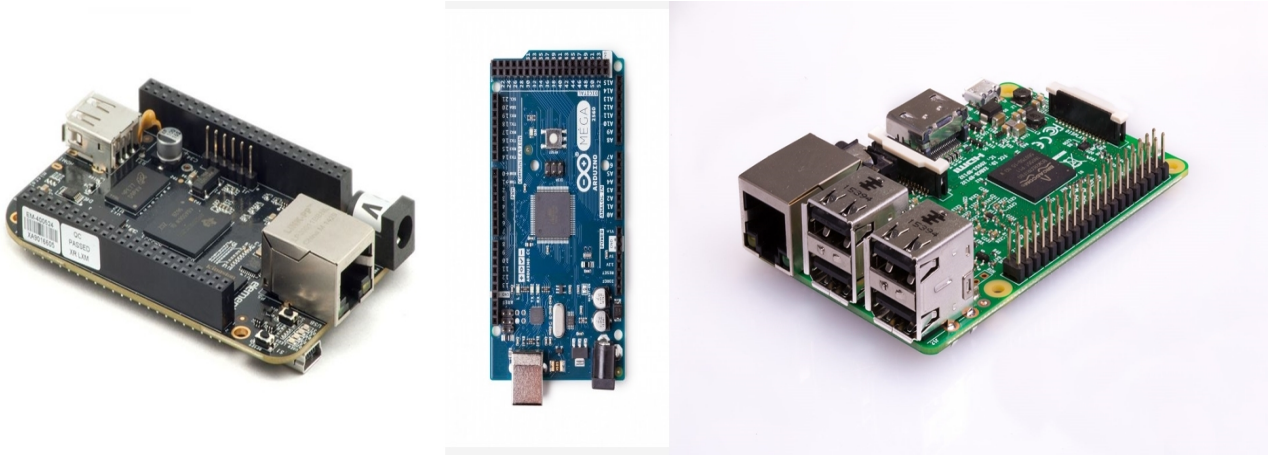
Tabell 7.4: Pughmatrise for kommunikasjon

7.5.5 Konklusjon for kommunikasjon

Vi har valgt å bruke WiFi som vår kommunikasjonsmetode, fordi WiFi er mye raskere enn de andre kommunikasjonsmetodene. Med WiFi har vi også flere frekvenser å velge mellom. Ulempen med WiFi er at strømforbruket er mye høyere enn de andre. Ut fra pughmatrise for de forskjellige kommunikasjonsmetodene som vi har tenkt å bruke i systemet, fikk WiFi mest poeng.

7.6 Single-board computer og mikrokontroller

For å kontrollere samtlige komponenter og overføre data har vi bestemt oss for å bruke en mikrokontroller eller en SBC (Single-Board Computer).



Figur 7.5: Single-board computer og mikrokontroller

7.6.1 Mikrokontroller

En mikrokontroller er en innvendig del av et Embedded system. En mikrokontroller er en liten og billig computer på en enkeltbrikke som inneholder en prosessor, ett lite minne og programmerbare inn- og utgangspinner. Meningen med mikrokontrollere er å bruke disse i automatisk kontrollerte enheter og produkter ved hjelp av forhåndsdefinerte og forhåndsprogrammerte oppgaver. Det finnes mange bruksområder for mikrokontrollere, noen av de er digitaltermometer, vaskemaskin, mikrobølgeovn, biler, robot osv.

Innhold i mikrokontrollere:

• CPU (central prosessing unit) som utfører forhåndsdefinerte operasjoner. Typisk i MHz.

• RAM (random access memory) som lagrer data i vilkårlig rekkefølge.

• ROM (read only memory) for lagring av data lokalt.

• Typiske grensesnitt er I2C, SPI og UART.

• Typisk 8 eller 16 bit prosessorer.

• ADC (analog to digital converter) for å konvertere analogt signal fra sensor til et mer PC-vennlig digitalt signal via bestemte pinner på kretskortet.

• DAC (digital to analog converter) for å konvertere digitale datasignaler til analoge signaler for sensorer via bestemte pinner på kretskortet.

• Seriell kommunikasjon er brukt til å overføre data med ett bit om gangen gjennom ett medium. Brukes for langdistansekommunikasjon mellom enheter eller komponenter.

• Parallell kommunikasjon er brukt til å overføre flere datablokker samtidig. Mye raskere kommunikasjon enn seriell, og brukes dermed i PCer med databusser. Ulempen med denne typen kommunikasjon er distansen, siden parallelle tilkoblinger trenger egen ledning.

7.6.2 Single-Board Computer

Single-board computere er oftest brukt til oppgaver som er krevende. Eksempler kan være matematiske kalkulasjoner, dokumentredigering, simulasjoner og til og med spilling. Som regel har SBC en rask CPU, et stort hukommelsesminne, en kraftig GPU og en stor lagringskapasitet. Et operativsystem er nødvendig for single-board computere. Det finnes mange bruksområder for SBC som værstasjoner, robot, overvåkingssystemer, droner og hjemmeserver. Ofte blandes SBC med SoC (system on chip), men disse er to forskjellige systemer der SBC er et kretskort med komponenter mens SoC er en chip med en eller flere CPUer og GPUer.

Innhold i single-board computer:

• CPU (central prosessing unit) som arbeider mye raskere enn CPUen i mikrokontrollere. Typisk i GHz.

• RAM (random access memory) som lagrer data i vilkårlig rekkefølge.

• ROM (read only memory) for lagring av data lokalt.

• Typiske grensesnitt er UART, USB, HDMI, Displayport, Ethernet og digitale pinner.

• Typisk 32 eller 64 bit prosessorer. Det vil si at en 32 bit prosessor kan håndtere 32 bit med binærdata samtidig.

Tabell 7.5 viser oss spesifikasjoner for tre forskjellige produkter, der Raspberry Pi 3 og BeagleBone Black er Single-Board Computere mens Arduino Mega er en mikrokontroller.

	Raspberry Pi 3	Arduino Mega	BeagleBone Black
Prosessor	CPU 1.2GHz	16MHz	CPU 1GHz
RAM	1GB	8k	512MB
Lagring	Micro SD	256k	4GB
I/0 piner	40	54	96
Analog pinner	0	16	16
Video utgang	2	0	1
Tilkoblingsmuligheter	7	0	2
Pris	38$	39$	55$
Vekt	42g	37g	40g
Dimmensjon	85x56mm	101x53mm	90x54mm
Nødvendighet for OS	Ja	Nei	Ja

Tabell 7.5: Spesifikasjoner for Raspberry Pi 3, BeagleBone Black og Arduino Mega

Tabell 7.6 viser oss matrisen for mikrokontroller med alternativene vi har og poengsumen hvert alternativ oppnår.

Pugh matrise for Mikrokontroller				
Kriterier	Poeng	Raspberry Pi 3	Arduino Mega	BeagleBone Black
Prosessor	4	+	-	=
RAM	4	+	-	=
Lagring	4	+	-	=
Digitale pinner	3	-	=	+
Analoge pinner	2	-	=	=
Videoutgang	2	+	-	=
Tilkoblingsmuligheter	4	+	-	=
Pris	2	+	=	-
Vekt	1	-	+	=
Dimensjon	1	+	-	=
Brukervennlig	4	+	=	-
Nødvendighet for OS	3	=	-	=
Sum		43,5	23	32,5

Tabell 7.6: Pughmatrise for mikrokontroller

7.6.3 Konklusjon for mikrocomputer og mikrokontroller

Vi har valgt Raspberry Pi 3 på grunn av en rask prosessor, et stort hukommelsesminne, en stor lagringskapasitet og mange tilkoblingsmuligheter. Raspberry Pi 3 støtter Windows 10 og kan programmeres med Python software.

7.7 Totalt energiforbruk

Vi skal bruke batterier istedenfor strømledning til vårt produkt. Derfor er det viktig for oss å vite energiforbruket til hver enkelte komponent som trenger strøm. Dette er for å beregne hvor stor kapasitet vi trenger på batteriene.

- Raspberry pi: 13W
- Motor: 200W
- Rangefinder: 50W
- Incremental encoder: 1W
- Proximity sensor: 2W
- Ultrasonic sensor: 1W

Summen blir 266.5W, men vi runder det opp til 300W for å forsikre oss. Hvis det kommer nye komponenter inn i prosjektet senere blir det da ingen store komplikasjoner. Ved å omdanne 300W til amperetimer, så får vi 300W = 25Ah. Vi anslår at batteriet skal kunne være i konstant kjøring i 4 timer, som er en halv arbeidsdag. Da trenger vi et eller flere batterier som skal kunne levere 100Ah før den må lades. Dermed har vi kommet frem til at vi må ha 2 batterier med 12V og 50Ah som parallell-kobles eller 1 batteri på 12V og 100Ah.

7.8 Spenningskilde

Med batteriløsningen blir hele systemet mer kompakt og alt på ett sted. Dette hjelper oss med å få en enklere operasjonsrunde med forflytting av vogn frem og tilbake til avstandspunktene. Med ledning blir dette upraktisk siden den skal anstrenges hele tiden, men hvis batteriløsningen byr på problemer har vi dette som en b-plan for energikilde. Oversikt over dette har vi i kravbackupdokumentet.

For valg av batteri har vi sett på forskjellige typer som blir mest brukt i industrien nå til dags. I pughmatrisen kommer det frem at AGM batterier ser mest relevant ut for oppgaven vår og derfor velger vi å gå for denne. Videre har vi sett litt på nye batteriteknologier som elektriske, poton flow, prieto og graphene batterier, men siden de er under utvikling er de mindre relevant for oppgaven vår i dette tilfellet.[3]

7.8.1 Litiumbatterier (Li-Ion og Li-Poly)

Litiumbatterier er dyre i forhold til andre batterier. De har mer kapasitet, mindre vekt samt raskere ladning. Motoren skal dra en vogn på 80 kilo og derfor er det viktig med kapasitet på batteriene. Sånn sett er litiumbatterier det beste valget, men prisen blir så høy at det ikke vil lønne seg for oss dette prosjektet. Tilgjengeligheten spiller en stor rolle i prosjektet vårt, der kommer også litiumbatterier dårligere ut enn blybatterier som man får i bilutstyr og verktøyforretninger som Biltema.[20]

7.8.2 Nikkelbaserte batterier (NiMh og NiCd)

Nikkelbaserte batterier ligger i sjiktet mellom AGM- og Litiumbatterier med tanke på batterienes spesifikasjoner, altså i form av vekt, kapasitet, ladning osv. Ulempen med nikkelbatterier er at for store mengder av grunnstoffet er farlig for helsa. Refererer til «energibudsjettet», der det klargjøres at energiforbruket vårt kategoriseres som energiforbruk i et kjøretøy, dermed blir batteriene store. Derfor blir disse batteriene mindre relevant for oppgaven vår.[25]

7.8.3 *Absorbent Glass Mat* (AGM) - Metallisk bly (Pb)

AGM batterier blir mest brukt i store energiforbruksystemer og er en type innenfor metallisk bly batterier. Siden oppgaven vår kategoriseres som et stort energiforbruksystem blir AGM batteritypen mest relevant for oss. Batteritypen er tilgjengelig i flere varianter siden den er kjent innenfor flere bruksområder når det gjelder kjøretøy. Dette hjelper oss med å tilegne batteriet til systemet vårt på en enklest mulig måte.[1]

7.8.4 Nyutviklede batteriteknologier

Elektriske og *Proton flow* batterier har vært under utvikling som ny batteriteknologi i noen år nå. Videre finnes det *Prieto* og *Graphene* batterier som er helt nye prosjekter i utviklingsfasen.[30]

Elektriske batterier er mindre relevant for oppgaven vår med tanke på andre type batterier siden det er mange små batterier som er parallelt koblet sammen. Dette blir upraktisk med tanke på bytting av batteriene senere når systemet er i bruk.

Proton flow batterier er under utvikling og har derfor ikke mange bruksområder nå til dags. Dette er en ny batteriteknologi som benytter en ny metode til å lagre energi i batteriene. Disse batteriene har vært under utvikling i noen år nå, teoretiske årsaker bak dette er verdt å studere videre.[32]

Prieto batterier fokuserer på å holde sikkerheten høy ved å ikke benytte seg av elektrolytt væske. Den er brannfarlig og farlig for helsa, og denne væsken brukes i andre batterier. Disse batteriene virker på en annen måte slik at de blir sikrere og har mer kapasitet, samt lenger levetid.[31]

Graphene batterier har selskapet *Grabat* utviklet. Virkemåten er unik og kapasiteten i disse batteriene er mye høyere i forhold til andre batterier vi har i dag. Siden teknologien utvikler seg daglig er det viktig å ha en effektiv energikilde som hjelper systemene/produktene å holde energistrømningen høy.[15]

7.8.5 BMS - *Battery management system*

BMS er innebygget i batterier for oppbevaring og ladbarhet. Systemet går ut på å gjennomføre en sikker metodikk for regulering av lading slik at den ikke skal overstige sin egen grense i sikkerhetsområdet. Med det menes hvor mye ett batteri kan lades med gitt strømtilførsel. Dette er til hjelp for å øke sikkerhet og unngå ulykker som brann og eksplosjon i batterier.[6]

7.8.6 Pughmatrise for batteri

Kriteriene henviser til hvilken form for innhold i de ulike batteriene det er snakk om[4]:

- **Kapasitet:** Kapasitet er hvor stor ladning/energi som er lagret i batteriet.
- **Vekt:** Grensen for dette er at vogna ikke skal overstige en totalvekt på 80 kg.
- **Kostnad:** Pris varierer kraftig fra type til type.
- **Levetid:** Estimert 3 til 5 år, noen 10 år.
- **Sikkerhet:** BMS benyttes i de fleste batterier.
- **Dimensjon:** Skal ikke være større enn et bilbatteri.
- **Tilkoblingsmulighet:** Enkele batteriklemmer skal være i bruk.
- **Lading:** Skal lades etter en arbeidsdag, har mulighet for 8 til 16 timers lading.
- **Selvutlading:** Om batteriet selvutlader drastisk eller ikke.
- **Resirkulering:** Ses hovedsaklig på som muligheter for gjenbruk.

Pugh matrise for batteri						
Kriterier	**Poeng**	**Li-Ion**	**Li-Poly**	**NiMh**	**NiCd**	**AGM (Pb)**
Kapasitet	5	+	+	=	=	=
Vekt	4	+	+	=	=	-
Kostnad	3	-	-	=	=	+
Levetid	3	+	+	=	=	+
Sikkerhet	5	-	-	=	=	+
Dimensjon	1	+	+	+	+	+
Tilkoblingsmulighet	2	=	=	=	=	+
Lading	3	+	+	=	=	-
Selvutlading	3	=	+	=	=	=
Resirkulering	3	-	-	=	-	+
Sum		35	36,5	33	31,5	38

Figur 7.6: Pughmatrise for batteri [5]

7.8.7 Konklusjon for spenningskilde

Som det vises i pughmatrisen vår, så velger vi å gå for AGM batterier. Grunnen til dette er at de er mest relevant for oppgaven vår. Tilgjengelighetsmessig er de enklere å få tak enn andre typer og disse er i hovedsak produsert for kjøretøy. Siden AGM batterier finnes i mange varianter som tidligere nevnt, er det ett mer bredt spekter å velge mellom, som igjen gjør det enklere å velge den riktige spenningskilden for systemet.

7.9 Batterilader

Som spenningkilde bruker vi to stykker 12VDC batterier som vi serielt kobler sammen. Dette gir totalt ut 24VDC som hovedspenningskilde. Siden batteriene er ladbare trenger vi en batterilader som kan lades med 24VDC. Vi valgte å bruke Biltema sin batterilader som kan lades både med 12VDC og 24VDC. Batterikapasitet ligger på 70Ah hver, vår kalibreringsvogn bruker maksimalt 300W og har en effektiv operasjonstid på fire timer. Dette tilsvarer en hel arbeidsdag for Miros sine ansatte, som betyr at ladetid kan være opptil 12 timer.

7.9.1 Ladekontakt

Siden vårt system skal ha en batteripakke så må batteriene lades. For å gjøre lading av batteriene lettere og autonomt så har vi bestemt oss for å bruke en magnetisk ladekontakt. Selskapet Rosenberger i Tyskland produserer slike magnetiske kontakter.

Figur 7.7: Ladekontakt

7.10 Omformer

Siden vi har en spenningskilde på 24VDC valgte vi å gå for to DC-spennings omformerme.

• 24VDC til 48VDC med 350W effekt. Denne blir brukt til CleatPaths motor, siden 24VDC vil gi motoren for liten effekt til å skape framdrift på kalibreringsvogna. Ved 48VDC vil motoren gis den kraft som trengs.

• 24VDC til 5VDC med 25W effekt. Denne blir brukt til Raspberry Pi 3 type B fordi Raspberry Pi opererer på 5VDC. Hvis denne spenningkilden overstiges kan vi skade Rasberry Pi, noe som fører til kortslutning av kretsen eller skade på andre sensorer og motorstyringskompomenter.

7.11 Sikkerhetssensor

For å minimalisere skader i prosjektet vårt har vi bestemt oss for å bruke en sensor som skal stå for sikkerhet. Vi har sett på mange forskjellige sensorer og kom frem til en enkel ultrasonic sensor. Sensoren skal detektere og gi varsel når den oppdager at objekter befinner seg i kjørebanen. Registreringsområdet til sensoren er fra tre centimeter til fire meter i avstand. Sensoren trenger strømforsyning på 5V og skal bli styrt av Raspberry Pi.

Figur 7.8: Sensor[48]

7.12 Utstyrliste for komponenter

Denne utstyrslisten er henvist til for å gi en oversikt over hvilke komponenter vi har valgt å bruke for løsning av problemstillingen. Dette er en detaljert oversikt over funksjonaliteten og bruksområdet komponentene har.

Første komponent er en Capacitive sensor som skal samarbeide med neste komponent Rotary encoder for nøyaktig måling av avstandspunktene. For at vognen skal kunne bevege seg til punktene autonomt har vi valgt å bruke en DC motor. Videre er ledninger og klemmer til hjelp for spenningsoverføring. DC/DC omformere som skal egne spenningen til motoren og mikrokontrolleren. Vi har en spenningskilde som består av et 12V batteri og en batterilader. Til slutt vises det en Raspberry pi 3 type B mikrokontroller og en ultrasonic sensor for sikkerhet.

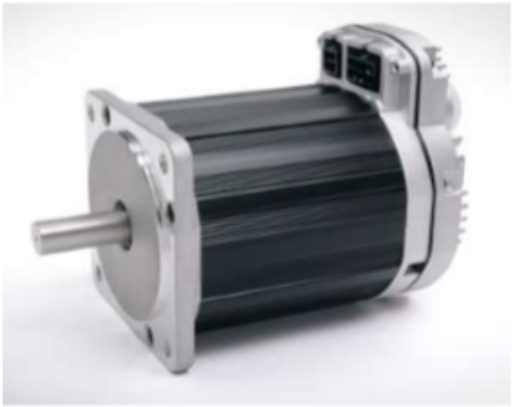
Komponentnummer	**1**
Navn	**Capacitive Sensor**
	**Funksjonalitet** Skal gjenkjenne små metallbiter skrudd fast i veibanen med halvmeter avstand mellom hver. **Bruksområde** Skal plasseres under vognen i vertikal linje med RF.

Figur 7.9: Capacitive sensor

Komponentnummer	**2**
Navn	**Rotary Encoder**
	**Funksjonalitet** Skal finjustere avstandsverdien ved hjelp av omdreiningene. **Bruksområde** Festes i hjulet bak som ikke er motordrevet. Dette for å unngå verdier fra eventuell spin i hjulene foran.

Figur 7.10: Rotary encoder

Komponentnummer	3
Navn	ClearPath-MCPV



Funksjonalitet
Skal styre vognen frem og tilbake med funksjoner gitt av mikrokontrolleren.
Bruksområde
Skal drive hjulene foran i vognen.

Figur 7.11: ClearPath-MCPV

Komponentnummer	4
Navn	DC Power kabel

Funksjonalitet
Skal muliggjøre spenningsoverføring mellom motor og batteri.
Bruksområde
Mellom batteri og motor.

Figur 7.12: DC Power kabel

Komponentnummer	5
Navn	Kontrollkabel

Funksjonalitet
Dette skal overføre spenningen mellom diverse komponenter.
Bruksområde
Spenningsforskyvning mellom komponenter.

Figur 7.13: Kontrollkabel

Komponentnummer	6
Navn	DC/DC omformer(48V)

Funksjonalitet
Dette skal omforme spenningen fra 12V til 48V.
Bruksområde
Justere spenningen fra batteriet til motoren.

Figur 7.14: DC/DC omformer (48V)

Komponentnummer	7
Navn	DC/DC omformer (5V)

Funksjonalitet
Dette skal omforme spenningen fra 12V til 5V.
Bruksområde
Justere spenningen fra batteriet til mikrokontrolleren.

Figur 7.15: DC/DC omformer (5V)

Komponentnummer	8
Navn	12V DC bilbatteri (65At)

Funksjonalitet
Skal virke som spenningskilde i systemet.
Bruksområde
Plassert i vognen, videre gi strøm til diverse komponentene.

Figur 7.16: 12V DC bilbatteri (65At)

Komponentnummer	9
Navn	Batterilader

Funksjonalitet
Skal lade batteriet.
Bruksområde
Skal festest foran reflektorvegg, ikke i vognen.

Figur 7.17: Batterilader

Komponentnummer	10
Navn	Batteriklemme

Funksjonalitet
Hjelper med enklere tilkobling og bedre strømforsyning.
Bruksområde
Skal festes i polene til batteriet.

Figur 7.18: Batteriklemme

Komponentnummer	11
Navn	Raspberry pi 3 type B

Funksjonalitet
Skal behandle verider for å holde kontroll på diverse komponenter for måling og styring.
Bruksområde
Skal plasseres midt mellom komponentene.

Figur 7.19: Raspberry pi 3 type B

Komponentnummer	12	
Navn	Ultrasonik avstandssensor	
		**Funksjonalitet** Sikkerhetsmessig vite om det er noen gjenstander foran vognen. **Bruksområde** Skal plasseres i hvert sitt hjørne av vognen.

Figur 7.20: Ultrasonic avstandssensor

7.13 Bill of materials

Tabell 7.7 viser foreløpig bill of materials for vårt prosjekt. Her vises alle de mest kritiske komponentene som vi trenger for å utføre dette proskjektet. Det vil senere følge med en diverse bill of materials som inkluderer alle forskjellige små kompomenter som blir benyttet til oppkobling.

Komponent nr.	Produkt	Produsent	Antall	Pris per stk	Sum	HyperLink
1	Capacitive Sensor	Carlo Gavazzi	1	786.00 kr	786.00 kr	Capacitive sensor
2	Rotary Encoder	Kubler	1	1385.00 kr	1385.00 kr	Rotary Encoder
3	ClearPath-MCPV	Teknic	1	520USD (4249 kr)	520USD(4249 kr)	DC motor
4	DC Power Cable,10ft	Teknic	1	19 USD (155.25 kr)	19 USD (155.25 kr)	Power Cable
5	Controller Cable, 10ft	Teknic	1	23 USD (187.94 kr)	23 USD (187.94 kr)	Controller Cable
6	DC/DC omformer(48V)	Mean Well	1	928.00 kr	928.00 kr	DC/DC omformer (48V)
7	DC/DC omformer (5V)	Mean Well	1	225.00 kr	225.00 kr	DC/DC omformer (5V)
8	12VDC bilbatteri (65Ah)	SØNNAK	2	1290.00 kr	2580.00 kr	12VDC batteri
9	Batterilader	Biltema	1	1299.00 kr	1299.00 kr	Batteri lader
10	Batteriklemme	MTA	4	20.90 kr	83.60 kr	Batteriklemme
11	Raspberry Pi 3 type B	Raspberry Pi	2	603.00 kr	1206.00 kr	Raspberry Pi
12	Ultrasonisk avstandssensorer	Raspberry Pi	4	55.00 kr	220.00 kr	Ultrasonisk avstand sensor
				Total sum:	13304 kr	

Tabell 7.7: Bill of materials

Komponent nr:	Produkt:	Produksent:	Antall :	Pris per stk:
1	M6 skrue 40mm	Ukjent	2	3.02 kr
2	M6 skive	BOSSARD	6	0.25 kr
3	M6 skrue 25mm	BOSSARD	4	1.64 kr
4	M6 mutter	Richo	6	0.83 kr
5	RPI hat RS232	Seed	1	106.00 kr
6	RPI hat RS422	Ukjent	1	305.00 kr
7	Terminal blokk blå	Wago	12	6.94 kr
8	Terminal blokk grå	Wago	6	8.11 kr
9	Aluminium stang	Ukjent	1	149.9 kr
10	Målebånd klasse 2	Stanley	1	349.9 kr
11	Sikring 10A	Schenider	2	392.00 kr
			Sum :	1845.00 kr

Tabell 7.8: Bill of materials for diverse deler

8 TEKNISK UTFØRELSE ELEKTRO

Dokumenthistorie			
Versjon	**Dato**	**Endret av**	**Aktivitet**
2.1	14/05/18	EAD	Dokument opprettet
2.2	19/05/18	TT, EAD	Tekniske ansvarsområder elektro
2.3	19/05/18	TT, DK	Kommunikasjon
2.4	19/05/18	EAD	Styring av vogn
2.5	19/05/18	TAT	Posisjonering
2.6	20/05/18	EAD	Adapter deler
2.7	20/05/18	VS	Revidert for publisering
3.0	21/04/18	TT	Publisert i rapport v3.0

Tabell 8.1: Dokumenthistorie for teknisk utførelse elektro

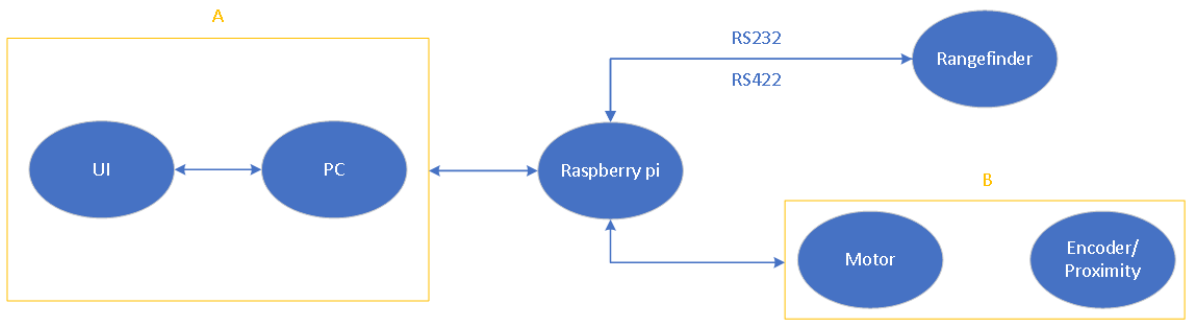
8.1 Teknisk ansvarsområde elektro

Navn	Thomas Trinh	Deivydas Kazokas	Tuan Anh Trinh	Ecir Ali Doganci
Tekniske ansvarsområder	Kommunikasjon mellom RPi og PC	Kommunikasjon mellom RPi og RF	Posisjonering (encoder og proximity)	Motorstyring (MSP)
Forklaring av ansvarsområder	RPi skal kunne sende og motta beskjeder/signaler/verdier fra PC (UI).	RPi skal kunne sende og motta beskjeder/signal/verdier fra RF med RS232 og RS422.	Posisjoneringa av vognen skal være presis med encoder og proximity som målemetode.	Motor skal kunne kjøre frem, tilbake og stoppe. Akselerasjon og bremsing.
Programmer og metoder vi har brukt.	Azure, MQTT, SQL, Mosquitto, Python shell, Jason, Dropbox, Node-RED, RS232/422 HAT, PuTTy, SFTP og Visual Studio Code med Python utvidelse.		Visual Studio Code med Python utvidelse for måleskript. PuTTy og SFTP for fil overføring.	Visual Studio Code med Python utvidelse for motorfunksjoner som styrer motoren. PuTTy og SFTP for fil overføring.
Samarbeid mellom medlemmer	Thomas samarbeidet med Deivydas. De har brukt de samme programmene og har utført det meste sammen for å få til kommuikasjons delen.		Tuan samarbeidet med Ecir. De fikk implimentert kodene sine sammen og fikk vogna kjørt til gitte distanserverdi med milimeter presisjon.	

Figur 8.1: Teknisk ansvarsområde elektro

8.2 Kommunikasjon

For at systemet vårt skal fungere funksjonelt må vi ha en kommunikasjon mellom våre enheter. Måten vi har tenkt å løse dette på er vist på fig 8.2.



Figur 8.2: Kommunikasjonsmetode

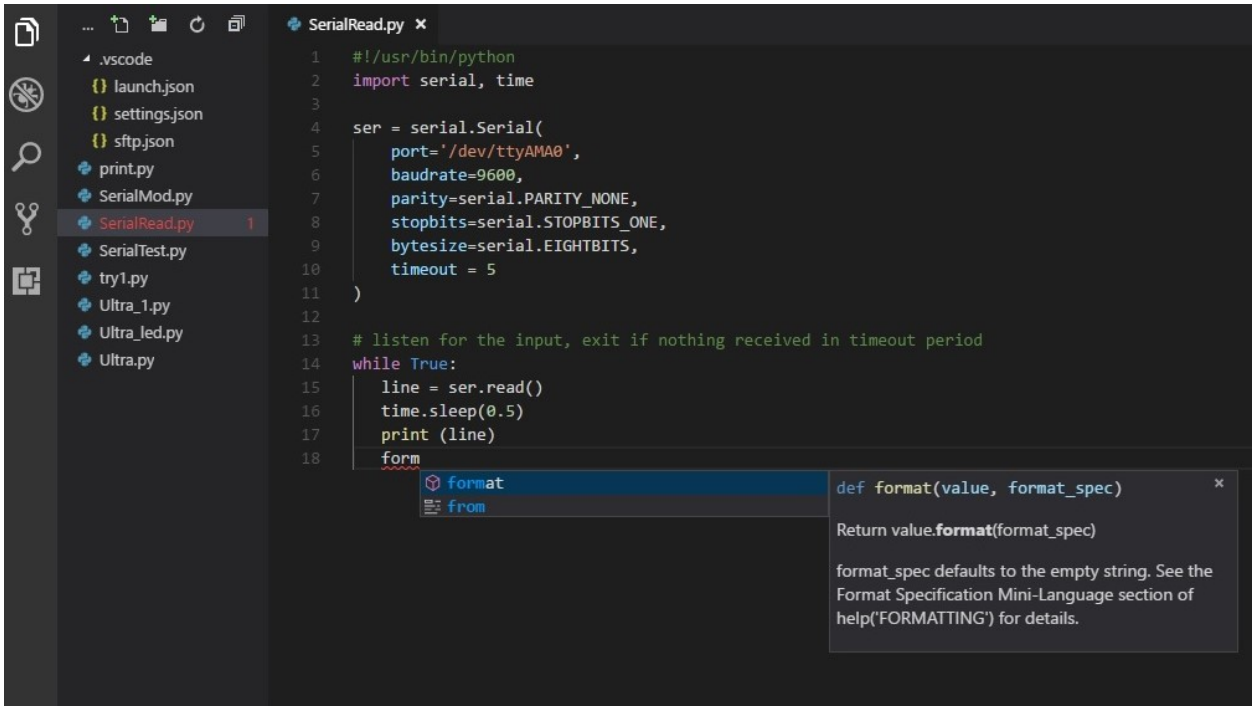
Videre skal vi ta for oss de forskjellige programmene og kommunikasjonsmetodene som vi har tenkt å bruke i systemet.

8.2.1 Programmer

Visual Studio Code

Elektrostudenter har brukt programvaren Visual Studio Code [51] for å lage skripter i kodespråket Python og JSON. Selv om Visual Studio Code er en enklere versjon av Visual Studio, støtter denne programvaren hundrevis av programmeringsspråk. Brukergrensesnittet til dette programmet er enkelt og inneholder lite, men har nødvendige funksjoner. Programvaren støtter operativsystemer som Linux, Windows og Mac.

Visual Studio Code inneholder mange hjelpeutvidelser som IntelliSense, syntaksutheving, automatisk brakettsammenstilling, utdrag og debugging (feilsøking). En av de viktigste utvidelsene som følger med i Visual Studio Code er IntelliSense vist i fig 8.3. Utvidelsen hjelper programmereren til å definere hva som skal inn i løkker og vil dermed holde skriptet uten stavefeil for funksjoner, variabler osv.



Figur 8.3: IntelliSense funksjon

Utvidelser for Visual Studio Code

Det finnes mange utvidelser til Visual Studio Code programvaren. Brukeren kan fritt laste ned og installere mange forskjellige utvidelser som Python, SFTP, Git og GitHub. Elektrostudenter benyttet denne utvidelsen til å lage flere Python skript i kodespråket. Videre ble det brukt SFTP utvidelsen for å overføre disse skriptene over til Raspberry Pi helt trådløst. Begge utvidelsene vises i fig 8.4.

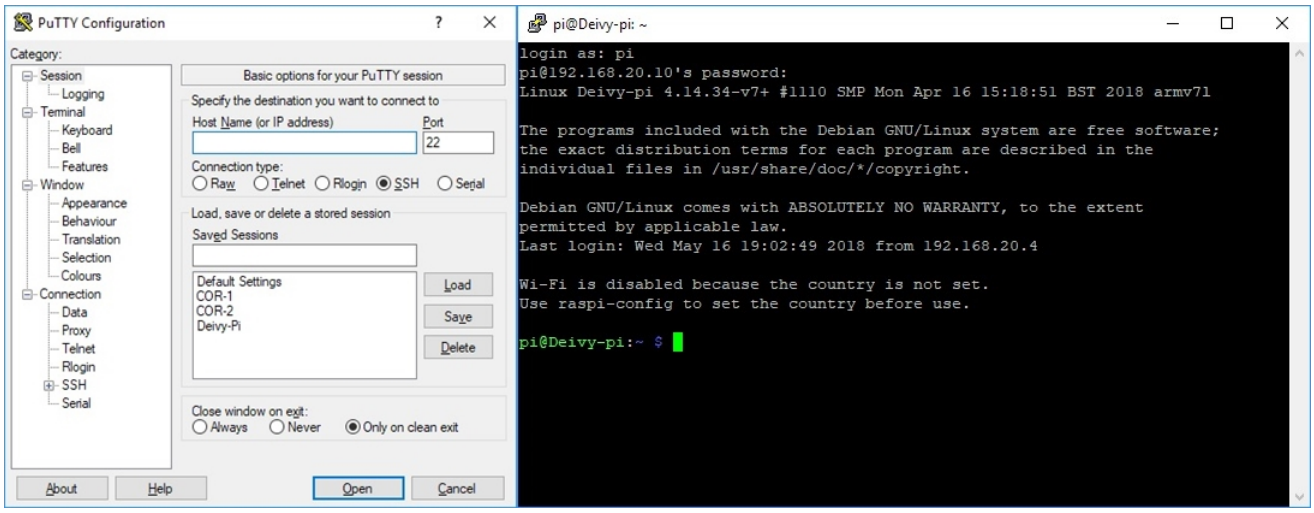
Figur 8.4: Python og SFTP utvidelse

Fordelen med Visual Studio Code er størrelsen. Visual Studio Code tar ganske lite plass og har kun IDE for koding. Som nevnt tidligere kan brukeren laste ned ønskede utvidelser og konfigurere programvaren slik brukeren ønsker det

Ulempen med Visual Studio Code er at programmet ikke har kapabiliteten til å visualisere skript i forhold til applikasjoner og ønskede visualiserte utganger. Til visualisering av slike skript, bør en full versjon av Visual Studio brukes.

PuTTy

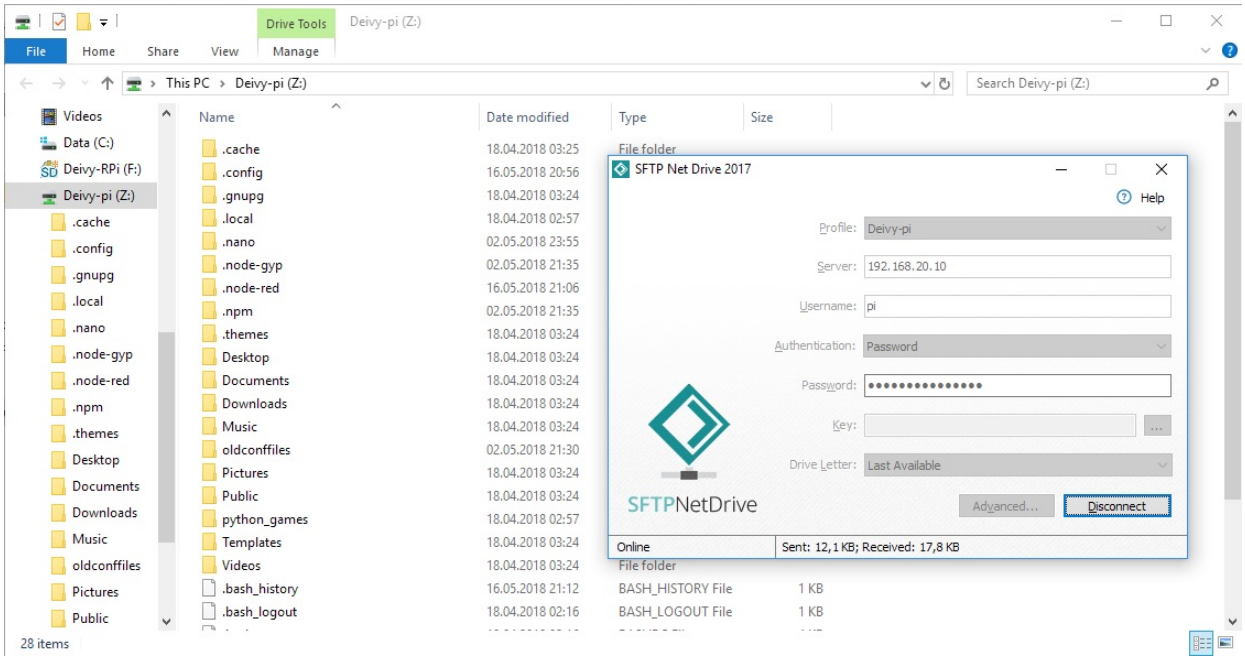
Siden Raspberry Pi er en liten datamaskin trengs det skjerm, tastatur og mus for å styre den. En annen mulighet er å styre Raspberry Pi via egen PC trådløst ved hjelp av PuTTy programvaren. PuTTy [33] er gratis og er et «open source» program som gir tilgang til terminalen for å utføre ønskede kommandoer. PuTTy ble først utviklet av Simon Tatham og videre utviklet av frivillige medlemmer. Programmet er enkelt og støtter mange tilkoblingsmuligheter som seriell kommunikasjon via en ledning, Secure Shell (SSH) trådløs kryptert tilkobling osv. I dette prosjektet ble SSH benyttet for å utføre produserte skript for Raspberry Pi fra egen PC. Prosedyren for å logge inn er vist i fig 8.5.



Figur 8.5: PuTTy

SFTP Net Drive

Secure File Transfer Protocol eller SFTP er basert på SSH (Secure Shell) protokoll [41]. Brukeren kan logge inn på eksterne lagringsplasser som Raspberry Pi med en IP adresse, brukernavn og passord som vist i fig 8.6. Programmet lager nettverksdisk for ekstern lagringsplass og fordelen med dette er at brukeren kan se alle filer og mapper som er lagret på maskinen. SFTP Net Drive programmet ble brukt til å overføre, undersøke og lagre ønskede filer fra PC til Raspberry Pi.

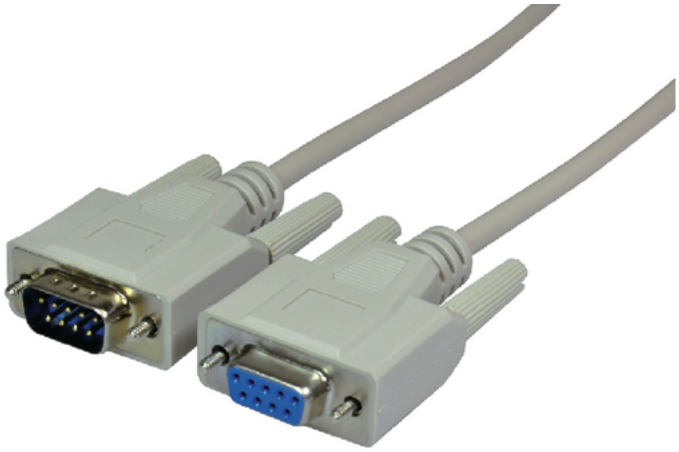


Figur 8.6: SFTP

8.2.2 Kommunikasjon mellom RF og PC/RPi

RangeFinder

RangeFinder har to serielle kommunikasjonstilkoblinger som er RS232 og RS422. Begge tilkoblingene bruker separate DB-9 kabler som vist i fig 8.7 for overføring av data signaler fra PC til RangeFinderen. Videre har RangeFinder Ethernet tilkoblingsmulighet, men det ble ikke brukt i dette prosjektet.



Figur 8.7: DB-9 kabel

Moxa uport 1150

I tidligere system ble det brukt en Moxa uport 1150 kabel som vises på fig 8.8. Moxa uport 1150 er en USB kabel for RS232 og RS422 kommunikasjon mellom PC og RangeFinderen. Moxa kabel følger med drivere til mange operative systemer som Mac, Windows, Linux og Raspbian (Linux basert system). Siden driveren for Moxa kabel støtter ikke nyeste versjon av Raspbian operativt system kunne vi ikke bruke USB kabel til å lese signalet fra RangeFinderen på Raspberry Pi.

Figur 8.8: Moxa uport 1150 USB kabel

UART

Etter en del undersøkelser har vi kommet frem til en mulig løsning med en RS422/RS485 Shield. Komponenten som blir nevnt senere i dokumentet, skal kommunisere serielt med RangeFinderen via UART pinner på Raspberry Pi. UART [47] er en mikrochip og står for «Universal Asynchronous Receiver/Transmitter» og kontrollerer grensesnittene for tilkoblede serielle enheter. UART mikrochipen konverterer mottatte bytes fra prosessoren og overfører til enkel-seriell-bit strøm som videre sendes til en seriellkoblet enhet. Videre vil chipen konvertere mottatt data fra enheter til bytes som er forståelig kommunikasjonsspråk for prosessoren og utføre funksjoner. En annen funksjon for UART er å legge til paritetsbit om det er ønskelig. Videre UART kontrollerer datastrøm med stopp og start bit. UART blir brukt mest til å kommunisere og utveksle data mellom enheter med seriell kommunikasjon, og RS232 kommunikasjonsprotokoll er den mest brukte.

Aktivering av UART

Det er en del konfigurasjoner som skal utføres før brukeren kan bruke UART pinnene til å lese og sende signalene. For konfigurering av Raspberry Pi informasjon ble hentet fra offisiell Raspberry Pi sin nettside [36]. Første steget er å aktivere UART pinner med nummer på brettet 8 og 10. Dette kan gjøres via en kommando som heter «sudo nano /boot/config.txt» og skrive inn «enable_uart=1» nederst i vindu slik det blir vist i Vedlegg M.

Endring i oppstartfase

Etter pinnene har blitt aktivert trengs det endringer i oppstartssekvensen. Grunnen til dette er at under oppstarten sjekker operativsystemet UART pinnene som blir brukt til systemets konsoll. For å stoppe at dette skjer hver gang Raspberry Pi blir slått på må cmdline.txt fil konfigureres. Ved å skrive «sudo nano /boot/cmdline.txt» inn i kommandoterminalen og fjerne «console=serial0, 115200» kan brukeren stoppe automatisk sjekk ved oppstart slik det blir vist i Vedlegg M.

Flytting av Bluetooth

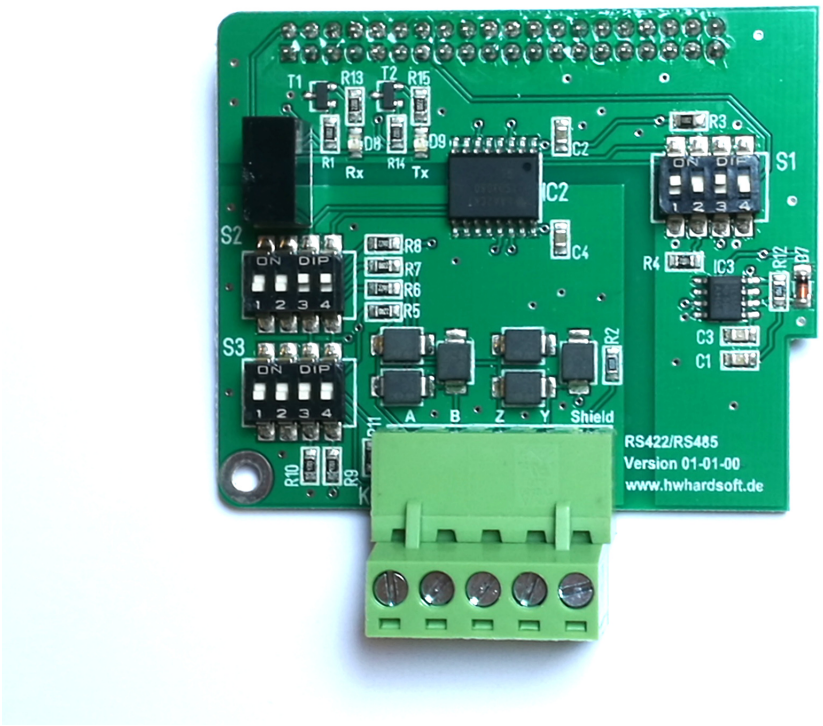
Raspberry Pi bruker nevnte UART pinner til Bluetooth. Dermed kan vi flytte Bluetooth til mini UART (ttyS0) eller skru av denne tjenesten hvis det er ingen behov for det. Flytting av Bluetooth til mini UART (ttyS0) kan gjøres ved å skrive «sudo nano /boot/config.txt» kommando på terminalvindu. Nederst i terminalen bør det skrives inn «dtoverlay=pi3-miniuart-bt» som vist i Vedlegg M (flytting av bluetooth).

Fjerning av Bluetooth

Hvis det er ikke behov for Bluetooth eller ønskelig å fjerne dette tjenesten kan det skrives en annen funksjon på «sudo nano /boot/config.txt» filen. For dette bør det skrives «dtoverlay=pi3-disable-bt» nederst i terminalen som vist i Vedlegg M (fjerne bluetooth). Grunnen for at Bluetooth skal flyttes eller fjernes er at vi skal bruke pinnene 8 og 10 til UART seriell kommunikasjon med full båndbredde.

RS422/485 HAT

Etter nevnte konfigurasjoner til Raspberry Pi har vi montert en komponent som skal kommunisere med RangeFinderen via RS422 kommunikasjonsprotokoll. Komponenten blir laget av Hartmut Wendt og vises på fig 8.9 (shield). Komponenten bruker spenningspinner og UART pinner som ble konfigurert tidligere og detaljert tilkobling kan sees i Vedlegg H. På kretskortet er det en del brytere som kan konfigureres til ønsket bruk. Dokumentasjon til kretskortet og konfigurasjon av brytere kan sees på hjemmeside til produsenten [16]. I vår tilfelle konfigurerte vi kretskortet til RS422 kommunikasjonsprotokoll og RangeFinderens Tx-A og Tx-B ble koblet inn på A og B terminalene på RS422/RS485 kretskortet for å motta signalene fra RangeFinderen.



Figur 8.9: Shield for RS422/RS485 [37]

Python skript for kommunikasjon

Ved hjelp av Python skript kan vi lese signalet fra RangeFinderen. Skripten vises på fig 8.10. For at Python skal lese seriellkommunikasjon trengs det installasjon av seriellbibliotek og det kan lastes ned ved hjelp av «sudo apt-get install python-serial» kommando i terminalvinduet til Raspberry Pi. Skripten bruker konfigurerte porten for UART som heter «ttyAMA0». Båndbredde settes til «9600» bits per sekund. Siden dette er standard for generell kommunikasjon og økning i datahastighet vil skape flere problemer underveis. Videre velger vi ingen paritetsbit, grunnen til dette er at vi ikke skal summere bitene for å sjekke at disse er i oddetall eller partall. Stoppbit og startbit er synkroniseringsbitene og er en bit hver i vårt tilfelle. Bytestørrelse ble satt til 8 bits. Dette betyr at opp i en byte sendes det 8 bits og med båndbredde på 9600 bits per sekund sendes det 960 bytes med informasjon i løpet av et sekund. Videre i skripten lages det en «while» løkke for å lese data som strømmer inn fra RangeFinderen, denne dataen blir skrevet ut på et terminalvindu.

```
2  import serial, time
3
4  ser = serial.Serial(
5      port='/dev/ttyAMA0',
6      baudrate=9600,
7      parity=serial.PARITY_NONE,
8      stopbits=serial.STOPBITS_ONE,
9      bytesize=serial.EIGHTBITS,
10     timeout = 5
11 )
12
13 # Listen to the input, if nothing is received in timeout period exit
14 while True:
15     line = ser.read()
16     time.sleep(0.5)
17     print (line)
```

Figur 8.10: Python skript for lesing av data

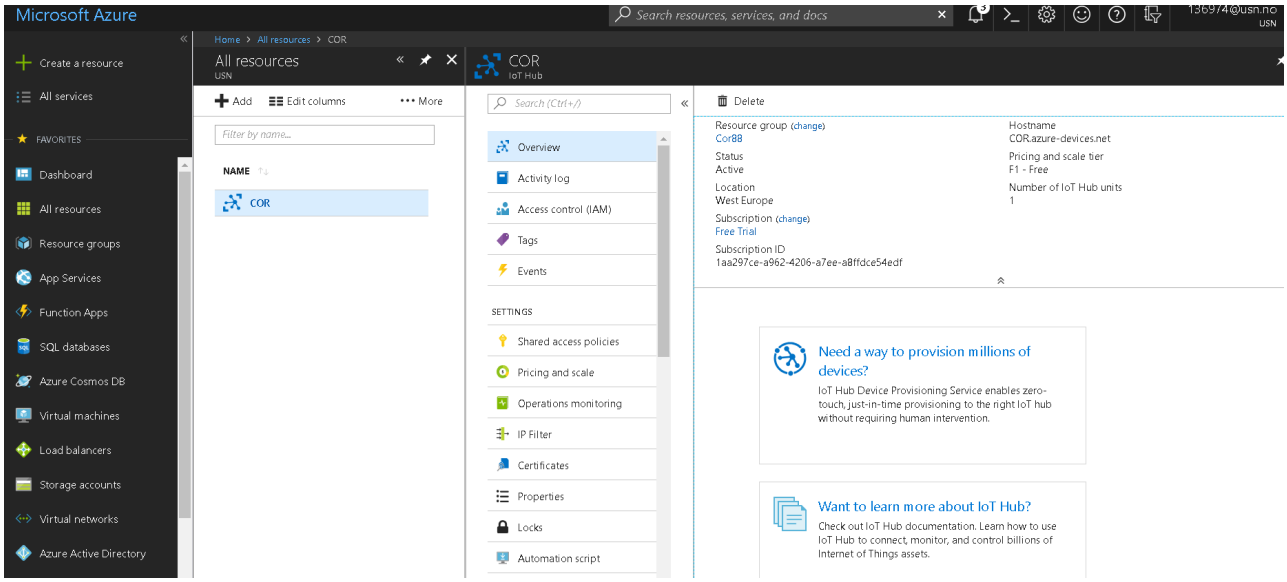
8.2.3 Azure IoT

IoT (Internet of Things) handler om å koble enheter til internett, hvor det er mulig å overføre data, samt styre enheten. Det er også mulig å sende data mellom hverandre så lenge de er koblet til internett. Dette kan brukes til å styre en lyspære til å slå seg av og på via en mobil applikasjon/PC.

Det er veldig enkelt å koble enheter til hverandre, alt de trenger er internett eller at de er koblet til det samme nettverket. Dermed kan dette brukes som vår kommunikasjonsmetode mellom RPi og PC, siden vi skal overføre data frem og tilbake [17].

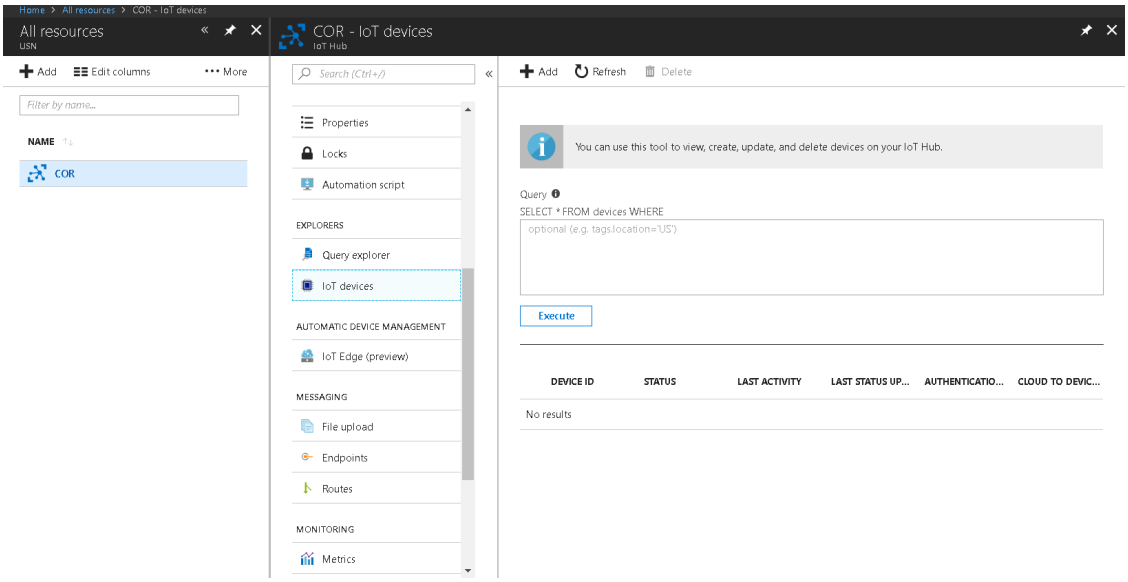
Microsoft Azure IoT inneholder de funksjonene som vi trenger til å kommunisere mellom enheter. Med Azure kan vi lage et brukergrensesnitt, lagre data, analysere data og styre andre enheter ved å gi dem en kommando eller input. For å bruke Azure må det registreres en bruker, og det finnes en gratis «versjon» som kan brukes med begrenset tilgang. Det finnes utallige funksjoner på Azure, men for å bruke Azure optimalt så man oppgradere «versjonen». De funksjonene vi trenger til kommunikasjon er hvordan vi kan overføre data mellom enheter, lagring av data og styre enheter med Azure. Dette er tilgjengelig i «gratis» versjonen av Azure [2].

Azure IoT Hub er en «cloud». Med IoT Hub får vi en egen hostname som skal brukes til å koble andre enheter til Hub'en. På Hub'en finnes det mange funksjoner som er tilrettelagt til bruk av IoT som vist på figur 8.11.



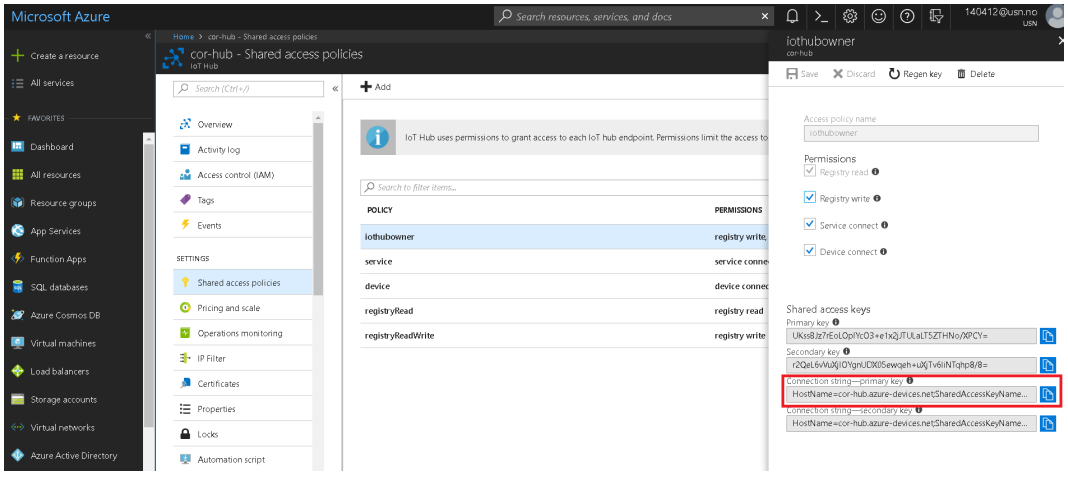
Figur 8.11: Azure IoT Hub

På IoT devices er det en oversikt over hvilken enhet som er koblet til Hub'en som vist på fig 8.12. devices. Her er det mulig å endre, lage nye eller slette enheter som er koblet til Hub'en. Det er to ting som trengs for å koble en enhet med Hub'en, det første er «hostname» og det andre er «device ID». Alle «Device ID» som blir koblet til Hub'en har en egen identifikasjonsnøkkel.



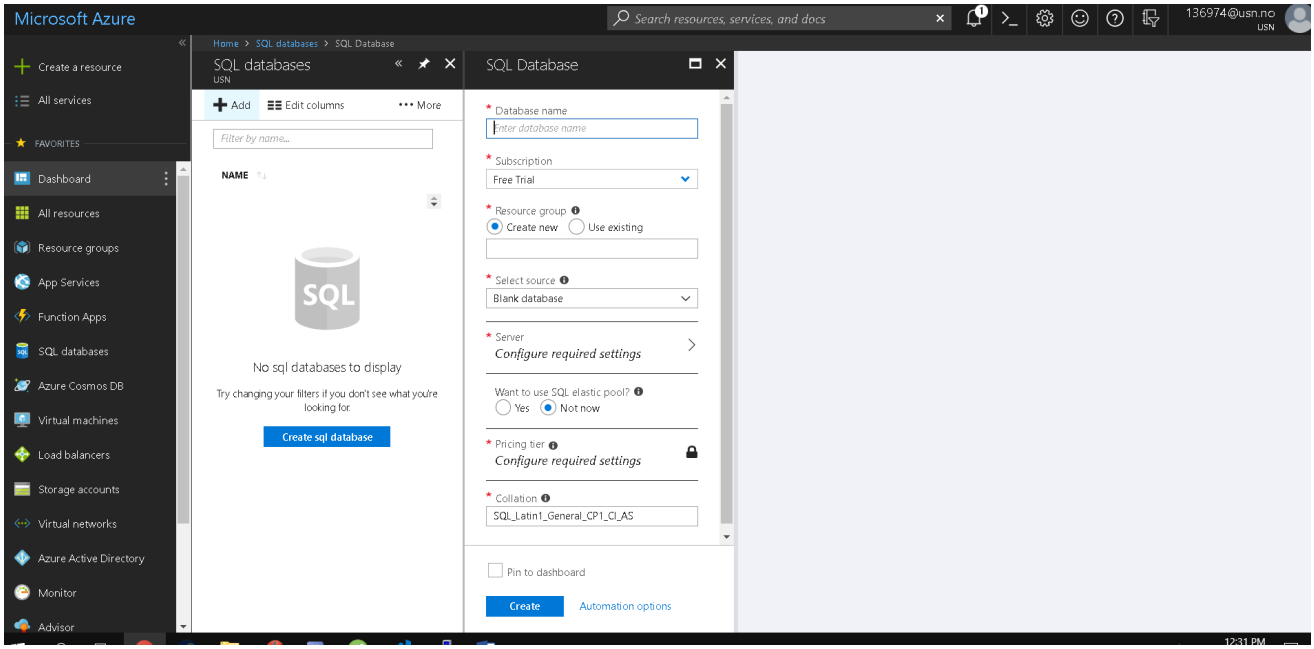
Figur 8.12: Device oversikt

For å koble «hostname» og «Device ID» sammen må vi ha en «connection string- primary key». Denne er viktig fordi den forbinder enhetene til samme kommunikasjonskanal. Den røde markeringen som vist på fig 8.13 er der man kan finne «connection string» [12]



Figur 8.13: Connection string

På Azure er det mulig å koble til en SQL-database til lagring av data. Dette skjer ved at vi kobler SQL forbindelsen med Azure som vist i fig 8.14. Brukergrensesnittet skal kunne sende data til SQL-databasen som er koblet til azure, slik at azure kan videreføre dataen videre til RPi ved hjelp av Node-RED.



Figur 8.14: SQL

Denne metoden mente vi var den mest optimale vi kunne bruke til å kommunisere med våre enheter. Vi kan både sende og motta data gjennom IoT Hub'en, alt blir lagret i SQL-databasen. Det finnes utallige funksjoner som kan implementeres med tanken på videreutvikling av systemet.

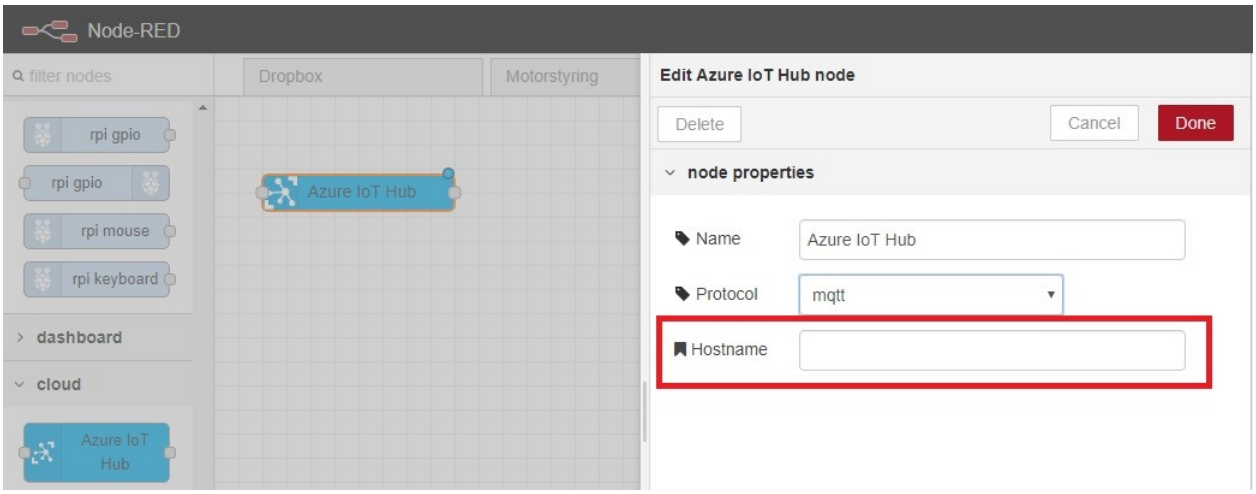
8.2.4 Node-RED

Node-RED [26] er et «flow» basert programmeringsverktøy som er basert på JavaSkript og er en del av «Internet of Things». De første utviklerne av programmet var IBM Emerging Technology team og ble videreutviklet av JS Foundation. Node-RED er visualisert programmering der brukeren kan benytte bokser som kalles for noder. Nodene har godt definerte funksjoner og blir brukt til å behandle og styre informasjon eller data. For å lage enkle kretser med noder trengs det grunnleggende kunnskap om programmering. Grunnen til dette er at de fleste av nodene inneholder en eller annen form for koding som blir utført i bakgrunnen.

Henvisningene for nedlastning og installasjon av slik programvare for Raspberry Pi er vist på installasjonsdokumentet på Node-RED offisielle nettside [27]. Som standard Node-RED har ingen autorisert innlogging til programmet og hvis Node-RED installeres på en enhet er det anbefalt å sette opp innloggingsfunksjon til programmet. Oppsetting av bruker og passord er vist på Node-RED offisielle nettside [28].

Azure IoT Hub Node

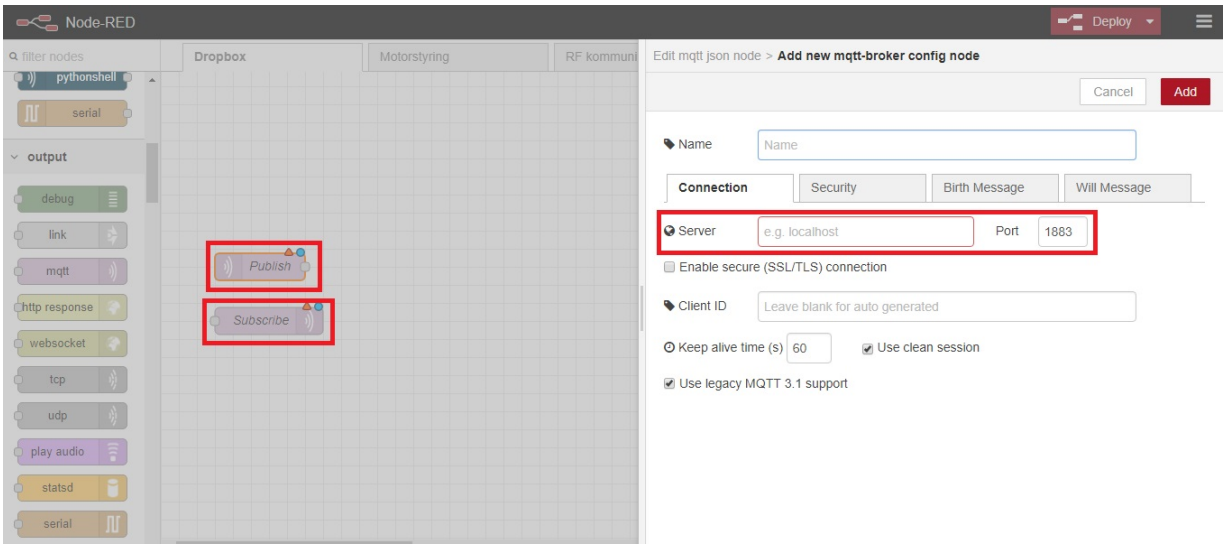
På node-RED finnes det en node for Azure IoT Hub, så da kan vi implementere vår Azure IoT Hub med Node-RED. Det er veldig viktig at den kan brukes i Node-RED, siden Node-RED er selve kjernen i vår kommunikasjonsmetode. For at Azure IoT Hub noden skal fungere trengs det hostname til IoT Hub'en slik det er vist på fig 8.15. Henvisningene for nedlastning av slik node er vist i Vedlegg A.



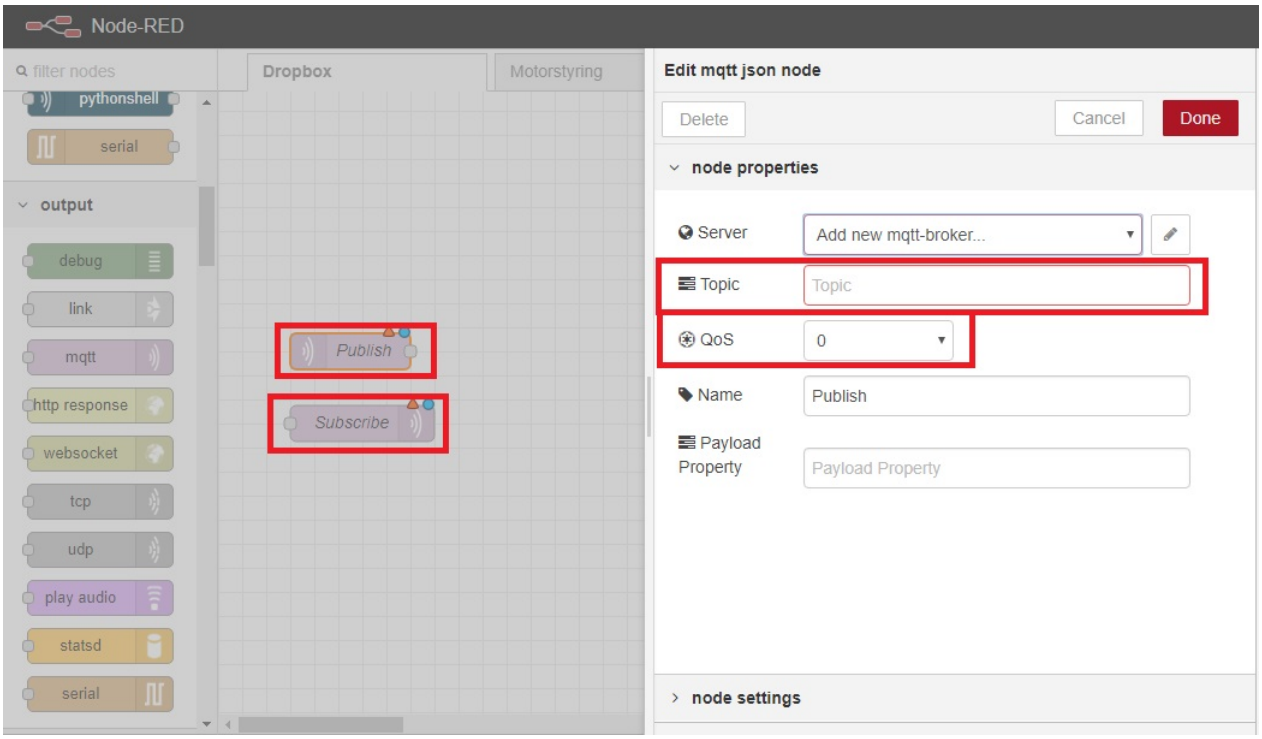
Figur 8.15: Azure IoT Hub Node

MQTT Node

På Node-RED finnes det en node for MQTT. Inni på noden til publisering og abonnering må nodene være koblet til samme «localhost» og «port» slik det er vist på fig 8.16. Videre må det defineres hvilket emne de publiserer eller abonnerer, slik at dataen kan blir overført frem og tilbake som det er vist på fig 8.17. Det er også mulig å velge hvilken type QoS vi vil sende dataen i. Henvisningene for nedlastning av slik node er vist i Vedlegg B.



Figur 8.16: MQTT tilkobling til server



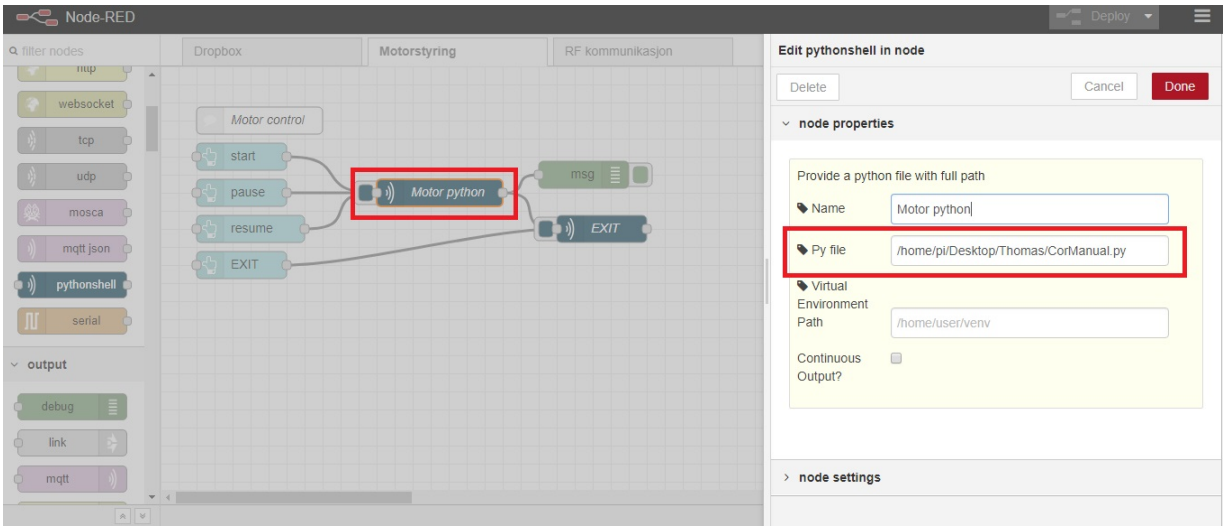
Figur 8.17: MQTT publisering og abonnering

Python Shell Node

Python shell er et terminalt vindu som brukes til å kommandere Raspberry Pi. Terminalen kan hente en Python script og utføre det. Terminalen kan bare utføre en skript om gangen. Endringer i skripten mens terminalen utfører skripten gir ikke resultat. Dermed må endringen skje før eller etter at terminalen har utført skripten.

På Node-RED finnes det en node for Python Shell, som kan utføre Python skripten. Det må skrives inn på noden hvor skripten ligger på Raspberry Pi slik det er vist på fig 8.18. På Node-RED kan vi utføre flere Python skripter både i serie og parallelt. Henvisningene for nedlastning av slik node er vist i Vedlegg C.

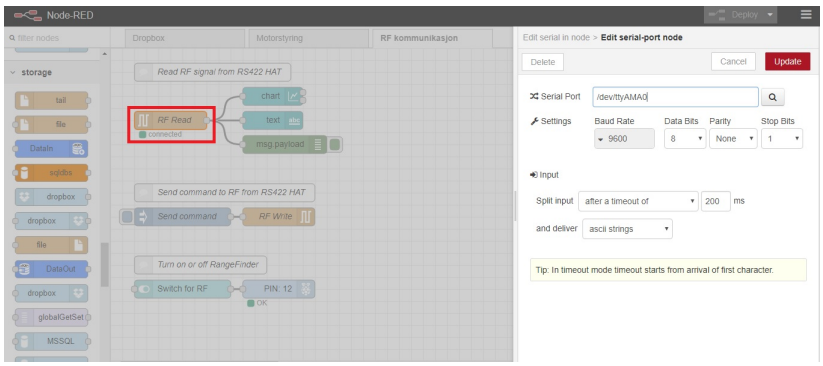
Vogna posisjonerer seg til de distanseverdiene vi taster inn på terminalen. Siden vi ikke skal bruke terminalen direkte til å utføre dette, må vi finne ut en annen metode til å gi oss distansene.



Figur 8.18: Python skript plassering lokalt

Serial Node

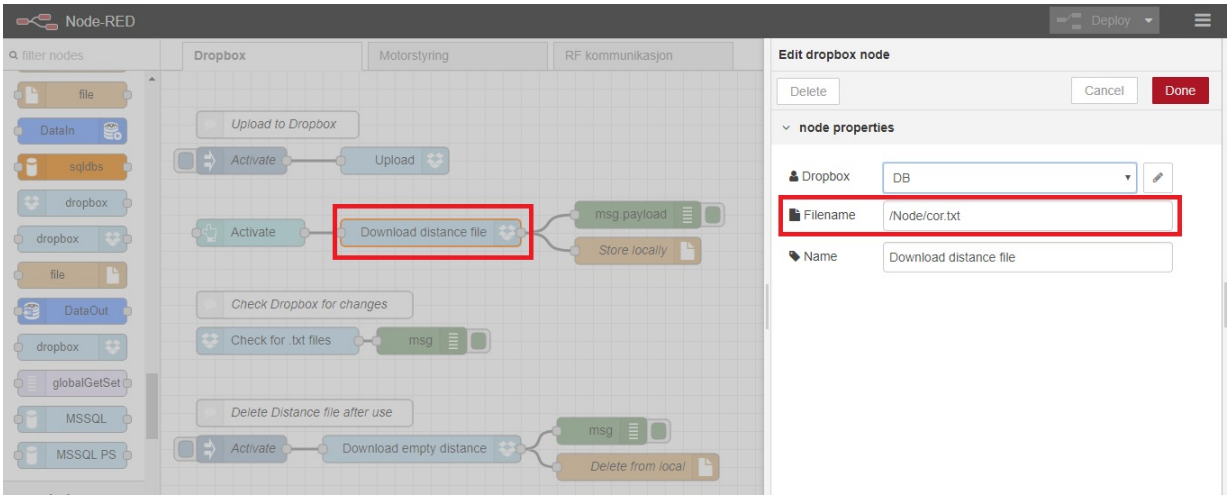
For å slippe og bruke separat Python skript til kommunikasjon med RangeFinderen kan brukeren laste ned «Serial node» på Node-RED. Henvisningene for nedlastning av slik node er vist i Vedlegg D. I «Serial Port» feltet til noden skal det skrives inn «/dev/ttyAMA0» slik det er vist på fig 8.19. AMA0 seriellport er UART pinnene på Raspberry Pi som ble konfigurert tidligere. Videre skal det velges båndbredde på 9600 bits per sekund med 8 bits datastørrelse samt skal det velges ingen paritetsbit og i siste feltet skal det velges en stopbit. Utgangen på signalet vises både på feilsøkingsvindu og på grafisk brukergrensesnitt.



Figur 8.19: Serial Node

DropBox Node

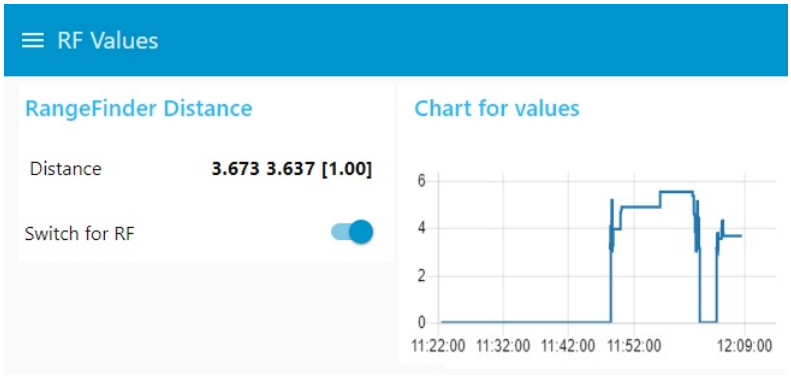
For å sende og motta konfigureringsfiler enkelt over enheter har vi bestemt oss for å bruke skylagringstjeneste. Dropbox node viste seg å være ett veldig nyttig verktøy for slike operasjoner. Henvisningene for nedlastning av slik node er vist i Vedlegg E. Som vist på fig 8.20 blir en tekstfil med distanseverdier lastet ned fra en mappe på Dropbox og vises i debug msg.payload vindu. Videre så lagres distansene ved hjalp av lagringsnode på en fil med JSON format lokalt på Raspberry Pi. Etter distanseverdier er lagret som JSON fil kan Python skript enkelt hente verdiene og utføre bestemte funksjoner.



Figur 8.20: DropBox Node

Grafisk brukergrensesnitt Node

For å visualisere utgangene eller inngangene på Node-RED kan brukeren laste ned en «Dashboard Flow». Henvisningene for nedlastning av slik node er vist i Vedlegg F. Flowen ble brukt til å visualisere RangeFinder distanseverdiene. Etter at Raspberry Pi har mottatt distanseverdiene kan Node-RED visualisere disse verdiene på hvilken som helst enhet med en nettleser ved å skrive inn «localhost:1880/ui» adresse. I fig 8.21 vises distanseverdier som ble skrevet ut i sanntid. Brukeren kan se nåværende distanseverdi av RangeFinder samtidig en graf der verdiene blir lagret i en viss tidsperiode. For å slå av og på kommunikasjon mellom RangeFinder og Raspberry Pi kan en bryter brukes som vist i fig 8.21.

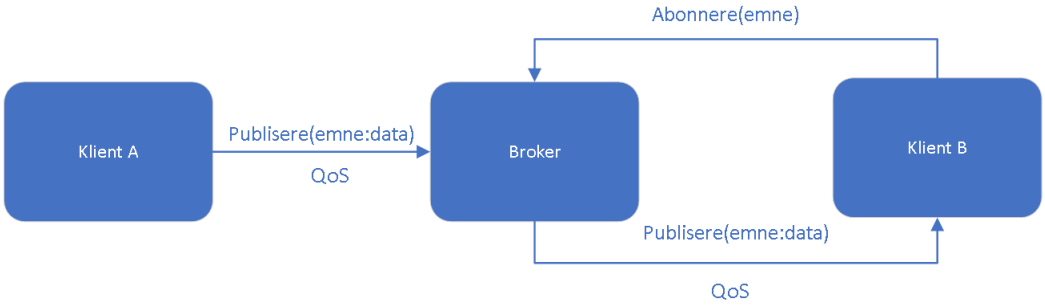


Figur 8.21: Grafisk brukergrensesnitt til RF

8.2.5 MQTT

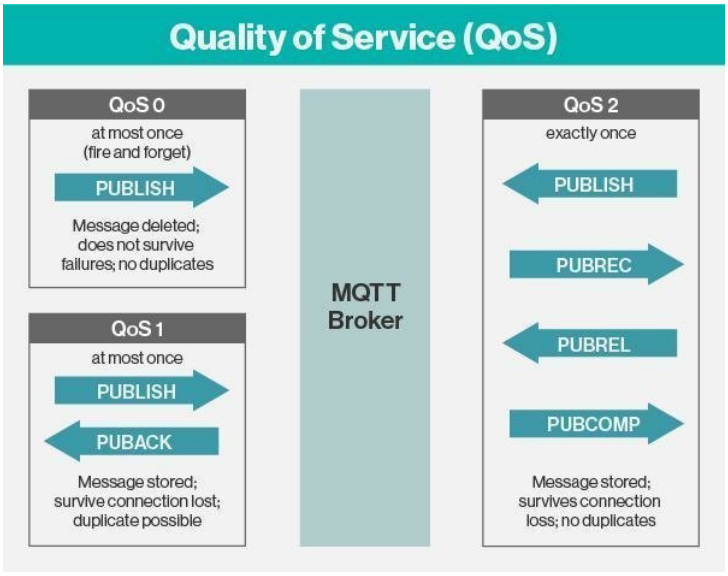
MQTT (Message Queing Telemetry Transport) er en enkelt beskjed protokoll for overføring av data, basert på TCP/IP forbindelse. Det brukes til kommunikasjon mellom maskiner eller IoT (Internet of Things). Andre liknede protokoller som fungerer på nesten samme måte er CoAP (Constrained Application Protocol) og AMQP (Advance Message Queing Protocol) [24].

For å kommunisere mellom hverandre må MQTT inneholde klienter og en broker, der både klienter og brokeren må ha TCP/IP. MQTT bruker en publisering og abonnering metode for å kommunisere. Klient A publisering et emne med data til en broker, og Klient B abonnerer det samme emnet til brokeren. Da sender brokeren data som ligger i emnet fra Klient A til Klient B. Forbindelsen mellom Klient A og klient B kommuniserer ikke direkte med hverandre, de går gjennom en broker som er vist i fig 8.22 publisering og abonnering.
Med MQTT kan det brukes tre typer QoS (Quality of Service): QoS0, QoS1 og QoS2.



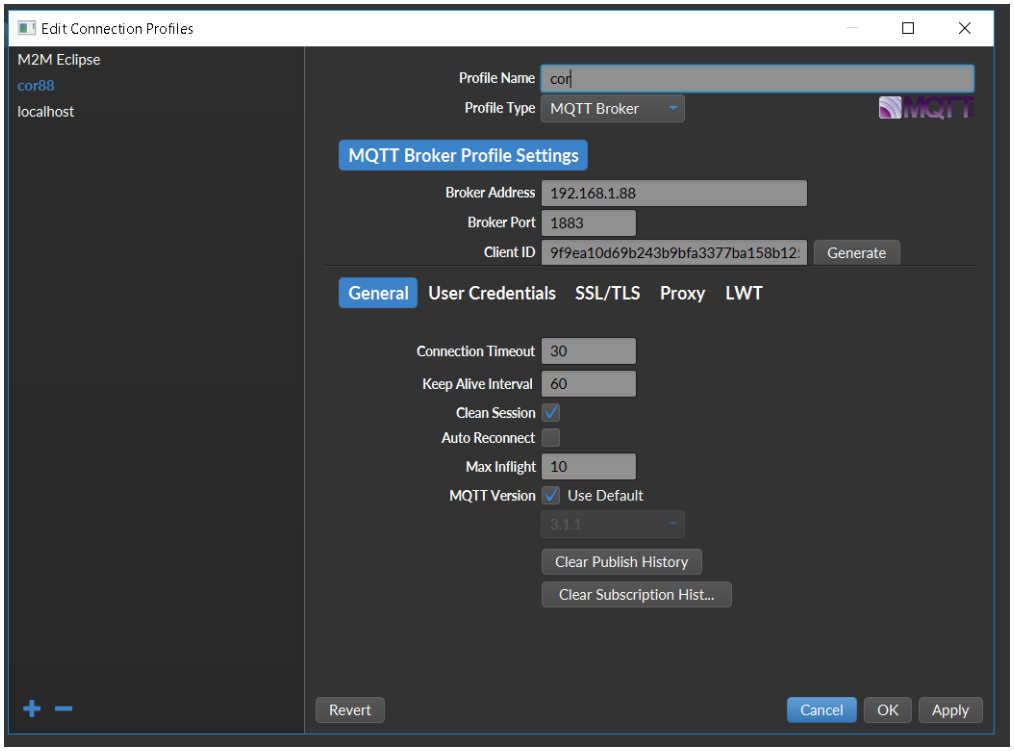
Figur 8.22: Publisering og abonnering

Det fins mange open-source brokere: ActiveMQ, JoramMQ, Mosquitto, RabbitMQ og EMQTT. Brokerene har forskjellige bruksområder, vi har valgt Mosquitto. En av grunnene til at vi valgte Mosquitto er at det er veldig egnet til RPi, siden den bruker programmeringsspråket Python. En broker kan ha mange forskjellige abonnenter samtidig. Brokeren publiserer bare til den klienten som har abonnert brokeren med samme emne. En klient kan både publisere og abonnere samtidig. QoS forteller oss hvordan brokeren skal behandle beskjeden fra en klient til en annen klient som er vist i fig 8.23.



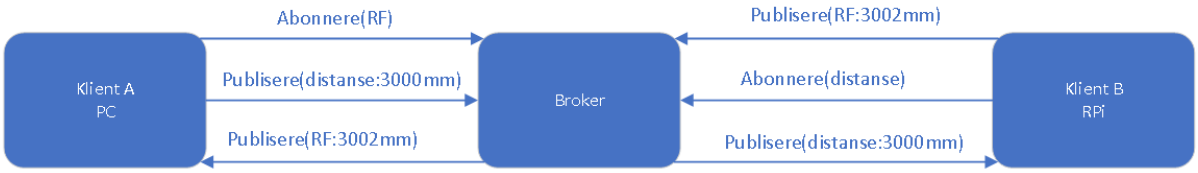
Figur 8.23: QoS

MQTT.fx er et program som er veldig enkelt å bruke til å sende og motta MQTT data. Inni i programmet trengs det å definere en broker adresse, broker port og Client ID som er vist i fig 8.24. Det er også mulig å beskytte andre fra å bruke det ved å lage en bruker og passord inni i «User Credentials» eller ved å bruke SSL/TLS.



Figur 8.24: MQTT.fx

MQTT passer til vår kommunikasjons metode mellom RPi og PC. RPi skal motta distanseverdier fra PC/brukergrensesnitt ved bruk av Node-red og MQTT.fx, så skal RPi skal implementere disse distanseverdiene inn i Python koden som er i Node-red. Etter at RPi har kjørt til de distanse skal RPi sende distansen den får av RF til PC/brukergrensesnitt som er vist i fig 8.25. PC/brukergrensesnitt er da Klient A og RPi er klient B.



Figur 8.25: MQTT med RF

8.2.6 JSON

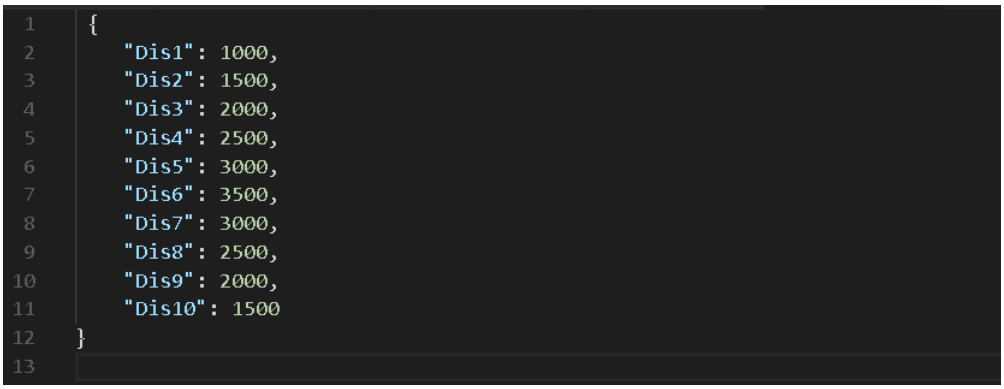
JSON (JavaScript Object Notation) er et tekstformat til datautveksling. JSON er et veldig enkelt programmeringsspråk som kan brukes til å motta, sende og lagre data. JSON har veldig nær tilnærming til JavaScript enn andre programmeringsspråk, men kan brukes i alle programmeringsspråk. Den største forskjellen mellom JSON og andre programmeringsspråk er at i JSON finnes det ikke en syntaks for kommentar. JSON er en av de vanligste dataformatene som brukes til asynkrone nettlesere og kommunikasjon mellom servere. Det er to viktige strukturer ved bruk av JSON, de er nøkkel og verdi. Hvor nøkkel er hva som skal være definert og verdi er hva nøkkel er. Verdi kan være i datatypen: string, boolean, number, array, object eller null [18].

JSON kan implementeres i et Python skript og det kan videre brukes på Node-RED. For at skriptet skal vite distanseverdiene den skal kjøre har vi implementert inn en JSON kode på Python skripten som vist på fig 8.26.

```
49  # Input distanser via JSON
50  file = open('Dis.json')
51  distanser = json.loads(file.read())
52  Dis1 = ((distanser['Dis1']))
53  Dis2 = ((distanser['Dis2']))
54  Dis3 = ((distanser['Dis3']))
55  Dis4 = ((distanser['Dis4']))
56  Dis5 = ((distanser['Dis5']))
57  Dis6 = ((distanser['Dis6']))
58  Dis7 = ((distanser['Dis7']))
59  Dis8 = ((distanser['Dis8']))
60  Dis9 = ((distanser['Dis9']))
61  Dis10 = ((distanser['Dis10']))
62
```

Figur 8.26: Funksjon for henting av JSON fil

For at Python skripten skal kunne hente inn distansene verdiene må vi ha en JSON fil hvor distansene er definert som vist i fig 8.27. Distanser definert i JSON. Disse distanseverdiene får vi fra brukergrensesnittet.



Figur 8.27: Distanse definert i JSON

Kommunikasjonen starter med at brukeren taster inn de distanseverdiene på brukergrensesnittet. Distansene blir lagret som en tekstfil på DropBox. Node-red gjør tekstfilen om til en JSON-fil, og laster det opp til RPi. Da henter Python skripten JSON-filen og implementerer det i skriptet fordi vi har skrevet en funksjon til det. Da skal vognen kjøre til de distansene. Imens dette skjer skal RPi motta distanseverdien som RangeFinder har målt i de forskjellige stoppene, disse distanseverdiene blir lagret til DropBox av Node-RED.

8.2.7 Konklusjon for valgt kommunikasjonsmetode

Vi fikk som nevnt problemer med enkelte av metodene. Ved bruk av Microsoft Azure IoT fikk vi lagret dataen på SQL-database, men det ble lagret på en komplisert måte. Det gjorde at det ble vanskelig for oss å jobbe med det videre. Et annet problem var at vi fikk bare dataen til å gå en vei. Ved starten av planleggingen mente vi at denne metoden var optimal for å få enhetene til å kommunisere med hverandre. Det som trakk metoden ned var at vi måtte betale for å bruke alle funksjonene optimalt. Vi fant ut senere at gratis versjonen varte bare en måned, og at man ikke får bruke det videre med mindre man oppgraderer versjonen.

På bakgrunn av dette gikk vi over til MQTT metoden. Denne metoden ga oss også problemer som vi ikke fikk løst på grunn av tidsklemmen. Vi fikk sendt data begge veier, men vi kom ikke på en måte å lagre dataen på. Vi klarte heller ikke å definere inngangen som vi skriver inn som distanse til å implementere i Python skripten. Et annet problem at vi ikke fikk kommunikasjon fra brukergrensesnittet til MQTT.fx programmet.

Til slutt gikk vi for den siste metoden, som er lagring av tekstfil til JSON-fil på DropBox med Node-RED. På denne metoden fikk vi en kommunikasjon mellom vårt brukergrensesnitt og Node-RED. Det er en veldig enkelt og ryddig måte å gjøre det på, men ikke optimalt slik vi ønsket.

For videreutvikling av kommunikasjonssystemet kan vi implementere at når vogna kommer til de forskjellige distansene, skal Raspberry Pi sende den virkelige distansen som vognen har kjørt til DropBox via Node-RED. Slik at brukergrensesnittet kan hente distansene fra DropBox med SQL, og sammenlikne den virkelige distansen den har kjørt med distansen vi tastet inn og med RangeFinder-distansen. Dette er for å se for avviket mellom dem og ha en oversikt over hvor presis enkelte deler fungerer.

8.3 Styring av vogn

8.3.1 Motorkomponent

Etter at vi fikk motoren, var vår første oppgave å få den til å kjøre fritt. Først koblet vi batteriene i serie for å øke spenningen til 24V. Spenningskabelen fig.7.12 ble avisolert på den ene siden for å få tilkoblingsmulighet for begge polene. Spenningskabelen hadde to utganger,

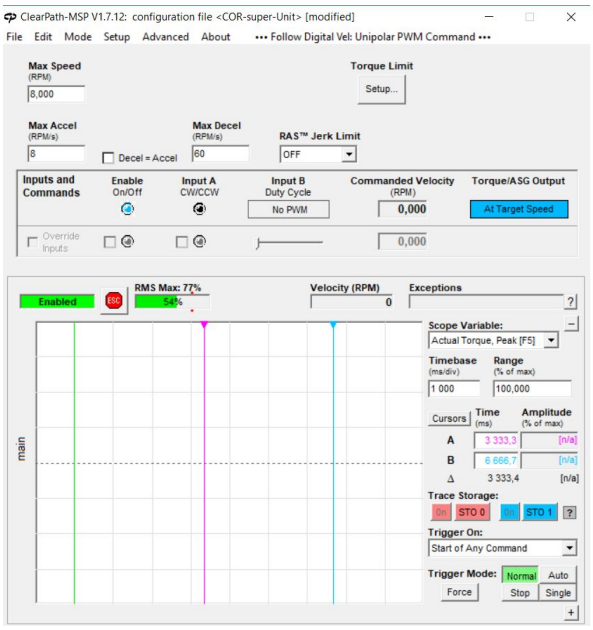
en positiv(rød) og en negativ(blå) ledning. Motoren vi brukte opererte mellom 24V - 72V spenning. For å øke spenningen fra batteriene til DC motoren ble det brukt en DC/DC omformer, som omformet spenningen fra 24V til 48V. Etter å ha koblet omformeren til motoren, koblet vi omformeren til batteriet. Slik det er vist i Vedlegg H.

8.3.2 Clearpath MSP programvare

Software programmet til DC motoren heter ClearPath MSP referert til [7] Nettside hvor man kan laste ned MSP. Helt på starten etter å ha koblet til spenningen blir programmet tilgjengelig for styring av hastighet og diverse funksjoner med en USB-kabel. Henvisninger til dette er referert i [8]

Hastighet og akselerasjon

Vi bruker MSP til å justere hastigheten og akselerasjonen slik det er vist i fig 8.28. Både hastigheten og akselerasjonen ligger på 8 RPM. Altså tar det et sekund for at motoren er i topp hastighet.



Figur 8.28: Unipolar

$$v = linrhastighet = m/s \tag{8.1}$$

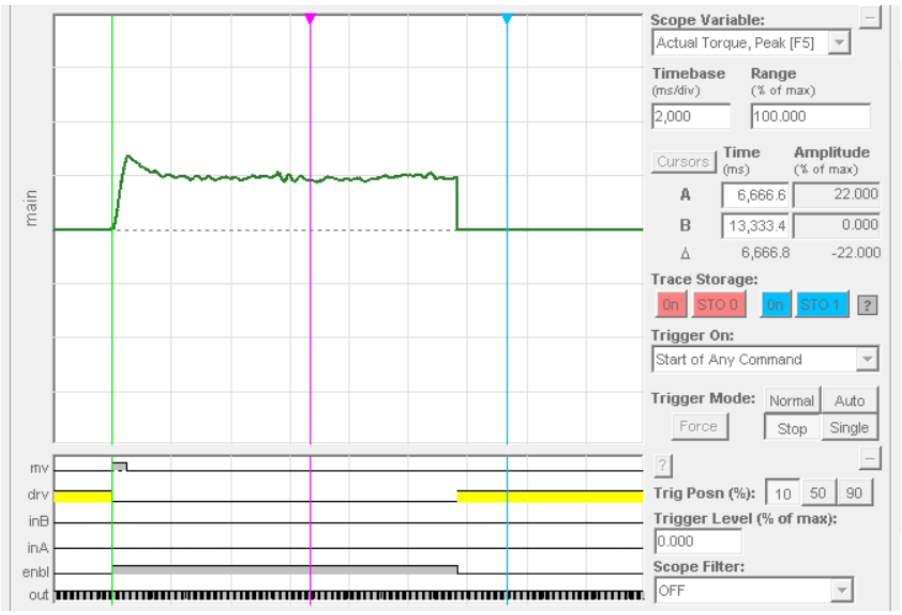
$$r = radiusphjulet = 35mm \tag{8.2}$$

$$\omega = vinkelhastighet = 8RPM \tag{8.3}$$

$$\omega = RPM * 0.10472 = 8 * 0.10472 \tag{8.4}$$

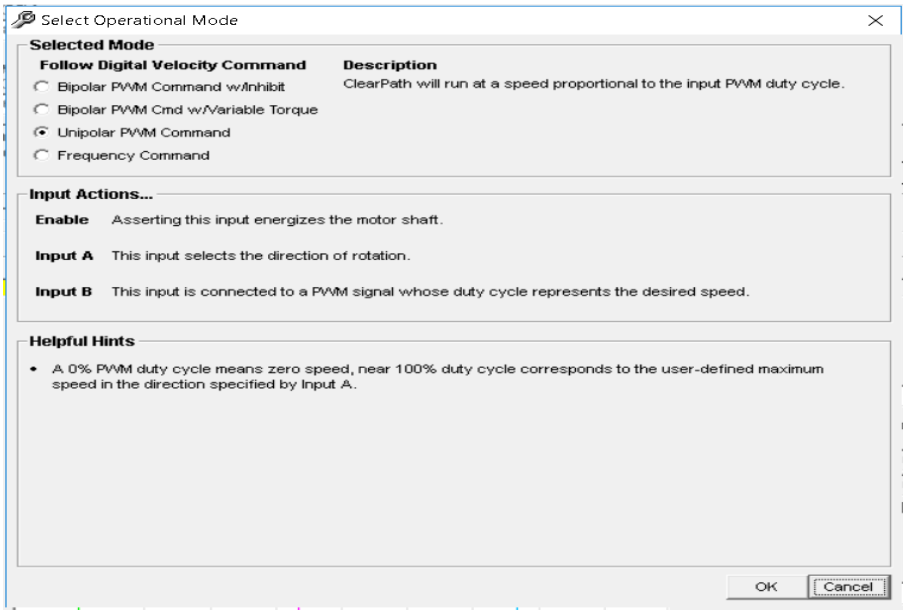
$$v = r * \omega = 35 * 8 * 0.10472 = 0.0293216m/s \tag{8.5}$$

I denne grafen fig.8.29 vi hvordan hastigheten forandrer seg gjennom perioden motoren kjører. Helt på starten av grafen ser vi den akselerer høyt og deretter gjevner seg ut etterhvert som den kommer i topp hastighet. Dette gjelder ikke når den skal bremse, da stopper motoren brått. Selv om motoren stopper brått skaper det ingen problemer fysisk.



Figur 8.29: Graf til fart

Unipolar PWM Command



Figur 8.30: Unipolar polar Mode

Det finnes forskjellige funksjoner («Mode» i MSP) for styring av motoren. Det vi har valgt å bruke er «Unipolar PWM Command» slik det er vist i fig.8.30 Denne funksjonen gir tilgang til noe som heter PWM (Pulse Width Modulation). Dette går ut på at én kan overstyre signalene som blir sendt via kablene med en prosentverdi i koden. Forandringen i prosenten overstyrer hastigheten til DC motoren.

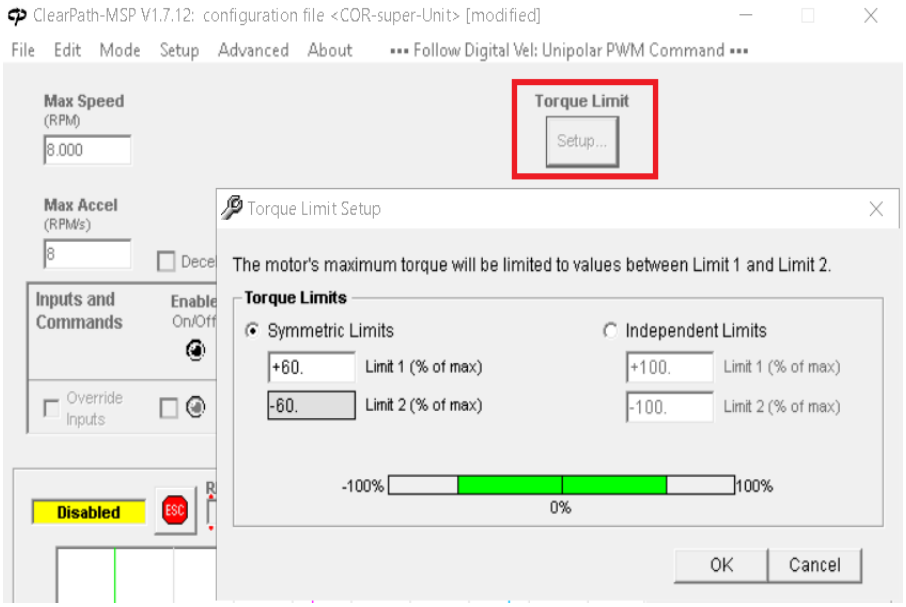
Slik det er vist i [34] og i Python koden fig.8.31. kan kodelinjen 23 «pwm.ChangeDutyCycle(. . .)» forandres med å taste inn prosentverdier mellom 0-100 i stedet for . . . . Dette vil forandre hastigheten prosentvis. Dette var nødvendig siden vi trengte en motorfunksjon for akselerasjon og en for bremsing. Problemet var at DC motoren stoppet brått. Dette var for å forbedre systemet, men ettersom det ikke ble et problem fysisk brukte vi ikke dette.

```
21  pwm = GPIO.PWM(InputB, 100)          # Access PWM signals
22  pwm.start(0)                         # Gives the start procentage
23  pwm.ChangeDutyCycle(100)             # Decide the procntage value for the velocity
```

Figur 8.31: PWM kode

Momentjustering

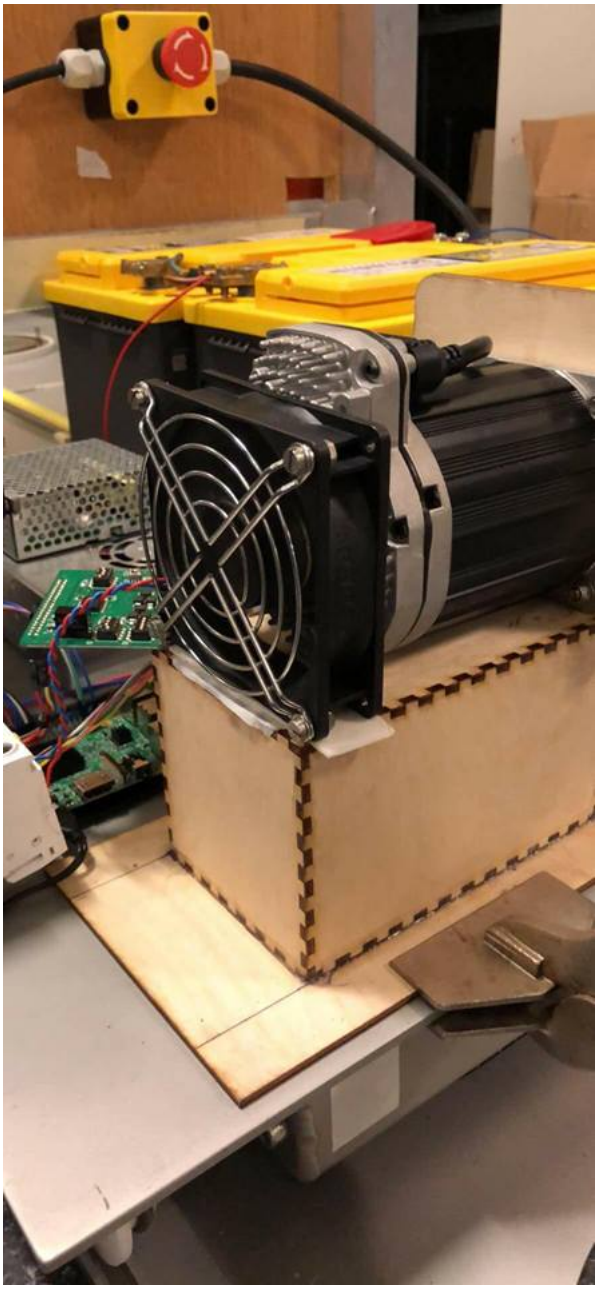
For å klare å justere motormomentet var det en funksjon som var tilgjengelig i programmet vist i fig.32 I denne funksjonen var det sånn vi kunne justere momentgrensen i prosentvise verdier. Dette var til hjelp med å løse et problem vi fikk under kjøring av vognen fysisk med alle delene opp i. Problemet var at vognen begynte å destabilisere seg ettersom momentet til DC motoren var for mye. Problemet ble løst med å prøve forskjellige prosentverdier. Til slutt fikk vi vognen til å kjøre stabilt i en gitt hastighet og momentverdi som stemte med beregningene.
Slik det er beregnet i Motor kapitlet 7.3 trenger vi 4Nm moment. Vi bruker 45%momentgrense. Som Fig.8.32 viser med at 100% for DC motoren er rundt 9Nm, blir 45% av dette tilnærmet 4Nm.



Figur 8.32: Mometprosent

Motorvifte

Vi oppdaget et problem med at DC motoren begynte å bli varm etter mange tester. Dette forårsaket at motoren ble ustabil og skrudde seg av etterhvert. Nødløsning for dette var å finne en motorvifte. Det var tilstede i Miros AS som vi tok i bruk slik vi ser det i fig.8.33



Figur 8.33: Motor med vifte

8.3.3 Motorstyring skript

Motorstyringskode

Først og fremst måtte vi avisolere kontrollkabelen for å få koblet ledningene til mikrokontrolleren og styre DC motoren med RPi. Kretsskjema for dette er vist i Vedlegg H. Tanken bak dette var å få kjørt DC motoren med Python kode. Med koden vist i fig 8.34. Hadde vi tre motorfunksjoner som var framover (fra reflektorveggen), bakover (mot reflektorveggen) og stopp. Disse er i bruk i hovedkoden vi bruker for motorstyring.

```
48  def forward() :
49      GPIO.output(Enable, GPIO.HIGH)    # ON/OFF, when its GPIO.HIGH its ON.
50      GPIO.output(InputA, GPIO.LOW)     # Pin for backwards direction, when GPIO.HIGH goes motor backwards.
51      GPIO.output(InputB, GPIO.HIGH)    # Pin for forwards, when GPIO.HIGH it goes forwards.
52
53  def backward() :
54      GPIO.output(Enable, GPIO.HIGH)
55      GPIO.output(InputA, GPIO.HIGH)
56      GPIO.output(InputB, GPIO.HIGH)
57
58  def stop() :
59      GPIO.output(Enable, GPIO.LOW)
60      GPIO.output(InputA, GPIO.LOW)
61      GPIO.output(InputB, GPIO.LOW)
62
```

Figur 8.34: Motorfunctions

Akselerasjon og bremsing kode

Vi har kommet fram til to metoder som gir DC motoren en akselerasjon og en bremsing. Beklageligvis har vi ikke fått disse til å fungere med hovedkoden. Dette er på grunn av tids- og kunnskapsmangel.

Første metoden

Denne metoden går ut på «while» løkker som blir brukt for å øke, holde og minke en verdi. Denne verdien representerer prosenten som velger hastighetsnivået. Dette gir oss en akselerasjon, en konstant hastighet og en bremse funksjon. I koden skjer dette etter sekunder slik det er vist i Vedlegg K, ettersom vi hadde hatt mer tid kunne vi forandre tids variabel om til en verdi som måleverdiene hadde bestemt.

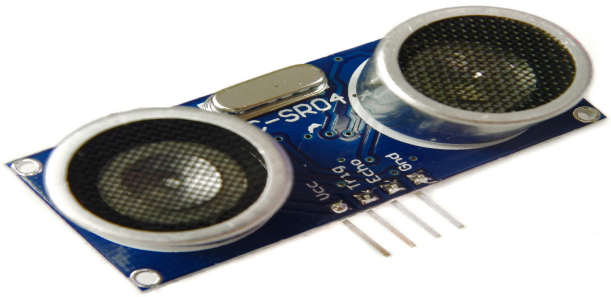
Andre metoden

Denne metoden er bare for bremsing, hastigheten forandres gradvis tre ganger innenfor «if» løkker med gitte avstandsverdier. Da avstandsverdien viser 200mm igjen går den i første løkken og halverer hastigheten til 50% helt til det er igjen 100mm. Fra 100mm til 0.2mm går hastigheten ned til 30% og da den kommer til 0.2mm stopper motoren helt, refererer til Vedlegg J.

8.3.4 Ultasonic sensor

Prosjektet inneholder en motor som beveger på en vogn, dette kan føre til skader på arbeidstedet. For å forhindre skader bør vogna inneholde en eller flere sikkerhets sensorer. Vi har valgt å bruke en enkel Ultrasonic sensor for å sikre at kjørebanen er fri for objekter. Spesifikt HC-SR04 som vises på fig 8.35. En ultra sonisk sensor er en enhet som kan måle avstand til et objekt ved hjelp av lydbølger. Målingen foregår ved å sende ut en lydbølge, og videre lytte etter samme lydbølge. Ved å telle tiden det tar når lydbølgen blir sendt til den treffer et objekt og reflekterer tilbake kan vi regne ut avstanden. Formelen for slik utregning vises under i formel 8.6 . Sensoren har en registreringsområdet som er fra tre centimeter og opp til fire meter i avstand.

$$Distanse = \frac{lydhastighet * telttid}{2} \tag{8.6}$$



Figur 8.35: Ultra sonic

Skript til Ultrasonic sensor

For å styre sensoren fra Raspberry Pi trengte vi å koble opp slik det blir vist på kretstegning på side XX. Videre trengte vi å lage et Python skript der vi importerte viktige biblioteker og definerte pinner, variabler og utregningsfunksjonen til sensoren. Hele Python skriptet er vist i Vedlegg N.
For utregningsfunksjonen til distansen trengte vi å definere en separat funksjon. Funksjonen heter «get_distance():» og blir vist på fig 8.36. Trigger pinnen blir aktivert i veldig kort tid av Raspberry Pi slik at sensoren skal sende ut en lydbølge. Deretter slår Raspberry Pi på Echo pinnen og teller tiden det tar til lydbølgen treffer objektet og returnerer tilbake. Videre blir reisetiden for signalet konvertert til distanse ved hjelp av formel 8.6.

```
# Distance function
def get_distance():
    # Turn Trig pin on for a short time
    GPIO.output(TRIG, True)
    time.sleep(0.0001)
    GPIO.output(TRIG, False)

    # Listen to Trig signal and derive time of travel
    while GPIO.input(ECHO) == False:
        start = time.time()

    while GPIO.input(ECHO) == True:
        end = time.time()

    sig_time = end-start

    # Convert signal time to cm and return value
    distance = sig_time / 0.000058
    return distance
```

Figur 8.36: Utregningsfunksjon for distanse.

Etter at sensoren har regnet ut distanseverdi kan vi fritt definere hva sensoren skal utføre under valgte distanser. I dette tilfellet har vi valgt at «Stop» variabelen skal være 20 centimeter. Det vil si at skriptet skal skrive ut «Go» tekst på kommandoterminalen. Det røde LED-lyset er skrudd av for alle distanseverdier over 20 centimeter. Hvis distanseverdien er under 20 centimeter skal det røde LED-lyset bli slått på og «STOP» tekst vil bli skrevet ut på kommandoterminalen. Skriptet for funksjoner vises på fig.8.37.

```
# Variable with random value to enter while loop
while (100 > Stop):
    # Call for get_distance function with new variable
    distance = get_distance()
    # Pause for one second
    time.sleep (1)
    # If distance in cm is desired uncomment line under
    #print('Distance: {0:.1f} cm'.format(distance))
    # If distance to object is more then Stop variable, print "Go" and set red LED to low
    if (distance > Stop):
        print('Go')
        RED_LOW()

    # If distance to object is less then Stop variable, print "Stop" and set red LED to high
    elif (distance < Stop):
            print('STOP')
            RED_HIGH()
```

Figur 8.37: Funksjoner for sensoren.

8.4 Posisjonering

Ett av kravene vi har mottatt fra Miros As omhandler en bevegelig del som skal posisjoneres til bestemte punkter sendt fra UI. Den bevegelige delen er en kalibreringsvogn som vi har modernisert ved å implementere vårt konsept og vår visjon for kontroll av posisjonering og motorstryring. Konseptet vårt vil oppfylle kravet fra Miros As. Posisjoneringsdelen inneholder flere elektroniske komponenter og sensorer som samarbeider for en stabil og nøyaktig distansemåling av kalibreringsvogn. For å generere og kalkulere nøyaktig posisjon til kalibreringsvogna benyttes sensordata i form av signaler fra encoder og proximitysensor. Alle verdier på kommende bilder eller kodebeskrivelser er gitt i millimeter for avstand og sekunder for tid.

8.4.1 Målemetoder

Generelle verdier.

Kalibreringsvogna skal ved hjelp av referanseverdier beveges til nøyaktig posisjon, som er gitt som avstand fra RF til reflektorvegg. Ved nullpunktet, hvor to metalrammer treffer hverandre, er 857mm gitt som avstand fra RF til reflektorvegg. Dette er verdi som reprensenterer WallToRF verdi i fig 8.38. EncoderEnd er verdien hver gang proximitypuls aktiveres. Encoderpuls tilbakestilles da til 540 puls for encoder distanse avstand.

```
#  General value
End = 0.2                  # The distand to stop or break the engine
tid = 8                   # how many secound it take to measuring SM-140 range
Home = 867                 # Home posisjon from SM-140 to reflect wall
WallToRF = 857             # Distance between SM-140 too reflect wall in mm
EncoderEnd = 540           # Encoder puls when proxi reset
```

Figur 8.38: Generelle verdi.

Distansemåling for encoder.

Encoderen som har blitt brukt til kalibreringsvogna kommer fra Kubler og genererer 2048 pulsslag per rotasjonsrunde. Encoderen har vibrasjonsresistans, og har operasjonstemperatur på -20 grader til 85 grader. Ved å feste encoder til kalibreringsvognas hjul, som har en diameter på 69mm, får vi 2048 pulsslag for hver hjulrotasjon.

$$Omkrets = 69mm * \pi = 216.7698931mm \tag{8.7}$$

$$PPr = \frac{omkrets}{antallpuls} = \frac{216.769831}{2048} = 0.1058446744mm \tag{8.8}$$

Ut fra ligningene over får vi en pulsoppløsning på 0.1058446744 mm for hvert pulsslag i encoderen, dette blir definert som lengde i pythonkoden. En verdi på 0.10584mm per puls betyr at vi vil få svært nøyaktige målinger hvis teorien stemmer i praksis.

```
clkState = GPIO.input(InA)                          # Define input InA from Encoder for forward rotation
dtState = GPIO.input(InB)                           # Define input InB from Encoder for backward rotation

# Code taken from Github
# https://github.com/modmypi/Rotary-Encoder/blob/master/rotary_encoder.py
#------------------------------------------------------------------------------------------------------------#

if clkState != clkLastState:                        # If encoder value is not same as previous encoder value ru
        if dtState != clkState:
            counter +=1                             # By forward rotation encoder value pluss one counter

        else:
            counter -=1                             #  By backward rotation encoder value minus one counter
        #print (counter)
        avstand1 = counter * lengde                 # Distance measuring value from encoder counting multiply b
        avstand = avstand1 + avstand2 + WallToRF    # Distance measuring value combi from proximity and encoder

        print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen1)   )
        clkLastState = clkState

#------------------------------------------------------------------------------------------------------------#
```

Figur 8.39: Kode linje til encoder.

Encoderen har to signalinnganger; InA og InB. - InA er clkState, som er pulstelling for rotasjon fremover. - InB er dtState, som er pulstelling for rotasjon bakover. Ut fra koden fig 8.39 blir forrige encoderpuls sammenlignet med eksisterende pulssignal. Hvis de er like betyr det at hjulene ikke er i bevegelse.

Ved bevegelse av kalibreringsvogna vil encoderpuls endre seg avhengig av rotasjonsretning og hastighet. Siden encoderen har en pulsoppdateringshastighet på 160kHz vil ikke hastigheten til kalibreringsvogna har alt for stor innvirkning.

For bakoverrotasjon vil encoder InB signal som er dtState aktiveres og trekker fra seg eksisterende encoder pulser. Ved foroverrotasjon vil encoder InA signal aktiiveres og legger til eksisterende pulser. Ved encoder puls generering multipliserer vi med puls oppløsning for finne ut hvor langt kalibreringvogn har reist ved hjelp av antall runde kalibreringvogn hjulene har rotert. Dette er kun for å finne avstand som har blitt generet av encoder puls telling. Avstand som encoder generer blir definert som avstand1 i koden fig 8.40.

Distansemåling for proximity.

Bruk av encoder alene vil i noen tilfeller være tilstrekkelig ved korte avstandsmålinger, men ved lengre distanser vil kvaliteten på encoderens pulstellinger påvirkes. Det vil si at encoderen kan hoppe eller miste pulser underveis ved lengre distanser. Ved å tilbakestille encoderens pulstellinger hver halvmeter kan vi eliminere dette problemet.
Proximitysensoren som blir implementert i kalibreringsvogna er produsert av Carlo Gavazzi, og har en måleavstand på 12mm. Ved hver halvmeter fra nullpunktet, som er WallToRF verdi, plasserer vi en aluminiumbit. Denne biten har en dimensjon på 30mm i lengde, 15mm i bredde og 15mm i høyde. Det er viktig at alle aluminiumbiter er lik i dimensjon siden ellers kan den påvirke målekvaliteten til kalibreringsvogna.
Hver gang proximitysensor passerer denne aluminiumbiten vil det legges til eller trekkes fra 500mm i kalibreringsvognas målingsavstand avhengig om det kjøres fremover eller bakover.

```
# For Proximity
if GPIO.input(proxi)==1 :
    proxypa = 1

else:
    proxyav = 1
    if (proxyav + proxypa  == 2):                    # Proxi counter one by ever
        antallProxi += 1                             # Forward proxi pluss one p
        proxyav = 0                                  # Proxi off value to 0
        proxypa = 0                                  # Proxi off value to 0
        counter = EncoderEnd
                                        # Reset encoder counter to 0
        avstand2 = antallProxi * proxiValue                  # Distance measuring
        #print (" Avstand 2 i mm: ",avstand2)
```

Figur 8.40: Kode linje til proximity.

Det er definert to proximityverdier, en kalt proxyav og en kalt proxipa som vist i kodelinjen. Når proximitysensor går fra lavt inngangssignal til høyt inngangssignal blir proxypaverdi satt til en. Når proximitys inngangssignal går fra høyt til lavt, vil proxyavverdi settes til en, og det sjekkes om proxiav og proxipas verdier i sum er satt til to. Hvis de er det vil det legges til en puls i antallProxi. For å unngå at proximitysensoren utfører feiltellinger, har vi definert at proximitys pulstelling øker med en ved promitypassering. For å legge til en proximitypuls må proximity gå fra lav tilstand til høy tilstand og tilbake til lav tilstand igjen. Dette for å unngå at proximity legger til flere promitypulser ved proximitypassering eller når proximitysensor stopper rett over aluminiumbit.
AntallProxiverdi sier oss hvor mange ganger proximity sensor har beveget over aluminiumbiter. For å finne ut distanse målt fra proximitys verdi så tar vi antallproxis verdi og multipliserer den med proxiValue som er 500mm. For hver proximitysensorpassering blir encoderens pulstelling satt til samme verdi som EncoderEnd verdi. EncoderEnd verdi blir satt til 540 pulsslag, dette tilsvarer 57.15mm. Grunnen til at EncoderEnds verdi er så høy er at proximitysensor har en diameter på 16.7mm. Ved objektdeteksjon må det dekkes over 80% av proximitysensorens overflate for å få en stabil deteksjon. Deteksjon av 80% overflate tilsvarer 13.36 mm for stabilt høyt signal og 13.36 mm for stabilt lavt signal. Ved å legge til lengden til aluminiumbit på 30 mm vil vi få en total lengde på 56.72 mm. Dette tilsvarer en encoderpuls på 535.8 pulsslag, de resterende fire av EncoderEnd pulsslag kommer av oppdateringshastigheten til proximitysensor.

Kombinert distansemålinger fra encoder og proximity.

Tidligere ble det beskrevet hvordan distansemålingene fra encoder og proximity kombineres og hvordan proximity tilbakestiller encoderpuls ved hver proximitypassering. Total avstand fra encoderen blir definert som avstand1 i koden, mens total avstand fra proximity blir definert som avstand2. Ved å legge til total proximityavstand, total encoderavstand og avstand fra vegg til RF vil vi få en presis sanntidsdistanse for kalibreringsvogna.

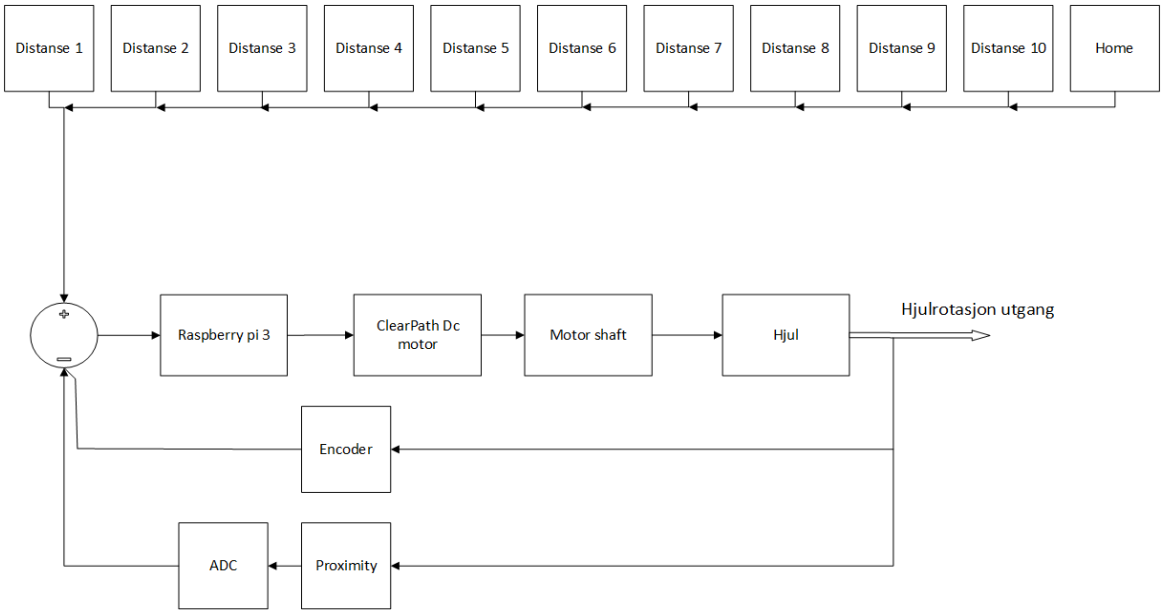
```
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
```

Figur 8.41: Kode linje til kombimert måling.

8.4.2 Posisjonering og distanserpunkter.

Posisjonering av kalibreringsvogn.

Prosessen med å posisjonere kalibreringsvogna til eksakt ønsket verdi foregår slik; operatør definerer punkter via UI, disse blir videre sendt til Raspberry Pi og derfra blir de implementert inn i Pythonkoden for ti-distansers målerunde.



Figur 8.42: Lukket sløyfe.

Som tidligere nevnt i målemetoder for distanseoppmåling får vi sanntidsavstandsverdi til kalibreringsvogna ved å implementere sanntidsverdier fra sensorer, disse skal sammenkobles med kommende distanseverdier som kalibreringsvogna skal til. For å finne ut eksakte distanseverdier kalibreringsvogna har igjen til neste distanseverdi, bruker vi en nedtellingsfunksjon.

Forover rotasjon.

```
            AvstandIgjen1 = Dis1 - avstand
            if avstand < Dis1 :
                forward()
            if  AvstandIgjen1 <= End :
                stop()
                print ("Runde 1 ferdig")
                break
sleep(tid)
```

Figur 8.43: Rotasjon forover.

I fig.8.43 vises kodelinjer for rotasjon fremover. Ved rotasjon fremover vil den nåværende distanseverdien til kalibreringsvogna være mindre enn distanseverdien som kalibreringsvogna skal til. - AvstandIgjen1: Avstanden som er igjen til neste distanseverdi. Ved bevegelse av kalibreringsvogna vil avstandigjen1-verdi minske. Når Avstandigjen1 nærmer seg nullverdi vil det aktiveres en bremsefunksjon, derfra ventes det i et visst antall sekunder før man tar RF måling og sender dette til kalibreringsrapporten og starter neste målerunde.

Bakover rotasjon.

```
            AvstandIgjen2 = avstand - Dis2
            if avstand > Dis2 :
                backward()
            if  AvstandIgjen2 <= End :
                stop()
                print ("Runde 2 ferdig" + str( avstand))
                break
sleep(tid)
```

Figur 8.44: Rotasjon bakover.

I fig.8.44 vises kodelinjer for rotasjon bakover. Ved bakover rotasjon vil distanseverdien kalibreringsvogna skal til være mindre enn den nåværende distanseverdien. - AvstandIgjen2: Avstandsvariabel som tar nåværende avstand og subtraherer den med distanseverdien kalibreringsvogna skal til. Ved bevegelse av kalibreringsvogna vil avstandigjen2-verdi minske. Når Avstandigjen2 nærmer seg nullverdi vil det aktiveres en bremsefunksjon, derfra ventes det i et visst antall sekunder før man tar RF måling og sender dette til kalibreringsrapporten og

starter neste målerunde. Ved rotasjon bakover vil proximitypulser trekkes fra hver gang aluminiumbit passeres. Encoderpulscounter tilbakestilles da til – EncoderEnd.

Ved ti distanser punkter kjøring.

```
# For distance 3 forward
while avstand < Dis3 :···
sleep(tid)

# For distance 3 backward
while avstand > Dis3 :···
sleep(tid)


# For distance 4 forward
while avstand < Dis4 :···
sleep(tid)

# For distance 4 backward
while avstand > Dis4 :···
sleep(tid)


# For distance 5 forward
while avstand < Dis5 :···
sleep(tid)

# For distance 5 backward
while avstand > Dis5 :···
sleep(tid)

# For distance 6 forward
while avstand < Dis6 :···
sleep(tid)

# For distance 6 backward
while avstand > Dis6 :···
sleep(tid)


# For distance 7 forward
while avstand < Dis7 :···
sleep(tid)

# For distance 7 backward
while avstand > Dis7 :···
sleep(tid)


# For distance 8 forward
while avstand < Dis8 :···
sleep(tid)
```

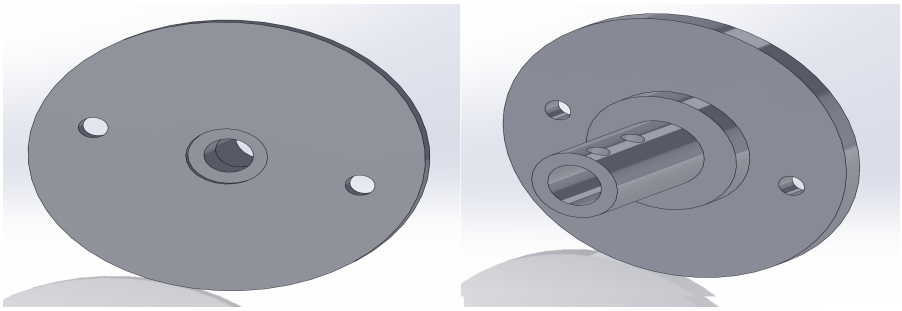
Figur 8.45: Eksempeler på forover og bakover posisjonering.

For å fullføre ti kjørerunder vil motorrotasjonsretningen avhenge av avstand fra nåværende posisjon og neste målepunkt. Hvis avstanden til neste posisjon er større enn nåværende posisjon vil motorens utgangsaksel rotere med klokka, dvs rotasjon fremover. Hvis avstanden til neste posisjon er mindre enn nåværende posisjon vil motorens utgangsaksel rotere mot klokka, dvs rotasjon bakover. Ved siste kjørerunde, som er runde ti, vil kalibreringsvogna automatisk kjøre hjem til start posisjon, som er Home-verdi.

8.5 Adapter deler

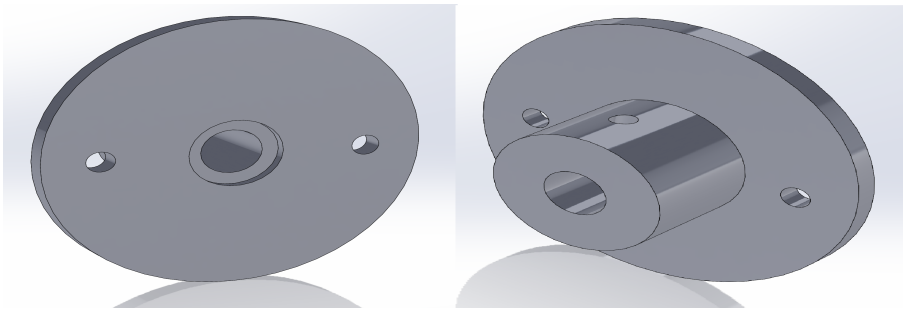
8.5.1 Motorovergang

Denne modellen hjelper oss med å få feste motorakselen til vognbeltet. Dette fører til at vognen beveger seg etter motorakselbevegelsen. Den skal virke på måten som en motoraksel adapter og driver vognbeltet stabilt. Det ble skissert to forskjellige modeller siden den første versjonen ikke fungerte optimalt som nevnt tidligere. Det ble utført to fysiske forsøk med den første modellen, hvor begge forsøkene ga negativt funksjonalitet. Første modellen er henvist i fig.8.46 og siste modellen er henvist i fig.8.47



Figur 8.46: Motor overgang versjon 1.

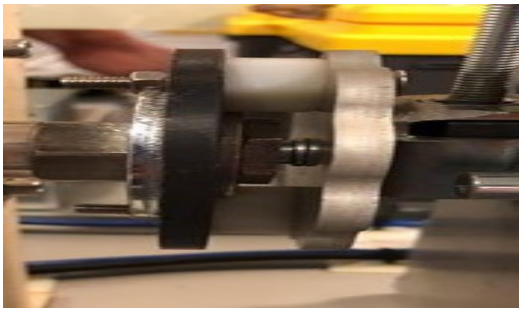
Problemet med første modellen var at materialet(plastikken) som drev beltet til vognen, ikke hadde bred nok vegg for å klare seg. Noe som medførte til at den knakk og ikke kunne brukes lenger. Med det forsøkte vi å 3D printe den siste modellen, denne var bredere, mer stabil og hadde bare et skruehull til motorakselen. Denne modellen skulle fungere mer optimal, men kanskje ikke fullstendig siden materialet ikke er hardt nok. Vi hadde ikke muligheten for å 3D printe siste modellen og prøve den fysisk, dette var på grunn av at både 3D printerne på skolen var opptatt og at tidsfristen for implementering i Miros AS ikke var lang nok.



Figur 8.47: Motor overgang versjon 2.

Det vi trengte var en nødløsning som skulle løse problemet for kobling mellom DC motoren og vognen. Nødløsningen gikk ut på at vi brukte en akselfeste skrue og boret et hull på overflaten av modellen gjennom hele, dermed skrudde vi fast akselfeste skruen i modellen og til slutt

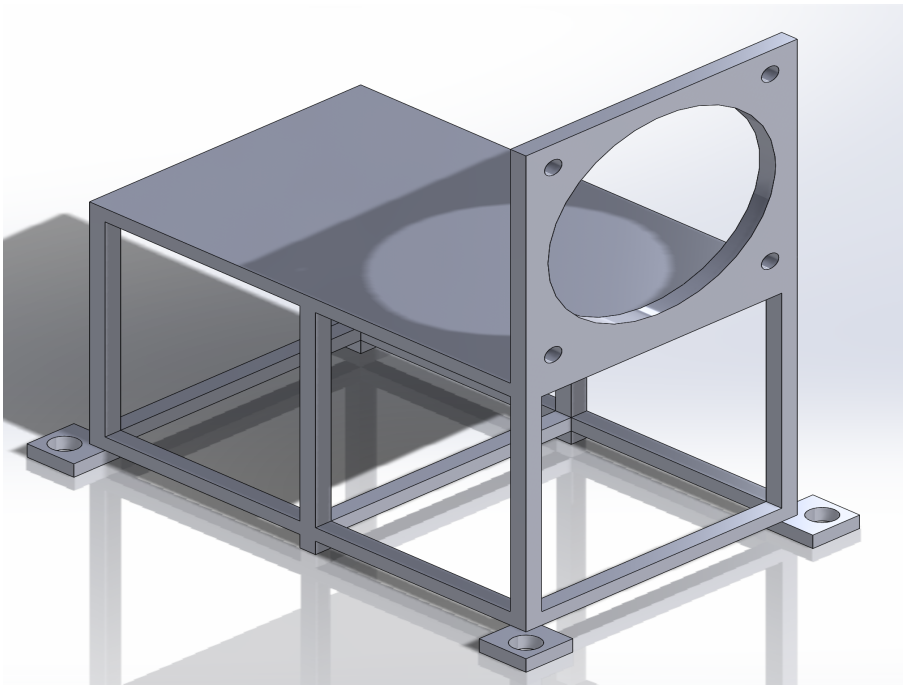
motorakselen. Dette ble hardt nok for at vognen bevegde seg sammen med motorakselen og fungerte optimalt etter dette slik vi ser det i fig.8.48



Figur 8.48: Nødløsning for motor overgang.

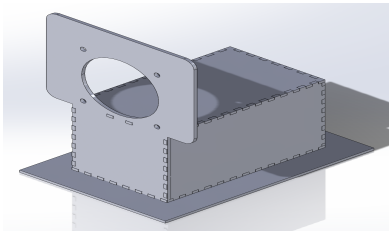
Motor ramme.

I denne delen skisserte vi to forskjellige 3D modeller, dette for å få feste DC motoren på vognen. Bare en av dem er i bruk nå. Modellen holder DC motoren ca. 14cm over overflaten av vognen og hjelper med å få motorakselen i horisontal linje med vognbeltet. Slik vi ser det i fig.8.49



Figur 8.49: Motor ramme versjon 1.

Første modellen som er henvist i fig.8.49 var ikke mulig å få printe ut i 3D siden den var alt for stor og ustabil for en slik printer. Med dette fikk vi råd om å laserkutte treverk og sette sammen delene med super- og trelim. Det er siste modellen henvist i fig.8.50 som er i bruk nå.



Figur 8.50: Motor ramme versjon 2.

8.5.2 Encoder til hjul

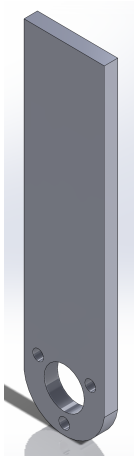
Denne modellen hjelper oss med å få feste Inkrementell encoderen på det ene hjulet bak. Slik det er henvist i fig 8.51 er modellen avhengig av Encoder-til-vogn modellen for å oppnå

optimal funksjonalitet. Det vi brukte var en dobbeltsidig teip for å få feste denne modellen i hjulet. Først tenkte vi å bore to hull for å skru den fast. Dette kunne skape problemer med at vi måtte være forsiktige og ikke skade hjulet, derfor valgte vi å gå for dobbeltsidig teip som fungerte optimalt og kan erstattes veldig enkelt seinere.

Figur 8.51: Encoder til hjul.

8.5.3 Encoder til vogn

Denne modellen skal holde Inkrementell encoder fast på plass og skal bevege seg sammen med vognen stabilt på veibanen som er henvist i fig.8.52 Det samme gjelder denne modellen om at den er avhengig av Encoder-til-hjul modellen for å fungere optimalt.



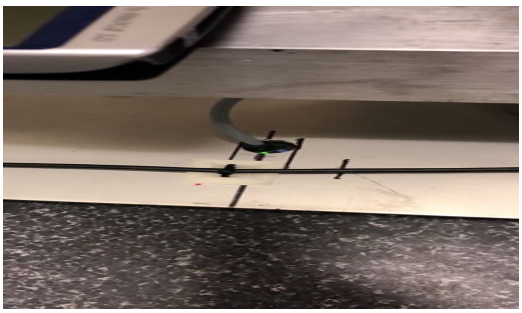
Figur 8.52: Encoder til vogn.

Vi 3D printet denne modellen litt lenger med tanke på lengden fra underflaten av vognen til midten av hjulet hvor Inkrementell encoderen skal være plassert. Etter å ha målt nøyaktig hvor lang den avstanden er, saget vi bort den delen som var for lang og festet den slik det er vist i fig.8.55

Figur 8.53: Encoder til vogn.

8.5.4 Proximity til vogn

For denne delen skisserte vi ikke en 3D modell i SolidWorks, i stedet tenkte vi å bore et hull i vognen hvor Proximity/Capacitive sensoren skal ligge slik vi ser det i fig.8.54, dette ble fysisk ustabilt og derfor måtte vi gjøre en endring på den.



Figur 8.54: Encoder til vogn.

Endringen gikk ut på å finne en L-formet metallskive. Denne festet vi med dobbeltsidig teip på den ene siden av metallskiven i linje med innsiden av vognen, og på den andre siden av metallskiven var det et hull som sensoren var skrudd fast i med to muttere slik det er vist i fig.8.55

Figur 8.55: Encoder til vogn.

9 TEKNISK UTFØRELSE DATA

Dokumenthistorie			
Versjon	**Dato**	**Endret av**	**Aktivitet**
2.1	15/04/18	VA	Dokument opprettet
2.2	17/04/18	VA	Første versjon av GUI
2.3	20/04/18	VA	Kalkuleringskode
2.4	26/04/18	VS	WEB-interface(UI flyttes til sky)
2.5	04/05/18	VS	Endelig UI (design)
2.6	07/05/18	VA	Rapportgenerering
2.7	10/05/18	VS	SQL Database
2.8	11/05/18	VA	MQTT
2.9	14/05/18	VS	Endelig UI (funksjoner)
2.10	21/05/18	VS	Revidert for publisering
3.0	21/05/18	VA	Publisert i rapport v3.0

Tabell 9.1: Dokumenthistorie for teknisk utførelse data

9.1 Brukergrensesnitt

9.1.1 Intro brukergrensesnitt

Når et system skal automatiseres og optimaliseres for operatører samt kunder, så er det viktig med et grundig gjennomført arbeid av brukergrensesnitt både i forhold til design og funksjonalitet. Miros har fra før av et grensesnitt som benyttes til overvåkning av distansemålinger og lagring i tillegg til utskrift av rapport.

For å automatisere systemets kjørerunde må det legges til funksjonalitet som igangsetting av ny kjørerunde, avbrytning av kjørerunde, omstart av kjørerunde om ønsket ved unøyaktig målerunde, i tillegg til spesifisering av målepunktene systemet skal kjøre til for å gjøre avstandsmålinger.

Under følger en tidlig versjon av grensesnittet som ble utarbeidet som en skisse før andre presentasjon.

Figur 9.1: Skissering av brukergrensesnitt

9.1.2 Miros AS grensesnitt

Miros sitt nåværende system inneholder en applikasjon med ulike oversikter over nåværende kjørerunde i sanntid og en detaljert rapportoversikt med en rekke felter med informasjon. Applikasjonen inneholder flere grafer som viser ulike synsvinkler og moduser for å få en mest mulig fullstendig oversikt over hver enkelte kjørerunde. Den har i tillegg ett vindu som viser detaljerte grafverdier de siste 24 timene. Vi vil nå gi en grov oversikt over de enkelte vinduene og deres funksjonalitet.

Summary

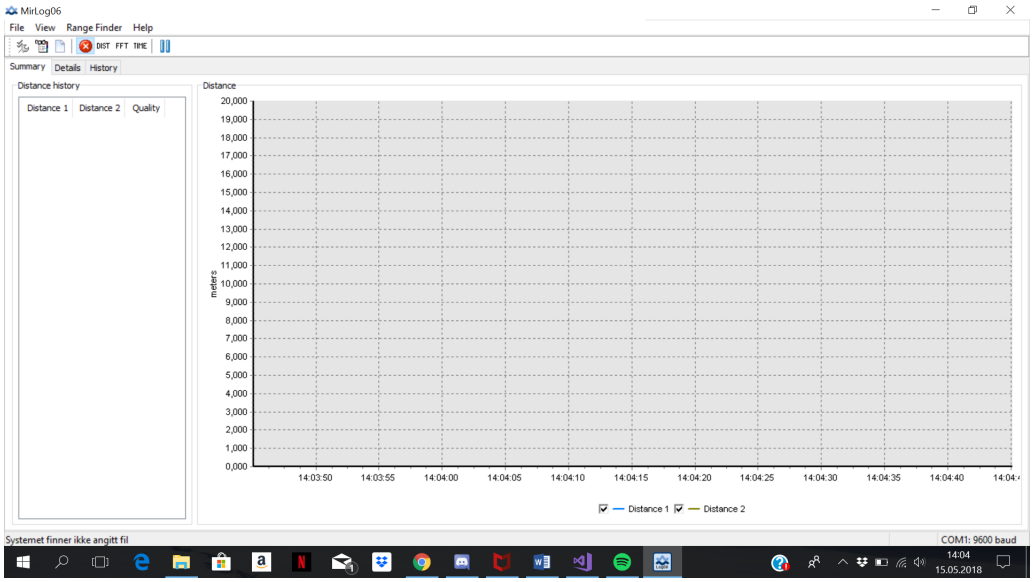
Dette vinduet er standardvinduet som vises i det operatøren starter applikasjonen. Her vises en oversikt med distansemålinger og avvik mellom disse målingene, og en graf med målt distanse i nåtid. Operatøren har her en mulighet til å vise normalens verdi, sensorens verdi eller sammenligne de ved å vise begge verdier samtidig i grafen.

Details

Her gis det et mer detaljert innblikk i nåværende kjørerunde. Til venstre vises en rekke tekniske data, disse gir den samme informasjonen som vises i den genererte rapporten. Tre forskjellige grafer viser mer detaljerte visuelle innblikk i kjørerunden. Modusene som vises her er Time series, spectrum i tillegg til vanlig distansegraf.

History

History-vinduets oppgave er å gi brukeren enn detaljert oversikt over kjørerunder det siste døgnet. Her vises de samme grafene som er gitt i details, men de vises innenfor et tidsrom som tilsier det siste døgnet.



Figur 9.2: Miros brukergrensesnitt

9.1.3 Web grensesnitt

Under en periode i prosjektet ble det utviklet et brukergrensesnitt som etter planen skulle fungere som et sky-basert grensesnitt lastet opp som en nettapplikasjon til Microsoft Azure. Denne utgaven ble mot sluttfasen av prosjektet forkastet og erstattet med en lokal applikasjon, siden det viste seg å bli litt for krevende og kommunisere fra et grensesnitt i en nettsky med tiden vi hadde til rådighet.

Men selv om det ikke ble noe endelig resultat av denne prosessen, så var den nyttig i utvikling av et tenkt design, spesielt i forhold til bestemmelse av målepunkter, som fikk tilegnet en egen side med et design som ble ganske likt sluttresultatet. Til å utvikle siden benyttet vi Web Forms App via Microsoft Visual Studio, dette siden programmet har innebygd funksjonalitet for opplasting til Microsoft Azure

Design

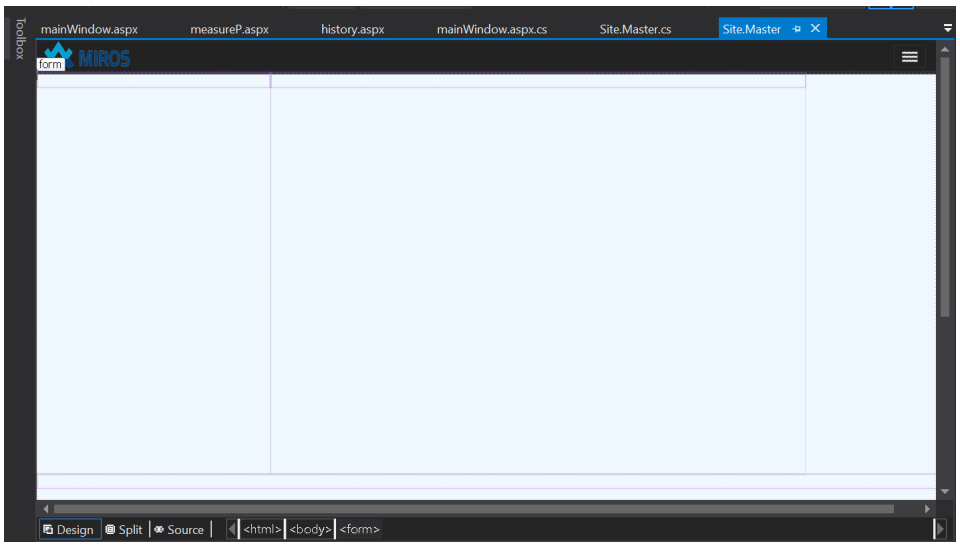
Det finnes både fordeler og ulemper ved å designe ett web basert grensesnitt med Web Forms. For å utnytte fordelene best mulig samtidig som å unngå og støte på ulempene, så er erfaring

svært nyttig. En bratt læringskurve med en god del omgjøring underveis var nødvendig for oss da ingen medlemmer hadde erfaring med programmet.

Å plassere de ulike elementene på en fornuftig og god måte er en utfordring med Web Forms, spesielt etter hvert som sidene vokser og flere elementer legges til. Mye design foregår på en «drag and drop» metode, som sparer en del innskriving av koder, men som ofte kan lure en til å tro at designet er mindre utfordrende enn det faktisk er. Bruk av paneler til å dele siden inn i ulike seksjoner er fordelaktig for å unngå uforutsette problemer når nye elementer legges til. Planlegging av disse ulike seksjonene før implementering vil derfor bli veldig tidsbesparende når man jobber med dette programmet.

En fordel i Web Forms er at prosjektene har en innebygget skisseringside kalt «Site.Master», hvor man kan legge til paneler som automatisk arves og legges til i de ulike sidene. Elementer som menyer og andre ting som ønskes å vises i samme seksjon av hver side, kan også legges til her for å spare mye tid. Man slipper da å kopiere denne koden til samtlige sider.

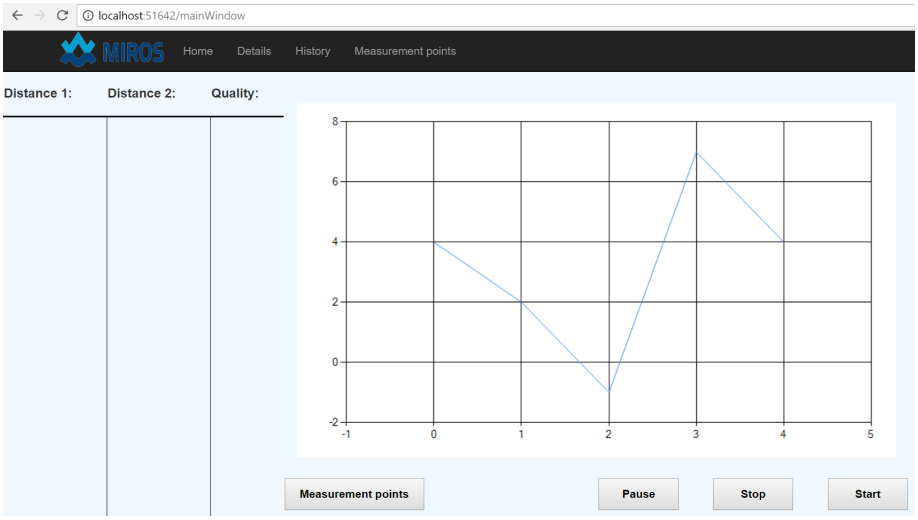
I vår «Site.Master» er det lagt til en navigasjonsmeny øverst, og deretter tre paneler under menyen. Et panel er plassert til venstre for utvikling av tabeller og lignende, mens det til høyre for dette ligger et slags hovedpanel hvor grafer og større menyer som endring av målepunkter plasseres. Et panel er også plassert på langs nederst på siden, et område som hovedsakelig er planlagt å inneholde knapper og andre valgmuligheter.



Figur 9.3: MasterPage

Når planen ble endret vekk fra nettbasert grensesnitt så var antallet sider det samme som det endelige grensesnittet endte med. En hovedside som inneholder knapper for å starte, stoppe og midlertidig pause kjøring, samt en knapp som sender brukeren til siden hvor man endrer målepunkter. I tillegg til knapper så viser hovedsiden en skissert graf og en tabell ment for sammenligning av målinger.

Som Miros sitt grensesnitt har nettapplikasjonen en side kaldt details som skal vise rapportinfo i venstre panel og tre grafer med ulike moduser i hovedvinduet. Historysiden er også lik, det vil si at de samme grafene som i details er skissert, men disse skal planlagt vise



Figur 9.4: WebUI

informasjonen opptil et døgn tilbake i tid.

En ny side som også ble brukt i det endelige grensesnittet ble utviklet. Siden «Measurement points» er en side som inneholder ti tekstbokser hvor brukeren taster inn ønskede målepunkter. En knapp er laget under for lagring av punktene, og nåværende målepunkter skal bli vist i en tabell i venstre panel.

9.1.4 COR grensesnitt

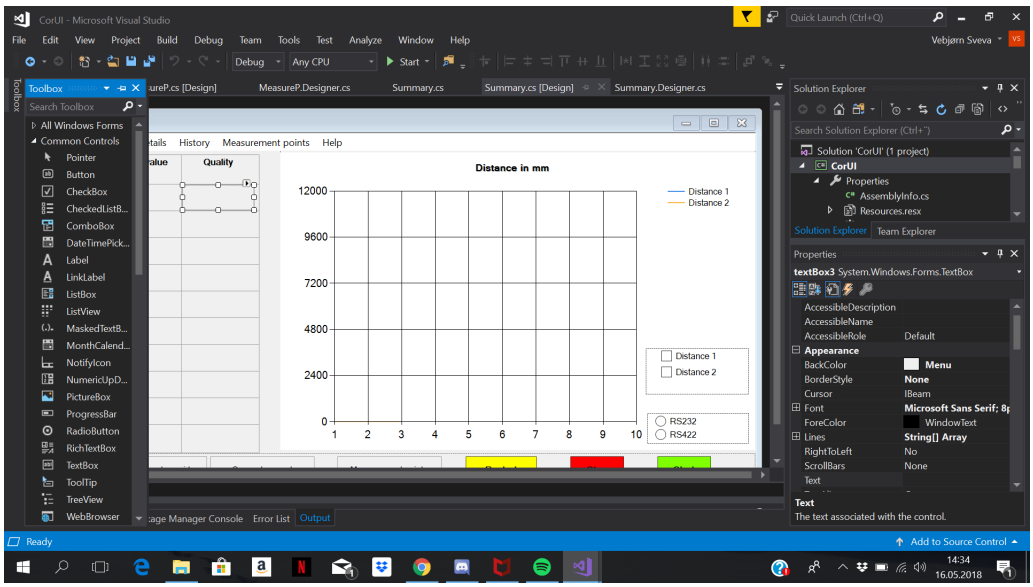
Med hensyn til kommunikasjon mellom grensesnitt og Raspberry Pi, så falt det en endelig beslutning på å utvikle en lokal applikasjon som må installeres på maskinen. Grensesnittet er utviklet med Visual Studio som plattform. Programmet benyttet heter Windows Forms, og er også koblet opp mot en SQL database for lagring av målepunkter, noe vi kommer tilbake til senere.

Windows Forms

Windows Forms er et sett med håndterte kodebiblioteker spesielt designet for utvikling av innholdsrike applikasjoner. Programmet inneholder et grafisk grensesnitt slik at utvikleren hele tiden har oversikt over sidenes oppsett og design. I en toolbox på venstre side ligger det en rekke kontroller som hver er en konkret instans av en klasse i kodebibliotekene. På høyre side vises klasser og benyttede biblioteker i Solution Explorer. Lenger ned på venstresiden ligger panelet Properties, som inneholder en rekke valgmuligheter for hver kontroll, alt fra bakgrunnsfargen på selve siden til om grafen din skal vises som lineær eller som et stolpediagram. Kontrollen man styrer velger man enkelt og greit ved å klikke på den i det grafiske grensesnittet [52].

Design

I Windows Forms er det en fordel å ha mest mulig erfaring med programmet når man skal designe utseendet. Det finnes paneler og måter å gruppere kontroller på, dette kan være utfordrende uten erfaring. Panelene må plasseres i forhold til hverandre ved hjelp av Anchoring og Docking, og kontrollene må plasseres på samme måte i henhold til hvor i



Figur 9.5: WinForms

panelene de skal plasseres. Har man lite erfaring vil det i første omgang føles enkelt å plassere kontroller og tabeller, men problemer vil oppstå med en gang man skal skalere vinduet i testprogrammet til fullskjerm, eller når man får en rekke kontroller på begrensede områder.

Ved hjelp av anchoring kan du feste kontroller eller paneler til bestemte avstander i forhold til kanten av enten ett panel eller selve siden. Når vinduet endres størrelse på vil da kontrollen beholde den avstanden til for eksempel høyre side av panelet eller siden dersom det er ønskelig.

Docking fester kontroller til kanter av siden eller panelet. Det vil si at dersom du vil ha en menylinje til å alltid være plassert øverst på siden og dekke toppen helt fra venstre til høyre hjørne, så velger du «Top» på docking. I tillegg til hjørner kan du også velge «fill» når du docker. Elementet vil da dekke hele siden eller panelet. Dette brukes ofte innenfor paneler til for eksempel grafer som skal dekke store områder av en side uavhengig av om vinduet er maksimert eller ikke.

Designet av siden besto av få krevende beslutninger, siden designet har vært utarbeidet i andre prosjekt tidligere. Men kontrollene fungerer på en litt annen måte i dette programmet enn i Web Forms, så noen endringer ble det naturligvis.

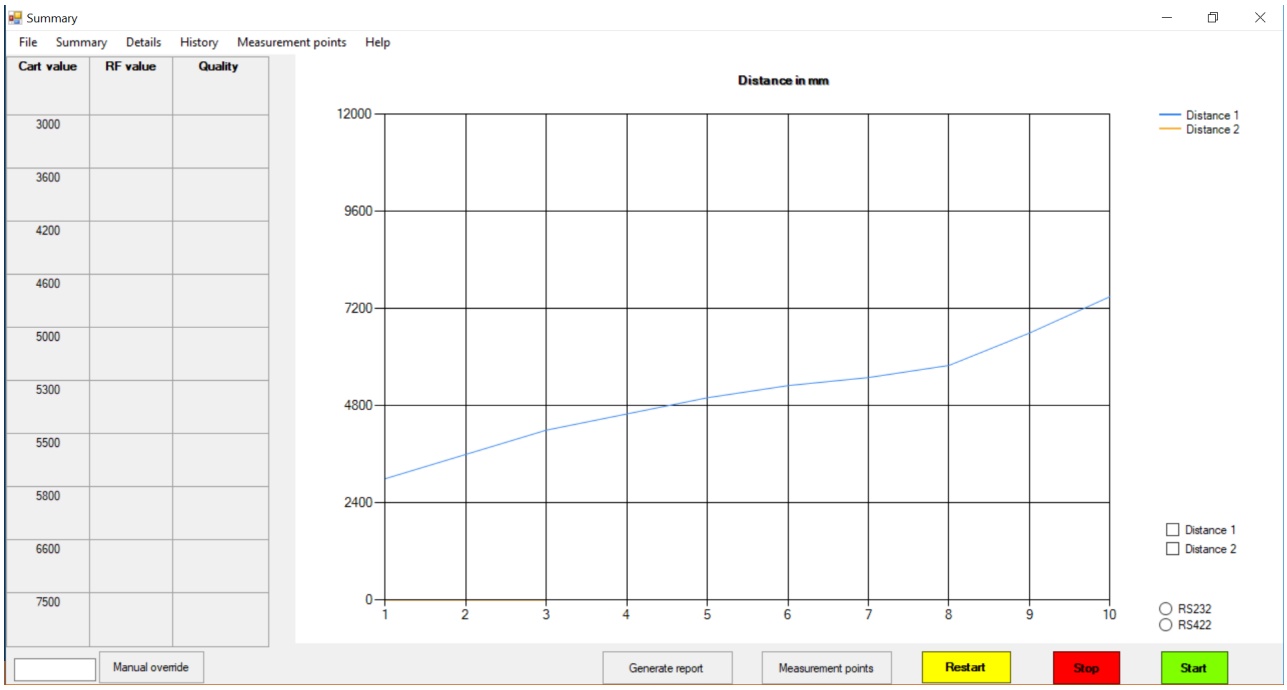
Øverst på hver side ligger en «MenuStrip» kontroll, dette er det samme som en klassisk menylinje i Windows-applikasjoner. I vårt prosjekt benyttes denne menylinjen som en klassisk navigasjonsmeny.

Applikasjonens vinduer/sider

Applikasjonen vår består av fire sider, der tre av disse også finner sted i Miros sitt nåværende program. Den fjerde siden er nyutviklet og heter «Measurement Points», siden skal brukes til endring av lagrede distansepunkter. Vi vil her gi en gjennomgang av samtlige sider, både i forhold til valg av design og implementerte funksjoner, samt funksjoner planlagt for senere implementering.

Summary

Dette er siden som åpnes når brukeren starter programmet. Designmessig er siden fordelt på tre områder; en tabell på venstre side, en graf på resten av den midtre delen, og en linje nederst med en rekke valgmuligheter. Disse i tillegg til navigasjonsmenyen øverst som vises på samtlige sider.



Figur 9.6: Summary

Funksjoner

Knappene i summary-vinduet utløser de fleste funksjonene som er gitt i kravene for systemets software. Her vil det gis en oversikt over hva knappene gjør og hvordan funksjonene utføres på kodenivå.

Start

Start-knappen er knappen som skal sette en kjørerunde og dermed det bevegelige systemet i gang. Når denne knappen trykkes hentes måleverdiene fra databasen. Disse verdiene legges da i et gitt format i en tekstfil slik at den er lesbar for Node-Red og Raspberry Pi [43].

```
private void button1_Click(object sender, EventArgs e)
{
    System.IO.File.WriteAllText(@"C:\Users\vebsv\Dropbox\Node\Stop.txt", @"{ ""Stopp"": 0}");
    StreamWriter myFileStart = new StreamWriter(@"C:\Users\vebsv\Dropbox\Node\cor.txt");
    SqlConnection con = new SqlConnection(conString);
    con.Open();
    if (con.State == System.Data.ConnectionState.Open)
    {
        string query1 = "SELECT  TOP 9 * FROM MeasurePoints";
        SqlCommand command1 = new SqlCommand(query1, con);
        SqlDataReader reader1 = command1.ExecuteReader();
        myFileStart.Write("{");

        try
        {

            while (reader1.Read())
            {

                myFileStart.WriteLine(String.Format(@"""Dis{0}"": {1},",
                    reader1["PointID"], reader1["DistanceValue"]));

            }

        }
        catch (Exception ex)
        {
```

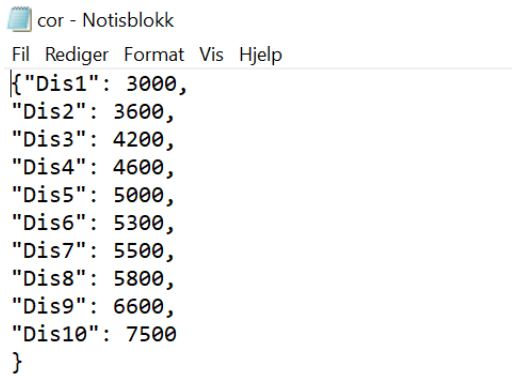
Figur 9.7: Start 1

Denne koden viser logikk for de ni øverste distansene. Dette fordi den nederste linjen må formatteres annerledes for at det skal bli lesbart. Det utføres en SQL query til å hente radene fra databasen.

```
    string query2 = " SELECT * FROM MeasurePoints WHERE PointID = '10'";
    SqlCommand command2 = new SqlCommand(query2, con);
    SqlDataReader reader2 = command2.ExecuteReader();

    try
    {

        while (reader2.Read())
        {

            myFileStart.WriteLine(String.Format(@"""Dis{0}"": {1}",
                reader2["PointID"], reader2["DistanceValue"]));

        }

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());

    }
    finally
    {
        reader2.Close();
        myFileStart.Write("}");
        myFileStart.Close();
```

Figur 9.8: Start 2

Koden for det siste punktet.



Figur 9.9: Distansepunkter

Punktene sendes i tekstfil i dette formatet.
For at det skal være mulig å hente data eller sende data til databasen, må man først skape en connection. Denne linken ble brukt til å gjøre dette [9].

Stopp

Når brukeren trykker på denne knappen sendes det en verdi til Node-Red som indikerer at kjørerunden skal stoppes umiddelbart.

Restart

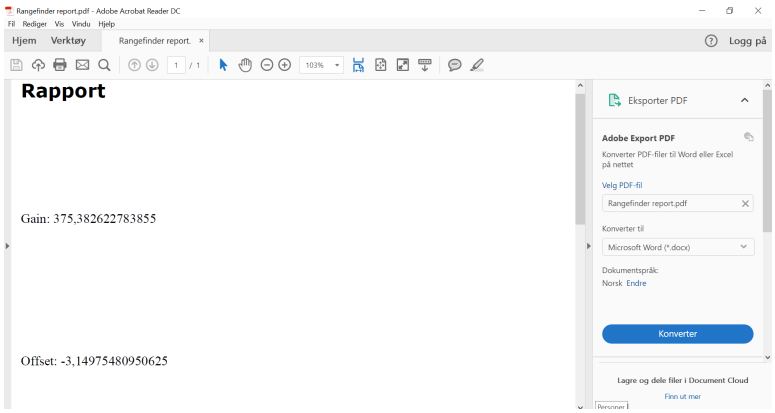
Denne knappen er ny i den endelige utgaven. Den benyttes til å gjenoppstarte en kjørerunde som har blitt stoppet med stopp-knappen. Den tar enkelt og greit verdien som ber systemet om å stoppe og setter den til null.

```
}

private void btnStop_Click(object sender, EventArgs e)
{
    System.IO.File.WriteAllText(@"C:\Users\vebsv\Dropbox\Node\Stop.txt", @"{ ""Stopp"": 1}");

}

private void btnRestart_Click(object sender, EventArgs e)
{
    System.IO.File.WriteAllText(@"C:\Users\vebsv\Dropbox\Node\Stop.txt", @"{ ""Stopp"": 0}");

}
```

Figur 9.10: Stopp og restart

Generate report

Ved å trykke på denne knappen genererer systemet en PDF som skal inneholde viktig informasjon som måleverdier med avvik, og matematiske data som kalkulert gain og offset. Denne pdf-en inneholder kun eksempelverdier foreløpig da det ikke har vært tilstrekkelig med tid til å sette kalkuleringsmodulen inn i funksjonen.



Figur 9.11: Rapport PDF

Slik ser den PDF-en ut for øyeblikket når brukeren trykker på Generate report-knappen.

Manual override

Det er satt krav til at systemet vårt skal kunne manuelt kjøres til en valgt distanse utenom kjørerundene. Et tekstfelt til venstre for knappen sendes i en tekstfil som en enkel distanse når brukeren trykker på knappen.

```
private void btnManuell_Click(object sender, EventArgs e)
{
    int box_intManuell = 0; Int32.TryParse(txtManuell.Text, out box_intManuell);
    //checks if textbox is left empty when submit button is pressed

    if (string.IsNullOrEmpty(txtManuell.Text))
    {

    }

    else if (box_intManuell > 12000 || box_intManuell < 3000 && box_intManuell != 0)
    {
        MessageBox.Show("Value entered must be between 3000mm and 120000mm");
    }

    else
    {
        System.IO.File.WriteAllText(@"C:\Users\vebsv\Dropbox\Node\Manuell.txt", @"{""DisManual"": " + txtManuell.Text + "}");
    }
}
```

Figur 9.12: Manuell

Tabell

Denne tabellen består av tre kolonner: «Cart value», «RF value» og «Quality». Det er planlagt at «Cart value» viser verdiene gitt fra proximity sensor og incremental encoder, «RF value» viser verdiene fra Rangefinderen, mens «Quality» viser måleavviket mellom verdiene. For øyeblikket vises kun de valgte distansepunktene fra databasen i «Cart value» kolonnen som en test, men dette skal etter planen endres på.

Cart value	RF value	Quality
3000		
3600		

Figur 9.13: Summary tabell

Et kort utsnitt av tabellen som viser to av verdiene som er hentet fra databasen

Graf

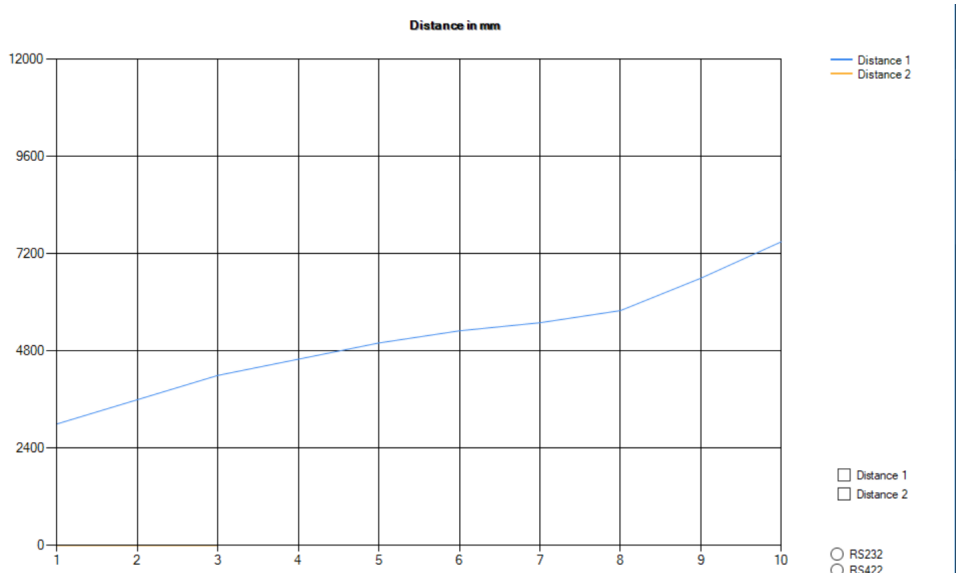
Dette er grafen som skal brukes til å visualisere avviket mellom målingene gjort av normalen og Rangefinderen. Det er lagt til to såkalte «chart series», en for normalens verdi og den andre for Rangefinderens. Som med tabellen har vi hentet distansepunktene fra databasen og lagt disse inn i «distance 1» serien [14].

Planen for fremtiden er som med tabellen og hente målte verdier og ha disse i seriene. Vi har også laget to «checkboxes» som skal gi brukeren muligheten til å velge hvilke linjer som skal vises. En funksjon for å zoome inn for å se millimeterne nærmere vil også være nyttig.

```
DataTable table = new DataTable();
SqlDataAdapter adapter = new SqlDataAdapter();

SqlConnection con = new SqlConnection(conString);
con.Open();

string command = "SELECT * from MeasurePoints";
SqlCommand cmd = new SqlCommand(command, con);
adapter.SelectCommand = cmd;

adapter.Fill(table);

//Set DataTable as data source to Chart
this.chart1.DataSource = table;

//Mapping a field with x-value of chart
this.chart1.Series[0].XValueMember = "PointID";

//Mapping a field with y-value of Chart
this.chart1.Series[0].YValueMembers = "DistanceValue";

//Bind the DataTable with Chart
this.chart1.DataBind();
```

Figur 9.14: Graf kode

Dette bildet viser koden som henter data fra databasen og binder de til grafen.



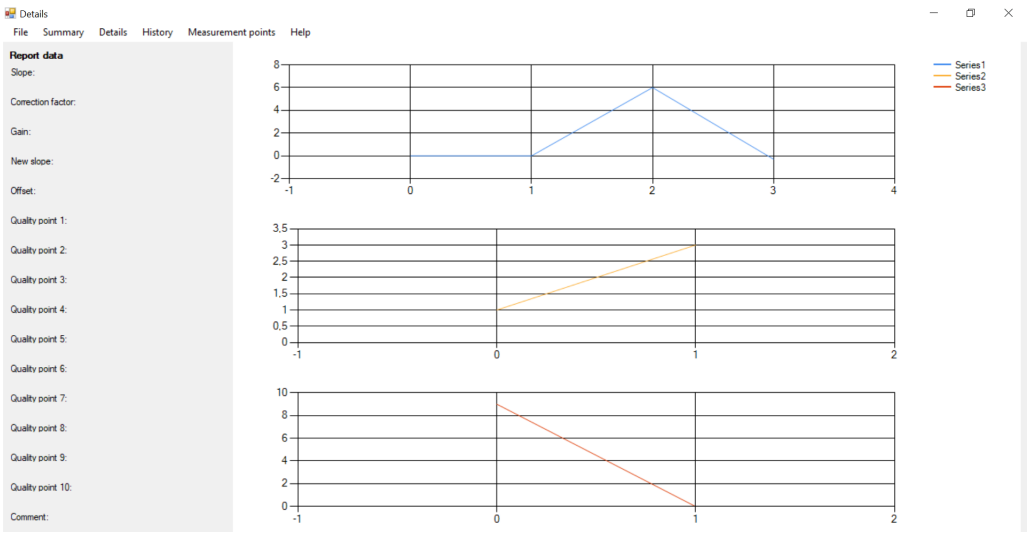
Figur 9.15: Graf bilde

Bilde av grafen slik den ser ut nå med verdier fra databasen som linje med punkter.

Vi har også lagt til to «radio buttons» som skal gi brukeren mulighet til å velge tilkoblingsmulighet, RS232 eller RS422. Fordelen med disse knappene er at en må være valgt hele tiden, så det er ikke mulig å starte en kjørerunde uten at et alternativ er valgt.

Details

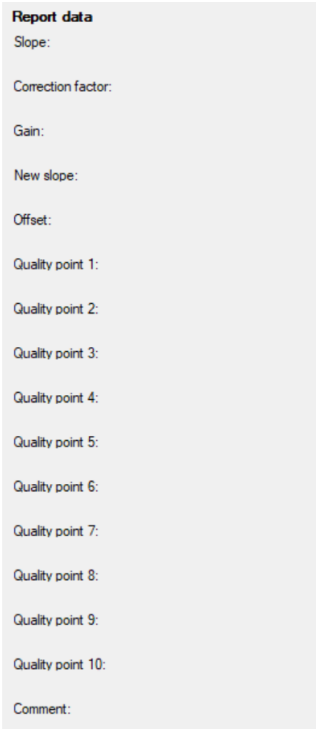
Denne siden skal vise den samme informasjonen som Miros sitt vindu med samme navn gjorde. Siden er i delt i to deler, en del hvor tabellen med rapportdata skal vises, og en del med tre ulike grafer skal vises. Funksjonene til denne siden er planlagt, men ikke implementert.



Figur 9.16: Details

Rapporttabell

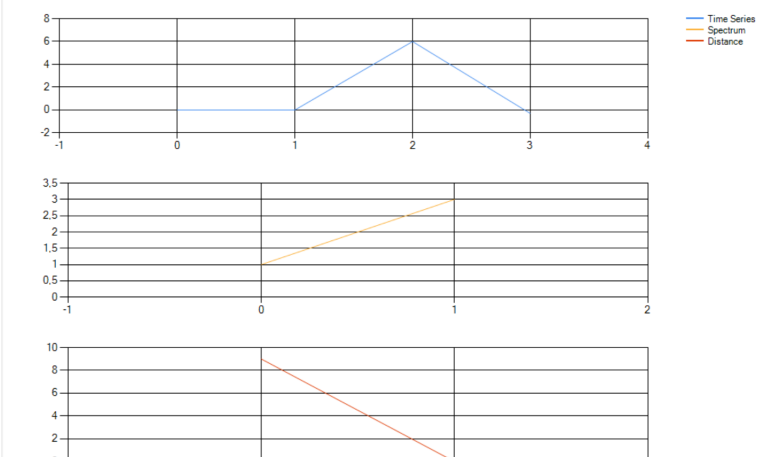
Denne tabellen skal inneholde informasjon som skal vises i den genererte rapporten. For øyeblikket ligger feltene klare med eksempeldata som vises i bildet under.



Figur 9.17: Details tabell

Grafer

Siden inneholder tre grafer som skal inneholde samme moduser som Miros sitt program har. Disse modusene er time series, spectrum og en vanlig distansegraf.



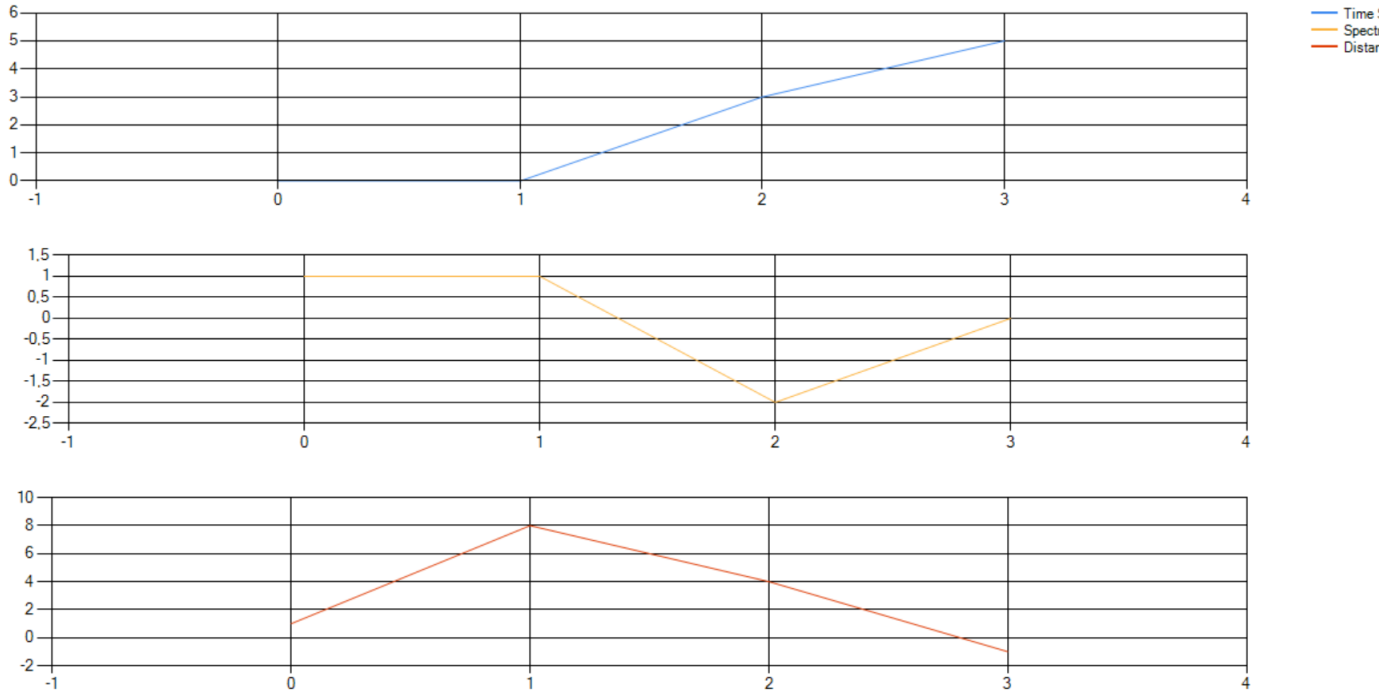
Figur 9.18: Details grafer

History

Denne siden skal også vise den samme informasjonen som vinduet med samme navn gjør i Miros sin applikasjon. Denne siden har bare et stort område hvor funksjoner som planlegges å tilføre er en tidsoverføring fra siste døgnet med målinger.

Grafer

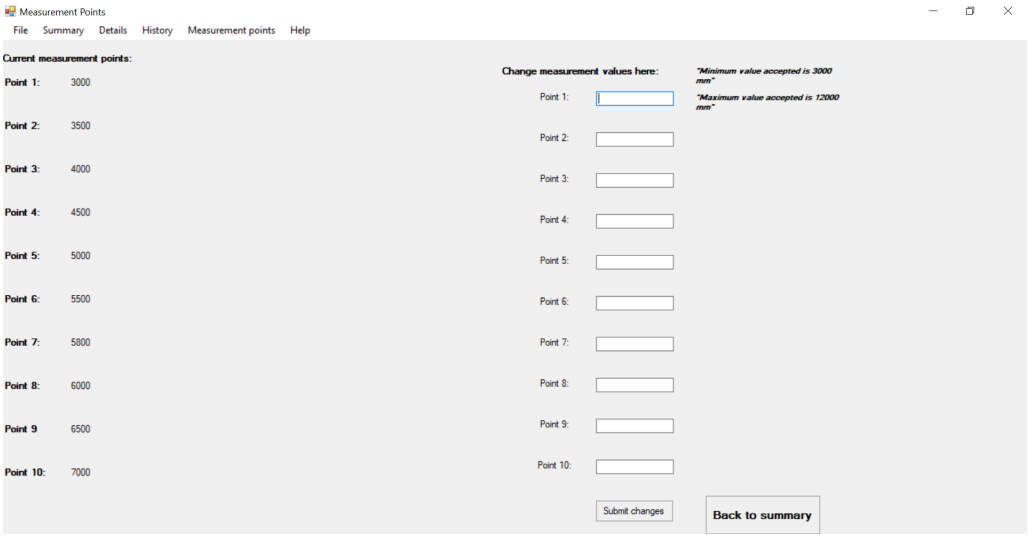
Grafene på History-siden viser de samme modusene som grafene på Details-siden. Forskjellen er at siste distansegrafen skal vise målinger utført siste døgnet istedenfor i nåtid.



Figur 9.19: History grafer

Measurement points

Dette er siden hvor de lagrede målepunktene kan endres. Denne siden er nyutviklet for å tilfredsstille automatiseringskravet. Siden er designet i to deler, en med tabell som viser nåværende kjørepunkter og en med tekstfelter og knapper for å endre på disse punktene.



Figur 9.20: Measurement points

Vi vil her gå gjennom funksjonalitet som er implementert på denne siden.

Målepunkter

Det ligger ti tekstbokser på denne siden som tar inn ønskede distanseverdier og lagrer de til databasen. For å unngå og få noen annet enn gyldige tallverdier inn er det lagt inn en funksjon som sier ifra når bokstaver tastes inn, samt en grense på verdi mellom 3000 og 12 000mm

```
private void txtP10_KeyPress(object sender, KeyPressEventArgs e)
{
    char ch10 = e.KeyChar;

    if (!Char.IsDigit(ch10) && ch10 != 8 && ch10 != 13)
    {
        e.Handled = true;
        MessageBox.Show("Please enter a valid value.");
    }
}
```

Figur 9.21: TxT keypress

Koden som sjekker at kun tallverdier eller backspace tastes inn. En feilbeskjed i form av en «messagebox» vil sendes til brukeren hvis bokstaver eller annet tastes inn.

```
else if (box_int10 > 12000 || box_int10 < 3000 && box_int10 != 0)
{
    MessageBox.Show(error);
}
```

Figur 9.22: TxT grenseverdi

Kode som sjekker om verdien som tastes inn er innenfor en grense på 3000 og 12 000 mm.

Submit changes

Dette er knappen hvor man lagrer punktene som man har tastet inn. Først slettes den gamle verdien som man skal endre fra databasen. Så vil verdiene som er tastet inn bli sendt til hvert sitt felt i databasen, og fra databasen sendes disse punktene ned igjen til tabellen på venstre side. Slik kan brukeren da kontrollere om inntastede verdier stemmer overens med det brukeren har ønsket.

```
//if a value is entered, old value is deleted and replaced by the new value
string deleteString1 = "DELETE FROM [dbo].[MeasurePoints] WHERE PointID = '1'";
string deleteString2 = "DELETE FROM [dbo].[MeasurePoints] WHERE PointID = '2'";
string deleteString3 = "DELETE FROM [dbo].[MeasurePoints] WHERE PointID = '3'";
string deleteString4 = "DELETE FROM [dbo].[MeasurePoints] WHERE PointID = '4'";
string deleteString5 = "DELETE FROM [dbo].[MeasurePoints] WHERE PointID = '5'";
string deleteString6 = "DELETE FROM [dbo].[MeasurePoints] WHERE PointID = '6'";
string deleteString7 = "DELETE FROM [dbo].[MeasurePoints] WHERE PointID = '7'";
string deleteString8 = "DELETE FROM [dbo].[MeasurePoints] WHERE PointID = '8'";
string deleteString9 = "DELETE FROM [dbo].[MeasurePoints] WHERE PointID = '9'";
string deleteString10 = "DELETE FROM [dbo].[MeasurePoints] WHERE PointID = '10'";

//Strings for inserting values from textboxes to database
string q1 = "insert into MeasurePoints(PointID, DistanceValue)values(' 1 ' , '" + txtP1.Text.ToString() + "')";
string q2 = "insert into MeasurePoints(PointID, DistanceValue)values(' 2 ' , '" + txtP2.Text.ToString() + "')";
string q3 = "insert into MeasurePoints(PointID, DistanceValue)values(' 3 ' , '" + txtP3.Text.ToString() + "')";
string q4 = "insert into MeasurePoints(PointID, DistanceValue)values(' 4 ' , '" + txtP4.Text.ToString() + "')";
string q5 = "insert into MeasurePoints(PointID, DistanceValue)values(' 5 ' , '" + txtP5.Text.ToString() + "')";
string q6 = "insert into MeasurePoints(PointID, DistanceValue)values(' 6 ' , '" + txtP6.Text.ToString() + "')";
string q7 = "insert into MeasurePoints(PointID, DistanceValue)values(' 7 ' , '" + txtP7.Text.ToString() + "')";
string q8 = "insert into MeasurePoints(PointID, DistanceValue)values(' 8 ' , '" + txtP8.Text.ToString() + "')";
string q9 = "insert into MeasurePoints(PointID, DistanceValue)values(' 9 ' , '" + txtP9.Text.ToString() + "')";
string q10 = "insert into MeasurePoints(PointID, DistanceValue)values(' 10 ' , '" + txtP10.Text.ToString() + "')";

//Strings that changes the value in the read-only text boxes when user hits submit button
string CV1 = "SELECT DistanceValue FROM MeasurePoints WHERE PointID = '1'";
```

Figur 9.23: Submit DB

Dette er koden for sletting av rader i databasen, og sending av nye verdier til databasen. Nederst vises koden som sender verdier fra databasen til en tabell.

```
//if not, queries shown above will be executed
SqlCommand del = new SqlCommand(deleteString1, con);
SqlCommand cmd = new SqlCommand(q1, con);
SqlCommand newV1 = new SqlCommand(CV1, con);

del.ExecuteNonQuery();
cmd.ExecuteNonQuery();

using (SqlDataReader change1 = newV1.ExecuteReader())

    while (change1.Read())
    {
        this.txtV1.Text = (change1["DistanceValue"].ToString());
    }
this.txtP1.Text = "";
```

Figur 9.24: Submit query

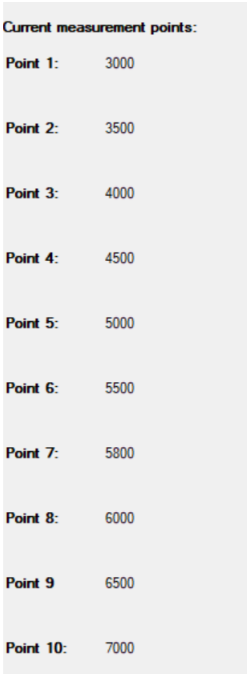
Her vises koden som bruker SQL-bibliotek til å sende etterspørsler om å slette, gi nye verdier og sette disse inn i tabell med gjeldende målepunkter

Back to summary

Denne knappen fungerer som en alternativ vei tilbake til hovedsiden.

Tabell med gjeldene målepunkter

Denne tabellen ligger til venstre på siden og viser til enhver tid punktene som ligger lagret i databasen.



Figur 9.25: Measure tabell

9.2 SQL database

SQL står for Structured Query Language, og er et programmeringsspråk som brukes til å behandle databaser og behandle dataene lagret i databasene. Bruksområdene for behandling av data inkluderer endring av data og struktur i lagrede tabeller. Eksempler på dette er å legge til, oppdatere og slette rader med data samt å bruke sett med data fra andre tabeller for vedlikehold av mer avanserte systemer. For informasjon om hvordan man installerer Microsoft SQL Server, se vedlegg L.

Serveren

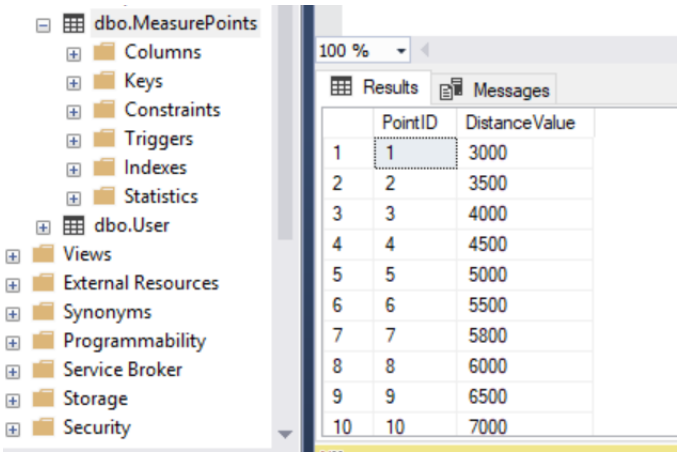
I prosjektet har vi brukt Microsoft SQL Server til å lagre data med informasjon om gjeldende målepunkter. Begrunnelsen for dette er at det kun skal kreves at brukeren trykker på start-knappen for at en ny kjørerunde skal settes i gang. Valg av servertjeneste er gjort både med tanke på fremtiden og at Visual Studio har innebygde funksjoner for å koble applikasjoner til databaser.

Database/tabell

Vi har opprettet en database som vi har gitt navnet COR. Dette er senteret for lagring av tabeller hvor det kan legges til funksjonalitet som en brukertabell for å lagre brukere eller administratorer til brukergrensesnittet vårt. Vi har per nå en tabell for å forenkle automatiseringsprosessen.

Measurepoints

Dette er navnet på tabellen som er i bruk i vårt nåværende system. Denne tabellen skal forenkle kjøringsprosessen ved å holde på distansepunkter og verdier. Kort gjennomgang av kolonnene følger under.



Figur 9.26: DB tabell

Kolonner

Tabellen består av to kolonner; «PointID» for identifisering av punkter og «DistanceValue» for lagring av verdier.

PointID

Denne kolonnen gir hvert målepunkt en unik identitet. Denne identiteten gjør at de riktige radene endres på når noen endrer punkter i brukergrensesnittet.

Distance value

I denne kolonnen lagres verdiene som tilsier hvilke distanser systemet skal måle i form av millimeterverdier mellom 3000 og 12 000.

9.3 Webside

Under en periode i prosjektet ble det utviklet et brukergrensesnitt som etter planen skulle fungere som et sky-basert grensesnitt lastet opp som en nettapplikasjon til Microsoft Azure. Denne utgaven ble mot sluttfasen av prosjektet forkastet og erstattet med en lokal applikasjon, siden det viste seg å bli litt for krevende og kommunisere fra et grensesnitt i en nettsky med tiden vi hadde til rådighet.

Men selv om det ikke ble noe endelig resultat av denne prosessen, så var den nyttig i utvikling av et tenkt design, spesielt i forhold til bestemmelse av målepunkter, som fikk tilegnet en egen side med et design som ble ganske likt sluttresultatet. Til å utvikle siden benyttet vi Web Forms App via Microsoft Visual Studio, dette siden programmet har innebygd funksjonalitet for opplasting til Microsoft Azure

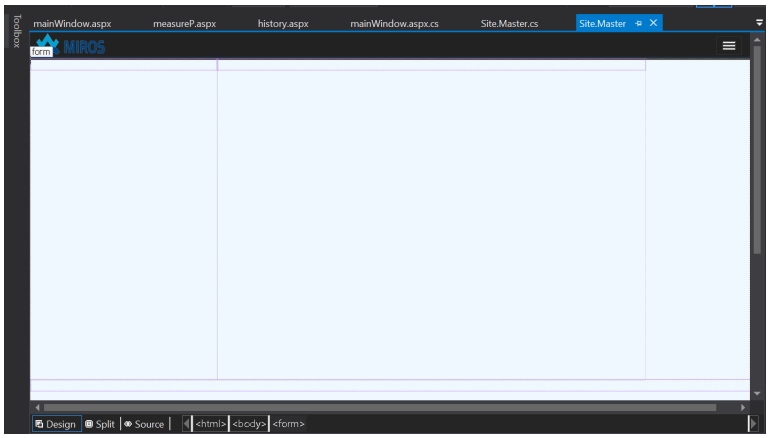
Design

Det finnes både fordeler og ulemper ved å designe ett web basert grensesnitt med Web Forms. For å utnytte fordelene best mulig samtidig som å unngå og støte på ulempene, så er erfaring svært nyttig. En bratt læringskurve med en god del omgjøring underveis var nødvendig for oss da ingen medlemmer hadde erfaring med programmet.

Å plassere de ulike elementene på en fornuftig og god måte er en utfordring med Web Forms, spesielt etter hvert som sidene vokser og flere elementer legges til. Mye design foregår på en «drag and drop» metode, som sparer en del innskriving av koder, men som ofte kan lure en til å tro at designet er mindre utfordrende enn det faktisk er. Bruk av paneler til å dele siden inn i ulike seksjoner er fordelaktig for å unngå uforutsette problemer når nye elementer legges til. Planlegging av disse ulike seksjonene før implementering vil derfor bli veldig tidsbesparende når man jobber med dette programmet.

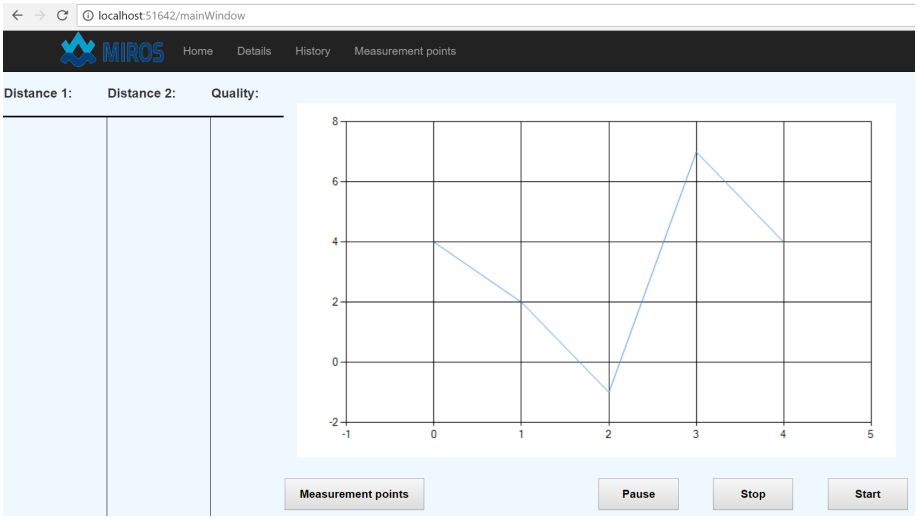
En fordel i Web Forms er at prosjektene har en innebygget skisseringside kalt «Site.Master», hvor man kan legge til paneler som automatisk arves og legges til i de ulike sidene. Elementer som menyer og andre ting som ønskes å vises i samme seksjon av hver side, kan også legges til her for å spare mye tid. Man slipper da å kopiere denne koden til samtlige sider.

I vår «Site.Master» er det lagt til en navigasjonsmeny øverst, og deretter tre paneler under menyen. Et panel er plassert til venstre for utvikling av tabeller og lignende, mens det til høyre for dette ligger et slags hovedpanel hvor grafer og større menyer som endring av målepunkter plasseres. Et panel er også plassert på langs nederst på siden, et område som hovedsakelig er planlagt å inneholde knapper og andre valgmuligheter.



Figur 9.27: MasterPage

Når planen ble endret vekk fra nettbasert grensesnitt så var antallet sider det samme som det endelige grensesnittet endte med. En hovedside som inneholder knapper for å starte, stoppe og midlertidig pause kjøring, samt en knapp som sender brukeren til siden hvor man endrer målepunkter. I tillegg til knapper så viser hovedsiden en skissert graf og en tabell ment for sammenligning av målinger.



Figur 9.28: Web UI

Som Miros sitt grensesnitt har nettapplikasjonen en side kaldt details som skal vise rapportinfo i venstre panel og tre grafer med ulike moduser i hovedvinduet. Historysiden er også lik, det vil si at de samme grafene som i details er skissert, men disse skal planlagt vise informasjonen opptil et døgn tilbake i tid.

En ny side som også ble brukt i det endelige grensesnittet ble utviklet. Siden «Measurement points» er en side som inneholder ti tekstbokser hvor brukeren taster inn ønskede målepunkter. En knapp er laget under for lagring av punktene, og nåværende målepunkter skal bli vist i en tabell i venstre panel.

9.4 Kalkuleringskode

Intro

En av oppgavene vi startet med da vi gikk inn i tredje fase var å gjøre klar en kode som skulle kalkulere et nytt Gain (Gain er en målingsmetode av egenskapene til en «to ports krets» til å øke strømmen eller amplituden til et signal fra input til output porten) og en ny korreksjonsfaktor for RangeFinderen. Vi ble tilsendt et Excel-ark fra Miros med diverse grafer og info om hva de gjør for å regne ut et nytt Gain og ny korreksjonsfaktor. Oppgaven ble da å sette denne informasjonen inn i en metode for å automatisere utredningsprosessen, noe som vil føre til tidskutt og effektivisering av kalibreringsprosessen til Rangefinderen.

Målinger		Lasermålt	RangeFinder	Avvik
		mm	mm	mm
Distanse 1		4009	4010	1
Distanse 2		3576	3577	1
Distanse 3		4349	4350	1
Distanse 4		4372	4372	0
Distanse 5		4908	4907	-1
Distanse 6		5370	5368	-2
Distanse 7		5864	5862	-2

Figur 9.29: Kalkuering 1

Intercept:	7,262979		
Slope:	0,998372		
Opprinnelig Gain:		374,383	
Korreksjonsfaktor:		1,0016311	
Gainkorrigert RangeFinder:			4016,54085
			3582,83457
			4357,09543
			4379,13132
			4915,00398
			5376,75593
			5871,56171

Figur 9.30: Kalkuering 2

Nytt Gain:		374,99367		
Ny Offset:		7,2748259		
ETTER GAIN OG OFFSETKORREKSJON:				
			RangeFinder	Avvik
			4009,26602	0,266022369
			3575,55974	-0,440258755
			4349,82061	0,820608009
			4371,85649	-0,143507038
			4907,72915	-0,270850221
			5369,4811	-0,518897338
			5864,28688	0,286882974

Figur 9.31: Kalkuering 3

Fremgangsmåte

Som standard er Gain-verdien for Rangefinderen satt til 374,383. For å se om vi hadde gjort riktige kalkuleringer, satte vi inn de samme verdiene som Rangefinder og Lasermåler hadde blitt tildelt i Excel-arket. Verdien ble satt inn i hvert sitt Array. For å få så nøyaktig måling som mulig bruker vi data-typen «double» slik at vi kan bruke desimaltall. Det første Utregningskoden må gjøre er å regne ut stigningstallet til grafen, denne verdien har vi gitt navnet Slope. Den blir regnet ut med ligningen (RFmax-RFmin)/(ECmax-ECmin)=Slope, der RF står for Rangefinder og EC står for Encoder. Deretter brukte vi Stigningstallet til å finne korreksjonsfaktoren ved å dividere 1 på stigningstallet. 1/Slope=correction_factor

```
double[] Encoder = {4009, 3576, 4349, 4372, 4908, 5370, 5864}; // Encoder array for the different distances measured.

double [] RF = {4010, 3577, 4350, 4372, 4907, 5368, 5862}; //RangeFinder array for the different distances measured.

double Slope = 0;


Slope = (RF[RF.Length-1] - RF[0]) / (Encoder[Encoder.Length-1] - Encoder[0]); // calculating Slope

double s1 = Slope;

Console.WriteLine(s1);

double correction_factor = 1 / s1; // correction factor
```

Figur 9.32: Kalkulere slope og correctionfactor

Korreksjonsfaktoren ble også brukt til å regne ut et nytt Gain. Dette ga oss en ny Gainverdi på 374,99364. For å kontrollere at Gain er riktig korrigert kan vi multiplisere alle Rangefinder-verdier med Korreksjonsfaktoren.

```
double Gain = 374.383; // standard gain for all RangeFinders



Gain += correction_factor; // get a new Gain by multiplying it with the correction factor.

for (int i = 0; i < RF.Length; i++) // we multiply the correction factor with all the RF values.
{
    RF[i] = RF[i] * correction_factor;
    Console.WriteLine(RF[i]);
}
```

Figur 9.33: Kalkulere gain

Deretter må det regnes ut en ny Slopeverdi slik at den kan brukes til å regne ut avvik. (RFmax-RFmin)/(ECmax-ECmin)=New_Slope. Offset blir regnet ut slik: Offset=(RFmin-(New_Slope*ECmin)) Offset blir så subtrahert med alle verdier i det korrigerte Rangefinder-arrayet. Dette vil gi oss de nye distansene til Rangefinderen når den har blitt kalibrert.

```
double NS = 0; // the new slope from calculating the new values from RF.


NS = (RF[RF.Length - 1] - RF[0]) / (Encoder[Encoder.Length - 1] - Encoder[0]);

double New_Slope = NS;

double Offset = 0;

Offset = RF[0] - (New_Slope * Encoder[0]); // calculating the new intersection (Offset)

Console.WriteLine(Offset);

for (int i = 0; i < RF.Length; i++)
{
    RF[i] = RF[i] - Offset;
}

Console.WriteLine("");

for (int i = 0; i < RF.Length; i++)
{
    Console.WriteLine(RF[i]);
}
```

Figur 9.34: Kalkulerer offset

9.5 Rapportgenering

Introduksjon

Ett av kravene som ble satt av Miros var at resultater fra kalibreringsjiggen skulle bli lagret i en rapport i form av en utskriftsbar PDF. Den skulle bli lagret i Microsoft Azure og skal også bli lagret lokalt. Så utfordringen ble da å utvikle en kode som skaper en PDF-fil med informasjon hentet fra Encoder og Rangefinder.

Fremgangsmåte

For å skape en slik kode brukte vi et kodebibliotek som heter PDF Sharp. PDF Sharp er «open source» og ligger tilgjengelig i GitHub. Informasjon om koden og hvor vi kunne hente den lå på nettsiden [10]

Etter at biblioteket ble lastet ned, ble det satt inn i PDF-genereringskoden. For å skape et PDF-dokument måtte vi først kalle på en ny instans av metoden PDF dokument, deretter ble en instans av PDFpage metoden kalt for å skape en side for PDF-dokumentet. Videre brukte vi xGraphics, xTextFormater og XFont for å utforme PDF dokumentet

```
PdfDocument pdf = new PdfDocument(); //create a new instance of a PDF Document

PdfPage pdfPage = pdf.AddPage(); // create a new instance of a PDFPage

XGraphics graph = XGraphics.FromPdfPage(pdfPage); //

XTextFormatter tfx = new XTextFormatter(graph);

XFont font = new XFont("Verdana", 20, XFontStyle.Bold);
```

Figur 9.35: PDF form

Deretter lager vi en DrawString slik, og gir den en plassering på toppen av PDF-dokumentet slik at den fungerer som tittel. Så gir vi xFont en ny skriftstørrelse og lager tre nye DrawString for Gain, Offset og Avstander.

```
graph.DrawString("Repport dockument for Calibrationjigg.", font, XBrushes.Black,new XRect (10, 0, pdfPage.Width.Point, pdfPage.Height.Point), XStringFormats.TopLeft );
font = new XFont("Times New Roman", 14, XFontStyle.Regular);
graph.DrawString("Gain: " + d , font, XBrushes.Black, new XRect (10,150, pdfPage.Width.Point, 150), XStringFormats.TopLeft );
graph.DrawString("Offset:", font, XBrushes.Black, new XRect(10, 300, pdfPage.Width.Point, 300), XStringFormats.TopLeft);
graph.DrawString("Distances: ", font, XBrushes.Black, new XRect(10, 450, pdfPage.Width.Point, 450), XStringFormats.TopLeft);
```

Figur 9.36: PDF innhold

For å lagre dokumentet må vi bruke en String til å gi informasjon om filnavnet til dokumentet. Til slutt kaller vi på metoden pdf.save og setter inn i filbanen hvor vi ønsker å lagre dokumentet.

```
string pdfFileName = "COR-RangeFinder_Report";

pdf.Save("C:\\Users\\Windows User\\Documents\\" + pdfFileName + ".pdf");
```

Figur 9.37: PDF lagring

9.6 MQTT

For kommunikasjon mellom kode i C og Python, valgte vi å bruke MQTT.

Via en «Broker» kan klienter sende eller motta (publisere/abonnere) «Emner», som inneholder data som blir brukt av de forskjellige programmene. For eksempel kan et array med distanseverdier som har blitt skrevet inn i brukergrensesnittet bli publisert i brokeren fra C. Så for å motta emnet må Pythonkoden abonnere på emnet sendt fra C. Python mottar emnene via Node-red og konsollkommandoer. Problemet vi hadde var å etablere kontakt mellom C-klienten og brokeren. Vi la inn en kode som skulle skrive ut om klienten klarte å abonnere på et emne, men etter mange mislykkede forsøk endte det med at vi ga opp MQTT og gikk over til å bruke tekstfiler som vi legger inn i en sky-basert server.

Kode

```
public partial class Form1 : Form
{
    // We first create a client Instance
    //private readonly  MqttClient client_new;
    MqttClient client_new = new MqttClient("192.168.1.235" /*" broker.hivemq.com"*/);

    public Form1()
    {
        // the address for our MQTTbroker
        //Byte IPA = 192.168.1.235;
        //string broker = "192.168.1.235:1883";

        InitializeComponent();

        //byte code = client_new.Connect(Guid.NewGuid().ToString());
        byte code = client_new.Connect(Guid.NewGuid().ToString(), "COR-1", "Bachelor2018");
    }
```

Figur 9.38: MQTT connect kode

For å ta i bruk MQTT i C var vi nødt til å laste ned M2MQTT-biblioteket ifra NUGET ved å bruke open source kode lånt fra [23].

Vi begynte med å legge inn alle bibliotekfilene vi trengte. Deretter skapte vi en ny instans av MqttClient slik at vi kunne bruke MQTT-biblioteket. I denne instansen satte vi inn IP adressen til brokeren vår.

Så etter å ha laget et MQTT-objekt måtte vi kalle på metoden Connect() for å koble til brokeren vår. Vi spesifiserte brukernavn og passord i Connect()-metoden ettersom det var noe vi hadde satt inn i brokeren.

```
public void Subscribe()
{
    client_new.MqttMsgUnsubscribed += client_MqttMsgSubscribed;

    ushort msgID = client_new.Subscribe(new string[] { "/beskjed" }, // here is a list of topics We want to subscribe to
        new byte[] {MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, MqttMsgBase.QOS_LEVEL_AT_LEAST_ONCE });// here is a related list of QoS levels with one for each topic


}
void client_MqttMsgSubscribed(object sender, MqttMsgSubscribedEventArgs e)
{
        Debug.WriteLine("Subscribed for id = " + e.MessageId);
}
```

Figur 9.39: MQTT subscribe kode

Deretter utviklet vi en metode for å abonnere, Subscribe(), hvor vi ba om å koble oss til emnet «beskjed» som vi hadde satt inn i brokeren. Innholdet i emnet skulle bli skrevet ut i debuggeren.

```
public void Publish()
{
    client_new.MqttMsgPublished += Client_MqttMsgPublished;

    ushort msgID = client_new.Publish("/my_topic", // topic
        Encoding.UTF8.GetBytes("MyMessageBody"), // message Body
        MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE,// QoS level
        false); // retained
}

public void Client_MqttMsgPublished (object sender, MqttMsgPublishedEventArgs e)
{
    Debug.WriteLine("MessageID = " + e.MessageId + " Published = " + e.IsPublished);
}
```

Figur 9.40: MQTT publish kode

Vi prøvde også å lage en publiseringsmetode Publish(), hvor vi prøvde å publisere et emne med innholdet «MyMessageBody».

Dessverre fikk vi som nevnt aldri kontakt med brokeren, og etter mange forskjellige forsøksmetoder så endte det med at vi måtte gi opp denne koden.

9.7 C#

C# er et Multi-Paradigm programmeringsspråk som omfatter: Strong typing (streng stavekontroll), imperativ (bruker påstander som endrer programmets tilstand), Functional(behandler beregning som en evaluering av matematiske funksjoner), generisk og objekt orientert programmering. C# ble designet og utviklet av Microsoft og ble lansert i år 2000 [11].

Historie

I Januar 1999, satte Anders Hejlsberg sammen et team som skulle bygge ett nytt programmeringsspråk som på den tiden ble kalt Cool (C-like Object oriented Language). Microsoft endret navnet på PGA varemerket. Da .NET prosjektet ble annonsert for offentligheten hadde navnet blitt endret til C#. Programmeringsspråket fikk innflytelse fra C++, Eiffel, Java, F# og mange andre programmeringsspråk [11].

Hvorfor brukte vi C#?

Grunnen til at vi valgte C# er at det tilbyr muligheter som vil hjelpe datastudentene med å oppfylle kravene som vi fikk fra Miros. Som nevnt ovenfor så er C# objektorientert og blir ofte brukt av firmaer for utvikling av brukergrensesnitt. I tillegg hadde datastudentene erfaring med C++, så det tok ikke lang tid for de å sette seg inn i det nye programmeringsspråket. Windows forms utvikles i C# og er en av de mest praktiske programmene for utvikling av grensesnitt siden den har innebygget funksjonalitet for å koble seg til en database.

Hvorfor brukte vi Visual Studio

Det ble tidlig bestemt at vi skulle bruke Visual Studio som programvare for koding ettersom datastudentene hadde erfaring med programmet fra tidligere fag, og da kunne hjelpe elektrostudentene med å sette seg inn i programmet. En annen viktig grunn var at USN hadde gitt oss gratis tilgang til programmet, som da bidro til en naturlig motivasjon for å bruke programmet.

9.8 Hjemmeside

Gjennom prosjektets faser jobbes det med å utvikle en nettside som informerer om bachelorprosjektet vårt. Her legges det ut informasjon om prosjektoppgaven, prosjektgruppen COR, oppdragsgiver MIROS samt oppdatering på prosjektets utvikling.

Nettsiden, som vist i fig 6.8, er utviklet med html som programmeringsspråk og Notepad++ som IDE. Praktiske kodebibliotek som bootstrap har vært svært nyttig for utvikling av nettsidens design.

Kort info om de ulike sidene og deres innhold følger under.

Figur 9.41: Nettside

Hjem

Dette er områdets forside, og inneholder kort info om nettsidens formål, kort om prosjektgruppen og hva du kan vente deg å finne når du navigerer deg rundt på webområdet.

Om oss

Denne siden inneholder et portrettbilde av hvert gruppemedlem med tilhørende informasjon som navn, studieretning og hovedansvarsområdet under bachelorprosjektet.

Prosjektbeskrivelse

Her kan du lese kort om oppdragsgiveren vår, problemstillingen som legges til grunn for prosjektet, og ut fra problemstillingen beskrivelse av oppgaven vi er tildelt.

Arbeidet

På denne siden legges det ut info om prosjektets framgang og iterasjonene som har blitt gjennomført i de ulike fasene, med problemstillinger og milepæler som oppstår underveis.

Galleri

Gallerisiden skal gi et mer visuelt innblikk i hvordan arbeidsprosessen vår har vært og prosjektets omfang, gjennom interessante bilder av prosjektets ulike spektre.

I tillegg til at prosjektbeskrivelsesiden viser kort info om oppdragsgiver MIROS, så ligger logoen deres til høyre på hver side. Et klikk på denne vil føre brukeren til oppdragsgiverens hjemmeside.

Dokumenthistorie			
Versjon	**Dato**	**Endret av**	**Aktivitet**
0.1	02/02/18	TAT	Dokument opprettet
0.2	05/02/18	TAT	Testtabell mal opprettet
0.3	05/02/18	TAT	Testplan opprettet
0.4	06/02/18	VS	Revidert for publisering
1.0	07/02/18	TAT	Publisert i rapport v1.0
1.1	16/03/18	TAT	Testplanendring
1.2	19/03/18	TAT	Testtabell 1.0 til 2.0 lagt inn
1.3	20/03/18	TAT	Testtabell 3.0 til 4.0 lagt inn
1.4	21/03/18	TAT	Testtabell 5.0 til 6.0 lagt inn
1.5	04/04/18	TAT	Revidert for publisering
2.0	06/04/18	TAT	Publisert i rapport v2.0
2.1	14/05/18	TAT	Test tabell T-1 til T-6 fjernet
2.2	15/05/18	TAT	Test rapport for T-2 og T-3 lagt inn
2.3	18/05/18	TAT	Test rapport for T-1 lagt inn
2.4	19/05/18	VA	Test rapport for T-4 lagt inn
2.5	19/05/18	TT	Test rapport for T-5 og T-6 lagt inn
2.6	21/05/18	TT	Finnpuse av test rapport
2.7	21/05/18	Alle	Revidert for publisering
3.0	21/05/18	Alle	Publisert i rapport v3.0

Tabell 10.1: Dokumenthistorie for test dokument

10.1 Innledning

Testanalyse har som hensikt å gi en oversikt over hvordan systemet skal oppfylle krav fra bedrift og sluttforbruker. Dette er viktig siden testresultat avgjør både hvorvidt sluttproduktet er tilfredsstillende nok for bedriften i form av funksjonalitet og ytelse.

10.2 Tabelloppsett

• **Krav ID:** Fra kravtabell.

• **Test ID:** Unik ID for hvert krav som testes.

• **RevisjonsID:** Gir oversikt over hvor mange ganger testen har blitt utført. Flere av testene kreves høy grad av nøyaktighet, noen ganger vil det derfor oppstå problemer, både i forhold til testverktøy eller andre hendelser.

• **Test type:** Hvilken form for test som har blitt utført, dvs: simulering*,fysisk.

• **Utstyr:** Hvilke utstyr som har blitt brukt under testing.

• **Kommentar:** Kommentarfeltet fylles ut med andre viktige hendelser, som beskrivelse av hvorfor testresultatet har blitt godkjent eller ikke. Dette gjøres for å kvalitetssikre vår arbeidsutførelse med tanke på kravspesifikasjon.

• **Teststed:** Forteller oss hvor og i hvilket miljø testen har blitt utført.

• **Testet av:** Person som har utført testen.

• **Godkjent av:** Person som har godkjent testen.

Krav ID :	**Test ID:**	**Revisjons ID :**	**Test type :**
******	********	******	*******
Krav beskrivelse :			
Test Beskrivelse :			
Utstyr :			
Resultat :			
Kommentar :			
Test dato : *********		Testet av : *******	
Test sted : *******		Godkjent av : *******	

Figur 10.1: Testoppsett eksempel.

• Simulering: Foregår tidlig i prosjektfasen. Her simulerer vi om konseptet vårt oppfyller kravene som vi har satt opp.
• Fysisk test: Her settes prototyper av konseptet vårt ut i praksis, ved at det fysiske systemets egenskaper testes. Ut fra fysiske tester kan vi detektere eventulle feil og mangler i vårt system.

10.3 Test rapport for T-1.

Test rapport for T-1.0

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-1.0	T-1.0		
Kravbeskrivelse :	Rangefinderen skal ha et maksimalt tillatt måleavvik på +/-5mm.		
Testbeskrivelse :	Etter kalibreringsrunden skal det sjekkes i kalibreringsrapporten om Rangefinderens måleverdi er innenfor grenseverdien.		
Utstyr :	PC, kalibreringsrapport.		
Resultat :	Ikke testet.		
Kommentar :			
Test dato : *******		Testet av : ******	
Test sted : *******		Godkjent av : ******	

Tabell 10.2: Test rapport T-1.0

Test utførelse: Denne testen ble ikke utført.
Test resultat:

Test rapport for T-1.1

Krav ID :	Test ID :	Revisjons ID :	Test type :
K-1.1	T-1.1		
Kravbeskrivelse :	Ved måleavvik innenfor grenseverdi skal rangefinderen verifiseres.		
Testbeskrivelse :	Sjekker om kalibreringsrapport blir generert hvis måleavvik er innenfor grenseverdi.		
Utstyr :	PC, server, skriver.		
Resultat :	Ikke testet.		
Kommentar :			
Testdato : *******		Testet av : ******	
Teststed : ********		Godkjent av : ********	

Tabell 10.3: Test rapport T-1.1

Test utførelse:
Denne testen ble ikke utført.
Test resultat:

Test rapport for T-1.2

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-1.2	T-1.2		
Kravbeskrivelse :	Ved måleavvik utenfor grenseverdi skal rangefinderen utføre ny kalibreringsrunde.		
Testbeskrivelse :	Kalibreringssystem regner ut nytt polynom som lastes opp til RF. Starter ny kalibreringsrunde med nytt polynom.		
Utstyr :	PC, Raspberry Pi 3, RF.		
Resultat :	Ikke testet.		
Kommentar :	Nytt kalibreringspolymon inneholder **offset** og **gain** til RangeFinder.		
Testdato : *********		Testet av : *******	
Teststed : *******		Godkjent av : *******	

Tabell 10.4: Test rapport T-1.2

Test utførelse:
Denne testen ble ikke utført.
Test resultat:

10.4 Test rapport for T-2.

Test rapport for T-2.0

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-2.0	T-2.0	R-2.0	Fysikk
Kravbeskrivelse :	Systemet skal utføre en operasjonrunde autonomt.		
Testbeskrivelse :	Kalibreringsvognen kan kjøre en kalibreringsrunde etter at referansepunkter har blitt lastet inn. Kalibreringsvognen skal kjøre uten ekstern hjelp.		
Utstyr :	PC, Raspberry Pi 3, DC motor, incremental encoder, capacity proximity sensor, DC spenningskilde.		
Resultat :	Godkjent		
Kommentar :	Sjekke med testtabell T-2.1 ,T-2.2, T-2.3		
Testdato : 08.05.2018		Testet av : Tuan Anh Trinh	
Teststed : Miros AS		Godkjent av : Ecir Ali Doganci	

Tabell 10.5: Test rapport T-2.0

Test utførelse: Kalibreringsvogn fullføre alle ti referanser punkter uten ekstern hjelp med en fart på 0.029m/s. Etter referanser punkter har blitt innplemtert til UI og laste opp til Raspberry Pi. Kalibreringsvogn kjøre til hvert referanser punkter og venter fem sekunder før den starte på neste referanser punkt ved end tiende referanser punkt kjøre kalibrering vogn tilbake til start posisjon.
Test resultat: Kalibreringsvogn utføre alle ti referanser punkter uten noe teknisk problem eller stoppe opp under kjøring til neste referanserpunkt.

Test rapport for T-2.1

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-2.1	T-2.1	R-5.0	Fysisk
Kravbeskrivelse :	Det bevegelige systemet skal automatisk flyttes til ti bestemte referansepunkter.		
Testbeskrivelse :	Etter at referansepunkter har blitt lastet inn, vil den bevegelige vognen kjøre til referansepunkter med feil margin på +- 2mm.		
Utstyr :	PC, målebånd klasse 2, kalibreringsvogn .		
Resultat :	Godkjent		
Kommentar :			
Testdato : 14.05.2018		Testet av : Tuan Anh Trinh	
Teststed : Miros AS		Godkjent av : Deivydas Kazokas	

Tabell 10.6: Test rapport T-2.1

Test utførelse:
ved ti refrenser punkt har blitt innplemtert til kalibrering vogn. Kjørers kalibrering vogn til de referanser punkter ved hvert stopp ble det tatt måling med klasse 2 målebånd. Det ble tatt tre kalibrering runde for å kvalitetsikre kalibrering vogn presisjon eventuelt menneskelig feil. Klasse 2 målebånd har en avvik på +-2.3mm ved 10 meter med en trekk kraft på 20 N [21].
Test resultat:

Runde 1			
Tast inn avstand:	Vognposisjon:	Målebånd :	Feil i mm :
3000	3000	2999	-1
3500	3500	3499	-1
4000	4000	3999	-1
4500	4500	4499	-1
5000	5000	5000	0
5500	5500	5500	0
5800	5800	5799	-1
6000	6000	5998	-2
6500	6500	6498	-2
7000	7000	6998	-2

Figur 10.2: Runde 1.

Runde 2			
Tast inn avstand:	Vognposisjon:	Målebånd :	Feil i mm :
3000	3000	3000	0
3500	3500	3499.5	-0,5
4000	4000	4000	0
4500	4500	4500	0
5000	5000	5000.5	0,5
5500	5500	5501	1
5800	5800	5800	0
6000	6000	6000	0
6500	6500	6500	0
7000	7000	6999	-1

Figur 10.3: Runde 2.

Runde 3			
Tast inn avstand:	Vognposisjon:	Målebånd :	Feil i mm :
3000	3000	2998	-2
3500	3500	3498	-2
4000	4000	3998	-2
4500	4500	4498.5	-1,5
5000	5000	5000	0
5500	5500	5500.5	0,5
5800	5800	5800	0
6000	6000	6001	1
6500	6500	6499	-1
7000	7000	7000	0

Figur 10.4: Runde 3.

Test rapport for T-2.2

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-2.2	T-2.2		
Kravbeskrivelse :	Systemets sensor skal automatisk konfigureres og eventuelt oppdateres ved oppstart/tilkobling.		
Testbeskrivelse :	RF får det nyeste kalibreringspolynomet ved oppstart. Ved måleavvik utenfor grenseverdi vil et nytt polynom beregnes og bli lastet opp til RF.		
Utstyr :	PC, UI, RF, Raspberry Pi 3.		
Resultat :	Ikke testet		
Kommentar :			
Testdato : *****		Testet av : ****	
Teststed : *****		Godkjent av : ******	

Tabell 10.7: Test rapport T-2.2

Test utførelse: Ble ikke utført. På grunn av opplastning til RF ikke ble gjennomført.

Test resultat:

Test rapport for T-2.3

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-2.3	T-2.3	R-1.0	Beregning
Kravbeskrivelse :	Systemet skal fullføre en kalibreringsrunde i løpet av maksimalt en time .		
Testbeskrivelse :	Systemet kjører en full kalibreringsrunde og genererer en kalibreringsrapport med RF innenfor grenseverdien.		
Utstyr :	PC, stoppeklokke, RF, kaliberingsrapport .		
Resultat :	Godkjent		
Kommentar :	Sjekke med testtabell T-1.1 , T-1.2		
Testdato : 14.05.2018		Testet av : Tuan Anh Trinh	
Teststed : Miros AS		Godkjent av : Thomas Trinh	

Tabell 10.8: Test rapport T-2.3

Test utførelse: Beregne hastelighet til kalibreringsvogn, tid det tar å monterer RF til kalibreringvogn med RS422 og spenningkilde.

Test resultat: Oppkobling tid til RF tar 10 minutter med kobling av spenningkilder og RS422 serielt kommumikasjon. Ved kjøring til ti bestemte punkter med den lengste punkt på 7000 mm og kjøre tilbake til hjemposisjon vil kalibreringsvogn klare en runde på 510 sekunder noe som tilsvarer 8 minutter og 30 sekunder.
Tilsammen lagt tid for oppkobling og nedkobling av RF pluss tiden det tar for å kjøre en oppmålingsrunde og kalibrering runde. Ti minutter x 2 = 20 minutter.
Oppmålingrunde og kalibrering runde = 17 minutter.
Totalt tid for å fullføre en kalibreringrunde med alle de nevnte prosses : 37 minutter.

10.5 Test rapport for T-3.

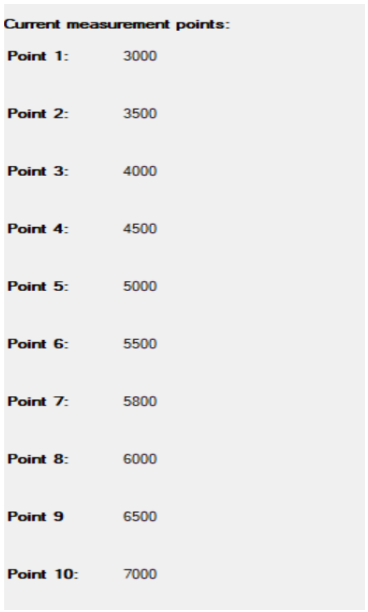
Test rapport for T-3.0

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-3.0	T-3.0	R-1.0	Fysisk
Kravbeskrivelse :	Systemet skal styres av et Windows-basert software.		
Testbeskrivelse :	Systemet blir styrt via UI, sjekk om UI kan gi referanseverdier til kalibreringsvogn.		
Utstyr :	PC, kalibreringsvogn, UI		
Resultat :	Godkjent		
Kommentar :	UI bestemmer referansepunkt og real-time verdier fra RF og kalibreringsvogn. Sjekke med testtabell T-2.0 , T-2.1		
Testdato : 13.05.2018		Testet av : Deivydas Kasokas	
Teststed : Miros AS		Godkjent av : Tuan Anh Trinh	

Tabell 10.9: Test rapport T-3-0

Test utførelse: UI sender referanser punkter verdi over til kalibreringsvogn som videre blir implentert til forskjellig distanser runder

Test resultat:



Figur 10.5: UI målepunkter.

Runde 1			
Tast inn avstand:	Vognposisjon:	Målebånd :	Feil i mm :
3000	3000	2999	-1
3500	3500	3499	-1
4000	4000	3999	-1
4500	4500	4499	-1
5000	5000	5000	0
5500	5500	5500	0
5800	5800	5799	-1
6000	6000	5998	-2
6500	6500	6498	-2
7000	7000	6998	-2

Figur 10.6: Runde 1.

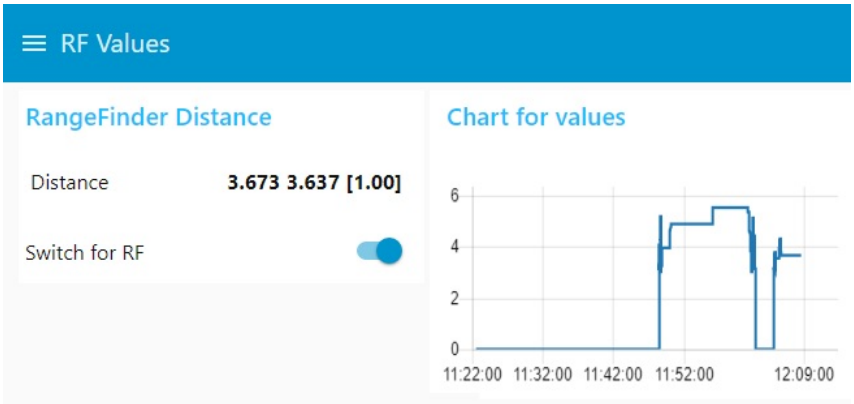
Test rapport for T-3.1

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-3.1	T-3.1		
Kravbeskrivelse :	Systemet skal vise real-time verdi fra RF og kalibreringsvogn via mobil applikasjon.		
Testbeskrivelse :	Bruk mobilapp og sjekk om real-time verdier fra RF og kalibreringsvogn har samme verdi som UI (android og IOS) .		
Utstyr :	PC, mobil (android eller IOS), kalibreringsvogn.		
Resultat :	Delvis godkjent		
Kommentar :	Kan vises via nettleser men ikke mobil applikasjon		
Test dato : 18.05.2018		Testet av : Thomas Trinh	
Test sted : USN		Godkjent av : Deivydas Kasokas	

Tabell 10.10: Test rapport T-3-1

Test utførelse: Real time verdi fra RF og kalibreringsvogn vises via internett basert webside fra node-red.

Test resultat:



Figur 10.7: IntelliSense funksjon

Test rapport for T-3.2

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-3.2	T-3.2	R-1.0	Fysisk
Kravbeskrivelse :	UI skal inneholde valgmuligheter for å starte og stoppe kalibreringsrunde.		
Testbeskrivelse :	Start og stopp funksjon i UI, aktiveres umiddelbart.		
Utstyr :	PC, kalibreringsvogn .		
Resultat :	Godkjent		
Kommentar :	UI har både en start- og stoppfunksjon implementert.		
Testdato : 13.05.2018		Testet av : Thomas Trinh	
Teststed : Miros AS		Godkjent av : Tuan Anh Trinh	

Tabell 10.11: Test rapport T-3-2

Test utførelse: Ved aktivering start funksjon fra UI vil kalibreringsvogn kjøre umiddelbart. Og ved aktivering av stopp funksjon fra UI vil kalibreringsvogn stoppe umiddelbart.

Test resultat: Kalibreringsvogn får start signal fra UI å starte med sine ti referanser punkter. Kalibreringsvogn avbryter sin kjøre runde ved UI aktiverer stopp funksjon og kalibreringsvogn stopper på samme sted uten å kjøre tilbake til hjem posisjon og operatør må dytte kalibreringsvogn tilbake til start posisjon.

Test rapport for T-3.3

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-3.3	T-3.3	R-1.0	Fysisk
Kravbeskrivelse :	UI skal inneholde valgmuligheter for å manuelt overstyre automatisk kalibreringsrunde.		
Testbeskrivelse :	Kalibreringsvogn stopper umiddelbart etter manuelt overstyring aktiveres.		
Utstyr :	PC, kalibreringsvogn.		
Resultat :	Godkjent		
Kommentar :			
Testdato : 14.05.2018		Testet av : Tuan Anh Trinh	
Teststed : Miros AS		Godkjent av : Thomas Trinh	

Tabell 10.12: Test rapport T-3-3

Test utførelse: Ved aktivering av ti referanser kjøring fuksjon har ikke operatør mulighet å overstyre kalibreringsvogn mens den er i operasjon utatt avbryte kjørerunde ved aktivering av stopp funksjon, eller aktivering av nødstopp.

Test resultat: Ved aktivering på stopp funksjon via UI vil kalibreringsvogn stopper umiddelbart å hoper ut av sin kalibrering runde. Ved aktivering av nødstop vil kutte inngangsspenning til hele systeme.

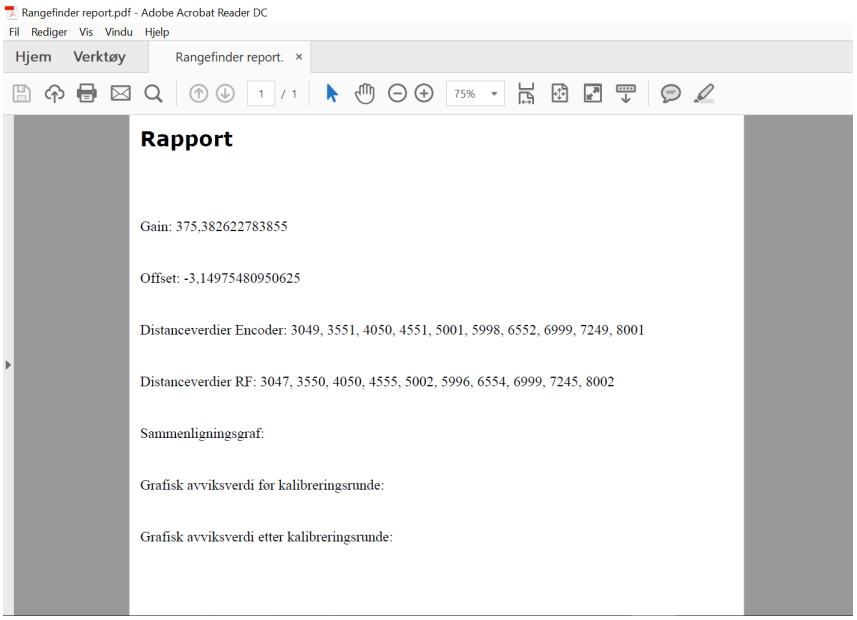
10.6 Test rapport for T-4.

Test rapport for T-4.0

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-4.0	T-4.0	R-1.0	Simulering
Kravbeskrivelse :	Systemet skal generere en rapport.		
Testbeskrivelse :	Sjekker om kalibreringsrapporten inneholder: målerundeverdier(verdi fra RF og kalibreringsvogn), RF og kalibreringsvogn sammenlikningsgraf, offset og gain, grafisk avviksverdi før og etter kalibreringsrunde.		
Utstyr :	PC.		
Resultat :	Delvis godkjent		
Kommentar :	Rapporten kan skrive ut både RF, Encoder, Gain, offsetverdier, men ikke grafisk avviksverdi før og etter kalibreringsrunden.		
Testdato : 15.05.2018		Testet av : Vetle Aasland	
Teststed : USN		Godkjent av : Vebjørn Sveva	

Tabell 10.13: Test rapport T-4.0

Test utførelse: Systemet genererer en PDF-fil som inneholder måleverdier fra både RF og Kalibreringsvognene i tillegg til de nylig kalkulerte Gain og offset verdiene, men vi rakk ikke å implementere de grafiske-avviks verdiene før og etter kalibreringsrunden og vi rakk ikke å implementere sammenlikningsgrafen for RF og Encoder.
Test resultat:

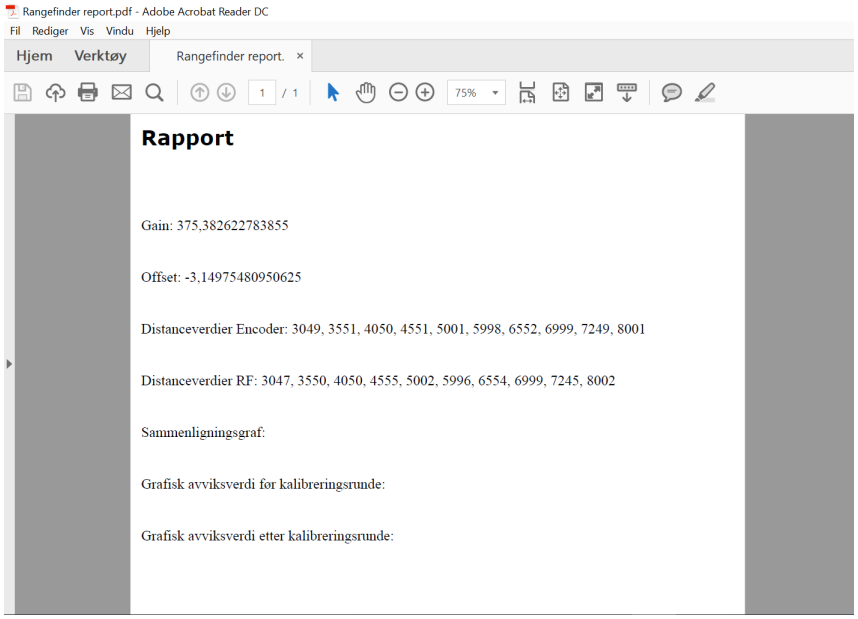


Figur 10.8: IntelliSense funksjon

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-4.1	T-4.1		
Kravbeskrivelse :	Systemet skal etter endt kjørerunde skrive ut en rapport.		
Testbeskrivelse :	Sjekker om rapporten blir skrevet ut via en printer etter endt kjørerunde.		
Utstyr :	PC, printer.		
Resultat :	Godkjent		
Kommentar :	PDF-fil blir lagret og med adobe kan du skrive den ut.		
Test dato : 18.05.2018		Testet av : Vetle Aasland	
Test sted : USN		Godkjent av : Vebjørn Sveva	

Tabell 10.14: Test rapport T-4.1

Test utførelse: PDF-filen generings koden er plassert sekvensielt i koden slik at den utføres etter at en måling har blitt utført. Etter at målingen er utført vil resultatene fra RF og Encoder bli sammenlignet og hvis de ligger utenfor grenseverdien satt av Miros, vil de verdien bli sendt inn i kalibreringskoden hvor de blir gain og offset korrigert og deretter settes verdiene inn i rapporten. For å skrive ut rapport dokumentet brukes Adobe for å skrive ut dokumentet.
Test resultat:

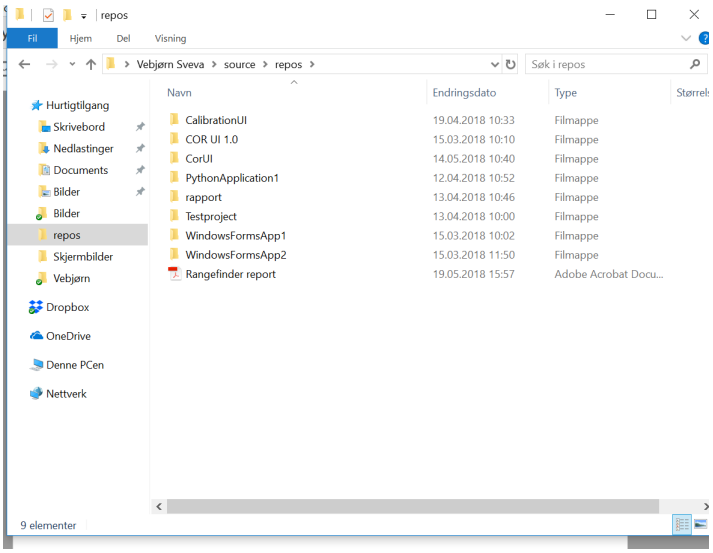


Figur 10.9: IntelliSense funksjon

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-4.2	T-4.2		
Kravbeskrivelse :	Rapporten skal lagres lokalt og på MS-Azure.		
Testbeskrivelse :	Sjekker om rapporten blir lagret lokalt og på MS-Azure.		
Utstyr :	PC, MS-Azure.		
Resultat :	Delvis godkjent		
Kommentar :	Rapporten lagres lokalt.		
Testdato : 18.05.2018		Testet av : Vetle Aasland	
Teststed : USN		Godkjent av : Vebjørn Sveva	

Tabell 10.15: Test rapport T-4.2

Test utførelse: Ved å bruke PDFsharp biblioteket fikk vi muligheten til å lage en kode som lagrer rapport-dokumentet i den valgte filbanen vi ønsker å plassere Dokumentet i.
Test resultat:



Figur 10.10: IntelliSense funksjon

Test rapport for T-4.3

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-4.3	T-4.3		
Kravbeskrivelse :	Rapporten skal inneholde en tidsmåling av hvert referansepunkt .		
Testbeskrivelse :	Sjekker om kalibreringsrapporten innholder tidsverdi for hver referansepunktsmåling.		
Utstyr :	PC, kalibreringsrapport.		
Resultat :	Ikke godkjent		
Kommentar :	Har ikke funksjon for tidsmåling i UI.		
Testdato : 18.05.2018		Testet av : Vetle Aasland	
Teststed : USN		Godkjent av : Vebjørn Sveva	

Tabell 10.16: Test rapport T-4.3

Test utførelse: Vi valgte å ikke implementer denne funksjonen. Vi nærmet oss tidsfristen og hadde mye som måtte fullføres. Og å implementer denne funksjonen vil ikke ha noen stor på virkning på selve kalibreringsprosessen.
Test resultat:

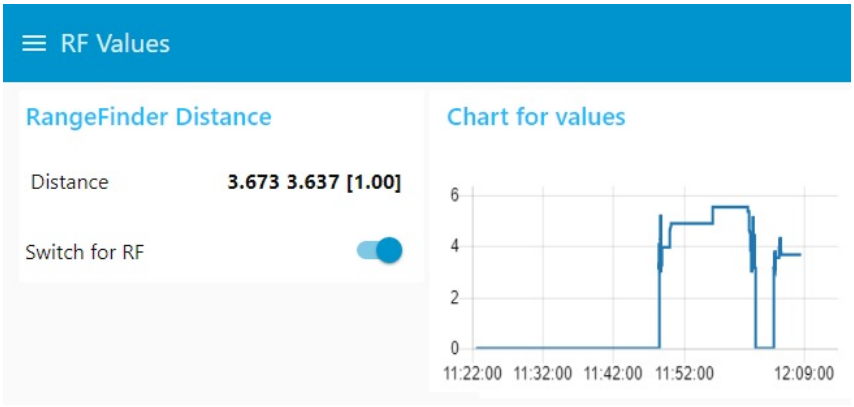
10.7 Test rapport for T-5.

Test rapport for T-5.0

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-5.0	T-5.0	R-1.0	Fysisk
Kravbeskrivelse :	Rangefinderen skal benytte toveiskommunikasjon.		
Testbeskrivelse :	Sjekker om RF sender og mottar data fra Raspberry Pi 3.		
Utstyr :	PC, RF, UI.		
Resultat :	Delvis godkjent		
Kommentar :	RF kan bare sende data til Raspberry Pi 3.		
Testdato : 13.05.2018		Testet av : Deivy Kazokas	
Teststed : Miros AS		Godkjent av : Thomas Trinh	

Tabell 10.17: Test rapport T-5.0

Test utførelse: RF kan sende data til RPi, men kan ikke motta. Dette førte til at kravet er delvis godkjent.
Test resultat:



Figur 10.11: IntelliSense funksjon

Test rapport for T-5.1

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-5.1	T-5.1	R-0.0	Fysisk
Kravbeskrivelse :	Kommunikasjonsmodulen skal benytte RS232 tilkobling i RF.		
Testbeskrivelse :	Kommunikasjonsmodulen sender og mottar data fra RF via RS232 kommunikasjonsprotokoll.		
Utstyr :	PC, RF, UI, Raspberry Pi 3.		
Resultat :	Ikke godkjent		
Kommentar :			
Test dato : *****		Testet av : ********	
Test sted : *****		Godkjent av : ******	

Tabell 10.18: Test rapport T-5.1

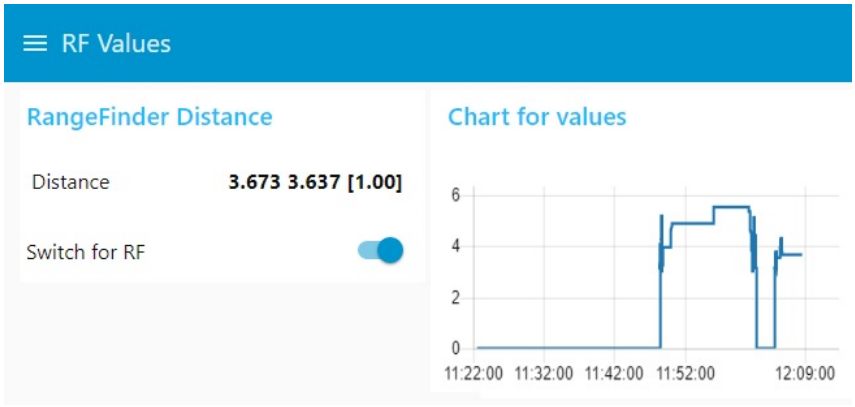
Test utførelse: Testen ble ikke utført, dermed er T-5.1 ikke godkjent. Grunnen til at vi ikke fikk testet dette er at vi fikk aldri den RS232-komponenten som vi bestilte.
Test resultat:

Test rapport for T-5.2

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-5.2	T-5.2	R-1.0	Fysisk
Kravbeskrivelse :	Kommunikasjonsmodulen skal benytte RS422 tilkobling i RF.		
Testbeskrivelse :	Kommunikasjonsmodulen sender og mottar data fra RF via RS422 kommunikasjonsprotokoll.		
Utstyr :	PC, RF, UI, Raspberry Pi 3.		
Resultat :	Delvis godkjent		
Kommentar :	RS422 kan bare hente data fra RF(Sm-140).		
Testdato : 13.05.2018		Testet av : Thomas Trinh, Deivydas Kazokas	
Teststed : Miros AS		Godkjent av : Tuan Anh Trinh	

Tabell 10.19: Test rapport T-5.2

Test utførelse: Testen er delvis godkjent, RS422 kan hente data fra RF, men ikke sende data til RF. Dette er fordi vi ikke fikk nok tid til å utføre det pga tidsklemmen vi hadde.
Test resultat:



Figur 10.12: IntelliSense funksjon

10.8 Test rapport for T-6.

Test rapport for T-6.0

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-6.0	T-6.0	R-1.0	Simulering/Fysisk
Kravbeskrivelse :	Software og elektro-mekaniske løsninger skal være modulbaserte .		
Testbeskrivelse :	Sjekke om software er mulig å modulbasere eller har mulighet for implementering i annen software. Elektromekaniske komponenter har mulighet for utskifting.		
Utstyr :	Skrutrekker, PC, UI .		
Resultat :	Godkjent		
Kommentar :	UI og kalibreringsvognskode er modulbasert, RPi kan byttes ut.		
Testdato : 11.05.2018		Testet av : Tuan Anh Trinh	
Teststed : Miros AS		Godkjent av : Thomas Trinh	

Tabell 10.20: Test rapport T-6.0

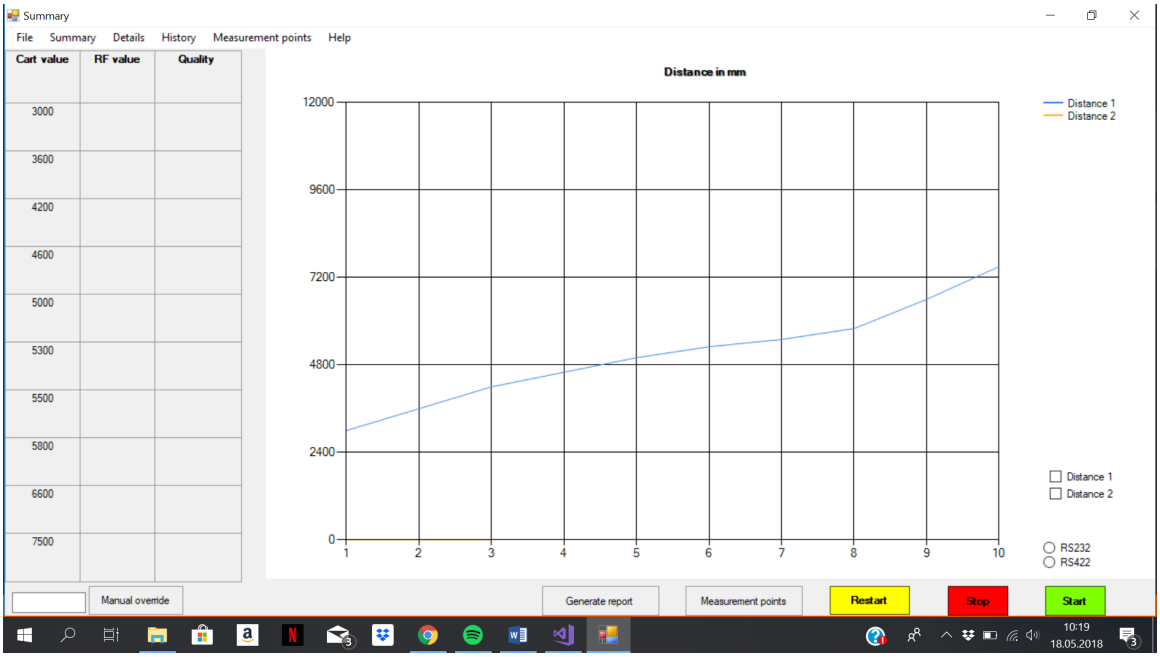
Test utførelse: UI kan byttes ut med en web-UI. Andre elektro-mekansike komponenter som implementeres i RPi, nye funksjoner kan skrives på Python skriptet. RPi kan byttes ut med en arduino hvis det er ønskelig.
Test resultat: Lett avmontering av elektroniske komponenter og UI er lett tilgjengelig for utbytting.

Test rapport for T-6.1

Krav ID :	Test ID:	Revisjons ID :	Test type :
K-6.1	T-6.1	R-1.0	Fysisk
Kravbeskrivelse :	Systemet skal utenom kalibreringsrunde kunne styres manuelt til gitte punkter.		
Testbeskrivelse :	Kalibreringsvognen kan manuelt styres med fysisk kraft eller UI.		
Utstyr :	DC motor, kalibreringsvogn.		
Resultat :	Godkjent		
Kommentar :	Kalibreringsvognen kan styres ved fysisk kraft og med UI.		
Testdato : 13.05.2018		Testet av : Thomas Trinh	
Teststed : Miros AS		Godkjent av : Tuan Trinh	

Tabell 10.21: Test rapport T-6.1

Test utførelse: Inni på UI finnes det en knapp der systemet kan manuelt styres til et punkt. Og motor kan bli slått av slik at det er mulig å flytte fysisk til det punktet som er ønsket.
Test resultat:



Figur 10.13: IntelliSense funksjon

11 REFERANSER

[1] *AGM batterier*. URL: https://www.solar-electric.com/learning-center/batteries-and-charging/agm-battery-technology.html. date accessed: 08.03.2018.

[2] *azure*. URL: https://azure.microsoft.com/nb-no/. date accessed: 12.04.2018.

[3] *Batteri informasjon*. URL: http://batteryuniversity.com/learn/. date accessed: 07.03.2018.

[4] *Batteri PDF - sammenligning*. URL: http://www.farnell.com/datasheets/1754479.pdf. date accessed: 12.03.2018.

[5] *Batteri PDF - sammenligning*. URL: https://hobbyking.com/media/file/172827882X365809X20.pdf. date accessed: 13.03.2018.

[6] *Battery management system*. URL: http://www.electronicdesign.com/power/look-inside-battery-management-systems. date accessed: 14.03.2018.

[7] *ClearPath motor*. URL: https://www.teknic.com/products/clearpath-brushless-dc-servo-motors/. date accessed: 14.03.2018.

[8] *ClearPath motor*. URL: https://www.teknic.com/model-info/CPM-MCPV-3432S-RLN/. date accessed: 14.03.2018.

[9] *connet*. URL: https://www.youtube.com/watch?v=V9r-Gp3uNCE. date accessed: 12.04.2018.

[10] *C-rapport*. URL: http://csharp.net-informations.com/file/create-pdf.htm. date accessed: 12.04.2018.

[11] *CS*. URL: https://en.wikipedia.org/wiki/C%5C_Sharp%5C_(programming%5C_language). date accessed: 12.04.2018.

[12] *Device-oppsett*. URL: https://catalog.azureiotsolutions.com/docs?title=Azure/azure-iot-device-ecosystem/manage_iot_hub. date accessed: 22.04.2018.

[13] *Embedded Systems*. URL: https://www.usn.no/studier/finn-studier/ingenior-sivilingenior-teknologi-og-it/bachelor-i-ingeniorfag-dataingenior/. dato: 25.01.2018.

[14] *Graf*. URL: https://www.syncfusion.com/kb/7683/how-to-bind-chart-with-data-from-sql-database. date accessed: 22.04.2018.

[15] *Graphene batterier*. URL: https://www.graphene-info.com/graphene-batteries. date accessed: 09.03.2018.

[16] *HAT-RS422*. URL: https://www.hwhardsoft.de/english/projects/rs485-shield/. date accessed: 20.04.2018.

[17] *IOT*. URL: https://internetofthingsagenda.techtarget.com/definition/IoT-device. date accessed: 10.04.2018.

[18] *JSON*. URL: https://www.json.org/. date accessed: 12.04.2018.

[19] *Kybernetikk og mekatronikk*. URL: https://www.usn.no/studier/finn-studier/ingenior-sivilingenior-teknologi-og-it/bachelor-i-ingeniorfag-elektroingenior/. dato: 25.01.2018.

[20] *Litium batterier*. URL: https://no.globtek.com/lithium-ion-battery-packs/. date accessed: 07.03.2018.

[21] *MaaleBand*. URL: http://www.hultafors.no/arkiv/maleband-med-presisjon/. date accessed: 15.05.2018.

[22] *Miros AS*. URL: https://miros-group.com. dato: 25.01.2018.

[23] *MQTT*. URL: https://www.hivemq.com/blog/mqtt-client-library-encyclopedia-m2mqtt. date accessed: 22.04.2018.

[24] *mqtt*. URL: https://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport. date accessed: 11.04.2018.

[25] *Nikkelbaserte batterier*. URL: http://batteryuniversity.com/learn/article/nickel_based_batteries. date accessed: 08.03.2018.

[26] *node-red*. URL: https://nodered.org/. date accessed: 12.04.2018.

[27] *node-red-install*. URL: https://nodered.org/docs/hardware/raspberrypi. date accessed: 12.04.2018.

[28] *node-red-UI*. URL: https://nodered.org/docs/security#usernamepassword-based-authentication. date accessed: 12.04.2018.

[29] Greta Hjertø NTNU. *Visjonsdokument*. URL: https://www.ntnu.no/iie/fag/maler-standarder/UP/Visjonsdokumentet_intro.htm. dato: 01.02.2018.

[30] *Nye batteriteknologier*. URL: https://www.pocket-lint.com/gadgets/news/130380-future-batteries-coming-soon-charge-in-seconds-last-months-and-power-over-the-air. date accessed: 09.03.2018.

[31] *Prieto batterier*. URL: https://www.prietobattery.com/how-it-works-2/chemistry/. date accessed: 09.03.2018.

[32] *Proton flow batterier*. URL: https://newatlas.com/proton-flow-battery-hydrogen-electricity-rmit/30818/. date accessed: 10.03.2018.

[33] *PuTTy*. URL: https://www.putty.org/. date accessed: 14.04.2018.

[34] *PWM*. URL: https://www.youtube.com/watch?v=9tActipVqIM&t=1984s. date accessed: 14.03.2018.

[35] RangerFinder. *SM-140*. URL: https://miros-group.com/miros-sm-140/. dato: 25.01.2018.

[36] *RpiUART*. URL: https://www.raspberrypi.org/documentation/configuration/uart.md. date accessed: 18.04.2018.

[37] *RS422-bilde*. URL: https://image.jimcdn.com/app/cms/image/transf/none/path/sae5d500d7e085f42/image/i5104b16205fc69e9/version/1516094301/image.jpg. date accessed: 18.04.2018.

[38] *Servo motor*. URL: http://www.electricaleasy.com/2015/01/how-does-servo-motor-work.html. date accessed: 14.03.2018.

[39] *servo motor*. URL: https://www.electronicshub.org/servo-motors/. date accessed: 19.03.2018.

[40] *Servo motor data blad*. URL: http://www.dtc.no/files/Produktkataloger%20andre/Datablad%20PM-motor%20EC180%20DT-b.pdf. date accessed: 14.03.2018.

[41] *SFTP*. URL: https://www.nsoftware.com/netdrive/sftp/. date accessed: 16.04.2018.

[42] KEN LUNN SIMON BENNETT JOHN SKELTON. *Schaum's Outlines UML Second Edition*. McGrwHill, 2004.

[43] *SQL-tekst*. URL: http://www.sqlservercentral.com/articles/Export/147145/. date accessed: 12.04.2018.

[44] *Stepper data blad*. URL: https://www.elfadistrelec.no/en/hybrid-stepper-motor-nm-trinamic-qsh8618-95-55-700/p/30100211?q=motor&filter_categoryCodePathROOT%2Fcat-L1D_379516=cat-L2D_379617&filter_categoryCodePathROOT%2Fcat-L1D_379516%2Fcat-L2D_379617=cat-L3D_525513&filter_categoryCodePathROOT%2Fcat-L1D_379516%2Fcat-L2D_379617%2Fcat-L3D_525513=cat-DNAV_PL_12101201&filter_categoryCodePathROOT=cat-L1D_379516&page=1&origPos=3&origPageSize=25&simi=99.65. date accessed: 14.03.2018.

[45] *Stepper motor*. URL: https://learn.adafruit.com/all-about-stepper-motors/what-is-a-stepper-motor. date accessed: 14.03.2018.

[46] *stepper motor*. URL: http://www.bristolwatch.com/arduino/arduino_unipolar_stepper.htm. date accessed: 19.03.2018.

[47] *UART*. URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0183g/index.html. date accessed: 18.04.2018.

[48] *Ultra sonic sensor*. URL: https://www.elfadistrelec.no/Web/WebShopImages/landscape_large/82/0f/30036820f.jpg. date accessed: 14.03.2018.

[49] *Unified prosess*. URL: http://bawiki.com/wiki/concepts/sdlc-process-models/unified-process/. dato: 06.02.2018.

[50] *Unified prosess*. URL: https://upload.wikimedia.org/wikiversity/en/a/aa/RationalUnifiedProcess.png. dato: 06.02.2018.

[51] *VS code*. URL: https://code.visualstudio.com/. date accessed: 16.04.2018.

[52] *W10*. URL: https://www.techopedia.com/definition/24300/windows-forms-net. date accessed: 12.04.2018.

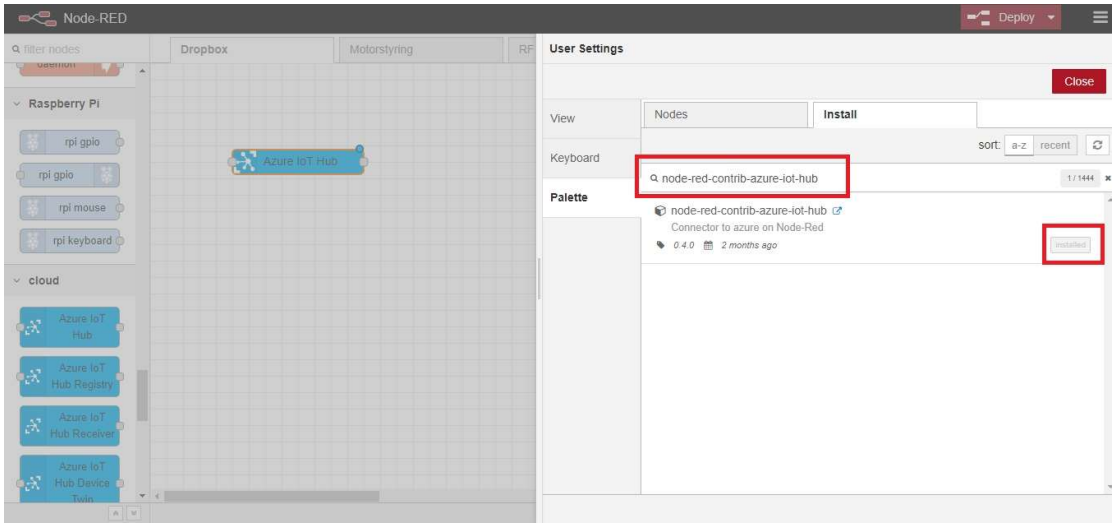
FIGURER

TABELLER

Vedlegg A

Nedlastning av Azure IoT Hub node.

Ved å trykke «Ctrl+Shift+P» i Node-RED vindu kommer brukeren til «Palette Manager». På installasjonsfane skal det søkes etter «node-red-contrib-azure-iot-hub» slik det er vist på fig. Install Azure og trykkes «install».

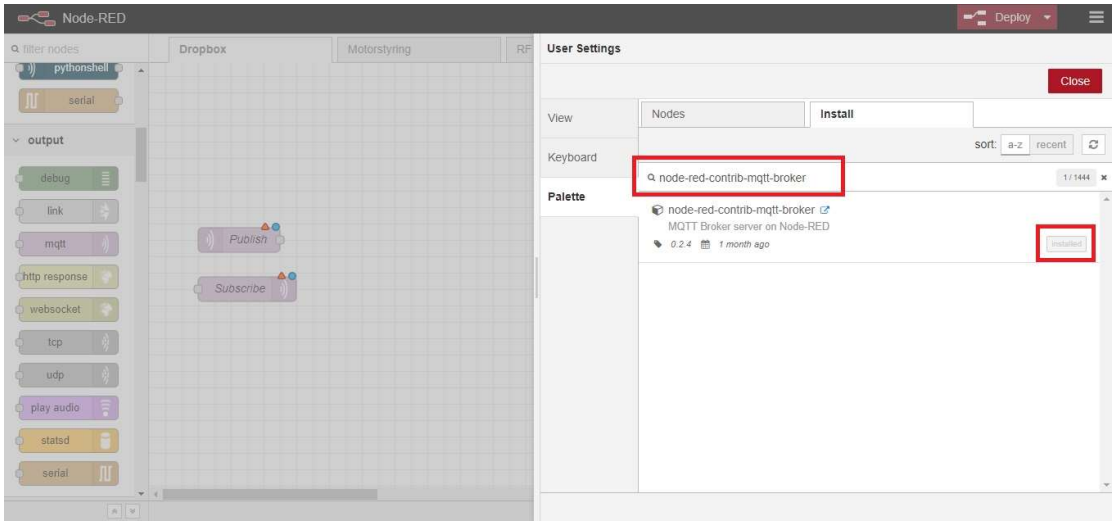


Figur Install Azure – Installasjon av Azure IoT Hub node

Vedlegg B

Nedlastning av MQTT node.

Ved å trykke «Ctrl+Shift+P» i Node-RED vindu kommer brukeren til «Palette Manager». På installasjonsfane skal det søkes etter «node-red-contrib-mqtt-broker» slik det er vist på fig. install MQTT og trykkes «install».

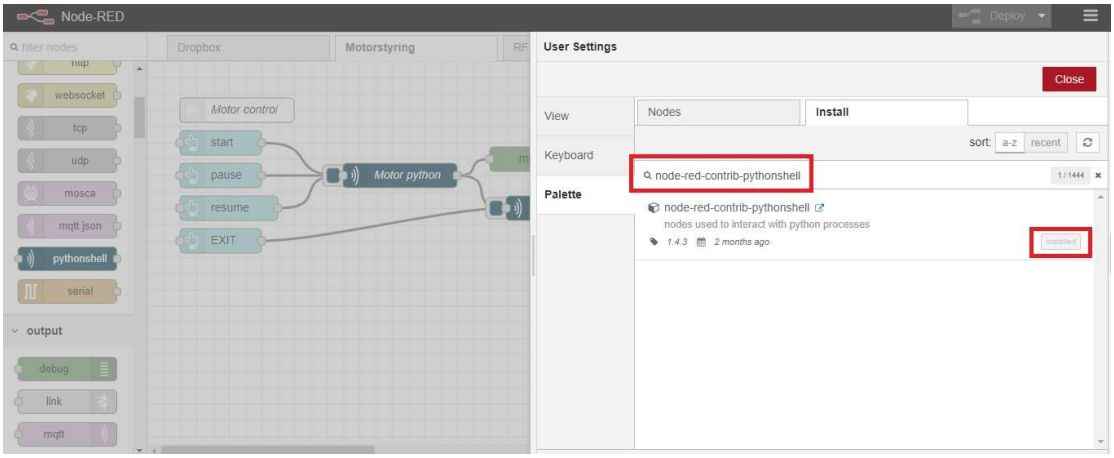


Figur Install MQTT – Installasjon av MQTT node

Vedlegg C

Nedlastning av PythonShell node.

Ved å trykke «Ctrl+Shift+P» i Node-RED vindu kommer brukeren til «Palette Manager». På installasjonsfane skal det søkes etter «node-red-contrib-pythonshell» slik det er vist på fig. install PythonShell og trykkes «install».

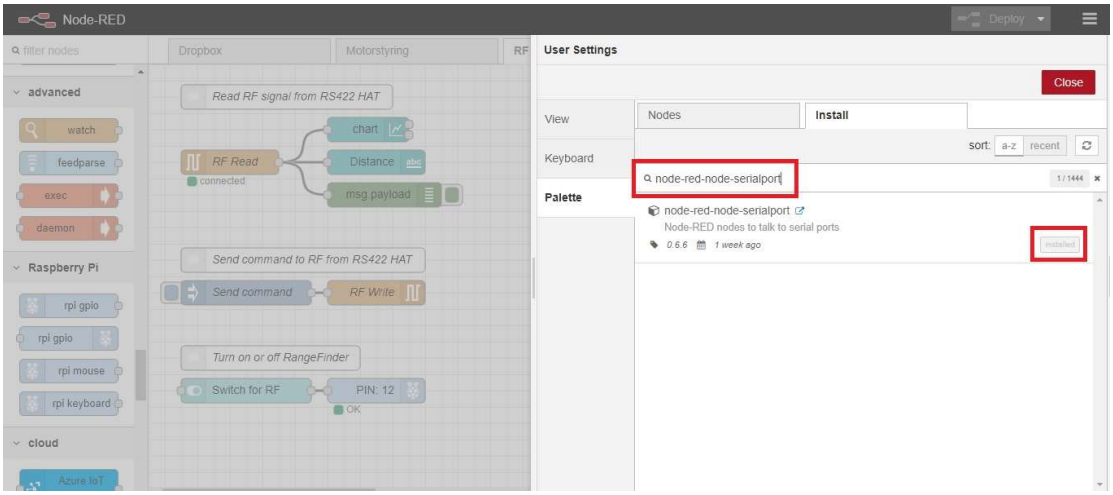


Figur install PythonShell – Installasjon av PythonShell node

Vedlegg D

Nedlastning av Serial node.

Ved å trykke «Ctrl+Shift+P» i Node-RED vindu kommer brukeren til «Palette Manager». På installasjonsfane skal det søkes etter «node-red-node-serialport» slik det er vist på fig. Install Serial og trykkes «install».

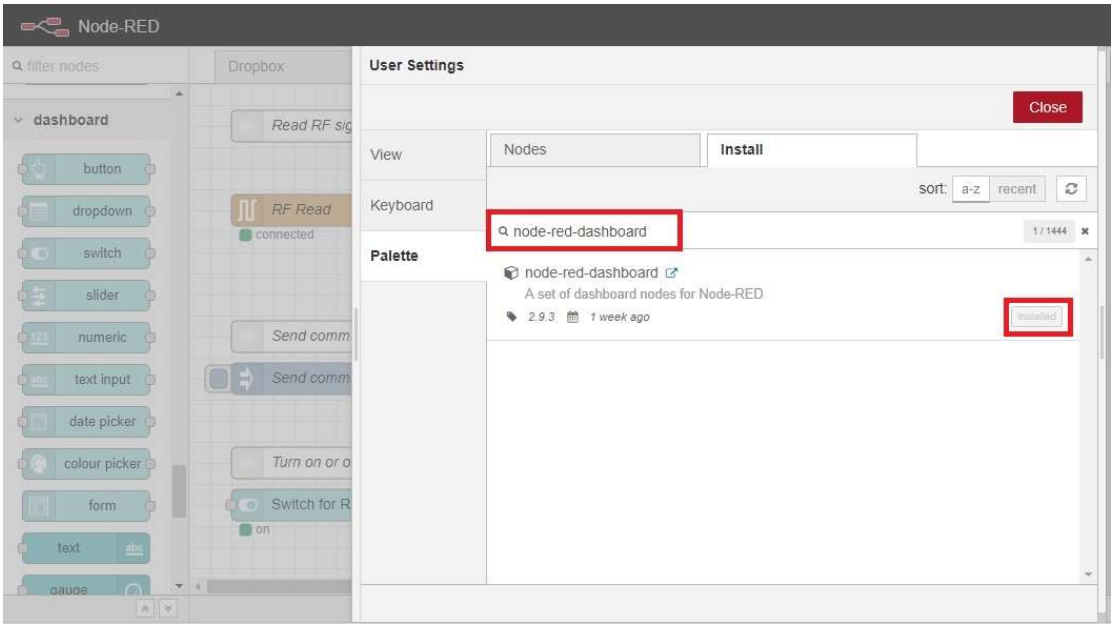


Figur install Serial – Installasjon av Serial node.

Vedlegg E

Nedlastning av Dropbox node.

Ved å trykke «Ctrl+Shift+P» i Node-RED vindu kommer brukeren til «Palette Manager». På installasjonsfane skal det søkes etter «node-red-node-dropbox» slik det er vist på fig. install Dropbox og trykkes «install».

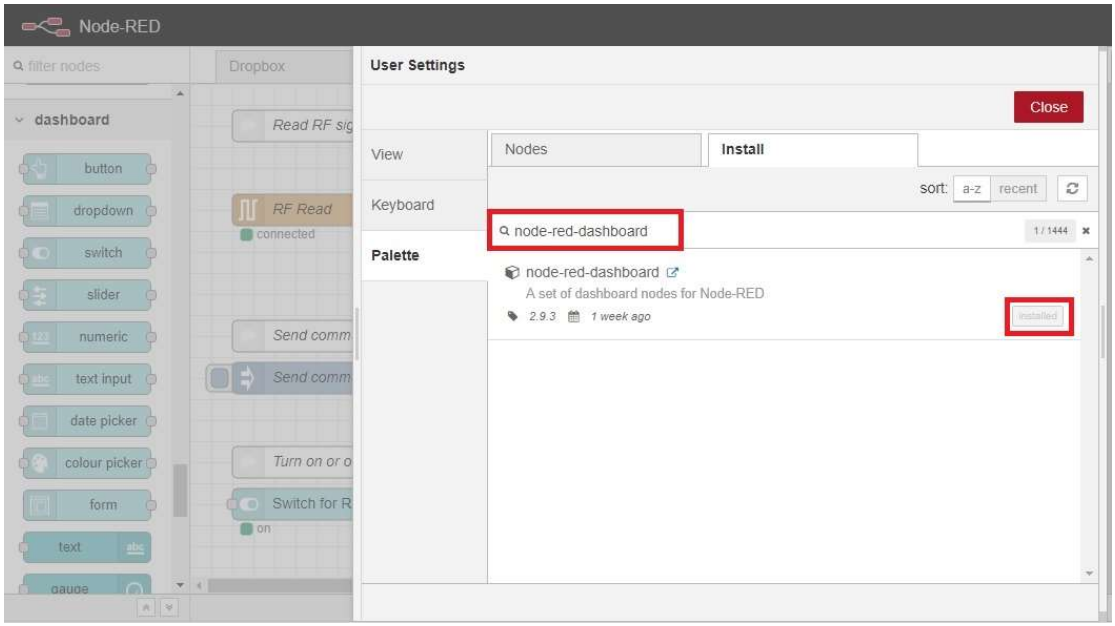


Figur install DropBox – Installasjon av Dropbox node

Vedlegg F

Nedlastning av grafisk brukergrensesnitt node.

Ved å trykke «Ctrl+Shift+P» i Node-RED vindu kommer brukeren til «Palette Manager». På installasjonsfane skal det søkes etter «node-red-dashboard» slik det er vist på fig. install GUI og trykkes «install».



Figur install GUI – Installasjon av grafisk brukergrensesnitt node

Vedlegg G

```
from time import sleep
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
from time import sleep
GPIO.setwarnings(False)
import json
# General value
End = 0.2 # The distand to stop or break the engine
tid = 8 # how many secound it take to measuring SM-140 range
Home = 867 # Home posisjon from SM-140 to reflect wall
WallToRF = 857 # Distance between SM-140 too reflect wall in mm
EncoderEnd = 540 # Encoder puls when proxi reset
# For Encoder input and value
InA = 36 # Encoder InA input from rasberry pi 3
InB = 35 # Encoder INB input from rasberry pi 3
antall = 0
counter = 0 # Encoder puls counter when wagon is moving
lengde = 0.106195485 # Puls revelosjon for every encoder counting from 34.5mm in wheel
radius and
2048 ppr
avstand1 = 0 # Distanse from encoder counting
# For Proximity input and value
proxi = 37
avstand2 = 0
avstand = 0
antallProxi = 0
proxypa = 0
proxyav = 0
proxiValue = 500 # Proximity measuring dispance
# For Motor input
Enable = 11 # BLU-wire ON/OFF
InputA = 13 # WHT-wire Forward/Backward
InputB = 15 # BLK-wire Configurate PWM value
# Input setup
GPIO.setup(Enable, GPIO.OUT) # Enable input
GPIO.setup(InputA, GPIO.OUT) # InputA input
GPIO.setup(InputB, GPIO.OUT) # InputB input
pwm = GPIO.PWM(InputB, 100) # Access PWM signals
pwm.start(0) # PWM starts at 0
file = open('Dis.json')
distanser = json.loads(file.read())
# For input distanscer
Dis1 = ((distanser['Dis1']))
Dis2 = ((distanser['Dis2']))
Dis3 = ((distanser['Dis3']))
Dis4 = ((distanser['Dis4']))
Dis5 = ((distanser['Dis5']))
Dis6 = ((distanser['Dis6']))
```

```
Dis7 = ((distanser['Dis7']))
Dis8 = ((distanser['Dis8']))
Dis9 = ((distanser['Dis9']))
Dis10 = ((distanser['Dis10']))
# GPIO.pin setup
GPIO.setup(proxi, GPIO.IN)
GPIO.setup(InA, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(InB, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
clkLastState = GPIO.input(InA)
# Function for forward rotation
def forward() :
GPIO.output(Enable, GPIO.HIGH)
GPIO.output(InputA, GPIO.LOW)
GPIO.output(InputB, GPIO.HIGH)
# Function for backward rotation
def backward() :
GPIO.output(Enable, GPIO.HIGH)
GPIO.output(InputA, GPIO.HIGH)
GPIO.output(InputB, GPIO.HIGH)
# Function for stop the engine
def stop() :
GPIO.output(Enable, GPIO.LOW)
GPIO.output(InputA, GPIO.LOW)
GPIO.output(InputB, GPIO.LOW)
# Code for first distance only forward
while avstand < Dis1 :
# For encoder
# Distance left from distance input to distance travel.
clkState = GPIO.input(InA) # Define input InA from Encoder for forward rotation
dtState = GPIO.input(InB) # Define input InB from Encoder for backward rotation
# Code taken from Github
# https://github.com/modmypi/Rotary-Encoder/blob/master/rotary_encoder.py
#-------------------------------------------------------------------------------------------------------#
if clkState != clkLastState: # If encoder value is not same as previous encoder value run
encoder measuring prosses
if dtState != clkState:
counter +=1 # By forward rotation encoder value pluss one counter
else:
counter -=1 # By backward rotation encoder value minus one counter
#print (counter)
avstand1 = counter * lengde # Distance measuring value from encoder counting
multiply by puls revelution
avstand = avstand1 + avstand2 + WallToRF # Distance measuring value combi from
proximity
and encoder counting and start posison start at 900mm
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen1) )
clkLastState = clkState
#-------------------------------------------------------------------------------------------------------#
# For Proximity
if GPIO.input(proxi)==1 :
```

```
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2): # Proxi counter one by every objekt passing. Counter only
pluss when proxi go from off to on to off again
antallProxi += 1 # Forward proxi pluss one proxi counter for every passing
proxyav = 0 # Proxi off value to 0
proxypa = 0 # Proxi off value to 0
counter = EncoderEnd
# Reset encoder counter to 0
avstand2 = antallProxi * proxiValue # Distance measuring value from Proximity
#print (" Avstand 2 i mm: ",avstand2)
AvstandIgjen1 = Dis1 - avstand
if avstand < Dis1 :
forward()
if AvstandIgjen1 <= End :
stop()
print ("Runde 1 ferdig")
break
sleep(tid)
# For distance 2 forward
while avstand < Dis2 :
# For encoder
AvstandIgjen2 = Dis2 - avstand
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen2) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi += 1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand < Dis2 :
```

```
forward()
if avstand > Dis2 :
backward()
if AvstandIgjen2 <= End :
stop()
print ("Runde 2 ferdig" + str( avstand))
break
sleep(tid)
# For distance 2 backward
while avstand > Dis2 :
# For encoder
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen2) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi = antallProxi -1 # Backward proxi minus one proxi counter for every passing
proxyav = 0
proxypa = 0
counter = - EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
AvstandIgjen2 = avstand - Dis2
if avstand > Dis2 :
backward()
if AvstandIgjen2 <= End :
stop()
print ("Runde 2 ferdig" + str( avstand))
break
sleep(tid)
# For distance 3 forward
while avstand < Dis3 :
# For encoder
AvstandIgjen3 = Dis3 - avstand
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
```

```
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen3) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi += 1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand < Dis3 :
forward()
if avstand > Dis3 :
backward()
if AvstandIgjen3 <= End :
stop()
print ("Runde 3 ferdig")
break
sleep(tid)
# For distance 3 backward
while avstand > Dis3 :
# For encoder
AvstandIgjen3 = avstand - Dis3
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen3) )
clkLastState = clkState
# For Proximity
```

```
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi = antallProxi -1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand > Dis3 :
backward()
if avstand < Dis3 :
forward()
if AvstandIgjen3 <= End :
stop()
print ("Runde 3 ferdig")
break
sleep(tid)
# For distance 4 forward
while avstand < Dis4 :
# For encoder
AvstandIgjen4 = Dis4 - avstand
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen4) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi += 1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand < Dis4 :
```

```
forward()
if avstand > Dis4 :
backward()
if AvstandIgjen4 <= End :
stop()
print ("Runde 4 ferdig")
break
sleep(tid)
# For distance 4 backward
while avstand > Dis4 :
# For encoder
AvstandIgjen4 = avstand - Dis4
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen4) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi = antallProxi -1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand > Dis4 :
backward()
if avstand < Dis4 :
forward()
if AvstandIgjen4 <= End :
stop()
print ("Runde 4 ferdig")
break
sleep(tid)
# For distance 5 forward
while avstand < Dis5 :
# For encoder
AvstandIgjen5 = Dis5 - avstand
```

```
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen5) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi += 1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand < Dis5 :
forward()
if avstand > Dis5 :
backward()
if AvstandIgjen5 <= End :
stop()
print("Runde 5 ferdig")
break
sleep(tid)
# For distance 5 backward
while avstand > Dis5 :
# For encoder
AvstandIgjen5 = avstand - Dis5
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen5) )
```

```
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi = antallProxi -1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand > Dis5 :
backward()
if avstand < Dis5 :
forward()
if AvstandIgjen5 <= End :
stop()
print ("Runde 5 ferdig")
break
sleep(tid)
# For distance 6 forward
while avstand < Dis6 :
# For encoder
AvstandIgjen6 = Dis6 - avstand
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen6) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi += 1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
```

```
#print (" Avstand 2 i mm: ",avstand2)
if avstand < Dis6 :
forward()
if AvstandIgjen6 < 0.2 :
stop()
print ("Runde 6 ferdig")
break
sleep(tid)
# For distance 6 backward
while avstand > Dis6 :
# For encoder
AvstandIgjen6 = avstand - Dis6
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen6) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi = antallProxi -1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand > Dis6 :
backward()
if avstand < Dis6 :
forward()
if avstand > Dis6 :
backward()
if AvstandIgjen6 <= End :
stop()
print ("Runde 6 ferdig")
break
sleep(tid)
# For distance 7 forward
while avstand < Dis7 :
```

```
# For encoder
AvstandIgjen7 = Dis7 - avstand
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen7) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi += 1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand < Dis7 :
forward()
if avstand > Dis7 :
backward()
if AvstandIgjen7 <= End :
stop()
print ("Runde 7 ferdig")
break
sleep(tid)
# For distance 7 backward
while avstand > Dis7 :
# For encoder
AvstandIgjen7 = avstand - Dis7
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
```

```
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen7) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi = antallProxi -1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand > Dis7 :
backward()
if avstand < Dis7 :
forward()
if AvstandIgjen7 <= End :
stop()
print ("Runde 7 ferdig")
break
sleep(tid)
# For distance 8 forward
while avstand < Dis8 :
# For encoder
AvstandIgjen8 = Dis8 - avstand
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen8) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi += 1
proxyav = 0
proxypa = 0
```

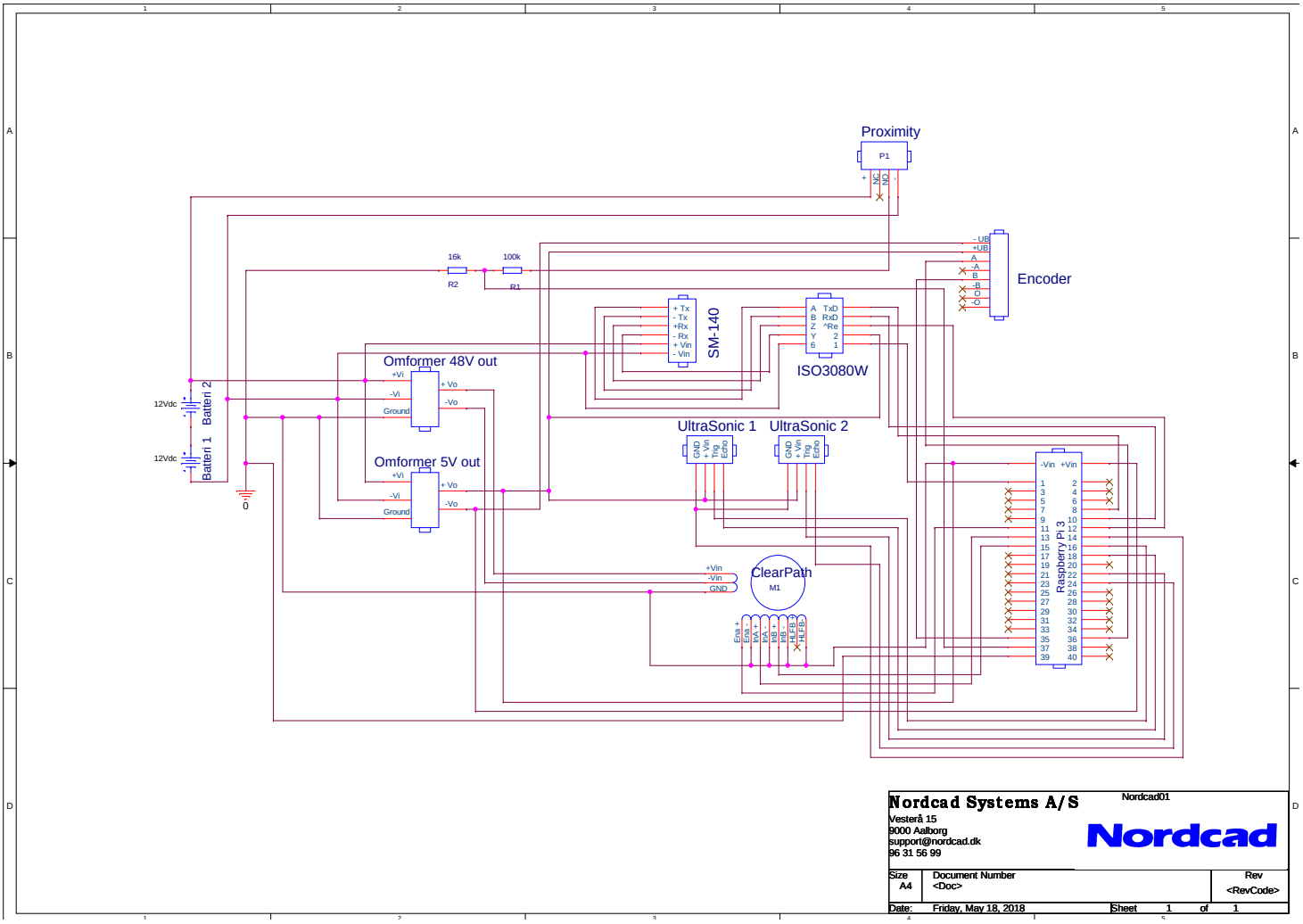
```
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand < Dis8 :
forward()
if AvstandIgjen8 < End :
stop()
print ("Runde 8 ferdig")
break
sleep(tid)
# For distance 8 backward
while avstand > Dis8 :
# For encoder
AvstandIgjen8 = avstand - Dis8
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen8) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi = antallProxi -1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand > Dis8 :
backward()
if avstand < Dis8 :
forward()
if AvstandIgjen8 < End :
stop()
print ("Runde 8 ferdig")
break
sleep(tid)
# For distance 9 forward
while avstand < Dis9 :
```

```
# For encoder
AvstandIgjen9 = Dis9 - avstand
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen9) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi += 1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand < Dis9 :
forward()
if AvstandIgjen9 <= End :
stop()
print ("Runde 9 ferdig")
break
sleep(tid)
# For distance 9 backward
while avstand > Dis9 :
# For encoder
AvstandIgjen9 = avstand - Dis9
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen9) )
```

```
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi = antallProxi -1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand > Dis9 :
backward()
if AvstandIgjen9 <= End :
stop()
print ("Runde 9 ferdig")
break
sleep(tid)
# For distance 10 forward
while avstand < Dis10 :
# For encoder
AvstandIgjen10 = Dis10 - avstand
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen10) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi += 1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand < Dis10 :
```

```
forward()
if AvstandIgjen10 <= End :
stop()
print ("Runde 10 ferdig")
break
sleep(tid)
# For distance 10 backward
while avstand > Dis10 :
# For encoder
AvstandIgjen10 = avstand - Dis10
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen10) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi = antallProxi -1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand > Dis10 :
backward()
if AvstandIgjen10 <= End :
stop()
print ("Runde 10 ferdig")
sleep(tid)
# The wagon move to home posisjon
while avstand > Home :
# For encoder
HomeIgjen = avstand - Home
clkState = GPIO.input(InA)
dtState = GPIO.input(InB)
#print(str(clkLastState) + str(dtState))
if clkState != clkLastState:
if dtState != clkState:
```

```
counter +=1
else:
counter -=1
#print (counter)
avstand1 = counter * lengde
avstand = avstand1 + avstand2 + WallToRF
print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(HomeIgjen) )
clkLastState = clkState
# For Proximity
if GPIO.input(proxi)==1 :
proxypa = 1
else:
proxyav = 1
if (proxyav + proxypa == 2):
antallProxi = antallProxi -1
proxyav = 0
proxypa = 0
counter = EncoderEnd
avstand2 = antallProxi * proxiValue
#print (" Avstand 2 i mm: ",avstand2)
if avstand > Home :
backward()
if avstand < Home :
forward()
if HomeIgjen <= End :
stop()
print ("Tilbake til stasjon ")
GPIO.cleanup ()
```



Nordcad Systems A/S
Vesterå 15
9000 Aalborg
support@nordcad.dk
96 31 56 99

Nordcad01

Nordcad

Size	Document Number	Rev
A4	<Doc>	<RevCode>

Date: Friday, May 18, 2018 Sheet 1 of 1

Vedlegg H

Spenning kilde i kretsen.

- 24 VDC serielt koble mellom to 12 VDC batteri
- 48 VDC fra omformer
- 5 VDC fra omformer
- Det er felles jord på alle komponenter

Proximity kobling til Raspberry.

- Proximity får inngangspenning på 24VDC
- +Vin går til 24 VDC
- – Vin går til felles jord
- Proximity blir koblet som NO (normaly open). Pinne NO går til spenningsdeler
- NO pinne går til R1.
- Fra R1 går utgangsignal fra NO til proximity til Raspberry inngangsignal 37.
- R2 går til felles jord

Encoder kobling til Raspberry.

- Encoder får inngangspenning på 5 VDC
- – Ub går til felles jord
- + Ub går til 5 VDC omformer
- A utgangsignal fra encoder går til Raspberry inngang 36
- B utgangsignal fra encoder går til Raspberry innhgang 37

Motor kobling til Raspberry.

- ClearPath motor får inngangspenning på 48 VDC
- GND fra motor går til felles jord
- Ena+ inngangsignal fra motor går til Raspberry utgangsignal 11
- Ena – inngangsignal fra motor går til fells jord.
- InA+ inngangsignal fra motor går til Raspberry utgangsignal 13
- InA – inngangsignal fra motor går til fells jord
- InB+ inngangsignal fra motor går til Raspberry utgangsignal 15
- InB – inngangsignal fra motor går til fells jord
- HLFB+ inngangsignal fra motor blir ikke brukt, dette er encoder signal fra motor
- HlFB- inngangsignal fra motor går til felles jord.

RF til ISO3080W.

- +Tx fra RF utganngsignal RS422 til inngannggsignal A på ISO3080W
- -Tx fra RF utganngsignal RS422 til inngannggsignal B på ISO3080W
- +Rx fra RF utganngsignal RS422 til inngannggsignal Z på ISO3080W
- -Rx fra RF utganngsignal RS422 til inngannggsignal Y på ISO3080W

ISO3080W til Raspberry.

- ISO3080W pinne 1 går til Raspberry pinne 1 (+ 3.3 VDC)
- ISO3080W pinne 2 går til inngangspenning +5 VDC
- ISO3080W pinne 6 går til felles jord
- TxD utgangsignal fra ISO3080W til Raspberry inngangsignal 8
- RxD utgangsignal fra ISO3080W til Raspberry inngangsignal 10
- Re utgangsignal fra ISO3080W til Raspberry inngangsignal 12

Ultrasonic sensor kobling til Raspberry.
- +Vin til UltraSonic 1 går til inngangspenning +5VDC
- +Vin til UltraSonic 2 går til inngangspenning +5VDC
- -Vin til UltraSonic 1 går til felles jord
- -Vin til UltraSonic 2 går til felles jord
- Trig til UltraSonic 1 inngangsignal går til Raspberry utgangsignal 16
- Echo til UltraSonic 1 utgangsignal går til Raspberry inngangsignal 18
- Trig til UltraSonic 2 inngangsignal går til Raspberry utgangsignal 22
- Echo til UltraSonic 2 utgangsignal går til Raspberry inngangsignal 24

Voltage divider eller spenningsdeler.

Siden proximity sensor få 24 VDC som inngangsspenning vil også utgangsspenning til Raspberry Pi 3 også få 24 VDC signal, dette føre til overspenning til Raspberry Pi og kan gi en ustabilt signal eller i verste fall kortslutte Raspberry. Får å forhindre dette ble det satt inn spenning fordeler som er R1 og R2 i fig(xx).

Formel får spenningfordeler : Vout = Vin * (R2/Ri + R2)

Ved å bruker 100k som R1, spenning inn er på 24 VDC og vi vil ha 3.3 VDC som utgangsspenning.

$$R1 = 100k\omega \tag{11.1}$$

Vin = 24 VDC ,
Vout = 3.3 V

$$\frac{Vout}{Vin} = \frac{R2}{R1 + R2} \tag{11.2}$$

$$(\frac{Vout}{Vin}) * (R1 + R2) = R2 \tag{11.3}$$

$$(\frac{3.3}{24}) * (R1 + R2) = R2 \tag{11.4}$$

$$R2 = 0.1375 * (R1 + R2) \tag{11.5}$$

$$R2 = 0.1375 * R1 + 0.1375R2 \tag{11.6}$$

$$R2\check{}0.1375R2 = 0.1375 * 100k\omega \tag{11.7}$$

$$0.8625R2 = 13750\omega \tag{11.8}$$

$$R2 = 13750/0.825 \tag{11.9}$$

$$R2 = 1594216k \tag{11.10}$$

Utifra ligning vil R2 ha en verdi på 16 k får å få 3.3 VDC som utgangsspenning fra proximity

Vedlegg I – MotorPWM

```
import RPi.GPIO as GPIO

from time import sleep

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

Enable = 7                    # BLU-wire ON/OFF
InputA = 12                   # WHT-wire
InputB = 36                   # BLK-wire

GPIO.setup(Enable, GPIO.OUT)       # Enable input
GPIO.setup(InputA, GPIO.OUT)       # InputA input
GPIO.setup(InputB, GPIO.OUT)       # InputB input, PWM signal kabel

D1 = int(input("Distance1: "))      # Types the distance value, changes the variabel by that.
print('Distance1: {}mm'.format (D1)) # Print the value

pwm = GPIO.PWM(InputB, 100)         # Access PWM signals
pwm.start(0)                  # Gives the start procentage
pwm.ChangeDutyCycle(100)           # Decide the procntage value for the velocity
GPIO.output(Enable, GPIO.HIGH)      # ON/OFF, when its GPIO.HIGH its ON.
GPIO.output(InputA, GPIO.LOW)       # Pin for backwards direction, when GPIO.HIGH goes
motor backwards.
GPIO.output(InputB, GPIO.HIGH)      # Pin for forwards, when GPIO.HIGH it goes forwards.
sleep(0.05)                   # Time for how long its gonna wait in this loop

GPIO.cleanup()
```

Vedlegg J

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
from time import sleep
import math
GPIO.setwarnings(False)

# for Encoder
InA = 36
InB = 35
antall = 0
lengde = 0.107
avstand1 = 0

# for Proximity
proxi = 37
avstand2 = 0
avstand = 0
antallProxi = 0
proxypa = 0
proxyav = 0

# For Motor input
Enable = 7 # BLU-wire ON/OFF
InputA = 12 # WHT-wire Forward/Backward
InputB = 16 # BLK-wire Configurate PWM value

GPIO.setup(Enable, GPIO.OUT) # Enable input
GPIO.setup(InputA, GPIO.OUT) # InputA input
```

```
GPIO.setup(InputB, GPIO.OUT) # InputB input
pwm = GPIO.PWM(InputB, 100) # Access PWM signals
pwm.start(0) # PWM starts at 0

# For input distanscer

Dis1 = int(input("tast inn distance 1: "))

GPIO.setup(proxi, GPIO.IN)
GPIO.setup(InA, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(InB, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

counter = 0
clkLastState = GPIO.input(InA)

def forward() :
   GPIO.output(Enable, GPIO.HIGH)    # ON/OFF, when its GPIO.HIGH its ON.
   GPIO.output(InputA, GPIO.LOW)     # Pin for backwards direction, when GPIO.HIGH goes
motor backwards.
   GPIO.output(InputB, GPIO.HIGH)    # Pin for forwards, when GPIO.HIGH it goes forwards.

def backward() :
   GPIO.output(Enable, GPIO.HIGH)
   GPIO.output(InputA, GPIO.HIGH)
   GPIO.output(InputB, GPIO.HIGH)

def stop() :
   GPIO.output(Enable, GPIO.LOW)
   GPIO.output(InputA, GPIO.LOW)
   GPIO.output(InputB, GPIO.LOW)
```

```
while avstand < Dis1 :
        # For encoder
        AvstandIgjen1 = Dis1 - avstand
        clkState = GPIO.input(InA)
        dtState = GPIO.input(InB)
        #print(str(clkLastState) + str(dtState))
        if clkState != clkLastState:
            if dtState != clkState:
                counter +=1

            else:
                counter -=1
            #print (counter)
            avstand1 = counter * lengde
            avstand = avstand1 + avstand2

            print("avstand i mm: " + str(avstand), "Avstand Igjen: " + str(AvstandIgjen1))
            clkLastState = clkState

        # For Proximity
        if GPIO.input(proxi)==1 :
            proxypa = 1

        else:
            proxyav = 1
            if (proxyav + proxypa  == 2):
                antallProxi += 1
                proxyav = 0
```

```
        proxypa = 0
        counter = 0
        avstand2 = antallProxi * 500
        #print (" Avstand 2 i mm: ",avstand2)
    if avstand < Dis1 :             # When a distance value gets a variable
      forward()                  # Forward defined in the code
      if 100 < AvstandIgjen1 < 200: # When the distancevalue is between 100 and 200
        print ("Break 30")       # Prints the DutyCycle when its in this loop
        pwm.ChangeDutyCycle(50)   # Changes the velocity to 50%
        forward()
        sleep(0.01)              # Gives signal/value every 1/100 second.
      if 0.2 < AvstandIgjen1 < 100: # When the distancevalue is between 100 and 0.2
        print ("Break 10")
        pwm.ChangeDutyCycle(30)   # Changes the velocity to 50%
        forward()
        sleep(0.01)              # Gives signal/value every 1/100 second.
      if AvstandIgjen1 < 0.2:       # When the distancevalue is lower than 0.2
        print ("Break 0")
        pwm.stop()               # Sets the velocity yo 0%, which means stop.
        stop()                 # Stop defined in the code
        sleep(0.01)              # Gives signal/value every 1/100 second.
```

Vedlegg K

```
# The motor accelerates, drives on konstant speed then breaks.

import RPi.GPIO as GPIO # Changes the name for pins on Raspberry pi 3

from time import sleep # Imports sleep function from time library
import math # Imports funksjons

GPIO.setmode(GPIO.BOARD) # To use the physical pins on the board
GPIO.setwarnings(False) # Warnings dosent show

Enable = 7 # BLU-wire ON/OFF
InputA = 12 # WHT-wire Forward/Backward
InputB = 16 # BLK-wire Configurate PWM value

GPIO.setup(Enable, GPIO.OUT) # Enable input
GPIO.setup(InputA, GPIO.OUT) # InputA input
GPIO.setup(InputB, GPIO.OUT) # InputB input

pwm = GPIO.PWM(InputB, 100) # Access PWM signals
pwm.start(0) # PWM starts at 0

D1 = int(input("Distance1: "))
print('Distance1: {}mm'.format (D1))



                        # Acceleration
counter = 0                     # Variable for count from 0-100 and 100-0.
counterhold = 0                 # Variable to hold the value 100.
```

```
while (counter < 100):              # counter variable starts from 0 goes to 100.
   print(counter)                   # Prints counter on screen.
   counter = counter + 1            # This adds one to counter variable.
   pwm.ChangeDutyCycle(counter)     # This changes DutyCycle (velocity for motor) with
percentage value.
   GPIO.output(Enable, GPIO.HIGH)   # ON/OFF, when its GPIO.HIGH its ON.
   GPIO.output(InputA, GPIO.LOW)    # Pin for backwards direction, when GPIO.HIGH goes
motor backwards.
   GPIO.output(InputB, GPIO.HIGH)   # Pin for forwards, when GPIO.HIGH it goes forwards.
   sleep(0.005)                     # The each time counter value adds to itself, so it takes 0.5
seconds from 0-100.


                                    # Holds the constant velocity
counterhold = counter               # counterhold now has the value 100.
while(counterhold > 0):             # Kode will be in this loop until the seconds goes in sleep
function.
   print(counter)
   counterhold = counterhold - 1    # counterhold goes from 100 to 0, but the while function
is for every value.
                                    # so it shows 100 every time since its counter variable
   pwm.ChangeDutyCycle(counter-1)   # Need to have the percentage value 99 to make it
move.
   GPIO.output(Enable, GPIO.HIGH)
   GPIO.output(InputA, GPIO.LOW)
   GPIO.output(InputB, GPIO.HIGH)
   sleep(0.05)
                                    # Breaking
while (counter > 0):                # When it goes in this loop the counter value goes from 100
to 0
   print(counter)
   counter = counter - 1            # This subtracts from counter value every time.
   pwm.ChangeDutyCycle(counter)
```

```
GPIO.output(Enable, GPIO.HIGH)
GPIO.output(InputA, GPIO.LOW)
GPIO.output(InputB, GPIO.HIGH)
sleep(0.005)
```

Vedlegg L

For at man skal kunne opprette og utvikle en database, så må det settes opp en server. Noen steg til hvordan det gjøres følger her.

Microsoft SQL server kan lastes ned fra Microsoft sin hjemmeside. Ved valg av developer edition vil du bli gitt et valg mellom å installere en såkalt «basic» versjon eller «custom». Velg custom her og du vil bli sendt til et installasjonssenter. Velg «installation» i menyen til venstre og deretter «New SQL Server stand-alone installation or add features to an existing installation».

I «SQL server 2017 setup» som nå dukker opp vil det ikke være nødvendig å gjøre noe før man når «feature selection» hvor man velger Database Engine Services. I «instance configuration» navngir man serveren og gir den en ID. I «database engine configuration» velger man metode for innlogging på serveren, som brukes for eksempel til SQL Server Management Studio, programmet hvor man oppretter databaser og tabeller. Velg Mixed Mode her og legg inn ønsket passord.

Etter fullført installasjon velger du i installasjonssenteret «Install SQL Server Management Tools». Du vil da bli sendt til en side med link til den nyeste versjonen av SQL Server Management Studio. Når du trykker linken får du en setup-fil som må lagres. Åpne da denne lagrede filen og bare trykk på installer. Når programmet er installert, åpner du SQL Server Management Studio og logger deg på med Servernavn (PCNAVN\navnet man ga serveren i installasjonen), velger SQL Server Authentication, setter login som «sa», og entrer passordet du valgte i installasjonen.

Under følger linken som ble brukt til å installere SQL Server:

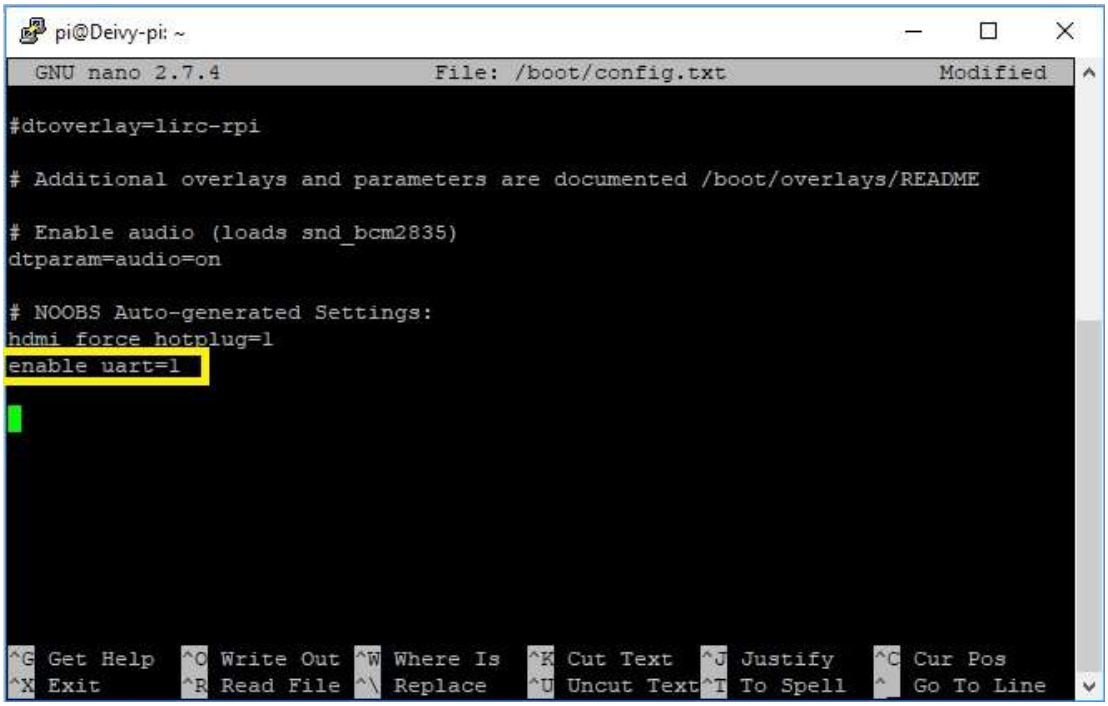
https://www.youtube.com/watch?v=yasfZuou3zI

Vedlegg M

Stegene for å konfigurere UART pinner på Raspberry Pi

Steg 1 - Aktivering av UART pinner på Raspberry Pi

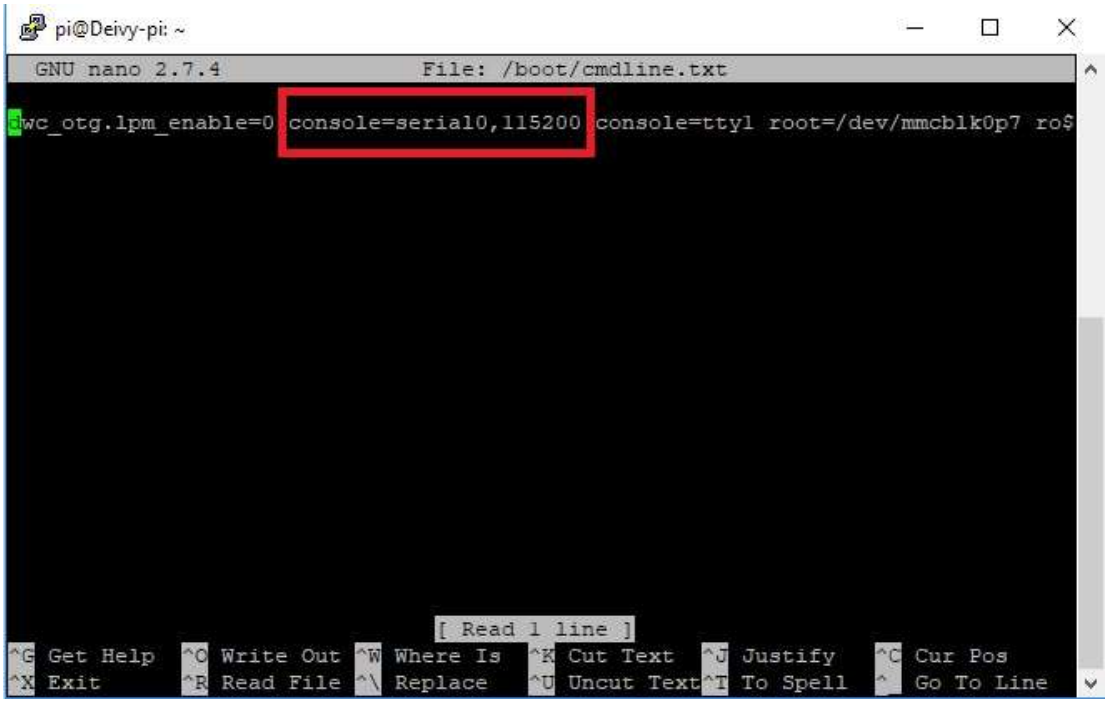
Skrive «sudo nano /boot/config.txt» i kommandoterminalen. Funksjon «enable_uart=1» skal skrives inn for å aktivere UART pinner helt nederst i filen på Raspberry Pi Slik det er vist på fig 10.



Figur 10 – Aktivering av UART pinner.

Steg 2 - Endring i oppstartfase for UART

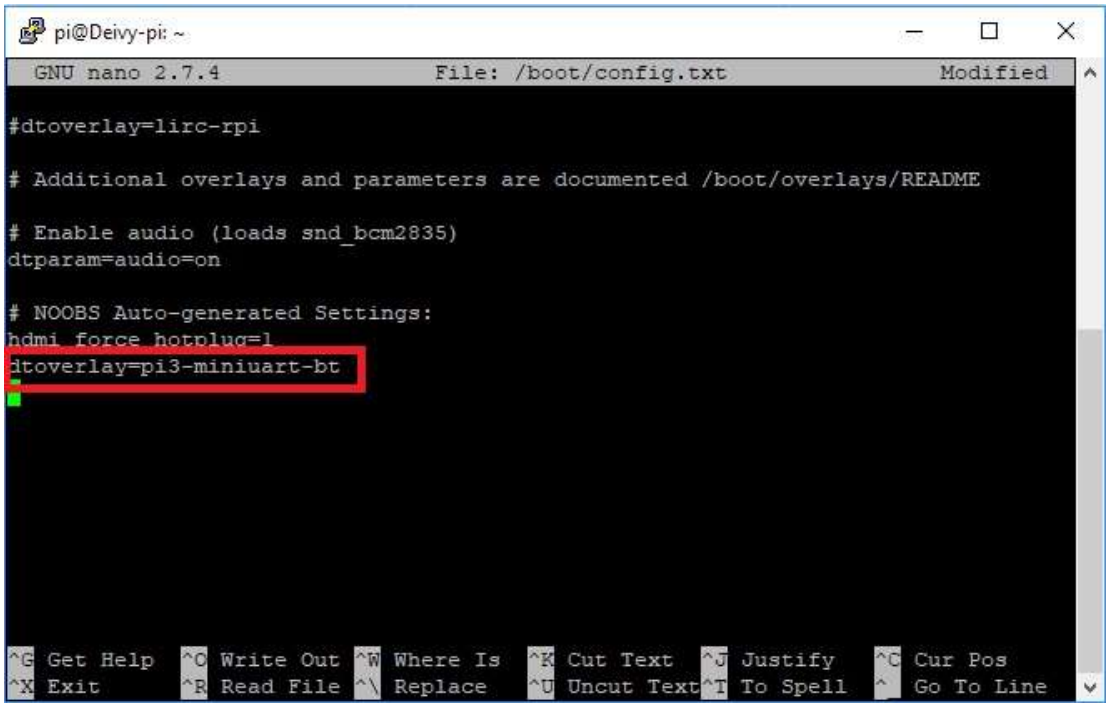
Skrive «sudo nano /boot/cmdline.txt» i kommandoterminalen til Raspberry Pi. Fjerne funksjon «console=serial0, 115200» som stopper automatisksjekk ved oppstart slik det er vist på fig 11.



Figur 11 – Endring i oppstarts sekvens

Steg 3 - Flytting av Bluetooth

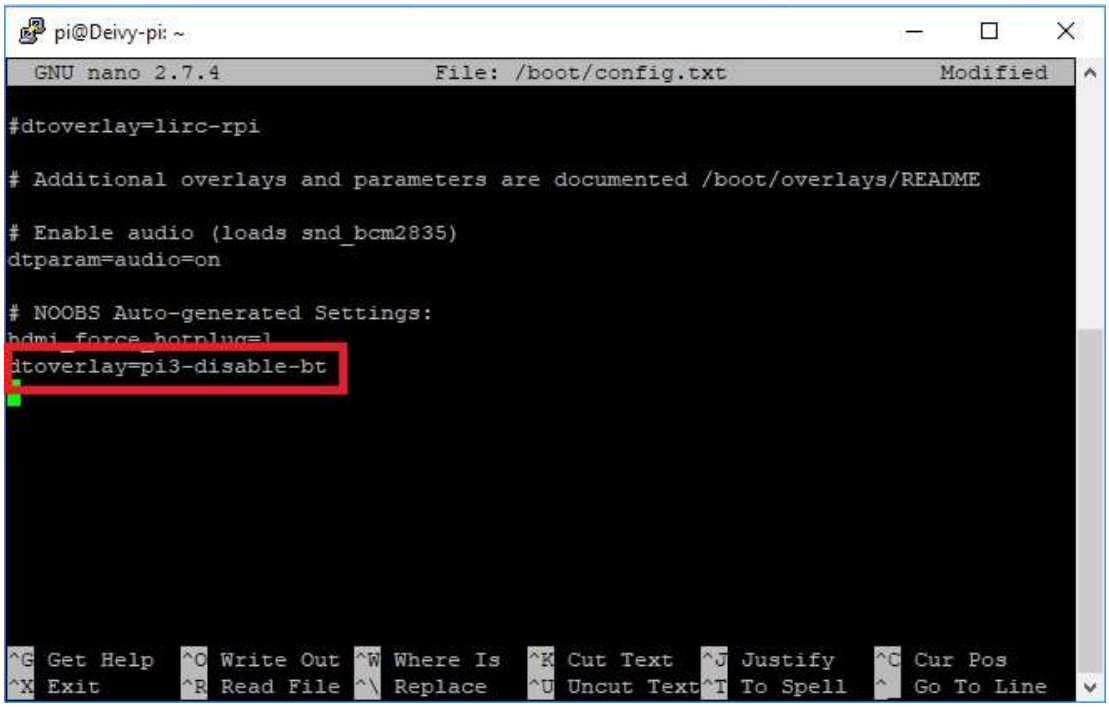
Skrive «sudo nano /boot/config.txt» i kommandoterminalen til Raspberry Pi. Funksjon «dtoverlay=pi3-miniuart-bt» skal skrives inn nederst i filen for å flytte Bluetooth tjenesten til mini UART (ttyS0) slik det er vist på fig 12.



Figur 12 – Flytting av Bluetooth

Steg 4 (valgfritt) – Fjerning av Bluetooth for UART.

Skrive «sudo nano /boot/config.txt» i kommandoterminalen til Raspberry Pi. Funksjon «dtoverlay=pi3-disable-bt» skal skrives inn nederst i filen for å deaktivere Bluetooth tjenesten slik det er vist på fig 13.



Figur 13 – Fjerning av Bluetooth tjeneste

Vedlegg N

```
import RPi.GPIO as GPIO
import time

# Turn off minor warrnings
GPIO.setwarnings(False)
# Reset pins from last use
GPIO.cleanup()
# Set GPIO pins in Board mode
GPIO.setmode (GPIO.BOARD)

# Pins on board for Ultrasonic
TRIG = 16
ECHO = 18

# Pins on board for LED
RED = 22

# Ultrasonic pins setup
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

# Stop variable in cm
Stop = 20

#LED setup
GPIO.setup(RED, GPIO.OUT)
# Set LED to low
def RED_LOW():
    GPIO.output(RED, GPIO.LOW)
# Set LED to High
def RED_HIGH():
    GPIO.output(RED, GPIO.HIGH)

# Distance function
def get_distance():
   # Turn Trig pin on for a short time
   GPIO.output(TRIG, True)
   time.sleep(0.0001)
   GPIO.output(TRIG, False)

   # Listen to Trig signal and derive time of travel
   while GPIO.input(ECHO) == False:
      start = time.time()

   while GPIO.input(ECHO) == True:
      end = time.time()

   sig_time = end-start

   # Convert signal time to cm and return value
```

```
    distance = sig_time / 0.000058
    return distance

# Variable with random value to enter while loop
while (100 > Stop):
    # Call for get_distance function with new variable
    distance = get_distance()
    # Pause for one second
    time.sleep (1)
    # If distance in cm is desired uncomment line under
    #print('Distance: {0:.1f} cm'.format(distance))
    # If distance to object is more then Stop variable, print "Go" and set red LED to low
    if (distance > Stop):
        print('Go')
        RED_LOW()

    # If distance to object is less then Stop variable, print "Stop" and set red LED to high
    elif (distance < Stop):
            print('STOP')
            RED_HIGH()
```